SECURITY TECHNIQUES FOR DRONES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jongho Won

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Elisa Bertino, Chair
School of Science

Dr. Dongyan Xu
School of Science

Dr. Ninghui Li
School of Science

Dr. Sonia Fahmy
School of Science

Approved by:

Dr. Voicu Popescu
Head of the School Graduate Program

To my beloved wife, Kangeun Lee

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude towards my advisor Prof. Elisa Bertino for her continuous support and guidance during my graduate studies. I have benefited greatly from her knowledge and advice, which was instrumental in the completion of my PhD. I would like to thank the members of my dissertation committee, Prof. Dongyan Xu, Prof. Ninghui Li, and Prof. Sonia Fahmy for their constructive comments and feedback. Special thanks goes to Prof. David Yau for his initial support and guidance.

During my graduate studies, I was also fortunate to collaborate with many current and past members of the Cyber Space Security Lab. I also wish to express my sincerest appreciation and thanks to ETRI and Dr. Seung-hyun Seo in Korea.

My internships at VMWare and Microsoft Research have been a great learning experience for me. I thank my mentors, Dr. Greg Bollella and Dr. Ranveer Chandra.

I would like to dedicate this work to my wife, Kangeun Lee, and my two children, Claire and Ethan. Although our stay was only for hard work, they were always patient and the reason for my happiness. Hence, I can never claim that this dissertation is a result of my solo exertion. Finally, to my parents, I thank you for the sacrifices you made for me. I have come to deeply believe that my most significant accomplishments are seldom achieved without your support.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AES	Advanced encryption standard
ECC	Elliptic-curve cryptography
HMAC	Hash message authentication code
ID-PKC	ID-based public-key cryptography
MMSE	Minimum mean square error
TPM	Trust Platform modules
UAV	Unmanned aerial vehicle
WBC	White-box cryptography
WBE	White-box encryption
WSN	Wireless sensor network

ABSTRACT

Won, Jongho Ph.D., Purdue University, May 2019. Security Techniques for Drones. Major Professor: Elisa Bertino Professor.

Unmanned Aerial Vehicles (UAVs), commonly known as drones, are aircrafts without a human pilot aboard. The flight of drones can be controlled with a remote control by an operator located at the ground station, or fully autonomously by onboard computers. Drones are mostly found in the military. However, over the recent years, they have attracted the interest of industry and civilian sectors. With the recent advance of sensor and embedded device technologies, various sensors will be embedded in city infrastructure to monitor various city-related information. In this context, drones can be effectively utilized in many safety-critical applications for collecting data from sensors on the ground and transmitting configuration instructions or task requests to these sensors.

However, drones, like many networked devices, are vulnerable to cyber and physical attacks. Challenges for secure drone applications can be divided in four aspects: 1) securing communication between drones and sensors, 2) securing sensor localization when drones locate sensors, 3) providing secure drone platforms to protect sensitive data against physical capture attacks and detect modifications to drone software, and 4) protecting secret keys in drones under white-box attack environments.

To address the first challenge, a suite of cryptographic protocols is proposed. The protocols are based on certificateless cryptography and support authenticated key agreement, non-repudiation and user revocation. To minimize the energy required by a drone, a dual channel strategy is introduced. To address the second challenge, a drone positioning strategy and a technique that can filter out malicious location references are proposed. The third challenge is addressed by a solution integrating

techniques for software-based attestation and data encryption. For attestation, free memory spaces are filled with pseudo-random numbers, which are also utilized to encrypt data collected by the drone like a stream cipher. A dynamic white-box encryption scheme is proposed to address the fourth challenge. Short secret key are converted into large look-up tables and the tables are periodically shuffled by a shuffling mechanism which is secure against white-box attackers.

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs), commonly known as drones, aircrafts without a human pilot aboard. The flight of drones may be controlled: either by a remote control from an operator, located on the ground station, or fully autonomously by onboard computers. They are often preferred for missions that are too "dull, dirty or dangerous" for manned aircrafts and mostly for military applications. However, over the recent years, drones have attracted the interest of industry and civilian sectors due to their versatility and maneuverability. Recent drones are often equipped with advanced sensor technologies, such as GPS, gyroscope, accelerometer, magnetometer and ultra-sonic sensors, and can thus be accurately controlled by remote operators. With the advance of drone software technologies, such as obstacle avoidance using computer vision techniques and geo-fencing, even hobbyists can fly drones safely.

Due to these reasons, drones are considered effective solutions for various civilian tasks, such as precision agriculture, search/rescue, structural health monitoring and surveillance. In many of these applications, they are used as mobile data collectors. For instance, using their cameras, drones can take high-precision photos and videos of large farming areas to analyze the health of crops or collect soil-moisture level data from pre-installed on-ground sensors. Sometimes, a drone itself can be a mobile sensor to measure the air quality, temperature/humidity or wind information in different locations. Also, drones can be utilized as mobile beacons to localize small physical objects embedding sensors and network interfaces.

Recently, the Wisconsin state police reported the use of drones for search and rescue missions. Microsoft CityNext partner DEA Drones [1] plans to use flying drones as an emergency medical response system for cities. PrecisionHawk [2] has been offering remote sensing and data processing services using drones for various applications such as precision agriculture, infrastructure monitoring and search/rescue. With the

recent advance of sensor and embedded device technologies, various sensors will be embedded in city infrastructure such as roads, traffic signals, sidewalks and bridges to monitor various city-related information, such as traffic conditions, air quality and structural health. In this context, drones can be effectively utilized in many safety-critical applications such as critical infrastructure monitoring, surveillance and alarm. For example, in structural health monitoring [3], 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. Even cars may play a role of sensors in roads. Drones can collect traffic information from cars to enhance travel efficiency and physical safety. Also, drones can be used for communicating with the sensors in order to send the sensors re-configuration instructions, such as instructing the sensors to change the sampling rates. In such context, drones represent a key technology for deploying novel monitoring applications.

Challenges for secure drone applications can be divided in four aspects: 1) securing communication between drones and sensors, 2) securing sensor localization when drones locate sensors, 3) providing a secure drone platform to protect sensitive data against physical capture attacks and detect modifications of drone software, and 4) protecting secret keys in drones under white-box attack environments.

1.1 Challenges for Secure Communications between Drones and Sensors

In many current and foreseen applications involving drones, including the Internet of Things (IoT), security is an important requirement. Drones, as many computing devices, are vulnerable to malicious attacks such as impersonation, manipulation, and interception. It is thus critical to address security requirements, such as authentication, non-repudiation, confidentiality and integrity, and securing communications between drones and other devices (referred to as smart objects in what follows), such as on-ground sensors. An important security building block is represented by cryp-

tography, which in turn requires a key management scheme. However, implementing a key management scheme suitable for wireless sensor networks (WSNs) that involve both smart objects and drones is quite challenging because of (1) the mobility and limited flight time of drones and (2) the constrained resources of smart objects. Most encryption key management schemes proposed for WSNs adopt a symmetric-key-based approach instead of an asymmetric-key-based approach in order to address the limited energy and processing capability of sensors [4, 5]. However, the symmetric-key-based approach suffers from high communication overhead and requires large amounts of memory space to store the shared pairwise keys. Also such approach is not scalable, not resilient against compromises, and unable to support adequate node mobility. Public key cryptography (PKC) is relatively more expensive than symmetric key encryption in terms of computational costs, but recent improvements in the implementation of elliptic curve cryptography (ECC) [6] have demonstrated the practical applicability of PKC to WSNs. In order to enhance scalability and flexibility, asymmetric key based approaches that use ECC and identity-based PKC have been proposed for WSNs [7–9]. However, ECC-based schemes with certificates [8] and pairing operation-based ID-PKC [7, 9] schemes, when directly applied to WSNs, suffer from certificate management overhead and computational overhead from pairing operations, respectively. Moreover, since drones, unlike sensors, are likely to record a wide range of information, they can become a target for physical capture. We thus need an approach to minimize information leakage in the event that attackers capture a drone.

1.2 Challenges for Secure Localization

Since sensors are usually deployed in unattended areas, their estimated locations can be severely distorted by even simple attacks such as replay attacks. For example, if the distance between a sensor and a beacon node is measured by the Received Signal Strength Indicator (RSSI), an attacker can reduce the distance by replaying

beacon signals with a strong signal strength. As a result, the attacker can break the location-based functions of WSNs. Another possible attack can be carried out against autonomous vehicles such as drones and self-driving cars equipped with GPS receivers. They are localized by receiving ranging signals from satellites. However, the GPS for civilian use does not provide any security measures such as encryption and authentication. Indeed, researchers have demonstrated the feasibility of attacks on real-world positioning systems and provided mitigation strategies. Zeng et al. [10] showed that a portable GPS spoofer can easily manipulate a navigation route of a car and guide the car to a wrong destination without being detected. Cho et al. [11] successfully demonstrated that the location-based ordering service provided by Starbucks is vulnerable to replay attacks. Tippenhauer et al. [12] showed that skyhook [13] which is a public WLAN-based positioning system is vulnerable to location spoofing attacks by jamming/replaying localization signals and by tampering with the service database.

1.3 Challenges for Building a Secure Drone Platform

Since drones may move around in unattended hostile areas with collected sensor data, they are vulnerable to cyber and physical attacks. An attacker can obtain secret keys and collected data in a drone by physically capturing the drone and then analyzing its memory. In addition, an adversary can install some malicious code on a drone by physically capturing a drone or using software vulnerabilities. The hidden malicious program can tamper the stored in the drone, or steal valuable data. Even worst, terrorists may make a compromised drone crash on humans, on a building or against a civil passenger airliner. It is thus critical to be able to detect compromised drones in order to reduce potential dangers and damages. An approach to detect compromised drones is through code attestation, i.e., the ground station or other drones verify that a given drone is still running the initial application and, hence, has not been compromised. To guarantee the integrity of the execution environment,

tamper-resistant security hardware such as Trust Platform Modules (TPM) [14] can be utilized. However, even a small increase in per device cost leads to a significant increase in overall production costs of high-volume drone manufacturing. In addition, the security hardware-based approach does not cover most of nowadays commercially available drones, such as drones for hobbyists, since they are not equipped with such security hardware.

An alternative to the use of TPM is the use of software-based attestation. Such technique attests code, static data, and configuration settings of an embedded device without requiring dedicated hardware and physical access to the device. Clearly, a software-based attestation technique will incur lower cost than an attestation technique that requires additional hardware. More importantly, a technique that works entirely in software can be used on legacy drones and is easily updatable. Due to such advantages, several software-based attestations techniques have been proposed [15–20] for resource-constrained embedded devices like sensors. The party performing the attestation (verifier) is physically distinct from the device being verified. Hence, the verifier cannot directly read or write the devices memory. Software-based attestation requires an external verifier since, without secure hardware, a (potentially) compromised device cannot be trusted to verify itself correctly. Most software-based attestation techniques are based on challenge-response protocols between the verifier (a ground station) and a prover (a target device). The verifier sends a challenge to the target device to prevent replay or pre-computation attacks. The target device computes a response to this challenge, using a verification procedure that is either pre-programmed into the embedded devices memory or downloaded from the verifier prior to verification. The verifier can locally compute the answer to its challenge, and can thus verify the answer returned by the target device. Since the verifier is assumed to know the exact memory contents and hardware configuration of the prover, it can compute the expected response and compare it with the received one. If the values match, the target device is genuine; otherwise, it has most likely been compromised. The existing software-based attestation schemes can be divided into four categories:

schemes based on the program counter [15, 16], schemes based on response time estimation [17], scheme based on self-modifying code [18], and scheme based on filling empty memory space [19, 20].

However, such software-based attestation techniques cannot be directly applied to drones due to the following reasons. First, not all platforms make the program counter available to software. Only a few micro-controllers, like the AVR micro-controllers, provide application software access to the program counter. Second, schemes based on response time, such as SWATT-based attestation, require very precise estimation of response times are and thus very sensitive to unpredictable network delays and also dependent from hardware platforms. Considering that the platforms of drones are diverse and the wireless communication channel between a drone and the ground station varies according to the network conditions, such as network traffic congestions or packet collisions, the timing-based approach is not suitable to drones. Third, as mentioned in [21], schemes based on self-modifying code are slow and notoriously difficult to implement, and thus are a questionable design choice for an attestation protocol for drones. Finally, drones need to collect data and store them in the program memory. Therefore, if the empty memory space is filled with pseudo-random numbers so that adversary would have no empty space to store its malware like in [19, 20], the drones cannot store collected data such as photos and videos.

In addition, code attestation itself does not guarantee the confidentiality of the collected data and of the messages exchanged between the drone and the ground station. An adversary may just want to steal secret key information from the drone and eavesdrop messages by decrypting them using the secret key. In the context of the white-box attack model, i.e., in untrusted execution environments, the traditional symmetric key-based cryptographic primitives with short secret keys, such as AES and HMAC, do not guarantee confidentiality and integrity. White-box attackers can see and manipulate the internal state of the memory of a victim device, by installing a malware program, or by capturing the device and analyzing the memory. Various white-box attacks, such as entropy attack [22], cold boot attack [23], and S-box

blanking attack [24], have shown that a short secret key, such as 128-bit AES key, could be successfully extracted. Using slightly longer secret keys may not be a viable solution since white-box attackers can carry the same attacks to extract the longer keys.

1.4 Challenges for Protecting Secret Keys in Drones under White-box Attack Environments

Like many networked computing devices, drones can be victims of traditional attacks, referred to as black-box attacks in what follows, such as eavesdropping, manipulation, replay attacks and man-in-the-middle attacks. Furthermore, attackers can launch stronger attacks, called white-box attacks. Researchers have already discovered vulnerabilities in consumer drones and demonstrated how to hijack them [25–29]. For instance, Maldrone [26] is the first drone malware that can be installed on drones while they are flying and allows attackers to take control of the drones. In addition, attackers may be able to launch firmware modification attacks [30] by utilizing reverse engineering tools [31, 32] since most drones available on the market support firmware upgrades. Moreover, since many drones are based on open-source software [33–35], attackers may be able to exploit known vulnerabilities in such software. Once attackers succeed in installing malware on the drones, their computing environments become untrusted. As a result, the attackers can steal secret keys from the drones and deceive data users into making incorrect decisions.

The concept of white-box cryptography (WBC) [36] was introduced in 2002 to protect software implementations of cryptographic algorithms in untrusted environments that are not equipped with hardware-assisted security mechanisms, such as the Trusted Platform Module [14] and the ARM TrustZone [37]. By untrusted environment, we refer to an environment in which the attacker has complete control of the device. Although WBC is originally intended for digital rights management (DRM), its applications are expanding to mobile devices and IoT devices [38, 39].

Recently, WBC has attracted attention from industry [40] since it does not require specialized hardware. In fact, even a small increase in per device cost leads to a significant increase in overall production costs of high-volume drone/robot manufacturing. In addition, white-box cryptography can be utilized on legacy systems, and can be upgraded by software updates.

In the context of the white-box attack model, i.e., in untrusted execution environments, traditional symmetric key-based cryptographic primitives with short secret keys, such as AES and HMAC, do not guarantee confidentiality and integrity. A white-box attacker can see and manipulate the internal state of the memory of a victim drone by obtaining a root privilege and installing malware. More seriously, the malware may be able to locate a short secret key, such as an 128-bit AES key or a private key for public-key cryptography, and send it to the remote attacker.

To protect the confidentiality of secret keys in such a white-box environment, several white-box cryptography solutions [36, 41–44] have been proposed. Such solutions hide a short key by converting it into one or more large look-up tables in order to make it hard for an attacker to extract the short secret key from the look-up table(s).

Although the existing white-box cryptography solutions provide a certain level of security against white-box attacks[1], none of them provide a method to securely change the look-up table after it is initialized. Therefore, once a white-box attacker succeeds in extracting a part of the look-up table from a vehicle, the attacker is able to permanently use the extracted partial table to decrypt/encrypt ciphertexts/plaintexts until the user is able to change the look-up table. Also, the attacker can decrypt all the past communications since the table is static. However, changing the look-up table in the white-box environments is not easy since the attacker can see the internal memory state by launching white-box attacks, while the look-up table is being changed.

[1]In fact, most existing solutions except [43, 44] fail to achieve a practical level of white-box security although they provide a competitive level of black-box security. In the white-box environments, an attacker can extract a short secret key in executable work-steps [45–49].

2 CERTIFICATELESS CRYPTOGRAPHIC PROTOCOLS FOR EFFICIENT DRONE-BASED SMART CITY APPLICATIONS

2.1 Introduction

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 chapter, we address all those requirements by designing, implementing, and testing a suite of efficient cryptographic protocols.

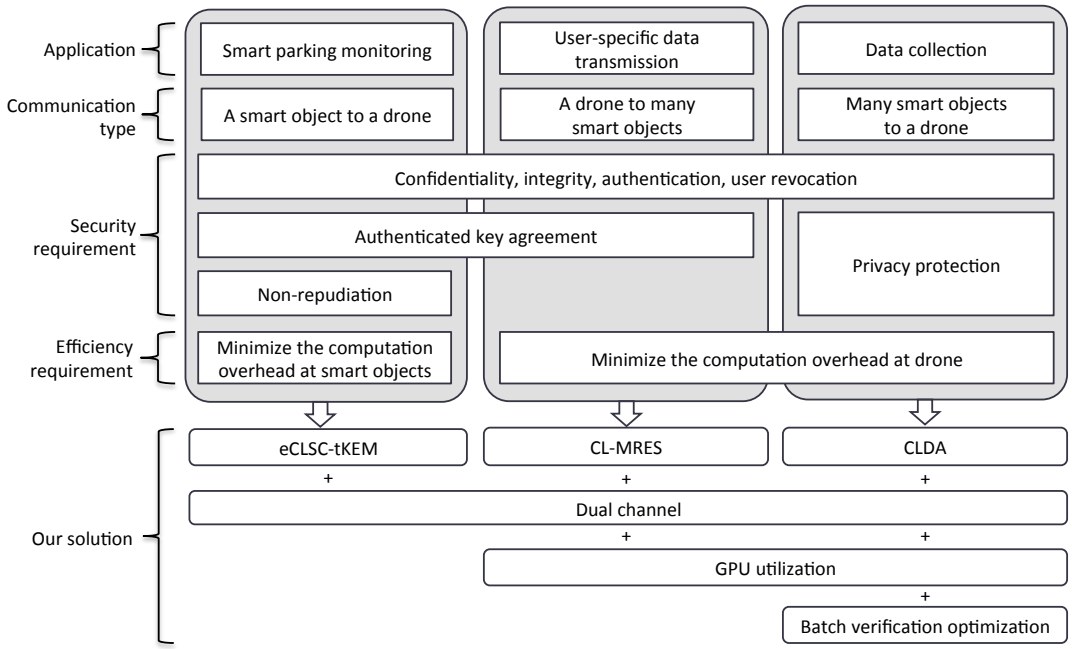


Figure 2.1.: Requirements and our solutions

2.2 Contributions and Protocol Overview

The contributions of the proposed solutions 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.

A suite of cryptographic protocols: As shown in Fig. 2.1, we consider three different communication types between a drone and smart objects, and their corresponding applications: 1) a smart object $\rightarrow$ a drone (secure monitoring), 2) a drone $\rightarrow$ many smart objects (user-specific data transmission), and 3) many smart objects $\rightarrow$ a drone (data collection). Fig. 2.1 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. 2.6.2 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) [50], 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 [50] 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 Certificate-Less 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.

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.

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.

2.3 Related Work

2.3.1 Mobile Data Collectors in WSN

Several studies [51–54] 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. [51] 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. [52] 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. [53] 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. [54] 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 chapter, 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 [55, 56] 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.

2.3.2 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) [62]. 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

Table 2.1. 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 [57]	9EM + 1V (8EM + 1V)	yes	yes	no	no
Sun's CL-AKA [58]	6EM (5EM)	yes	yes	no	no
Selvi's CLSC-TKEM [59]	4EM+1P+1EX (3EM+1P+1EX)	yes	yes	yes	no
Seo's CLSC-TKEM [60]	5EM (3EM)	yes	yes	yes	no
eCLSC-TKEM [61]	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.

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 [63].

Several pairing-free CL-AKA protocols [57, 58, 64, 65] 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 [58] and Yang's CL-AKA [57]. Recently, Li et al. [66] 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. [59] showed a security weakness in Li et al.'s CLSC-TKEM and presented an improved CLSC-TKEM. Since both CLSC-TKEM protocols [59,66] rely on bilinear pairing operations, they are not suitable for resource-constrained devices.

Seo et al. first proposed a pairing-free CLSC-TKEM protocol [60] 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 [50]. 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 2.1 summarizes the comparison between eCLSC-TKEM and existing CL-AKA and CLSC-TKEM.

2.3.3 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 [67] 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. [68] 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 [69] introduced the concept of multi-recipient key encapsulation (MR-KEM) and Barbosa et al. [70] 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 [69] 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. [71] 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 chapter, 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.

2.3.4 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 [72, 73]. They mainly focus on the optimized implementations of the Elliptic Curve-based ElGamal (EC-ElGamal) homomorphic encryption on resource-constrained devices. In this chapter, 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.

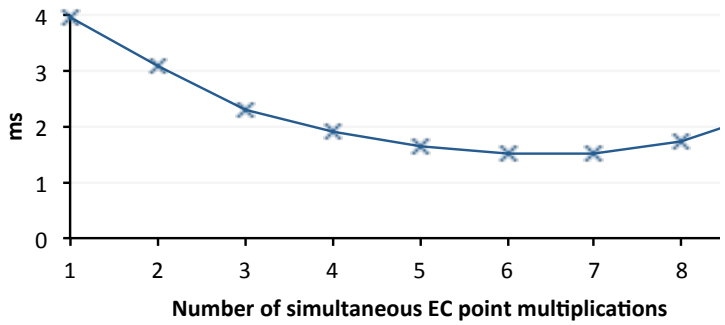


Figure 2.2.: Computation time per EC point multiplication on CPU

2.4 Background

2.4.1 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* [74,75] and *single-thread for one EC point multiplication* [76]. 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.

2.4.2 Simultaneous Multiple EC Point Multiplications

An optimization for simultaneous multiple EC point multiplications [77] 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 [78] ECC library and tested the technique on the CPU of the Nvidia Jetson TK1 developer kit [79]. Fig. 2.2 shows the computation time per EC point multiplication for calculating $\sum_{i=1}^{n} k_i \cdot P_i$, $(n = 1, 2, ..., 9)$ when secp160r1 is utilized for EC curve parameters. As shown in Fig. 2.2, 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.

2.5 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.

2.5.1 eCLSC-TKEM

eCLCS-TKEM meets all the security requirements, i.e., authenticated key agreement (AKA), non-repudiation and user revocation (see Table 2.1), while it minimizes the computational overhead at smart objects. Note that the CL-AKA protocols [57, 58] support only AKA. For non-repudiation, they must be extended with a digital signature scheme. Although the CLSC-TKEM protocols [59,60] 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 $H_0 : \{0,1\}^* \times G_q^2 \times \{0,1\}^* \rightarrow \mathbb{Z}_q^**$, $H_1 : G_q^3 \times \{0,1\}^* \times G_q \rightarrow \{0,1\}^n$, $H_2 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \rightarrow \mathbb{Z}_q^*$, and $H_3 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \rightarrow \mathbb{Z}_q^*$. Here, n is the key length of a 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: $W \cdot P = (d_A + l_A \cdot H + x_A \cdot H') \cdot P$
  $= d_A \cdot P + l_A \cdot P \cdot H + x_A \cdot P \cdot H'$
  $= (r_A + xH_0(ID_A, R_A, P_A, t_A)) \cdot P + U \cdot H + H' \cdot P_A$
  $= R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A$
- **[Decryption step]** Compute
  $T = (d_B + x_B) \cdot V\big(= (d_B + x_B) s_A \cdot P = s_A \cdot Y\big)$,
  $Y = (d_B + x_B) \cdot P\big(= (r_B + xH_0(ID_B, R_B, P_B, t_B) + x_B) \cdot P = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot$

$P_{pub} + P_B\big)$,
$K = H_1(Y, V, T, ID_A, P_A, ID_B, P_B)$, and $DEC_K(\tau)$ to obtain M.

2.5.2 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 in [80]). 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.

HybridEncryption: Given public parameters Ω, a list $L = \{ID_{B_1}, ..., ID_{B_n}\}$ 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 $ID_{B_i} \in L$, $i = 1, 2, ..., n$.
  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 $(V, \tau_1, \tau_2, ..., \tau_n)$.

HybridDecryption: Given ciphertexts $(V, \tau_1, \tau_2, ..., \tau_n)$, a list $L = \{ID_{B_1}, ..., ID_{B_n}\}$ 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.

2.5.3 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 [72] 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 verification: 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 $(\sum_{i=1}^{n} \sigma_i) \cdot P = \sum_{i=1}^{n} \left(R_i + H_0(ID_i, R_i, P_i, t_i)P_{pub}\right) + \sum_{i=1}^{n} H_i \cdot U_i + \sum_{i=1}^{n} H_i' \cdot P_i$, goes to the next step. Otherwise, outputs a verification failure error and verifies them individually. The correctness of the above equation is as follows:

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

- 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:

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

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

2.5.4 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 [81] 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. 2.3, 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 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.

2.6 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.

[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.

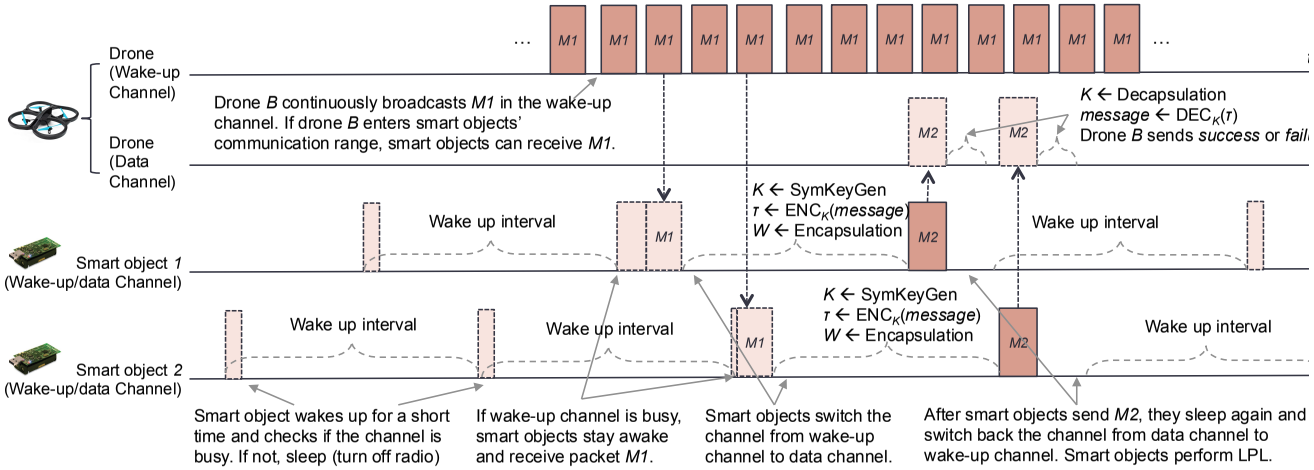


Figure 2.3.: Smart parking management. Solid-line rectangle: transmitted message, dash-line rectangle: received message. $M1 = \{ID_B, P_B, R_B, t_B\}$, $M2 = \{ID_A, P_A, R_A, t_A, U, V, W, \tau\}$, Decapsulation result transmissions are omitted.

2.6.1 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.

2.6.2 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 man-

agement. 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 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.

Protocol description: Fig. 2.3 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 .

Security analysis: The parking management based on eCLSC-TKEM meets all the security requirements described in Fig. 2.1 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 [80]. Theorem 2 in Appendix A [80] 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.

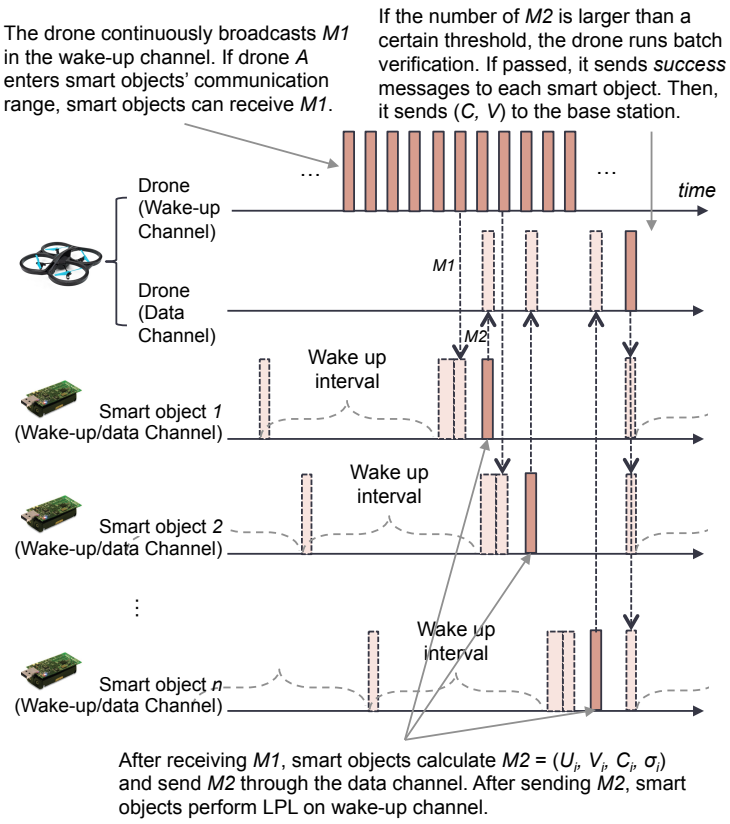


Figure 2.4.: Traffic monitoring (CLDA). Solid-line rectangle: transmitted message, dash-line rectangle: received message. $M1 = \{ID_B, P_B, R_B, t_B\}$. $M2 = \{U_i, V_i, C_i, \sigma_i\}$.

2.6.3 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.

Protocol description: Fig. 2.4 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.

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 $(V, \tau_1, \tau_2, ..., \tau_i, ..., \tau_n)$. Each car (i) decrypts the encrypted message (τ_i) using the HybridDecryption algorithm.

Security analysis: The traffic monitoring and management based CLDA and CL-MRES meet all the security requirements described in Fig. 2.1.

- *Confidentiality and integrity*: CLDA is a variant signed ElGamal encryption [82] 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 [80].
- *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.

2.7 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.

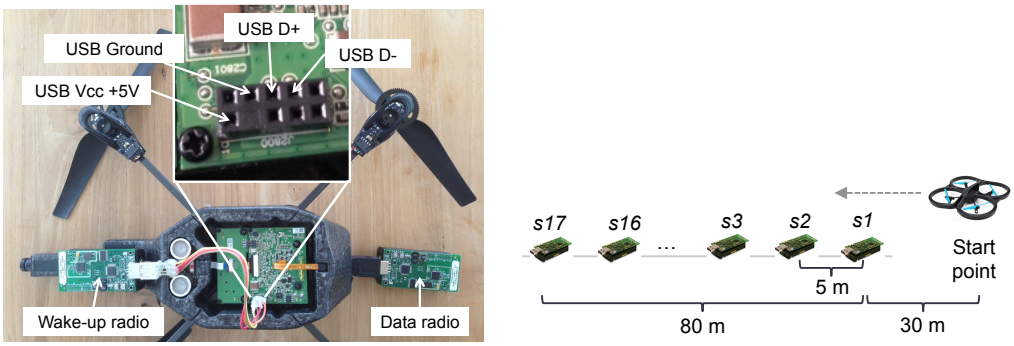
2.7.1 Experiment Setup of the Parking Management

To evaluate the performance of eCLSC-TKEM, we implemented our protocols on commercially available devices: AR.Drone2.0 [83] (as a medium-capacity drone) and TelosBs (as smart objects). To compare eCLSC-TKEM with other certificateless-based schemes, we also implemented CL-AKA [57, 58] and CLSC-TKEM [60].

Drone

AR.Drone2.0 [83] 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 [78] as a crypto-library.

We utilized two TelosBs as the drone's radio transceivers as shown in Fig. 2.5(a). One radio working in the data channel was plugged into the USB port next to the



(a) Dual radios attached on the drone

(b) Test-bed setup

Figure 2.5.: Experiment setup

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, 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.

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 [6] as an elliptic curve cryptography library.

Table 2.2. Comparison of protocols (unit: second)

Protocol	secp128r1	secp160r1	secp192r1
Yang's CL-AKA [57]	32.84	36.22	50.43
Sun's CL-AKA [58]	15.10	16.98	23.84
Seo's CLSC-TKEM [60]	13.37	13.87	18.77
eCLSC-TKEM [61]	9.25	9.61	13.03

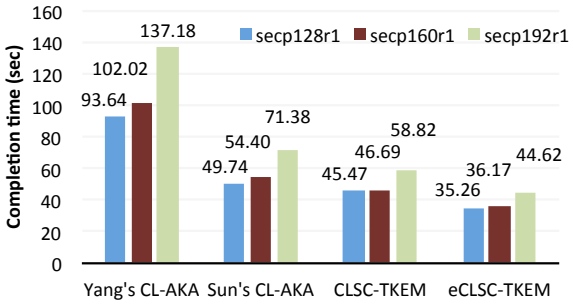


Figure 2.6.: Impact of key bit size

2.7.2 Experimental Results of the Parking Management

Network topology

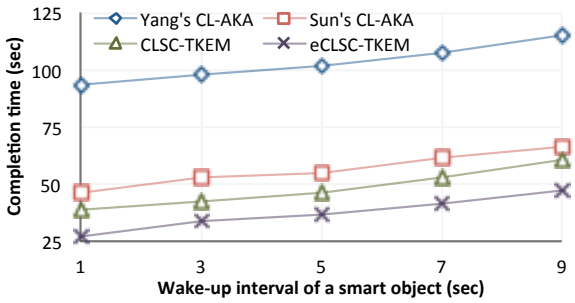
Fig. 2.5(b) shows the test-bed setup of our parking management system. We deployed the 17 smart objects in a line spacing them 5m apart and the drone started from the start point that was 30m apart from $s1$. The drone's altitude was set to 10m. In our test-bed, the drone's mission is to collect data 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.

Impact of key bit size

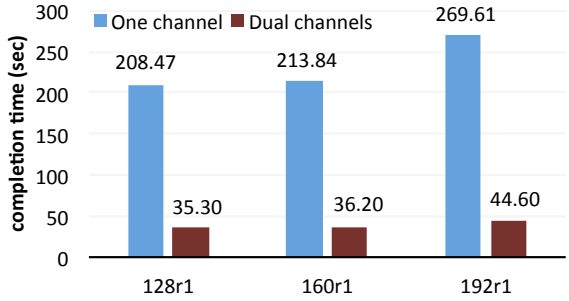
Fig. 2.6 shows the time required to complete the mission when the system used three different key bit sizes. We activated LPL for the smart objects with the wake-up interval of 5 seconds. When secp160r1 was used, the drone with our protocol took 36.2 seconds to complete the mission and completed the 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 2.2 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 2.2 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.

Impact of interval between wake-ups

Fig. 2.7(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



(a) Impact of interval between wake-ups



(b) Impact of the dual channel strategy

Figure 2.7.: Experimental results

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.

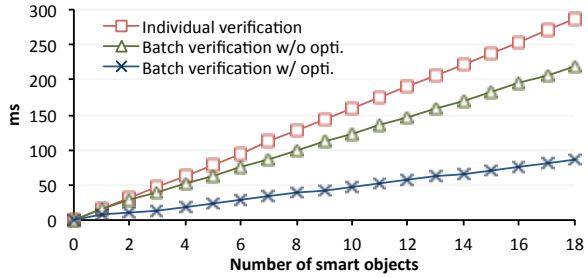
Impact of dual channel strategy

Finally, Fig. 2.7(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 strat-

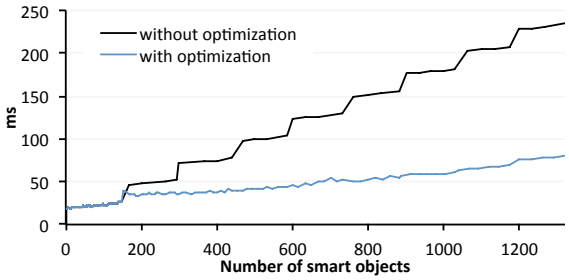
egy, the mission was completed approximately 6 times faster than when only one channel was used. The dual channel strategy allows smart objects to concurrently execute eCLSC-TKEM with a drone, and thus the drone's flight time can be significantly saved. When the dual channel strategy was used, 3 or 4 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.

2.7.3 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 [79] 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 [78] into CUDA-C functions in order to run them on the GPU. We utilized secp160r1 as an ECC parameters.



(a) CPU



(b) GPU

Figure 2.8.: The computation time for the signature verification on the CPU or GPU

Experimental results of CLDA

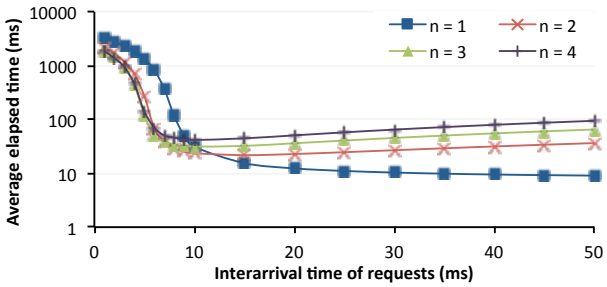
The overall performance of the CLDA protocol is dominated by the performance of the Batch verification algorithm. Therefore, we measured the execution time of the Batch verification algorithm on the CPU and the GPU. The drone collects a random value from virtual cars which run on a PC. They execute the 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.

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. 2.4.2. Fig. 2.8(a) shows the computation time of the three versions

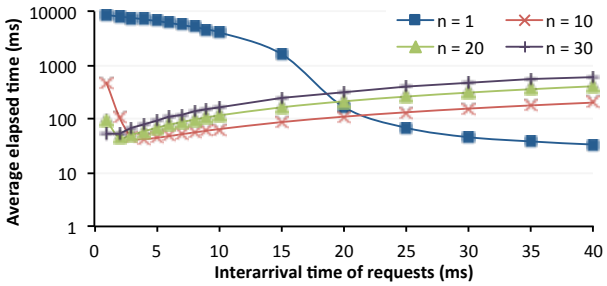
of the protocol on the drone. When the number of cars is 18, the drone running the optimized batch verification requires only 85.5ms, while the drone running the individual verification and the drone running the batch verification without optimization require 285.8ms and 218.4ms, respectively. This result 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. 2.8(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. 2.6.3, 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



(a) CPU



(b) GPU

Figure 2.9.: The average elapsed time observed by cars when the drone uses the CPU or GPU

and the batch verification times on the CPU and the GPU, are based on the real measurements.

Fig. 2.9(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

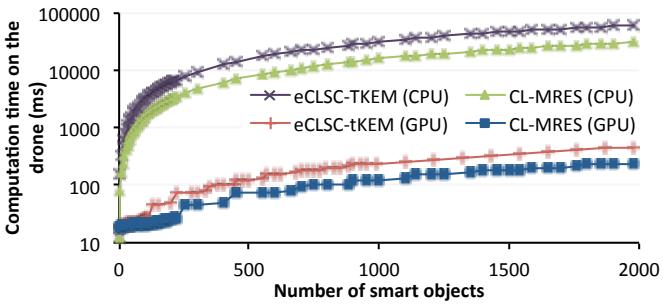


Figure 2.10.: The performance comparison between eCLSC-TKEM and CL-MRES on the CPU and GPU

elapsed time increases since the drone cannot execute the Batch verification algorithm until n signature verification requests are collected.

Fig. 2.9(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.

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. 2.10 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.

2.8 Summary

In this chapter, 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.

3 ROBUST SENSOR LOCALIZATION AGAINST KNOWN SENSOR POSITION ATTACKS

3.1 Introduction

Sensor Networks (WSNs) have started to play a critical role in many domains by providing sensing and monitoring services. Their applications range from military tasks to civilian tasks such as surveillance, fire detection and soil condition monitoring for agriculture. In these applications, the location information of sensors is essential to identify the origins of events or sensed data. In addition, location information is critical for many system functions, such as geographic routing [84] and location-based key management [85]. However, in many cases, the location of sensors can only be determined after deployment since sensors may be deployed via random scattering (e.g., from an airplane). Sometimes, their initial locations can change due to natural phenomena such as wind or rain, or be moved by users or other agents (e.g., mobile sensors). Equipping each sensor with a Global Positioning System (GPS) receiver is undesirable due to the high cost.

Recently, the Internet of Things (IoT) has extended the notions of sensors and sensor networks to a large variety of applications. Various IoT devices, often referred to as "things", such as security sensors, cameras, smart phones or autonomous robots are interconnected. In the near future, even small physical objects embedding sensors and network interfaces will be deployed for new IoT services such as e-health, e-marketing, intelligent transportation, logistic services and food packaging [86–88]. In many IoT applications, such IoT devices are mobile or carried by other agents. Their localization is essential to provide useful services such as navigation, location-aware energy management or location-based advertisement. However, their localization is

not easy since they might not have a GPS because of the cost or because they are located indoors. (IoT devices are also referred to as *sensors* in what follows.)

To tackle the sensor localization problem, several schemes have been proposed based on the use of a small number of static beacon nodes or mobile beacon nodes, which know their positions. Such schemes estimate sensor locations using various information received from beacon nodes such as beacon nodes' positions, distances from beacon nodes, angles of beacon signals or connectivity information. We refer to messages containing such information as *location references*.

However, if sensors are deployed in unattended hostile areas, the estimated locations can be severely distorted by even simple attacks such as replay attacks. For example, if the distance between a sensor and a beacon node is measured by the Received Signal Strength Indictor (RSSI), an attacker can reduce the distance by replaying beacon signals with a strong signal strength. As a result, the attacker can break the location-based functions of WSNs. Another possible attack can be carried out against autonomous vehicles such as drones and self-driving cars equipped with GPS receivers. They are localized by receiving ranging signals from satellites. However, the GPS for civilian use does not provide any security measures such as encryption and authentication. Indeed, researchers have demonstrated the feasibility of attacks on real-world positioning systems and provided mitigation strategies. Zeng et al. [10] showed the feasibility that a portable GPS spoofer could manipulate a navigation route of a car and guide the car to a wrong destination without being noticed. Cho et al. [11] successfully demonstrated that the location-based ordering service provided by Starbucks is vulnerable to replay attacks. Tippenhauer et al. [12] showed that skyhook [13] which is a public WLAN-based positioning system is vulnerable to location spoofing attacks by jamming/replaying localization signals and by tampering with the service database.

In the last decade, various kinds of sensor localization schemes have been proposed for use in adversarial environments. However, some schemes are not well-suited for general WSNs since they require additional hardware. For instance, such schemes rely

on nanosecond scale time synchronization for distance bounding protocol [89, 90], directional antenna [90] or ultra-sound [91]. Although some such schemes [92, 93] do not require the use of such additional hardware, their computationally expensive searching techniques do not make it possible for sensors to estimate their locations in a distributed manner. Considering current low-end sensor platforms, a localization algorithm should be computationally efficient.

To tackle these problems, several secure localization schemes based on Minimum Mean Square Estimation (MMSE) have been proposed [94–96]. Since they utilize a simple MMSE method [97], their computational and storage costs are low enough to be successfully implemented on real sensor platforms such as TelosB [98] and MicaZ [99]. However, we found that an attacker can considerably degrade the estimation accuracy if the positions of victim sensors are known to the attacker. We refer to such attacks as *known sensor position* attacks. In actual attack situations, it is not hard for an attacker to locate a victim sensor since he/she just needs to sniff link layer frames sent by the sensor and measure the distances to the sensor in several different locations.

In this chapter, we introduce two kinds of known sensor position attacks: *Aligned-Beacon-Position (ABP) attack* and *inside attack*. The ABP attack can easily distort sensor position estimates by exploiting the fact that benign beacon nodes are usually aligned in a line. For example, in many sensor localization techniques [100–102], a mobile beacon node moves in a straight line. In the case of indoor localization, beacon nodes may be located straight along hallways or passages. The inside attack thwarts the Degree-Of-Consistency (DOC) filtering algorithm, which is a traditional filtering algorithm to filter out malicious location information. Therefore, the inside attack increases sensor position estimation errors.

To protect against those attacks, we introduce two defense schemes. First, we propose a novel beacon placement strategy to protect against ABP attacks. Second, we propose a new filtering technique that can filter out malicious location references introduced by inside attacks. Finally, we propose a localization algorithm improved

with respect to accuracy and efficiency compared to the state-of-the-arts. We evaluate the impact of the two known sensor position attacks on the existing algorithms and the performance of our algorithm by simulation and test-bed experiments.

3.2 Related Work

Secure localization schemes based on MMSE utilize location information, called location reference, which consists of the position information (x_i, y_i) of a beacon node i and the distance d_i between the sensor and the beacon node i.

Liu et al. [95] proposed four schemes based on two approaches: a voting-based approach and a MMSE-based approach. In the voting-based scheme, the sensor deployment field is divided into small sub-regions and each sub-region is voted by location references. Such scheme provides more accurate sensor positions than other schemes at the cost of high computation and memory overhead. The other three schemes are MMSE-based. They filter out bad information and try to find out a subset of location references which produces the minimum mean square error by iteratively performing a MMSE. The three MMSE-based schemes have different search strategies. The first scheme, called Brute-Force MMSE (BF-MMSE), is based on a brute-force search. It iteratively performs a MMSE with all possible subsets of location references from the largest subset to the smallest subset until a result of the MMSE is lower than a threshold. Although BF-MMSE outputs relatively more accurate sensor positions than the other two MMSE-based schemes, it suffers from high computational overhead when the number of location references is large. The second scheme, called Greedy Attack Resistant MMSE (GAR-MMSE), is based on a greedy algorithm. Rather than performing a MMSE for all possible subsets, it drops one potentially malicious location reference for each round. Although this scheme is more efficient than the brute-forth scheme, it is weak against collusion attacks. The third MMSE-based scheme, called Enhanced-greedy Attack Resistant MMSE (EAR-MMSE), utilizes the fact that benign location references intersect each other. Each location reference has a

counter which is the number of other location references intersecting with itself. Like the greedy algorithms, it drops a location reference which has the lowest count per round. EAR-MMSE is the most efficient in terms of computational and storage cost compared to the other schemes. In addition, the accuracy of EAR-MMSE is higher than the accuracy of the greedy algorithm and comparable to other schemes, i.e., the voting scheme and BF-MMSE. However, its localization accuracy dramatically decreases when an attacker knows the true positions of sensors as discussed later.

Wang et al. [96] developed the Cluster-Based MMSE (C-MMSE) which tries to cluster benign location references by randomly selecting two location references as seeds. It reduces the time required to estimate a sensor position compared to EAR-MMSE by sacrificing the estimate accuracy.

Li et al. [94] proposed the Least Median Square (LMS) method based on the fact that the median is statistically robust against outliers. LMS tries to minimize the median of the square errors instead of minimizing the sum of the square errors. However, this method involves a number of MMSE operations to find the median, which is computationally expensive compared to EAR-MMSE.

Some other schemes focus on detecting malicious beacon nodes by checking round trip time [103] or based on cooperation with other nodes [103, 104]. These detection schemes are complementary to ours and can be combined for more robust localization.

Zeng et al. [93] introduced a *pollution attack*. In this attack, if the malicious location references given by an attacker are consistent with some benign location references, these benign location references become "polluted". Thus, the attack succeeds even with a small number of malicious location references. This attack is similar to the ABP attack in that both attacks exploit benign location references. However, Zeng et al. do not address how the attacker exploits benign location references. In other words, it is unclear, under which conditions the attacker is able to launch the pollution attack. Also, they do not provide a solution to prevent such attacks. They only propose an algorithm that checks if the estimated position of a sensor is correct or not.

3.3 Background

In this section, we first describe network and threat models. We then provide a brief overview of the MMSE and the notion of *Degree of Consistency*.

3.3.1 Network Model

In our network model, there are two kinds of nodes: beacon nodes and non-beacon nodes. Beacon nodes know their locations because they can directly obtain their current location information from other sources such as GPS receivers or manual inputs. In contrast, the locations of non-beacon nodes are unknown and have to be estimated by using beacon nodes. In what follow we use the term *sensor* as synonym for *non-beacon node*.

There are two possible approaches for localization: *distributed* and *centralized*. In a distributed approach, a beacon node sends a beacon signal which includes the beacon node's identity, the current time, and its current location information. After a sensor receives several beacon signals from different beacon nodes, it measures the distances from the beacon nodes using a distance measurement method such as the Received Signal Strength Indicator (RSSI), Time of Flight (ToF) or the Time Difference on Arrival (TDOA). A sensor collects a set of location references for its position estimation. A location reference consists of a triple $\{x_i, y_i, d_i\}$, where (x_i, y_i) is the position of the beacon node i and d_i is the measured distance. Since sensors are usually low-end embedded devices that are battery-powered, the computational cost for localization should be minimized.

On the other hand, in a centralized approach, all location references are sent to a base station. Then, the base station calculates the positions of all sensors. Although our scheme can be adapted for integration with any localization approach, for the sake of presentation, we will describe our scheme based on the distributed approach.

We assume that distance measurement errors are uniformly distributed in $[-\epsilon, \epsilon]$, as in [92, 93, 95]. Note that the RSSI-based distance measurement method has rela-

tively large ϵ compared to the ultra-sound-based distance measurement method. In our test-bed experiment, we utilize the RSSI-based distance measurement method since it does not require sensors to be equipped with additional expensive hardware such as an ultra-sound receiver and to perform a nano-second-scale time synchronization. However, any measurement method can be employed by our algorithm.

3.3.2 Threat Model

The goal of the attacker is to make the estimated locations of sensors far away from their true locations. We assume that the attacker is able to manipulate any element of a location reference $\{x_i, y_i, d_i\}$ by performing various attacks such as beacon node compromises, worm hole attacks and replaying beacon signals for distance reduction/enlargement. For example, if a beacon node is compromised, it can manipulate any element of a location reference. Since the beacon node has the secret key, the malicious location reference is successfully authenticated by sensors. In case of worm hole attacks and replay attacks, the attacker can just modify d_i. For instance, the attacker can reduce d_i by replaying beacon signals with increased signal strength.

We assume that beacon nodes are managed by a system administrator as in airport (or shopping mall) indoor positioning systems. A beacon signal sent from a beacon node to a sensor is encrypted using a pairwise key k established between the beacon node and the sensor. Then, the keyed-hash message authentication code ($HMAC_k(beacon\_node\_id,\ current\_time)$) is attached to the beacon signal to prevent a compromised beacon node from generating more than one identity. Therefore, sensors can utilize only one location reference derived from a beacon signal sent by a beacon node. In other words, the attacker must steal n different pairwise keys in order to generate n malicious location references.

We assume that an attacker has knowledge about the location of a target victim sensor. Multiple attackers can collude to make their malicious location references point to one false position so that the measured position of a sensor is misled to the

false position. However, our scheme can protect against such a colluding attack if the number of benign location references is higher than the number of malicious location references.

3.3.3 Minimum Mean Square Estimation

Suppose that a sensor obtains a set $\{l_1, l_2, ..., l_n\}$ of n location references from beacon nodes, where $l_i = (x_i, y_i, d_i)$, and the estimated location is $(\hat{x}, \hat{y})$. Then, the mean square error δ of this location estimation is as follows:

$$\delta = \frac{1}{n} \sum_{i=1}^{n} \left(d_i - \sqrt{(\hat{x} - x_i)^2 + (\hat{y} - y_i)^2} \right)^2$$

MMSE is a method which obtains the estimate $(\hat{x}, \hat{y})$ by minimizing δ. MMSE is a nonlinear least squares problem which is solved by well-known optimization methods like the Gauss-Newton method [105], the Gradient descent method [106] and the basic MMSE [97]. Since the first two methods iteratively update the estimated point from an initial random point, they are more accurate, but take more time than the basic MMSE. These methods also involves complex calculations such as Jacobian matrix formation [105, 106] and matrix multiplications, and their computational complexity increases as n increases [1]. Therefore, when a sensor localization algorithm is designed, the number of MMSE operations should be minimized.

3.3.4 Degree Of Consistency

Our algorithm utilizes the notion of *Degree Of Consistency* to filter out malicious location references.

Definition 1 (Degree Of Consistency [95]) *Each location reference l_i has a counter called* Degree Of Consistency (DOC). *The DOC of l_i is the number of other location references (l_j, $i \neq j$) which intersect l_i.*

[1] Note that the results of MMSE tend to become accurate as n increases due to distance measurement errors.

The way to check whether two location references l_i and l_j intersect is as follows. Considering the maximum measurement error ϵ, there are two circles for a location reference. For instance, the two circles of l_i are $\{x_i, y_i, d_i - \epsilon\}$ and $\{x_i, y_i, d_i + \epsilon\}$. Let d_{ij} be the distance between two beacon points, i.e., $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. If the following three conditions are all true, l_i and l_j do not intersect: (1) $d_{ij} > (d_i + \epsilon) + (d_j + \epsilon)$, (2) $d_{ij} + (d_i + \epsilon) < (d_j - \epsilon)$ and (3) $d_{ij} + (d_j + \epsilon) < (d_i - \epsilon)$. Otherwise, they intersect.

3.4 Known Sensor Position Attacks

In actual attack situations, it is not difficult for an attacker to locate a target victim sensor even if it is physically hidden from the attacker. For instance, the attacker just needs to sniff link layer frames like keep-alive messages and measure the distances from the target sensor in three or more different locations. Then, the attacker can locate the victim using the multilateration technique. In this section, we introduce the two attacks that exploit the knowledge that the attacker has on the sensor position: the *Aligned-Beacon-Position (ABP) attack* and the *inside attack*.

3.4.1 Aligned Beacon Position Attack

In WSNs, it is important to minimize the number of beacon nodes to save cost. Therefore, in many sensor localization schemes [100–102, 107], beacon nodes are deployed with a well-organized pattern to efficiently cover the entire area. For example, if the beacon nodes are mobile devices like drones, they usually cover an area with a line sweeping pattern like a lawn mower. If the beacon nodes are static, they might be deployed on each cell of a square or hex grid.

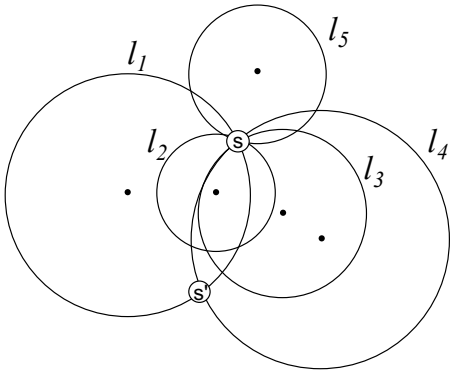


Figure 3.1.: ABP attack example. Black dots indicates beacon positions. l_1, l_2, l_3, l_4 and l_5 are benign location references. s is the true sensor position. s' is the false sensor position intended by an attacker.

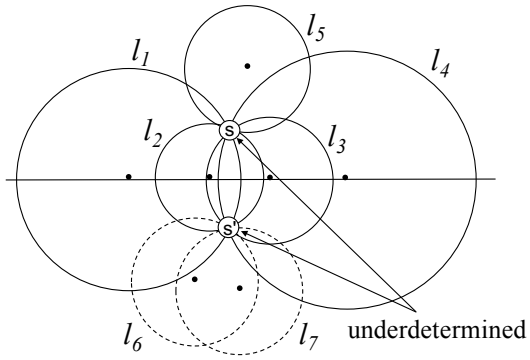


Figure 3.2.: ABP attack example. Black dots indicates beacon positions. l_1, l_2, l_3, l_4 and l_5 are benign location references. l_6 and l_7 are malicious location references. s is the true sensor position. s' is the false sensor position intended by an attacker.

Attack Example

As shown in Fig. 3.1, the beacon positions (i.e., the centers of the circles) of benign location references l_1, l_2, l_3, l_4 and l_5 are unaligned and consistent with the true sensor position s. In order to change the position estimate from s to s', the attacker must generate four or more location references that are consistent with s'.

However, if the beacon points of benign location references are aligned in a line and the attacker knows the actual sensor position, he/she can easily distort sensor position estimates by exploiting the benign location references. For example, in Fig. 3.2, assume that there are only l_1, l_2, l_3 and l_4. They are benign location references and their beacon positions are aligned in a line. Then, there are two possible locations for the sensor, i.e., s and s'. We call such sensors *underdetermined* sensors. By introducing l_5, the final location s can be determined. However, if the attacker knows the actual sensor position, he/she can distort the sensor position estimate to s' by introducing only two additional location references, i.e., l_6 and l_7, or compromising l_5. Such attack, referred to as an **Aligned Beacon Position (ABP) attack**, reduces attack costs by exploiting three or more benign location references whose centers are aligned in a line.

Attack Description

The following conditions must be satisfied for an attacker $\mathcal{A}$ to successfully launch an ABP attack:

- $\mathcal{A}$ finds the true position of the victim sensor s.
- $\mathcal{A}$ obtains information about lines $line_1, line_2, ..., line_n$, on which three or more beacons are located, and calculates the distances $dst_1, dst_2, ..., dst_n$ between the lines and s. Note that $\mathcal{A}$ only needs to know whether such line exists and where it is located.
- $\mathcal{A}$ generates N_m malicious location references pointing a false sensor position s'. s' is the point at the opposite side of s with respect to $line_i$ $(1 \leq i \leq n)$. N_m must be greater than N_b, where N_b is the number of benign location references whose beacon points are not on $line_i$. Note that the possible estimation errors that an ABP attack can cause are $2 \times dist_i$ $(1 \leq i \leq n)$ and are bound to d_{max}. Here, d_{max} is the maximum measurement distance.

3.4.2 Inside Attack

The inside attack is another known sensor position attack, which causes serious estimation errors. In this section, we show how an inside attack disables DOC-based filtering algorithms.

Attack Example

EAR-MMSE [95] is one of the most efficient localization algorithms that produce a good estimate $(\hat{x}, \hat{y})$ even when some of location references are malicious. The use of DOC makes EAR-MMSE simple and secure. However, if malicious location references are *inside* as we will describe, EAR-MMSE is prone to severe estimatation errors. Before we describe this problem, we overview EAR-MMSE.

As shown in Algorithm 1, EAR-MMSE first runs MMSE by using all location references to obtain the mean square error δ and the estimate $(\hat{x}, \hat{y})$. If δ is smaller than a certain threshold θ, it returns $(\hat{x}, \hat{y})$ as a final estimate. Otherwise, it selects a location reference whose DOC is lowest. EAR-MMSE runs MMSE again by inserting all location references except the selected reference. This procedure is repeated until δ is smaller than θ or the number of remaining location references is less than a certain number bigger than 2 [2].

As shown in Fig. 3.3, the DOCs of benign location references (l_1, l_2 and l_3) are more than or equal to 2, whereas the malicious location references l_4 has a DOC of 1. Therefore, if a malicious location reference is far from the true location of a target sensor, it will be removed. Now assume that a malicious location reference includes the true position of a sensor as shown in Fig. 3.4. In this case, all location references (l_1, l_2, l_3 and l_4) have a DOC of 3. However, EAR-MMSE does not have a specific tie-break rule and must randomly choose one of them.

If there are two malicious location references, EAR-MMSE may remove a benign location reference as shown in Fig. 3.5. Since the DOC of the benign location reference

[2]The multilateration requires at least 3 location references.

Algorithm 1 EAR-MMSE

1: $\mathcal{L} = \{l_1, l_2, ..., l_n\}$
2: **while** 1 **do**
3: $(\delta, \hat{x}, \hat{y}) \leftarrow MMSE(\mathcal{L})$
4: **if** $\delta < \theta$ **then**
5: **return** $(\hat{x}, \hat{y})$
6: **else**
7: **if** length of $\mathcal{L} < 3$ **then**
8: **return** $failure$
9: **else**
10: //get a location reference with the lowest DOC
11: $l \leftarrow findLocRefLowestDOC(\mathcal{L})$
12: $\mathcal{L} \leftarrow \mathcal{L} - \{l\}$
13: **end if**
14: **end if**
15: **end while**

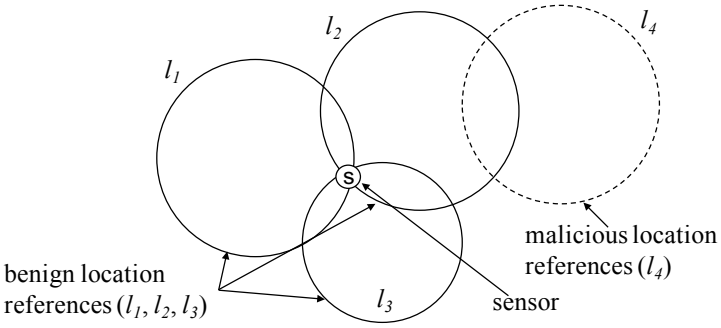


Figure 3.3.: Outside attack example. DOCs of l_1 and l_3 are equal to 2. DOC of l_2 is 3. DOC of l_4 is 1.

l_4 is the lowest DOC, i.e., 3, l_4 is removed in the first iteration. Then the DOCs of all location references become equal to 4 and one of remaining five location references

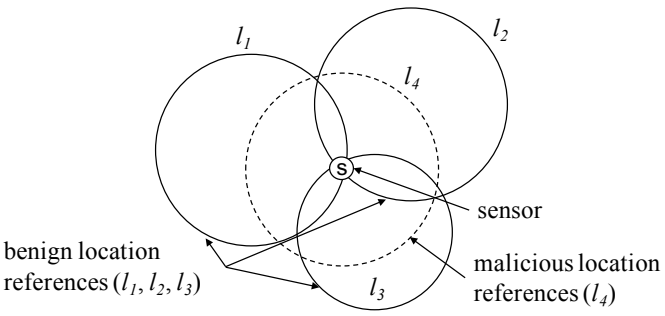


Figure 3.4.: Inside attack example 1. DOCs of l_1, l_2, l_3 and l_4 are equal to 3.

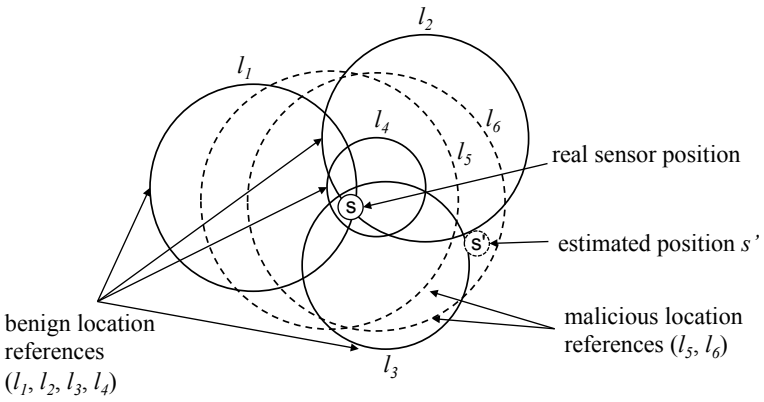


Figure 3.5.: Inside attack example 2. DOCs of l_1, l_2 and l_3 are equal to 5. DOCs of l_5 and l_6 are equal to 4. DOC of l_4 is equal to 3.

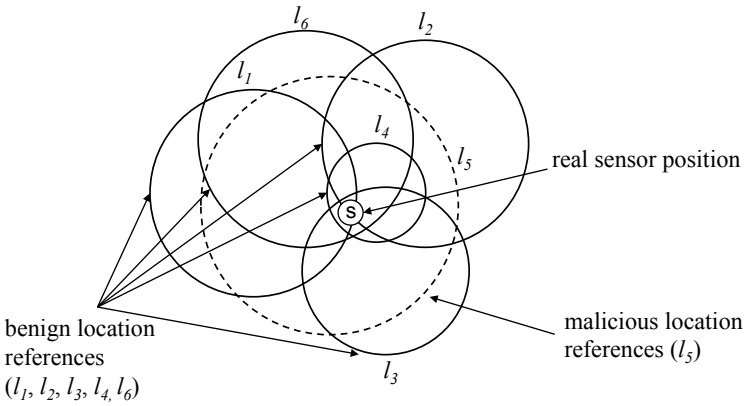


Figure 3.6.: Inside attack example 3.

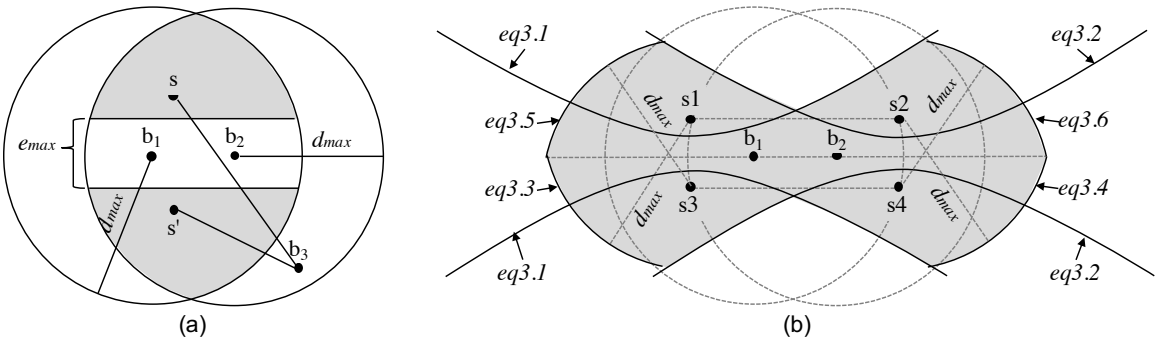


Figure 3.7.: ABP-attack-free beacon area

will be removed. If a benign location reference, e.g., l_1, is removed, the estimated position will be s' with a high probability. We call such a malicious location reference **inside-attack reference**, which practically is "inside" the union of other benign location references, includes the true sensor position and does not have the lowest DOC. The **inside attack** denotes an attack causing an estimation error using inside-attack references.

Attack Description

In order for an inside attack to be successful, an attacker has to generate an inside-attack reference that satisfies the following two conditions.

1) m inside-attack references must intersect at least $n - 2m$ benign references (n is the total number of references).

2) The m inside-attack references must intersect each other.

If the two conditions are satisfied, the DOCs of the inside-attack references and the victim benign references are all equal to $n-m-1$. For example, assume there are four benign references and three inside-attack references ($n = 7$ and $m = 3$). Since four benign references intersect each other, their DOCs are at least 3. Also, since the three inside-attack references intersect each other, their DOCs are at least 2. Now, if the three inside-attack references intersect with 1 ($= n - 2m = 7 - 2 \times 3$) benign

reference, e.g., l_1, then, the DOCs of the inside-attack references becomes 3 and the DOCs of l_1 becomes 6. Since the DOCs of benign references, except l_1, is also 3, one of six references (three inside-attack references and three benign references) will be randomly removed. If a benign reference is removed, the DOCs of all remaining benign references will decrease by 1, while the DOCs of three inside-attack references will not change. Therefore, in the next iteration, one of benign references, except l_1, will be removed. Finally, one benign reference and three inside-attack references will remain. This procedure is illustrated as follows: $\{6,3,3,3,3,3,3\} \rightarrow \{5,*,2,2,3,3,3\} \rightarrow \{4,*,*,1,3,3,3\} \rightarrow \{3,*,*,*,3,3,3\} \rightarrow \cdots$, where $\{DOC_1, DOC_2, ..., DOC_7\}$ are the DOCs of the references and DOC_5, DOC_6 and DOC_7 are the DOCs of inside-attack references. $*$ means that the reference is *removed*.

However, not all inside-attack references are harmful. If the circle line of an inside-attack reference is close to the true sensor position (x_0, y_0), it is not different from a benign reference. Therefore, to make an inside-attack reference harmful, the circle lines of the inside-attack references should be far from the true sensor position. In other words, an inside-attack reference with (x_0, y_0, d_{max}) is most malicious if it satisfies two conditions above.

Possible estimation errors which an inside-attacker can cause are bound to d_{max}. However, if the time-of-flights of RF signals (e.g, GPS) or the RSSI using powerful antennas ((>1km) [108]) is utilized as a distant measurement method in order to send/receive beacon signals from afar, the attacker's capability is not negligible since d_{max} becomes very large.

3.5 Defense Scheme

In this section, we first propose a beacon deployment strategy to prevent ABP attacks. Then, we present an Inside-Attack Filtering (IAF) algorithm.

3.5.1 ABP-attack-free Beacon Deployment

Many secure localization algorithms [93, 95, 96] locate sensors after filtering out malicious location references. They correctly work only when the number of location references supporting the true sensor position is greater than the number of location references supporting the false sensor position. Therefore, if an attacker launches an ABP attack against a sensor, the existing localization algorithms cannot correctly locate the sensor as described in Sec. 3.4.2. Note that the deployment strategy in what follows is intended to prevent an attacker from exploiting benign beacon nodes that are aligned in a line, rather than to filter out malicious location references from compromised nodes.

Determining Beacon Positions

To protect against ABP attacks, it is crucial to proactively prevent them by carefully deploying beacon nodes. We first define $e_{max}(> \epsilon)$, which is the maximum estimation error allowed by our ABP-attack-free deployment strategy. As shown in Fig. 3.7 (a), given two beacon positions b_1 and b_2, the gray area indicates the area in which a sensor can be located according to b_1 and b_2, and has a distance greater than $e_{max}/2$ from the line on which b_1 and b_2 are located. Since there are only two beacons, the sensor location is underdetermined. That is, if s is the true sensor location, s' is the false sensor location, and vice versa. When the third beacon node b_3 is deployed, it must *determine* the sensor position, i.e., whether s is true or s' is true, given the maximum measurement error ϵ. In other words, the b_3's position must satisfy $|\overline{b_3 s} - \overline{b_3 s'}| > \epsilon$. In real applications, ϵ increases as the measured distance increases. However, for the sake of simplicity, we assume that ϵ is constant.

In order to determine the area where the third beacon can be deployed, the four extreme sensor positions $s1$, $s2$, $s3$ and $s4$, as shown in Fig. 3.7 (b), must be considered. In Fig. 3.7 (b), the gray area indicates the area in which the third beacon is not allowed to be deployed by our ABP-attack-free beacon deployment strategy.

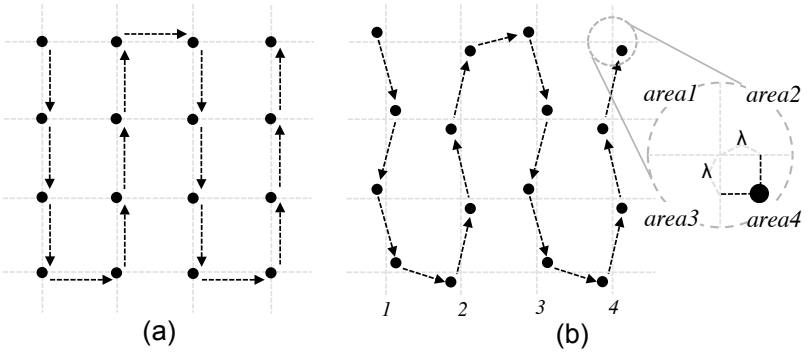


Figure 3.8.: Path planning for a mobile beacon

Let the positions of b_3, $s1$, $s2$, $s3$ and $s4$ be (x_{b3}, y_{b3}), (x_{s1}, y_{s1}), (x_{s2}, y_{s2}), (x_{s3}, y_{s3}) and (x_{s4}, y_{s4}), respectively. The third beacon position b_3 must satisfy the following equations in order to *determine* a sensor position.

$$\Big|\sqrt{(x_{b3}-x_{s1})^2+(y_{b3}-y_{s1})^2}- \sqrt{(x_{b3}-x_{s3})^2+(y_{b3}-y_{s3})^2}\Big| > \epsilon \tag{3.1}$$

$$\Big|\sqrt{(x_{b3}-x_{s2})^2+(y_{b3}-y_{s2})^2}- \sqrt{(x_{b3}-x_{s4})^2+(y_{b3}-y_{s4})^2}\Big| > \epsilon \tag{3.2}$$

However, if b_3 is far enough not to reach one of $s1$ and $s3$ (or one of $s2$ and $s3$), b_3 can be deployed even when Eq. 3.1 and Eq.3.2 are not satisfied. As shown in Fig. 3.7 (b), b_3 can be deployed if one of the following equations is satisfied.

$$(x_{b3}-x_{s1})^2+(y_{b3}-y_{s1})^2 > d_{max}^2 \tag{3.3}$$

$$(x_{b3}-x_{s2})^2+(y_{b3}-y_{s2})^2 > d_{max}^2 \tag{3.4}$$

$$(x_{b3}-x_{s3})^2+(y_{b3}-y_{s3})^2 > d_{max}^2 \tag{3.5}$$

$$(x_{b3}-x_{s4})^2+(y_{b3}-y_{s4})^2 > d_{max}^2 \tag{3.6}$$

Beacon Deployment Strategy

In order to efficiently cover an area with a small number of beacon nodes, a common approach is to deploy the beacon nodes with a well-organized way like deploying them at each cell in a square or hex grid.

In this section, we introduce an example path planning strategy for a mobile beacon under the assumption that the mobile beacon moves with a line sweeping pattern and stops at each intersection point in a grid to sends beacon signals. If the stopping points are naively set as shown in Fig. 3.8 (a), all the beacon nodes are aligned on horizontal, vertical or diagonal lines, and thus sensors are vulnerable to ABP attacks. To make beacon nodes unaligned, we divide the area surrounding each intersection point into four sub-areas and set four beacon stopping points at a distance of λ from the x-axis and the y-axis as shown in Fig. 3.8 (b). The mobile beacon stops at the points in $area1$ and $area2$ alternately in the odd-numbered lines and stop at the points in $area3$ and $area4$ alternately in the even-numbered lines. To find a right λ, we increase λ until when all the beacon stopping points satisfy the conditions presented in Sec. 3.5.1.

The path planning scheme proposed here increases the total traveling time of the mobile beacon. If the mobile beacon is a battery-powered agent like a drone, the increased time may degrade the usability of this path planning scheme. However, as we show in Sec. 3.6.1, our path planning scheme does not increase much the traveling time of the mobile beacon.

Security Analysis

Theoretically, our mobile beacon path planning strategy is secure against ABP-attacks if all the positions of adjacent beacons satisfy the equations in Sec. 3.5.1. Note that, if a sensor application requires beacon nodes to be densely deployed or d_{max} is too large, it may be impossible to find a combination of the beacon positions that satisfy the equations in Sec. 3.5.1. However, the more densely beacons are deployed, the

harder it is for an attacker to launch ABP-attacks since he/she needs more malicious beacon references. Note that, as we described in Sec. 3.4.1, the number of malicious references must be greater than the number of benign references.

3.5.2 Inside Attack Filtering Algorithm

In this section, we present an Inside Attack Filtering (IAF) algorithm and a secure localization algorithm, called IAF-MMSE, which efficiently estimates sensor positions using IAF and the DOC. Then, we provide the security analysis of IAF-MMSE.

Observations

Our IAF algorithm utilizes the following facts:

- A harmful inside-attack reference l_i has a large d_i and its beacon position (x_i, y_i) is close to (x_0, y_0).
- As we will discuss later in Theorem 1, more than half of all intersection points are concentrated inside a certain circle, called *boundary circle*, of which the center is (x_0, y_0) and the area is relatively small compared to the entire area. Intuitively, the reason is that the distances between the circle lines of the benign references and (x_0, y_0) are smaller than ϵ.

Based on these two facts, we can see that the more harmful an inside-attack reference is, the more intersection points are included by the reference. For example, in Fig. 3.5, the numbers of intersection points included by the benign references l_1, l_2, l_3 and l_4 are 3, 5, 5 and 3, respectively. However, the inside-attack references l_5 and l_6 include 15 and 14 intersection points respectively. The reason is that the inside-attack references include almost all intersection points of the benign references. However, the benign references may not include intersection points made by inside-attack references.

In addition, even if the fraction of inside-attack references increases, this observation still is valid. Let p_i^{bn} be the number of intersection points included by a benign

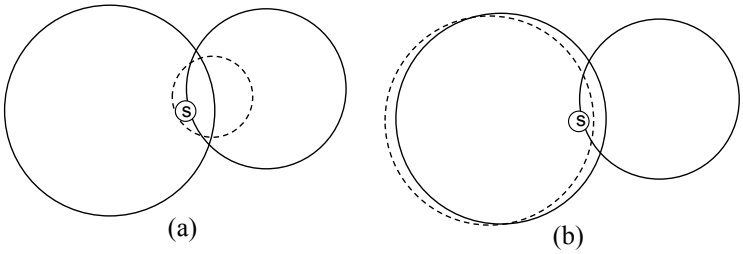


Figure 3.9.: Excluded location reference examples (dashed circle) for the analysis

reference and p_j^{in} be the number of intersection points included by an inside-attack reference. If the fraction of inside-attack references increases, both p_i^{bn} and p_j^{in} decrease with a high probability. The example in Fig. 3.6 is the same as the example of Fig. 3.5 except for l_6 which is benign in the example in Fig. 3.6, but is malicious in the example in Fig. 3.5. In Fig. 3.6, $\{p_1^{bn}, p_2^{bn}, p_3^{bn}, p_4^{bn}, p_5^{in}, p_6^{bn}\} = \{5, 7, 7, 5, 18, 9\}$, whereas in Fig. 3.5, $\{p_1^{bn}, p_2^{bn}, p_3^{bn}, p_4^{bn}, p_5^{in}, p_6^{in}\} = \{3, 5, 5, 3, 15, 14\}$. After l_6 becomes malicious, both p_i^{bn} and p_j^{in} for all i and j decrease except p_6^{in}. The reason is that the intersection points made by l_6 with benign references are removed.

For the purpose of our analysis, we assume that any benign reference is not completely included by others as the dashed-line circle in Fig. 3.9 (a). We assume that all benign references are sufficiently different (not like the dashed-line circle in Fig. 3.9 (b), see the details in Lemma 1) and that any two references have two intersection points or nothing. These conditions can be easily achieved if beacon nodes are deployed according to the beacon deployment strategy in Sec. 3.5.1.

Definition 2 (Outer reference) *Given a location reference* (x_i, y_i, d_i)*, its* outer reference *is* $(x_i, y_i, d_i + \epsilon)$*, where* ϵ *is the maximum measurement error. Benign outer references must include the real sensor position.*

Definition 3 (Boundary circle) *The* boundary circle *is a circle of which the center is the true sensor position and the radius* r *is* $\sqrt{d_{max}^2 - (d_{max} - \epsilon)^2} = \sqrt{2\epsilon d_{max} - \epsilon^2}$*, where* d_{max} *is the maximum measurement distance.*

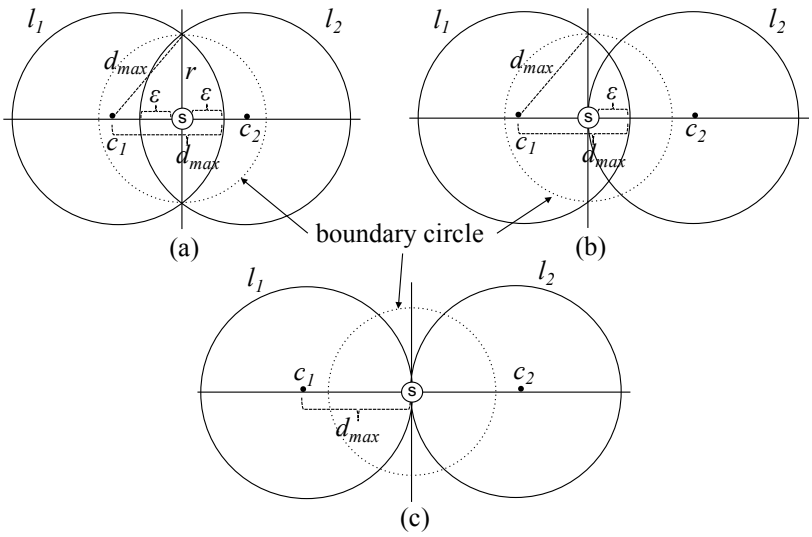


Figure 3.10.: The illustrations of three cases

Lemma 1 *Inside the boundary circle, there is at least one of two intersection points made by two outer benign references.*

Proof In this proof, we only consider the cases in which the measured distances (radiuses) of two benign outer references l_1 and l_2 are d_{max}. However, the proof can be generalized for any measured distances. There are three cases:
(1) As shown in Fig. 3.10 (a), the case a is when both l_1 and l_2 include the true sensor position and the lines of the circles are ϵ-away from the true sensor position.
(2) As shown in Fig. 3.10 (b), the case b is when one benign outer reference l_1 includes the true sensor position and the line of the circle are ϵ-away from the true sensor position. On the other hand, the line of the other benign outer reference l_2 lies on the true sensor position.
(3) As shown in Fig. 3.10 (b), the case c is when the lines of both l_1 and l_2 lie on the true sensor position.

Now, we rotate l_2 and check whether the intersection points of l_1 and l_2 are inside the boundary circle. In case a, as shown in Fig. 3.11 (a), if we rotate l_2, one of the two intersection points lies on the bold dashed-line unless the centers of

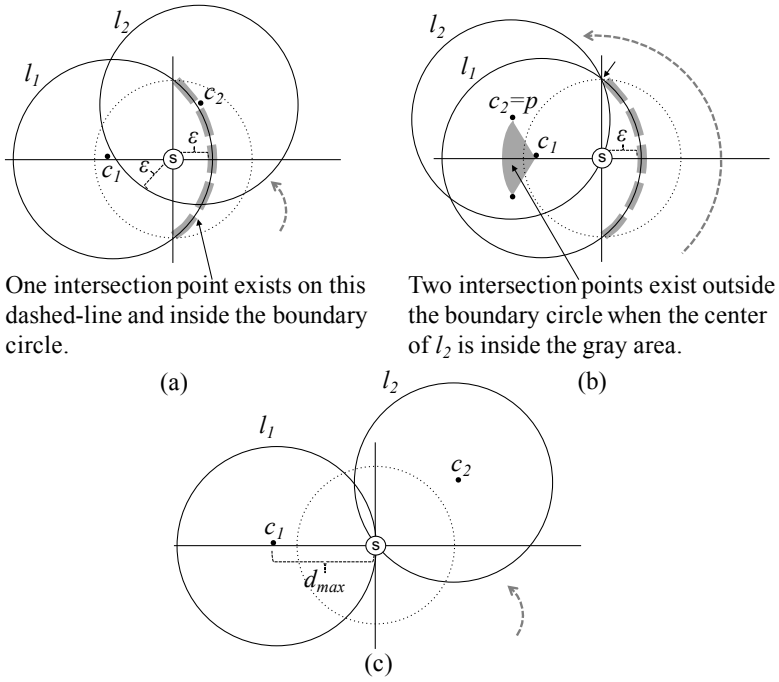


Figure 3.11.: The illustrations of l_2 rotation

the two references are completely congruent. In case b, as shown in Fig. 3.11 (b), if we rotate l_2, one of the two intersection points lies on the bold dashed-line until the center c_2 of l_2 reaches a point p. Let the true sensor position be $(0, 0)$. Then, $p = (\pm\sqrt{d_{max}^2 - r^2/4}, r/2)$, where $r(= \sqrt{2\epsilon d_{max} - \epsilon^2})$ is the radius of the boundary circle. Specifically, if c_2 is outside of the gray area, one of the two intersection points lies on the bold dashed-line. In case c, as shown in Fig. 3.11 (c), since l_1 and l_2 lie on the true sensor position, one intersection point exists on the true sensor position unless the two references are completely congruent. Therefore, one of the intersection points of the two benign outer references exists inside the boundary circle if their centers are sufficiently separated. ■

Theorem 1 *The number of intersection points inside the boundary circle is at least* $\binom{n}{2} \big(= n \times (n-1)/2 \big)$.

Algorithm 2 IAF

1: $\mathcal{L} = \{l_1, l_2, ..., l_n\}$ ▷ $l_i = (x_i, y_i, d_i)$
2: **for all** l_i **do**
3: // get the number of intersection points inside l_i
4: $np_i \leftarrow getNumOfInnerIntersectionPoints(\mathcal{L}, l_i)$
5: **end for**
6:
7: $np_m \leftarrow getMedian(np_1, np_2, ..., np_n)$
8:
9: **for all** l_i **do**
10: **if** $np_i > np_m + \eta$ **then**
11: $\mathcal{L} \leftarrow \mathcal{L} - \{l_i\}$
12: **end if**
13: **end for**

Proof The number of all intersection points is $\binom{n}{2} \times 2$. Since at least one of them is inside the boundary circle according to Lemma 1, the number of intersection points inside the boundary circle is at least $\binom{n}{2} \times 2 \times 1/2 = \binom{n}{2}$. ■

Inside-Attack Filtering

Based on the above observations, we developed a heuristic filtering algorithm called Inside-Attack Filtering (IAF). The algorithm works as follows:

1. It obtains the intersection points of all location references.
2. For each location reference l_i, it counts the number np_i of intersection points included in l_i.
3. It computes the median np_m of $\{np_1, np_2..., np_n\}$.

4. It removes the location references for which np_i is greater than np_m plus a threshold η.

The pseudo-code of IAF is shown in Algorithm 2. In step 3 (code-line 7), we utilize the *median* instead of the *mean* since the median provides a more representative value when there are outliers. η is a system parameter to adjust the filtering sensitivity. If η decreases, the detection rate increases, but the false positive rate also increases. On the other hand, if η increases, the false positive rate decreases, but the detection rate also decreases. η is set as follows:

$$\eta = \binom{n'}{2} - \binom{n'-1}{2} + \alpha,$$

where $n' = \lceil n/2 \rceil$ and $\alpha \geq 0$. n is the number of location references. For instance, if n is 7 and $\alpha = 0$, $\eta = \binom{4}{2} - \binom{3}{2} = 3$. When n is 7, there are at least 4 benign references and there are at least $\binom{4}{2}$ intersection points inside the boundary circle, which means that the inside-attack references include at least cintersection points if they are bigger than the boundary circle (Theorem 1). On the other hand, a benign reference approximately includes $\binom{3}{2}$ intersection points which are made by 3 other benign references.

α (≥ 0) is added in order to reduce the false positive rate, i.e., to prevent benign references from being removed by IAF. η without α is very conservative since it assumes the worst case, i.e., that $\lfloor n/2 \rfloor$ of location references are malicious. If α is small, IAF will aggressively filter out all malicious references, but sometimes it will remove benign ones (high true/false positive rate). On the other hand, if α is large, IAF will loosely filter out malicious references (low true/false positive rate). Notice that even if some malicious references are not filtered out by IAF, the unfiltered malicious ones can be filtered out by the rest of our entire IAF-MMSE algorithm discussed in Sec. 3.5.2.

IAF-MMSE

To estimate sensor positions, we propose the IAF-MMSE algorithm. Similarly to EAR-MMSE, IAF-MMSE checks the DOCs of location references. However, IAF-MMSE can remove multiple malicious references in one round, whereas EAR-MMSE removes one reference per round. Therefore, IAF-MMSE is able to reduce the number of MMSE operations. IAF-MMSE consists of three steps.

1. As shown in Algorithm 3, IAF-MMSE first removes all obviously malicious references based on the assumption that the majority of references is benign. In other words, if there are n references and more than half of them are benign, the DOC of a benign reference must be greater than or equal to $\lfloor n/2 \rfloor$. For example, if n is 7 and four of them are benign, the DOCs of the benign references must be at least 3. If a reference has a DOC of 2, it is removed since it is obviously malicious (Line 1∼7).

2. Inside-attack references are removed by IAF (Line 8).

3. After the inside-attack references are removed, IAF-MMSE iteratively runs MMSE using the remaining references similarly to EAR-MMSE. However, if multiple references are obviously malicious, they are removed simultaneously. This procedure is repeated until δ is smaller than θ or the number of remaining location references is less than 3. (Line 9∼28). The simultaneous removal reduces the number of MMSE executions and thus increases the computational efficiency.

Security Analysis

IAF-MMSE can estimate a sensor position by removing the effect of the malicious location references when there exist more benign references than malicious ones. To defeat IAF-MMSE, the attacker must first generate inside-attack references satisfying the conditions introduced in Sec. 3.4.2. If an inside-attack reference does not satisfy

Algorithm 3 IAF-MMSE

1: $\mathcal{L} = \{l_1, l_2, ..., l_n\}$ ▷ $l_i = (x_i, y_i, d_i)$
2: $(\mathcal{L}', k) \leftarrow findLocRefLowestDOC(\mathcal{L})$
3: // k is the lowest degree of consistency
4: // $\mathcal{L}'$ is a set of references with k degree
5: **if** $k < \lfloor n/2 \rfloor$ **then**
6: $\quad \mathcal{L} \leftarrow \mathcal{L} - \mathcal{L}'$
7: **end if**
8: $\mathcal{L} \leftarrow IAF(\mathcal{L})$
9: **while** 1 **do**
10: $\quad (\delta, \hat{x}, \hat{y}) \leftarrow MMSE(\mathcal{L})$
11: $\quad$ **if** $\delta < \theta$ **then**
12: $\quad\quad$ **return** $(\hat{x}, \hat{y})$
13: $\quad$ **else**
14: $\quad\quad$ **if** length of $\mathcal{L} < 3$ **then**
15: $\quad\quad\quad$ **return** $failure$
16: $\quad\quad$ **else**
17: $\quad\quad\quad (\mathcal{L}', k) \leftarrow findLocRefLowestDOC(\mathcal{L})$
18: $\quad\quad\quad$ // k is the lowest degree of consistency
19: $\quad\quad\quad$ // $\mathcal{L}'$ is a set of references with k degree
20: $\quad\quad\quad$ **if** $k < \lfloor n/2 \rfloor$ **then**
21: $\quad\quad\quad\quad \mathcal{L} \leftarrow \mathcal{L} - \mathcal{L}'$
22: $\quad\quad\quad$ **else**
23: $\quad\quad\quad\quad l_i \leftarrow selectRandomly(\mathcal{L}')$
24: $\quad\quad\quad\quad \mathcal{L} \leftarrow \mathcal{L} - \{l_i\}$
25: $\quad\quad\quad$ **end if**
26: $\quad\quad$ **end if**
27: $\quad$ **end if**
28: **end while**

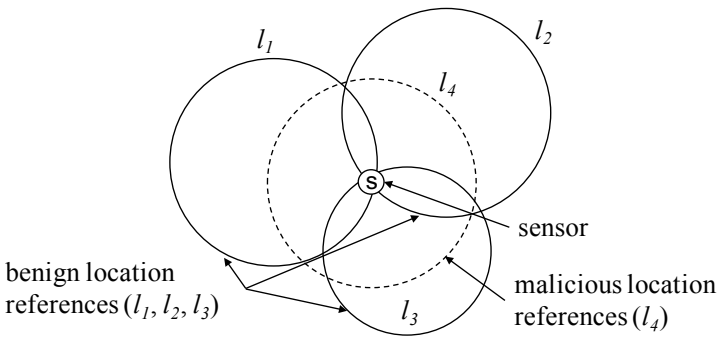


Figure 3.12.: An example showing that IAF-MMSE cannot filter out a malicious location reference.

the conditions, it will be filtered out due to the lack of DOC. Then, the attacker must decrease the measurement distance of an inside-attack reference l_i to reduce the number of intersection points inside l_i. In this case, although the probability that IAF filters out l_i decreases, the maliciousness of l_i also decreases at the same time.

IAF-MMSE obviously has a limit when all references have the same DOC and have a similar number of intersection points. As shown in Fig. 3.12, if a malicious location reference (l_4) is consistent with benign references (l_2 and l_3), the estimated location will be s' (false sensor location) rather than s (true sensor location) since DOCs of l_1, l_2, l_3 and l_4 are equal to 3. In addition, l_1, l_2, l_3 and l_4 include same number of intersection points. However, in such cases, any state-of-the-art secure localization techniques [95, 96] cannot correctly estimate the true sensor position since $\{l_2, l_3, l_4\}$ looks more consistent than $\{l_1, l_2, l_3\}$.

If the attacker obtains information about the benign references held by a target sensor, he/she can abuse the benign references in order to make the malicious references look consistent as in this example. Then, the attacker can introduce estimation errors even when the number of malicious references is smaller than the number of benign references. More details were discussed in [93].

However, we argue that this threat can be mitigated using other defense schemes compatible our scheme as follows: First, if RSSI is used as a distance measurement

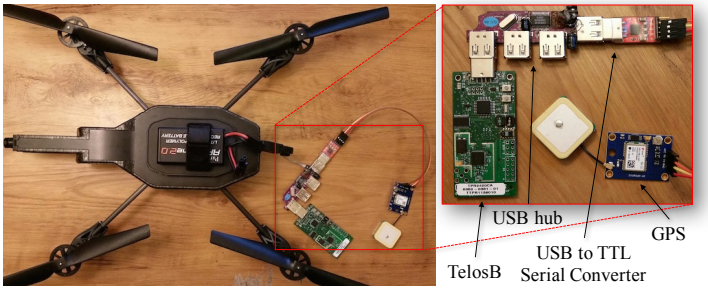


Figure 3.13.: Mobile beacon node prototype

method, encrypting beacon signals with varying the signal strength [109] makes hard for the attacker to learn about the benign location references. Second, the use of hidden/mobile beacon nodes [110] can also mitigate the threat for the same reason. Third, a defense scheme based on measuring the trustworthiness of data [111] can be utilized to filter out malicious location references. Finally, any detection algorithms, such as a scheme based on measuring round trip time [103], a scheme based on domain knowledge [112], or cooperative detection schemes [104,113], can be utilized to detect malicious beacon nodes with increased computational or communication costs.

3.6 Evaluation

In this section, we evaluate our ABP-attack-free path planning scheme and IAF-MMSE through a test-bed and simulations.

3.6.1 Test-bed Experiments

We conducted test-bed experiments in order to check how our ABP-attack-free path planning affects the total traveling time of a mobile beacon. Then, we assess the performance of IAF-MMSE in real environments.

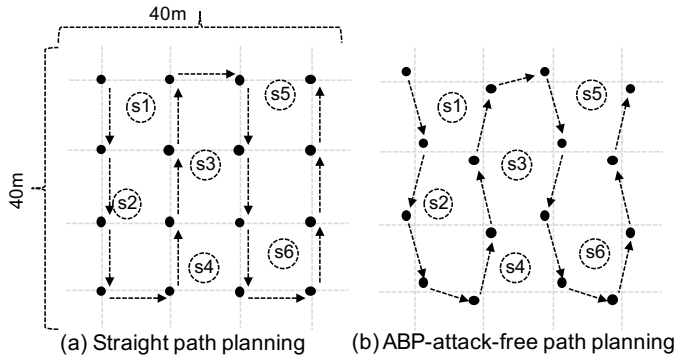


Figure 3.14.: Mobile beacon path

Setup

As sensors, we used TelosB [98]. EAR-MMSE [95] and IAF-MMSE were implemented on the sensors. For a beacon node, we developed a mobile beacon node using AR.Drone2.0 [83]. After the drone is booted, it works as an Wi-Fi access point. After a laptop is connected to the drone as an Wi-Fi client, the drone can be remotely controlled by sending UDP control commands from a laptop. For a GPS receiver, we attached the u-blox-6M [114] whose the estimation error is less than $2.5m$. The GPS receiver communicates with the drone through a USB-to-TTL serial converter (CP2101). To communicate with sensors, a TelosB is attached to the drone. As shown in Fig. 3.13, the GPS receiver and TelosB are connected to the drone through a USB hub. Six sensors were deployed in a $40m \times 40m$ area and the drone flew over the area as shown in Fig. 3.14. d_{max} is set to $16m$ and ϵ is set to $3.3m$ based on our measurements. e_{max} and θ are set to $13m$ and ϵ^2, respectively. The number of benign references received by each sensor is from 5 to 7. After receiving the beacon signals, each sensor measured the distance using RSSI.

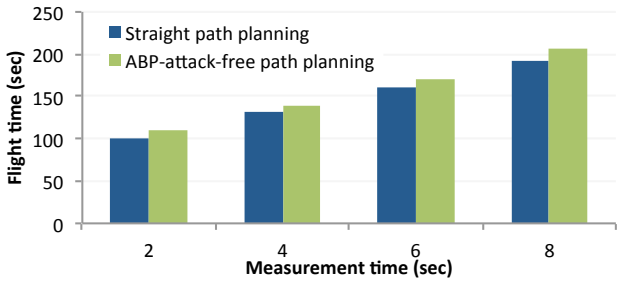


Figure 3.15.: Flight time comparison

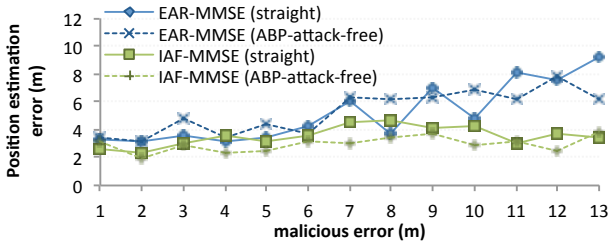


Figure 3.16.: Averaged position estimation errors of six sensors

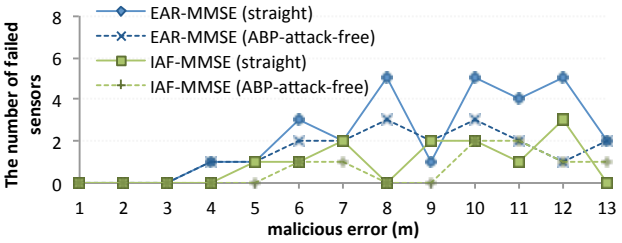


Figure 3.17.: Number of sensors which fail to be measured

Total Flight Time Comparison

We first compare the flight times of the drone with our ABP-attack-free path planning and the straight path planning as shown in Fig. 3.14. Based on the system parameters, we found that λ should be $3m$ to create an ABP-attack-free path as shown in Fig. 3.14 (b). The drone stopped at 16 waypoints and sent beacon signals for 2, 4, 6 or 8 seconds with its location information. The total traveling distances for the straight path and the ABP-attack-free path are 108 and 170, respectively.

However, as shown in Fig. 3.15, the differences in their flight times are less than 9.5% in all cases. The reason is that the drone spends most of its flight time to accurately stop at each waypoint. This result confirms that our ABP-attack-free path planning is not too expensive considering its security benefits.

Performance Comparison

Although the ABP-attack-free path planning prevents ABP-attacks, it cannot prevent inside-attacks. For the inside-attack, we manually input four inside-attack references for each sensor and increased their malicious errors. Each sensor estimated its own position using IAF-MMSE and EAR-MMSE, and sent the result to the base-station.

Fig. 3.16 shows the averaged position estimation errors of six sensors according to the malicious error. Overall the errors are high due to GPS errors and distance measurement errors. Although the drone stopped while sending beacon signals, signals strengths fluctuated according to the environment. The estimation errors go up and down as the malicious errors increase since both algorithms randomly select one reference when more than one references have same minimum DOCs.

When the malicious errors are small, there is no difference between four cases. However, even when the malicious errors become large, the estimation errors of IAF-MMSE do not increase much due to the use of IAF. Since EAR-MMSE does not distinguish inside-attack references from benign ones, it sometimes removes benign references, which also causes large errors. When the malicious error is $8m$, the accuracy of EAR-MMSE with the straight path looks better compared to others. The reason is that, as shown in Fig. 3.17, when the malicious error is $8m$, only one sensor can be successfully located if EAR-MMSE with the straight path is used. Although the ABP-attack-free path cannot prevent inside-attacks, it helps IAF-MMSE to improve its performance since it prevents inside-attack references from abusing benign references.

3.6.2 Simulation Results

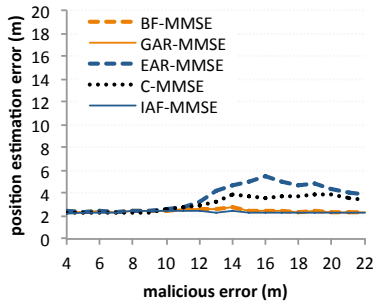
To solely focus on the performance of IAF-MMSE in various attack scenarios, we developed a simulator specialized for sensor localization. Since the considered algorithms are purely based on location references, we did not use general network simulators such as NS-2 [115]. IAF-MMSE is compared with other MMSE-based algorithms: BF-MMSE [95], GAR-MMSE [95], EAR-MMSE [95], and C-MMSE [96].

For all simulations, a target sensor (x_0, y_0) is located at the center of a $30m \times 30m$ area. 7 beacon nodes are randomly deployed in the area and some of them are malicious. The maximum distance measurement d_{max} is set to $26m$. Therefore, the target sensor obtains seven location references. The distance measurement errors are uniformly distributed in $[-\epsilon, \epsilon]$ like in [92, 93, 95] and ϵ is set to $4m$. The threshold η for IAF is set to 5. We utilize the basic MMSE [97] and the threshold θ is set to ϵ^2.

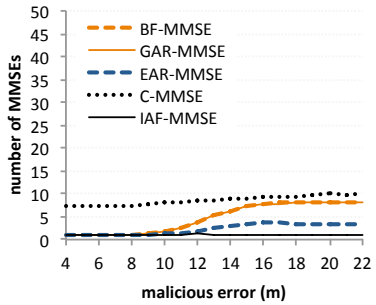
Two attack scenarios, i.e., *inside-attack scenario* and *non-inside-attack scenario*, are considered to show that our algorithm works well in any scenario.

Attack scenario 1 (inside-attack): The beacon position (x_i, y_i) of an inside-attack reference l_i is randomly located within $1.5m$ away from true sensor position. In this scenario, *malicious error* refers to a malicious modification of the measured distance d_i in an inside-attack reference. The number of malicious (inside-attack) references set to 1 or 3.

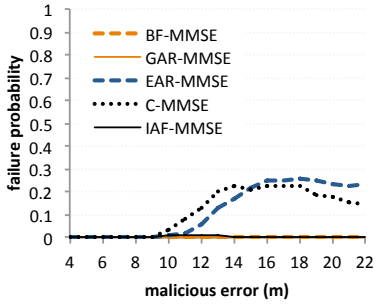
Attack scenario 2 (non-inside-attack): The attacker first declares a specific false sensor position (x_f, y_f) and then creates a malicious reference l_i on which (x_f, y_f) lies, i.e., $d_i = \sqrt{(x_i - x_f)^2 + (x_i - x_f)^2}$. In this scenario, *malicious error* refers to a malicious modification of the distance between the true sensor position and (x_f, y_f). There are two sub-scenarios: *non-collusion* scenario and *collusion* scenario. In the non-collusion scenario, multiple malicious references are provided by non-colluding attackers and thus they declare independent false sensor positions. On the contrary, in



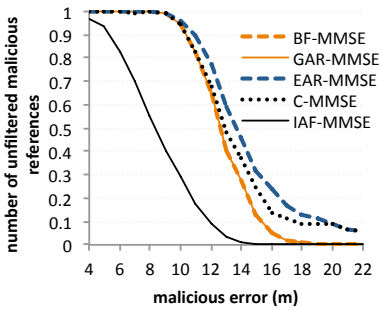
(a) Sensor position estimation error



(b) Number of MMSE operations



(c) Failure probability



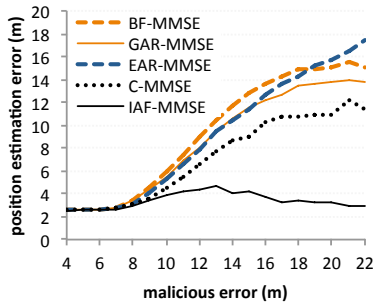
(d) Number of unfiltered inside-attack references

Figure 3.18.: Inside-attack scenario (attack scenario 1). One inside-attack reference exists.

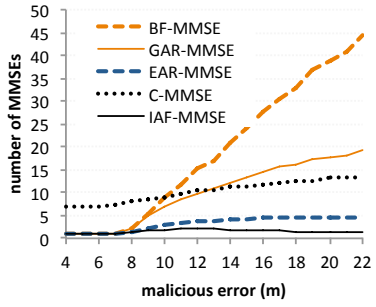
the collusion scenario, three malicious references are provided by colluding attackers so that they have a common false sensor position.

Attack scenario 1

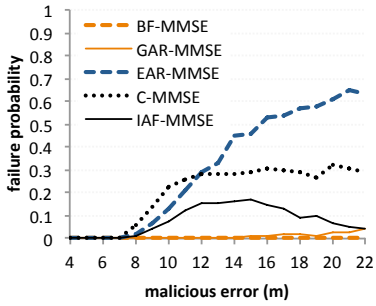
Fig. 3.18 and Fig. 3.19 shows the results when the number of inside-attack references is 1 and 3, respectively. When there is one inside-attack reference, the position estimation accuracy of IAF-MMSE is similar to BF-MMSE and GAR-MMSE (see Fig. 3.18(a)). However, the accuracy of EAR-MMSE and C-MMSE decreases as the malicious error increases since they cannot filter out the inside-attack reference.



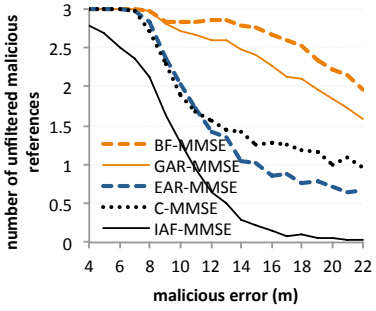
(a) Sensor position estimation error



(b) Number of MMSE operations



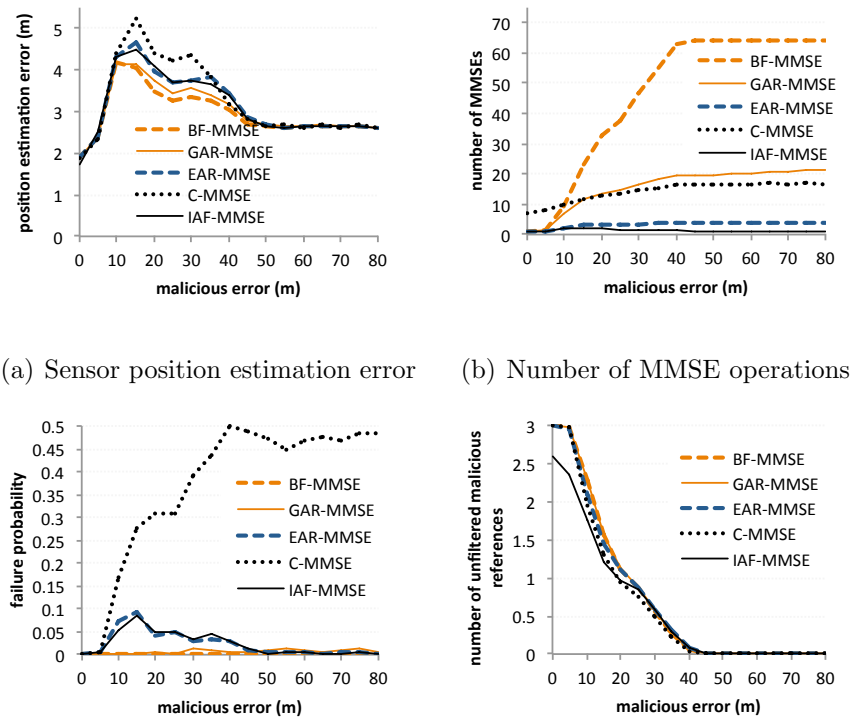
(c) Failure probability



(d) Number of unfiltered inside-attack references

Figure 3.19.: Inside-attack scenario (attack scenario 1). Three inside-attack references exist.

When there are three inside-attack references, the accuracy of IAF-MMSE is the best and does not decrease much even when the malicious error increases. Interestingly, the accuracy of BF-MMSE and GAR-MMSE as well as EAR-MMSE severely degrades as the malicious error increases. However, the reasons for the accuracy degradation are different. EAR-MMSE, as shown in Fig. 3.19(c), has the highest failure rate since it cannot distinguish malicious references from benign references and thus removes benign references with high probability. In the case of BF-MMSE and GAR-MMSE, although their failure rates are very low, the unfiltered rates (see Fig. 3.19(d)) are higher than other schemes, which means that BF-MMSE and GAR-MMSE seldom filter out inside-attack references when there are 3 inside-attack references. They



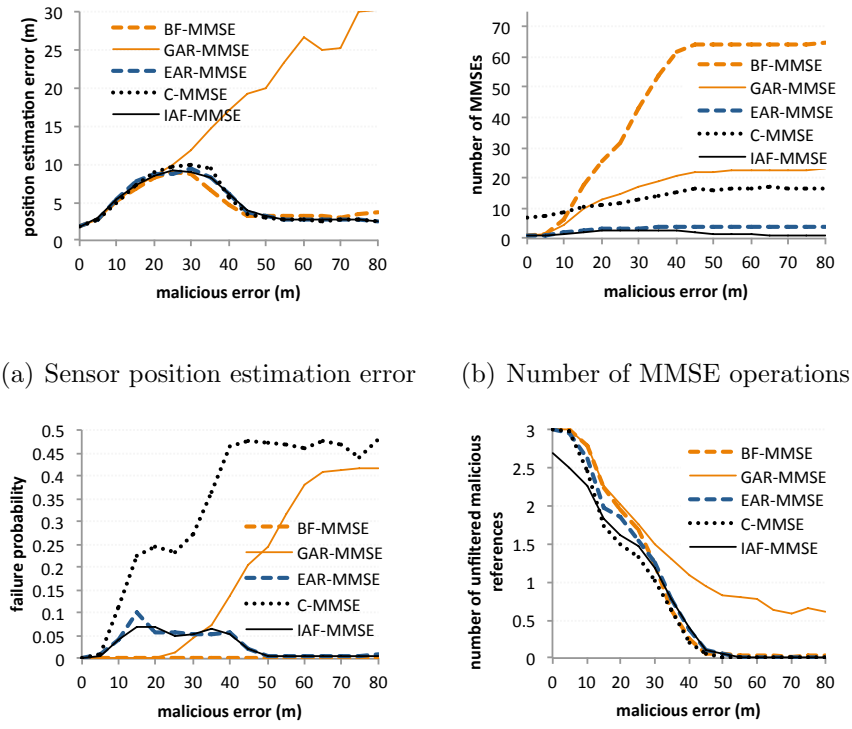
(a) Sensor position estimation error (b) Number of MMSE operations

(c) Failure probability (d) Number of unfiltered malicious references

Figure 3.20.: Non-inside-attack (attack scenario 2). Three malicious location references do not collude for a false sensor location.

just try to estimate the best sensor location with all references, which results in large estimation errors.

In terms of the number of MMSE operations, IAF-MMSE shows the best performance (see Fig. 3.18(b) and Fig. 3.19(b)). When there are 3 inside-attack references and the malicious error is $22m$, IAF-MMSE requires only 1.2 MMSE operations, whereas EAR-MMSE requires 4.7 MMSE operations on average. The reason is that IAF-MMSE removes inside-attack references before MMSE is iteratively executed.



(a) Sensor position estimation error

(b) Number of MMSE operations

(c) Failure probability

(d) Number of unfiltered malicious references

Figure 3.21.: Non-inside-attack (attack scenario 2). Three malicious location references collude for a false sensor location.

Attack scenario 2

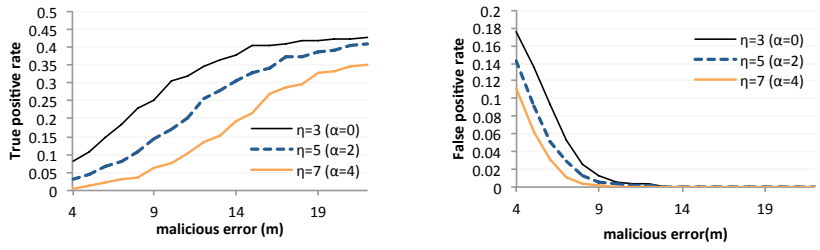
Non-collusion scenario: Fig. 3.20 shows the results when there are 3 malicious non-colluding references. All the algorithms provide similar position estimation accuracy (see Fig. 3.20(a)) when they succeed in estimating sensor positions. As shown in Fig. 3.20(c), the failure rates of all the algorithms, except C-MMSE, are low. In the case of C-MMSE, its failure rate is very high when the malicious error is large. C-MMSE randomly selects two references as seeds for clustering. However, the probability that two references are both benign is very low in this scenario.

IAF-MMSE requires the least number of MMSEs (see Fig. 3.20(b)). For instance, IAF-MMSE requires approximately 1 MMSE operation when the malicious error is $80m$, while EAR-MMSE, C-MMSE, GAR-MMSE and BF-MMSE requires about 4, 17, 21 and 64 MMSE operations, respectively. The reason is that IAF-MMSE can filter out multiple malicious references at the same time.

Collusion scenario: If multiple malicious references collude, the overall position estimation errors of all the algorithms become higher than the errors in the non-collusion scenario (see Fig. 3.21). As shown in Fig. 3.21(a), all algorithms except GAR-MMSE provide similar estimation errors and their estimation errors become highest when the malicious error is $25m$, while the estimation errors of GAR-MMSE linearly increase as the malicious error increases. When 3 malicious references collude, if one benign reference coincides with the false sensor position created by three malicious references by accident, it is difficult to distinguish between the true sensor position and the false sensor position. However, since EAR-MMSE and IAF-MMSE utilize the DOC to filter out such colluding references, the estimate error decreases as the malicious error increases.

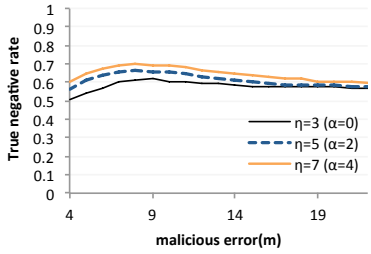
However, GAR-MMSE does not utilize the DOC. Therefore, once one benign reference is removed, the sensor position will be estimated to the false sensor position with a high probability. In the case of BF-MMSE, since it executes MMSE operations for all possible subsets of all location references, it can filter out colluding references at the cost of a high computational overhead. In the case of C-MMSE, although its estimation errors are similar to IAF-MMSE, its failure rate is high when the malicious error is large. It randomly selects two references as seeds for clustering and both must be benign. However, the probability that two references are both benign is very low in this scenario.

As shown in Fig. 3.21(c), the failure rates of GAR-MMSE increase as the malicious error increases. GAR-MMSE selects one potentially malicious reference from set $\mathcal{L} = \{l_1, l_2, ..., l_n\}$ by checking $\delta = MMSE(\{l_1, l_2, ...l_n\} - l_i)$ for all i $(= 1, 2, ..., n)$. When a location reference l_j is absent, if δ becomes lowest, l_j is permanently removed from

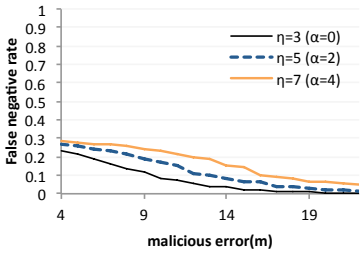


(a) True positive rate

(b) False positive rate



(c) True negative rate



(d) False negative rate

Figure 3.22.: IAF performance according to the threshold η when three inside-attacks exists (attack scenario 1)

$\mathcal{L}$. However, when multiple references are colluding, δ is not useful as an indicator for filtering since all δ values are very large. Thus, GAR-MMSE filters out benign location references with a high probability. For these reasons, the failure rate of GAR-MMSE increases as the malicious error increases. As shown in Fig. 3.21(d), GAR-MMSE cannot filter out malicious references well.

As shown in Fig. 3.21(b), IAF-MMSE requires the lowest number of MMSE operations. IAF-MMSE needs only 1 MMSE operation, while EAR-MMSE needs 4 MMSE operations when the malicious error is $80m$. A small number of MMSE operations is required for resource-constraint sensors, especially when iterative MMSE methods (e.g., Gauss-Newton method) are utilized.

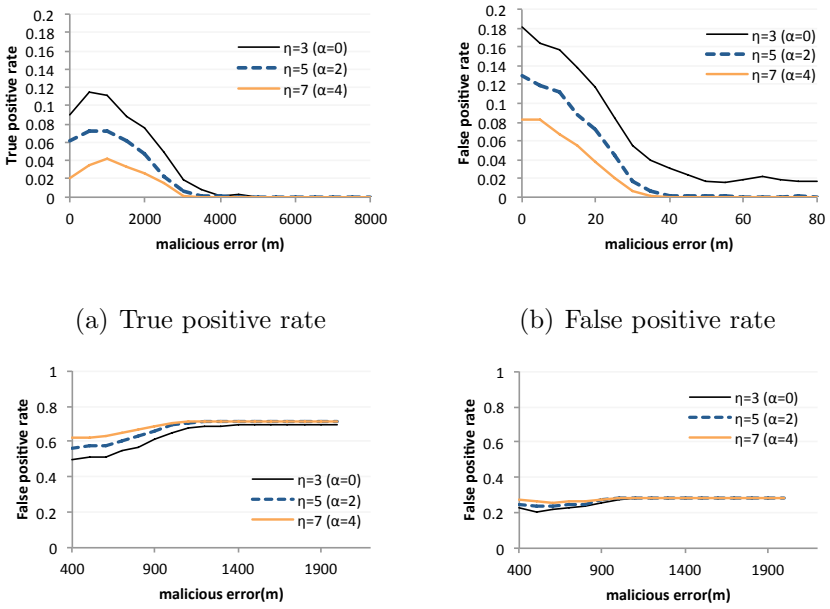
Inside-Attack Filtering Performance

Fig. 3.22 shows the performance of IAF when there are 3 inside-attack references and the threshold η is set to 3, 5 or 7. When a filtering algorithm filters out a reference, if the filtered reference was a malicious one, the filtering result is a *true positive* (see Fig. 3.22(a)). On the other hand, if the filtered reference was a benign one, the filtering result is a *false positive* (see Fig. 3.22(b)). If a filtering algorithm decides not to filter out a malicious reference, the result is a *false negative* (see Fig. 3.22(d)). On the contrary, if a filtering algorithm does not filter out a benign reference, the result is a *true negative* (see Fig. 3.22(c)). Overall, IAF filters out inside-attack references well and does not filter out benign references when the malicious error is large.

IAF-MMSE filters out inside-attack references based on the number of inner intersection points and η. Therefore, the size of η affects the performance of IAF-MMSE as discussed in Sec. 3.5.2. As shown in Fig. 3.22(a), the lower η is, the more aggressively IAF filters out inside-attack references. However, at the same time, more benign references are also filtered out (see Fig. 3.22(b)), which may result in a bad impact on the overall performance. For example, when 3 malicious references collude (attack scenario 2), if η is set to 3, the false positive rate becomes too large. Therefore, many benign references are removed by IAF (see Fig. 3.23(b)). On the other hand, if η is set to 7, the true positive rate is too small since IAF loosely filters out inside-attack references. When η is set to 5, IAF-MMSE performs well in any scenario.

Execution Time for Localization Algorithms

To assess the time required for a position estimation in a real sensor platform, we implemented two algorithms, i.e., EAR-MMSE and IAF-MMSE on TelosB [98]. TelosB is a representative sensor platform with a low-speed processor (MSP430 f1611, 4MHz). We selected EAR-MMSE since it results in the least number of MMSE operations after IAF-MMSE. We utilized same localization reference data generated by the simulator when there are 3 inside-attack references (attack scenario 1). Fig. 3.24



(a) True positive rate

(b) False positive rate

(c) True negative rate

(d) False negative rate

Figure 3.23.: IAF performance according to the threshold η when three colluding non-inside-attacks exists (attack scenario 2)

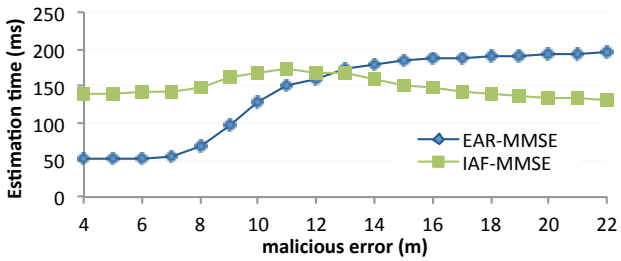


Figure 3.24.: Required time for a position estimation by TelosB

shows the time required for a position estimation. When the malicious error is low, IAF-MMSE requires more time than EAR-MMSE due to the execution of IAF. However, as the malicious error increases, IAF-MMSE requires less time than EAR-MMSE due to two reasons. First, the number of MMSE operations in IAF-MMSE does not increase as the malicious error increases, while the number of MMSE operations in

EAR-MMSE increases as already shown in Fig. 3.19(b). Second, as the malicious error increases, the number of intersection points decreases. Therefore, IAF spends less time to find intersection points.

3.7 Summary

In this chapter, we introduced two attacks exploiting knowledge about sensor positions, i.e., the *aligned-beacon-position attack* and the *inside attack*, by which an attacker can severely distort the estimated locations of sensors. We found that most state-of-the-art MMSE-based schemes are vulnerable to such attacks. To prevent the ABP attack, we introduced a beacon deployment strategy and showed its cost through test-bed experiments. To defend against the inside attack, we proposed a inside-attack filtering algorithm, which filters out malicious location references based on the number of intersection points. We also proposed an efficient algorithm which estimates sensor positions when the number of benign location references is greater than the number of malicious location references. Our simulations and test-bed experiments show that our algorithm can efficiently and accurately estimate sensor positions by filtering out inside-attacks and other malicious location references.

4 SECURING MOBILE DATA COLLECTORS BY INTEGRATING SOFTWARE ATTESTATION AND ENCRYPTED DATA REPOSITORIES

4.1 Introduction

Drones, like many networked computing devices, are vulnerable to malicious cyber and physical attacks, such as memory analysis, eavesdropping, impersonation and manipulation. An attacker can install a malicious program on a drone by physically capturing the drone or by exploiting software vulnerabilities. Once compromised, the drone is under the control of the attacker. The hidden malicious program can tamper with secrets stored in the drone, or steal valuable data. It is thus critical to verify that the software running on the drone is not compromised. An approach to detect compromised software is through code attestation, i.e., the ground station verifies that a given drone is still running the initial application and, hence, has not been compromised.

However, code attestation itself does not guarantee that the data in the drone's memory are not leaked. During the interval between two attestation executions, malware may steal the data collected by the drone and the secret encryption keys used to securely communicate with the ground station. If the drone's software is reverted to its original state after such attacks, the attestation technique cannot detect the compromise. Using the stolen secret keys, the attacker can send false control messages to the drone or send false sensor data to the ground station. Even worse, terrorists may make the compromised drone crash on humans or against a passenger airliner. Therefore, it is also crucial to protect secrets and data stored in the drone.

Many existing software-based attestation techniques [15–17, 116] cannot be used for mobile data collectors due to unpredictable network delays. Others [19,20] cannot

directly be adopted either since they do not address the security of the collected data. In addition, as pointed out by Castelluccia et al. [21], these schemes only consider program memory, but not data memory. Thus, an adversary can utilize free spaces of the data memory to hide malware and restore the original program during the attestation to correctly calculate a checksum. We refer to such an attack as *data memory attack*.

In this chapter, we propose an integrated solution that effectively supports three key security functions for mobile data collectors: 1) software-based attestation, 2) collected data encryption and 3) secret key leakage prevention. Our solution relies on the notion of *randomized data repositories*, that is, repositories filled in with random numbers, similar to the scheme in [19], filling the free program memory with random numbers. However, in our approach, unlike [19], such repositories achieve two security objectives: 1) software attestation and 2) encryption of collected data. Also, our solution prevents an attacker from misusing free spaces in data memory, i.e., for data memory attacks, by filling up the spaces with look-up tables for white-box encryption. The look-up tables for white-box encryption achieve three security objectives: 1) encryption of messages for communications, 2) secret key leakage prevention and 3) protection from data memory attacks. The main contributions of our work are as follows:

- We introduce a software-based attestation technique that fills up free memory spaces with pseudo-random numbers, which are also utilized to encrypt data collected by the drone like a stream cipher. Our software-based attestation technique is suitable for mobile networked devices like drones since it tolerates network delays. Also, the encryption using data repositories assures that an attacker cannot obtain any information about the collected data even if the attacker can see the whole memory space.

- We propose a group-based attestation to address the case in which a ground station needs to efficiently verify the software integrity of a fleet of drones.

- We propose to use a technique that converts short secret keys, like 128-bit AES keys, into large look-up tables which fill up the remaining space of the data memory. The tables prevent malware from residing in the data memory. The large size of the tables also makes it hard for the malware to send the tables to the remote malware operator.

Note that, even though our current design and implementation are focused on drones, our integrated solution can be adopted for other types of mobile devices that collect data.

4.2 Background

Code attestation: To guarantee the integrity of the execution environment, security hardware like Trust Platform Modules (TPM) [14] can be utilized. However, hardware-based security approaches do not cover most of the current commercially-available drones, such as drones for hobbyists, since they are not equipped with such security hardware.

An alternative to security hardware is the use of software-based attestation. Such a technique attests code, static data, and configuration settings of an embedded device without requiring dedicated hardware and physical access to the device. Obviously, software-based attestation techniques require lower cost than hardware-based attestation technique. Furthermore, since software-based techniques work thoroughly in software, they are easily updatable. Due to such advantages, several software-based attestation techniques have been proposed [15–20] for resource-constrained embedded devices like sensors.

Software-based attestation requires a remote verifier, which performs the attestation, since a potentially compromised device cannot be trusted to verify itself correctly without security hardware. Since the verifier is separated from the device (prover) being verified, the verifier cannot directly read or write onto the device's memory. Thus, most software-based attestation techniques are based on challenge-response protocols

between the verifier (the ground station) and a prover (a target drone). The verifier sends a challenge to the target device to prevent replay or pre-computation attacks. The prover computes a response to this challenge using a verification function. Since the verifier is assumed to know the exact memory contents and hardware configuration of the prover, it can compute the expected response and compare it with the received one. If the values are equal, the target device is untampered; otherwise, it is considered to be compromised.

Protecting secret keys: Code attestation itself does not guarantee the confidentiality of messages exchanged between the drone and the ground station. An adversary may just want to steal a secret key from the drone's memory and eavesdrop on messages by decrypting them using the stolen secret key. In the context of the white-box attack model [36], i.e., in untrusted execution environments, the traditional symmetric key-based cryptographic primitives with short secret keys, such as keys for AES and HMAC, do not guarantee confidentiality and integrity. White-box attackers can see and manipulate the internal state of the memory of a victim device, by installing a malware program, or by capturing the device and analyzing the memory. Various white-box attacks, such as entropy attack [22], cold boot attack [23], and S-box blanking attack [24], have shown that a short secret key, such as an 128-bit AES key, could be successfully extracted. In 2002, Chow et al. introduced the study of cryptographic implementations [36, 41] based on traditional block ciphers, such as AES and DES, in the context of the white-box attack model. Since then, several variants [117–119] have been proposed. However, they all have been practically cryptanalyzed [46, 47, 120].

Bogdanov and Isobe [121] proposed a white-box secure block cipher SPACE and its variant N-SPACE. Their security against key extraction attacks relies on the black-box security of traditional block ciphers such as AES, or Triple-DES. (N-)SPACE cipher is based on generalized Feistel networks (GFNs) that have been studied as a building block of block ciphers. The F function in the GFN is a traditional block cipher like AES. As shown in Fig. 4.1, at each round, the F function takes some output

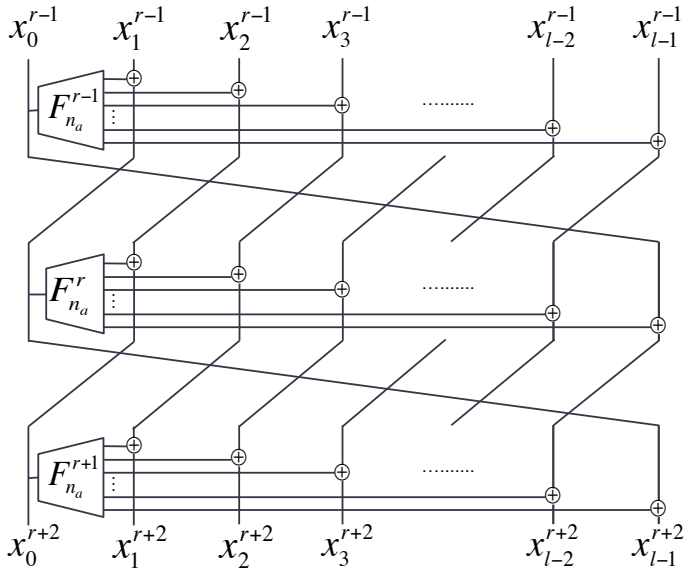


Figure 4.1.: The Feistel structure of SPACE [121]. This shows the encryption of an m-bit plaintext. $Xr = \{x_0^r, x_1^r, ..., x_{l-1}^r\}$ denotes m-bit state of round r and the size of each line, i.e., x_i^r , is $m_a(= m/l)$-bit.

bits (x_0^{r-1}) of the previous round as input bits, and encrypts them. The output bits of the F function are XOR-ed with the other output bits (x_1^{r-1}, x_2^{r-1}, ..., x_{l-1}^{r-1},) of the previous round. The key idea of (N-)SPACE is to change the key-dependent F function into table-lookups. Since the short secret key used to generate the look-up table is deleted after the table generation, an attacker cannot extract any information about the short secret key from the look-up table. Bogdanov and Isobe proposed four variants of SPACE(m_a, R), i.e., SPACE(8, 300), SPACE(16,128), SPACE(24, 128) and SPACE(32, 128), where m_a is the number of input-bits for the F function and R is the number of rounds. Their table sizes are 3.84 KB, 896 KB, 208 MB and 48 GB when the bit-size (m) of a plaintext is 128, respectively. The bit-size of each look-up table is given by $2^{m_a}\times(m\text{-}m_a)$, where 2^{m_a} denotes the number of table entries and (m-m_a) is the size of each entry. In SPACE, only one short secret key is involved to generate the look-up table. However, one of the nice features of N-SPACE is that the

look-up table size can be variable according to the number of short secret keys. We utilizes this feature for our work.

4.3 Related Work

The existing software-based attestation schemes can be classified into two categories: schemes based on response time estimation [15–17, 116], and schemes based on filling empty memory space [19, 20].

SoftWare-based ATTestation (SWATT) [17] and Indisputable Code Execution (ICE) [15, 16, 116] rely on response times of challenge-response protocols and are based on the fact that the checksum computation slows down noticeably if the adversary tempers with the device memory. However, such schemes cannot be utilized for drones since they require very precise estimation of response times, and are thus very sensitive to unpredictable network delays and also dependent on hardware platforms. As the platforms of drones are diverse and the wireless communication channel between a drone and the ground station varies according to the network conditions, such as network traffic congestion or packet collisions, the timing-based approach is not applicable.

On the other hand, schemes that fill the free program memory space with pseudo-random numbers [19, 20] make it hard for the adversary to store malware in memory. If the prover receives a challenge, it generates a checksum over the entire program memory using the challenge as a seed. However, as pointed out by Castelluccia et al. [21], these schemes only consider the program memory, but not the data memory. Therefore, the adversary can utilize the free data memory to hide malicious code and restore the original program during the attestation in order to successfully generate the correct checksum. In addition, such schemes cannot be directly applied to drones since drones need to collect data and store them in the program (non-volatile) memory.

All the existing software-based attestation schemes consider a single device for attestation, and thus are not scalable. The ground station (verifier) must execute the attestation procedure n times for n drones. Asokan et al. [122] proposed a Scalable Embedded Device Attestation (SEDA) protocol for embedded devices. They showed that the protocol can efficiently verify the software integrity of a large number of devices using read-only memory and a memory protection unit. SEDA is secure under the assumption that the attacker cannot access cryptographic secrets in the devices. Ibrahim et al. [123] proposed the *heartbeat* protocol and integrated it with SEDA to address physical attacks. The protocol makes the devices periodically broadcast *heartbeat* messages to the neighbors. After collecting the *heartbeat*s from the devices, the verifier can infer the occurrence of physical attacks if there are missing *heartbeat*s since the physical attacks cause the devices to be powered off. Our attestation protocol is similar to the SEDA-based protocols [122, 123] because our protocol also addresses multi-device attestation. However, our attestation protocol differs from the SEDA-based protocols, since they are hardware-assisted, while ours is purely based on software.

All the existing attestation protocols [15–17, 19, 20, 116, 122, 123] do not consider the case in which devices collect data in memory and do not cover the protection of the collected data, and thus cannot be directly applied to data collection applications.

4.4 System Model

Drone: We assume that the drones are mobile embedded devices equipped with low-end smartphone-like computing power and data memory. For instance, AR.Drone2.0 [83] is equipped with 1GHz ARM Cortex A8 CPU and 1GB RAM. We also assume that the drones as mobile data collectors have a large amount of program memory enough to store the data collected during their missions. The drones run a drone control program (DCP), which is a main program for flight control and mission execution. We assume that the maximum amount of the data memory to run drone operations, e.g.,

flight operations, video processing and encryption, is known and preassigned when the drone starts. We assume that the drones operating system (OS) has minimal functionality to operate the drone. Therefore, it is hard for an attacker to delete any parts of the OS/DCP and exploit the preassigned data memory space for malware without being detected. The ground station knows all the original memory contents in each drone.

Network: We consider a drone network composed of a large number of drones like in [122,123]. The drones can build an ad-hoc network to communicate with each other and cooperatively carry out a common mission such as surveillance and search/rescue. The drones can collect sensor data from on-ground sensors and image data by taking pictures/videos. The drones immediately send the collected data to the ground station. The drones store the collected data in their flash memory only when the data are too large to be sent via the communication network or the ground station does not need them right away. The ground station continuously checks the status of each drone by pinging it and monitoring its trajectory.

Adversary: We assume that the ground station is always secured. The attacker attempts to compromise the software running on the drone by injecting malicious code exploiting software vulnerabilities. However, we do not consider the case in which an attacker installs additional memory to place malware in it. The reason is that the ground station can easily infer that a drone is compromised if the drone is disconnected for a long time, its hardware is modified, and its trajectory is abnormal. Indeed, no software-only attestation technique [17–20] is able to guarantee the ability to detect compromised drones when additional memory is installed. Similarly, we do not consider the case in which an attacker uses an additional drone (d_{new}) that colludes with a compromised drone (d_{cmp}). Such attacks are not impossible. However, in this chapter, we assume that it is hard for the attacker to create a covert channel for collusion between d_{new} and d_{cmp} since d_{cmp} must be in the communication range of the ground station during attestation. For instance, the drone can be equipped with an external communication module, like a wi-fi (or 3G) dongle, which does not allow

any program in the drone to manipulate the signal power of the module on purpose. Thus, malware in the drone cannot create a covert channel by reducing the drone's signal power.

We assume that the attacker wants to steal the data stored at the drone, such as collected data, keys for encryption. The attacker can perform any cyber/physical attacks including the known white-box attacks. If the attacker succeeds in stealing a secret key shared with the ground station, he/she can successfully eavesdrop on messages from the ground station and send false messages to the ground station. We assume that malware in the drone can covertly send short secret keys, like 128-bit AES keys or private keys for public key cryptography, to the remote malware operator without being detected. However, we assume that remote malware operator cannot download the entire look-up table without being detected even if the malware can see and extract the look-up tables for two reasons. First, the use of a white-box attack tool like a debugger inevitably makes the drone control program slow. Therefore, the drone can be regarded as compromised if the drone's real-time operations become lagging. Second, the drone can intentionally use up the uplink bandwidth by sending dummy data whose priority is lower than the priority of other useful data like photos or videos. Since the available uplink bandwidth for the malware to send the look-up tables is limited, it is hard for the malware to complete the transmission of the look-up tables during the drone's flight or during the time between two attestations.

4.5 Requirements for a Mobile Data Collector

The design goals are summarized as follows.

- **Code attestation:** The ground station must be able to verify the genuineness of the drone program without strict time measurements. If a drone is compromised with malicious code, this compromise must be detected by a software-based attestation protocol. Also, the attestation protocol must be scalable so

that the ground station can efficiently verify the integrity of a large number of drones.

- **Collected data protection:** The data collected by a drone before the drone is compromised must not be decrypted even if an attacker captures the drone and sees its entire memory contents. Note that we only consider the protection of data collected before the drone is compromised. Only the ground station must be able to decrypt the encrypted data. The encryption method must be computationally efficient so that the drone can encrypt data generated at a high speed, like high-precision videos, in real time. RSA is not a viable solution since it is too computationally expensive. One possible solution is that the drone generates a temporary secret key (sk) for data encryption and encrypts sk using the ground station's RSA public key. However, this solution is insecure since sk must reside in the drone memory during encryption. The attacker can decrypt the data encrypted with sk after extracting sk from the memory. Similarly, the use of Elliptic Curve Cryptography is not secure. After the drone and the ground station agree on a shared key using Elliptic Curve Diffie-Hellman, the key resides in the drone memory during encryption. Also, the attacker may be able to extract the private key of the drone after analyzing the drone's memory.

- **Secret key leakage prevention:** It must be hard for attackers to obtain secret keys used to encrypt messages exchanged with the ground station. Protection of such keys must be achieved without security hardware.

4.6 Proposed Solution

4.6.1 System Overview

Memory layout: Fig. 4.2 shows the memory layout of our secure drone platform which is equipped with program memory and data memory. The program memory is a non-volatile storage, like flash memory, that contains the operating system and the

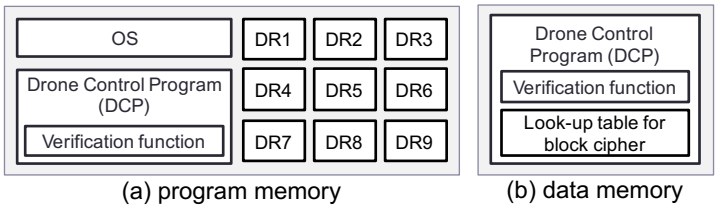


Figure 4.2.: Memory layout

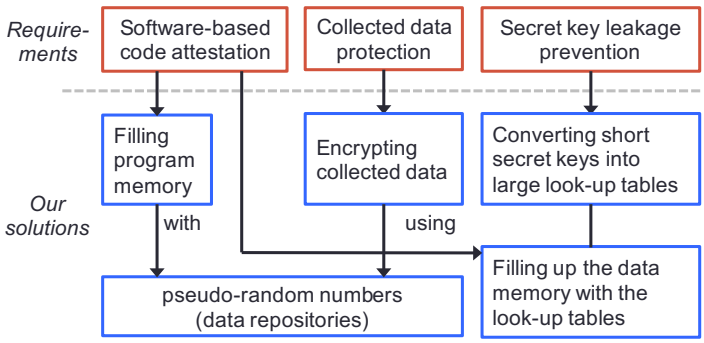


Figure 4.3.: Requirements and our solution

drone control program with the verification code. The rest of the program memory is filled up with *data repositories* (DRs), which consist of pseudo-random numbers (see Fig. 4.2 (a)). Note that DRs are logical partitions of the pseudo-random numbers and the sizes of all DRs are equal except the last one. Since the program memory is fully filled with DRs, virtual memory is disabled. Fig. 4.2 (b) shows the data memory layout. The data memory is a volatile storage that contains registers, stack, heap and static data segments for running programs. Notice that the data memory is filled up with the look-up table for the white-box secure block cipher.

Our solutions: Fig. 4.3 summarizes the requirements and our solutions. First, for code attestation, we adopt an attestation protocol based on filling empty memory space with pseudo-random numbers like in the approach by Yang et al. [19]. Second, the pseudo-random numbers are utilized to encrypt the collected data. The collected data and the pseudo-random numbers are XOR-ed like in a stream cipher. Finally, to

protect the messages exchanged between the drones and the ground station, we utilize a white-box secure block cipher with large look-up tables. Since short secret keys are deleted after they are converted to white-box secure look-up tables, the attacker cannot obtain any information about the secret keys. Even a malicious program in the drone cannot send them to the remote attacker without being detected due to its large size as mentioned in the adversary model. Also, since the look-up tables fill the data memory, the attacker cannot hide malware in the data memory during attestation.

4.6.2 Setup

We assume that there are n drones. Before the drones start their missions, they are in a secure place, which means that the drones and the ground station can communicate with each other via a secure channel. Also, while in the secure place, we assume that the drones have the genuine programs in their program memory and cannot be compromised. Let the free space in the program memory be l_f.

1) DR generation: The ground station selects N secret keys ($sk_1, sk_2, ..., sk_N$) and generates $DRx = PRNG(sk_x)$ ($1 \leq x \leq N$). $PRNG$ is a cryptographically secure pseudo-random number generator like AES-CTR. The size of a DR is l_{DR} and, thus, a drone can have K DRs ($K = \lceil l_f / l_{DR} \rceil \leq N$) in its program memory. The ground station randomly distributes DRs to the drones. DR^i_k ($1 \leq k < K$) denotes the k-th DR stored by drone i. For instance, the drone with id i may have $DR2$, $DR5$, $DR7$ and $DR8$ when K is 4 and $N \geq 8$. Then, DR^i_1, DR^i_2, DR^i_3 and DR^i_4 are $DR2$, $DR5$ $DR7$ and $DR8$, respectively.

2) Look-up table generation: The ground station generates a look-up table lt and sends lt to the drones. The size of the look-up table is l_{lt}, which is equal to the remaining space of the data memory. To fit the table into the remaining space of the data memory, we devise a variant of the SPACE cipher, called MF-SPACE. Note that in SPACE, the size of a look-up table is fixed, i.e., $2^{m_a} \times (m - m_a)$ bits.

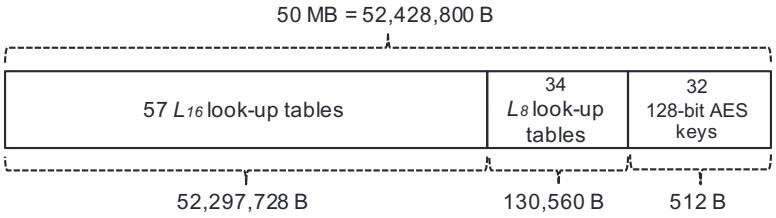


Figure 4.4.: Filling up a free space of data memory

For example, SPACE(8, 300) generates a look-up table using one secret key and its table size is 3,840 B (=$2^8 \times (128-8)/8$ bytes) when the plaintext size is 128-bit. For each round, SPACE looks up the only look-up table. However, MF-SPACE generates multiple look-up tables using multiple secret keys. Let L_8 be the look-up table whose size is 3,840 B, and L_{16} be the look-up table whose size is 908 KB (=917,505 B). For instance, as shown in Fig. 4.4, if the remaining space of the data memory is 50 MB (=52,428,800 B), the total size of the multiple look-up tables must be 50 MB. MF-SPACE first generates 123 (=57+34+32) AES keys. Here, 123 is the smallest number of AES keys that are used to generate look-up tables to fill up the data memory. That is, MF-SPACE generates 57 L_{16} tables (=57×917,505 B = 52,297,728 B) using the first 57 AES keys and fills up the data memory as much as possible with those 57 L_{16} tables. Then, it generates 34 L_8 tables (=34×3,840 B =130,560 B) using the next 34 AES keys and fills up the remaining data memory. Then, the remaining data memory 512 B can be filled with the last 32 AES keys.

For encryption/decryption, MF-SPACE looks up the 91 look-up tables and uses the 32 AES keys in the order according to which they have been generated. For the first 57 rounds, MF-SPACE looks up the L_{16} tables. For the next 34 rounds, it looks up the L_8 tables. For the next 32 rounds, it just executes AES using the 32 AES keys. After 123 rounds, MF-SPACE starts from the beginning and this process is repeated until the number of total rounds becomes 300. Note that 300 is the number of rounds recommended by Bogdanov et al. [121].

Note that the look-up tables have two objectives. First, they are used to encrypt/decrypt messages exchanged with the ground station. Second, they prevent malware from residing in the data memory. If the attacker deletes some of the tables, the ground station can conclude that the drone is compromised since the drone will fail to encrypt/decrypt messages.

3) Anti-steganography key generation: The ground station distributes an *anti-steganography key* (*ask*) to the drones before the drones start their missions, and periodically updates it. *ask* is a AES-128 key and is used to transform (i.e., encrypt) collected data to ciphertext (see Sec. 4.6.4). We call it *anti-steganography key* since this is not mainly intended for data encryption since such a short secret key can be leaked by a white-box attacker. Instead, we utilize the diffusion property of AES in order to prevent steganography attacks (as discussed in Sec. 4.7.2). In short, it is hard for an attacker to obtain particular output bits (malware) by manipulating some given input bits (e.g. an image) since changing one bit of the input will change the half of the output bits on average.

4.6.3 Message Encryption/Decryption

After a drone leaves the secure place for its mission, the drone must securely send data, such as collected data and telemetry, to the ground station. The ground station also has to securely send command/control messages to the drone. To encrypt/decrypt such messages, they utilize MF-SPACE.

We refer to $Enc(p)$ as the ciphertext of p and $Dec(c)$ as the plaintext of c. The ground station can keep the AES keys in memory rather than keeping all the look-up tables in order to save its storage space.

4.6.4 Collected Data Protection

The program memory of drone i is filled with the pseudo-random numbers and their logical partitions are DR_1^i, DR_2^i, ..., DR_K^i. If drone i collects some sensor

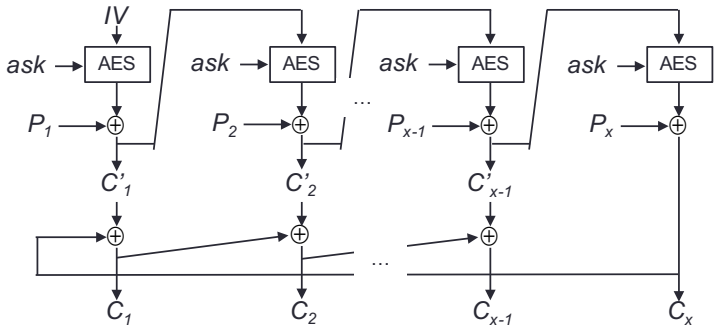


Figure 4.5.: AES-CFB Loopback ($C = E_{ask}(P)$)

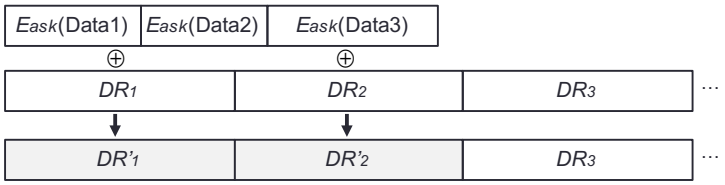


Figure 4.6.: Data encryption using DR ($\kappa = 2$)

data P, it first encrypts P using ask, i.e., $C = E_{ask}(P)$. Here, $E_{ask}(P)$ denotes AES encryption with the Cipher FeedBack LoopBack (AES-CFB-LB) mode in order to prevent steganographic attacks [124]. As shown in Fig. 4.5, AES-CFB-LB is the same as AES-CFB except that the last output C_x is XORed with the first intermediate output C'_1. Then, the next intermediate output C'_2 is XORed with C_1. In this manner, an intermediate output C'_i $(1 < i \leq x - 1)$ is XORed with the previous final output C_{i-1}. The goal of AES-CFB-LB is to make any single bit change of the plaintext statistically change the half of all the bits in ciphertext. Note that in AES-CFB a single bit change of P_i in a block only affects the ciphertext of the block (C_i) and the ciphertext of the following blocks (C_j) $(j > i)$.

Then, as shown in Fig. 4.6, each bit of C is XORed with the pseudo-random numbers (bits) in DR^i_1 like a stream cipher. If all bits in DR^i_k $(1 \leq k < K)$ are used for encryptions, the bits in DR^i_{k+1} are used to encrypt data sequentially. The

ground station can decrypt the collected data since it knows all DRs. If all drones have different DRs, only the ground station can decrypt the collect data.

Drone i keeps a variable κ, which denotes the last DR used for encryptions. In what follows, DR' denotes a DR with encrypted data. For example, if κ is 3, the drone has DR'^i_1, DR'^i_2, DR'^i_3, DR^i_4, ..., DR^i_K

4.6.5 Code Attestation

In this section, we present two code attestation protocols: a protocol for a single drone and a protocol for multiple drones.

Single drone attestation

We first explain how the ground station verifies the integrity of a single drone, say i. The ground station GS initiates the protocol as follows:

- **Step 1:** GS sends an initiation message to drone i.
- **Step 2:** i sends κ, DR'^i_1, DR'^i_2, ..., DR'^i_κ to GS.
- **Step 3:** GS checks if DR'^i_1, DR'^i_2, ..., DR'^i_κ contain any malicious instructions (The detection methods are discussed in Sec. 4.7.2). If so, GS concludes that i is compromised and aborts the protocol.
- **Step 4:** GS selects a random number R and sends $C(= Enc(R))$ to i.
- **Step 5:** i obtains $R(= Dec(C))$.
- **Step 6:** i takes R as an initial input for the checksum function and computes the checksum h of its entire program memory except DR'^i_1, DR'^i_2, ..., DR'^i_κ.
- **Step 7:** i sends $C' = Enc(h)$ to GS.
- **Step 8:** GS decrypts C' and checks whether $h = h'$, where h' is the checksum computed by GS. Since GS knows all the contents of the program memory of

i, GS can also calculate h'. If $h \neq h'$, the ground station concludes that i is compromised. Otherwise, GS concludes that i is genuine.

The checksum algorithm to obtain h is same as the algorithm in [17] except that our algorithm sequentially traverses the memory. The details of the algorithm will be presented in the final version of the chapter due to the page limit.

Multi-drone attestation

If the ground station is required to verify the software integrity of n drones using the single drone attestation, the ground station must execute the step 8 n times and thus this approach is not scalable. In this section, we describe how multiple drones cooperatively perform attestation and off-load the overhead from the ground station.

- **Step 1:** The ground station GS creates an attestation group in which all drones cooperatively run the protocol and can directly communicate with each other.
- **Step 2:** GS sends an initiation message to the n drones in the group.
- **Step 3:** Each drone i ($1 \leq i \leq n$) sends κ, DR'^{i}_{1}, DR'^{i}_{2}, ..., DR'^{i}_{κ} to GS.
- **Step 4:** GS checks if DR'^{i}_{1}, DR'^{i}_{2}, ..., DR'^{i}_{κ} contain any malicious instructions. If so, GS concludes that i is compromised, starts to create another attestation group except i and goes back to the step 2.
- **Step 5:** GS assigns new IDs to the group members. The IDs are temporarily used during attestation.
- **Step 6:** As shown in Fig. 4.7, GS generates attestation sequences for OS/DSP and DRs (except the DRs with collected data, i.e., DR') in the ascending order of the drones' ID, and announces them to all the group members.
- **Step 7:** GS selects a random number R and broadcasts $C = Enc(R)$.

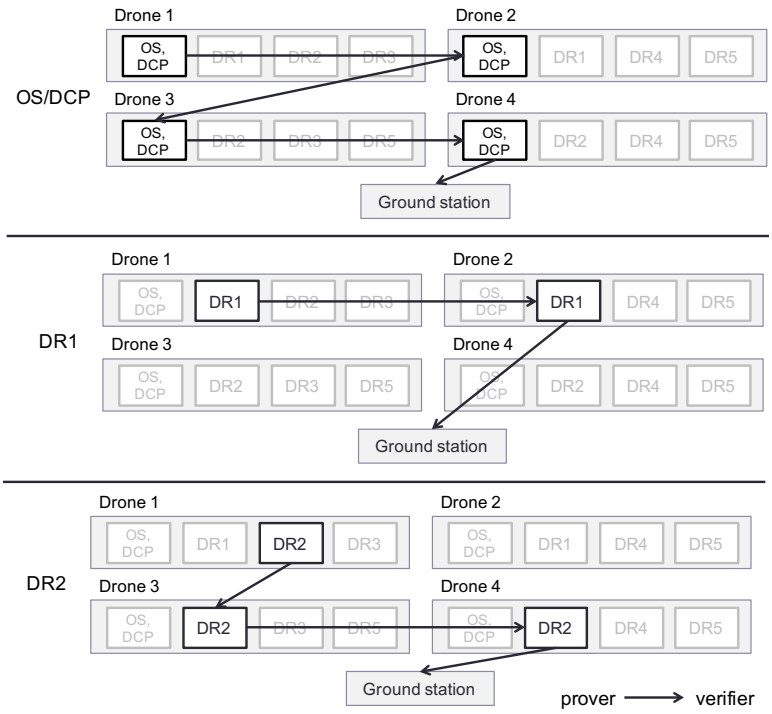


Figure 4.7.: Attestation sequences for each DR and OS/DCP when drones have no collected data

- **Step 8:** i obtains $R(= Dec(C))$.
- **Step 9:** i takes $(R + i)$ as an initial input for the checksum function and computes the checksums for OS/DCP and DR_k^i $(\kappa + 1 \leq k \leq K)$.
- **Step 10:** i sends each checksum to each *drone verifier* v. Drone verifiers are determined by the ascending attestation sequences. For instance, in Fig. 4.7, drone 2 is the drone verifier of drone 1 with respect to OS/DCP and $DR1$ since drone 1 sends $h_{OS/DCP}$ and h_{DR1} to drone 2. Likewise, drone 3 is the drone verifier of drone 1 with respect to $DR2$ and $DR3$ since drone 1 sends h_{DR2} and h_{DR3} to drone 3. The last drone, i.e., drone 4, sends $h_{OS/DCP}$, h_{DR2}, h_{DR4} and h_{DR5} to the ground station.

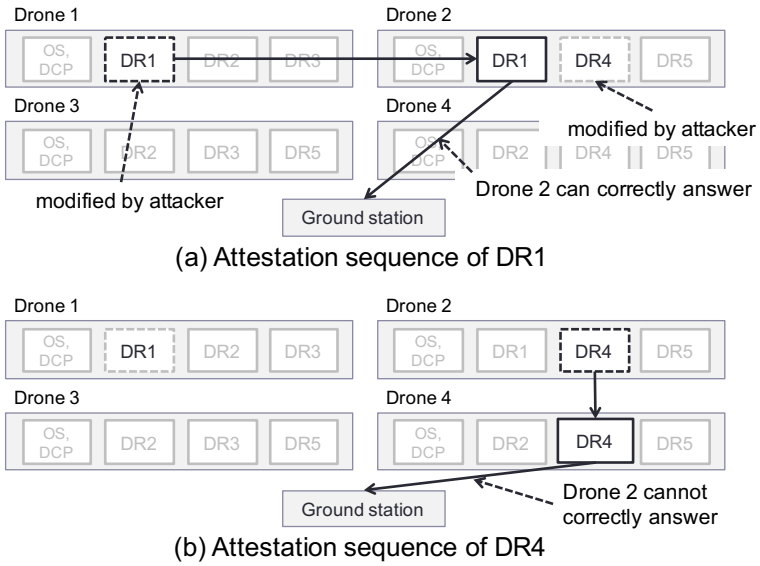


Figure 4.8.: Attestation sequences of $DR1$ and $DR4$.

- **Step 11:** v computes h'_{DRx} and compares it with the checksum h_{DRx} received from i. If they are not equal, v sends $failure(i, k)$ to the ground station. Otherwise, v sends a message $success(i, k)$.

- **(Optional) Step 12:** If GS receives $failure(i, k)$ messages, it performs the single mode attestation for i.

Security analysis: We now show how this protocol guarantees that malicious modifications of the program memory in group members are detected. Security against generic attacks on attestation is analyzed in Sec. 4.7.

Theorem 2 *The* multi-drone attestation protocol *detects any modifications to the program memory in the group members.*

Proof We provide the sketch of the proof with an example. ■

For example, as shown in Fig. 4.8 (a), suppose that the attacker captures drone 1 and modifies $DR1$. Since drone 2 is the verifier of drone 1 with respect to $DR1$, the attacker must also tamper with the program memory of drone 2 so that drone

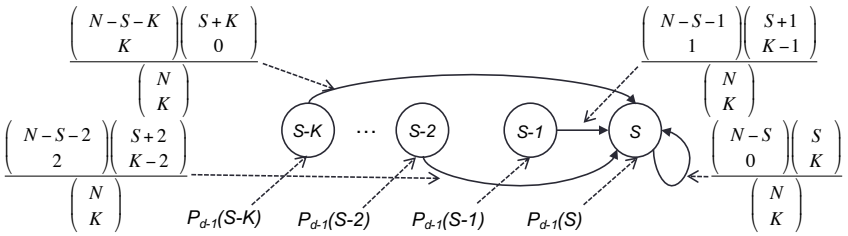


Figure 4.9.: Markov chain

2 can deceive the ground station. Assume that the attacker modifies $DR4$ in drone 2. Although drone 2 can correctly answer to the challenge with respect to $DR1$, the modified $DR4$ must be verified by the drone 4 (see Fig. 4.8 (b)). Since any modifications of drone 4 are detected by the ground station, the ground station can conclude that some drones including drone 4 are compromised.

Performance analysis: The multi-drone attestation protocol imposes much less computational overhead on the ground station. Let Ω denote the overhead when the ground station performs the checksum function for one DR. If the ground station runs the single drone attestation protocol for n drones, the computational overhead is $n \times K\ \Omega$. Since the overhead is proportional to n, the single drone attestation protocol is not scalable. However, the multi-drone attestation protocol imposes a cost of at most $N\ \Omega$ on the ground station since the ground station just needs to calculate checksums for each DR like in the example in Fig. 4.7 (K is 3 and N is 5.). We omit OS/DSP for the sake of simple analysis. The expected number of distinct DRs when n drones join the attestation group can be calculated using a Markov chain. As shown in Fig. 4.9, since a drone has K distinct DRs, for every join, there are $K+1$ possible transitions to reach the state $\mathcal{S}$, where $\mathcal{S}$ means the state when the group has $\mathcal{S}$ distinct DRs. Given N and K, the probability that the group has $\mathcal{S}$ ($\leq N$) distinct DRs after the d-th drone joins is $P_d(\mathcal{S}) = \frac{\sum_{t=0}^{K} \binom{N-\mathcal{S}-t}{t}\binom{\mathcal{S}+t}{k-t} P_{d-1}(\mathcal{S}-t)}{\binom{N}{K}}$. The expected number $\mathcal{D}$ of distinct DRs after the d-th drone joins is $E[\mathcal{D}] = \sum_{t=0}^{N} t P_d(t)$.

Fig. 4.10 shows the analytical results on the computational overhead when the single drone attestation protocol is used and when the multi-drone attestation proto-

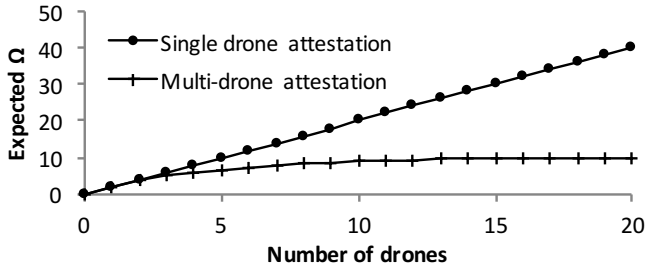


Figure 4.10.: Single drone attestation vs. multi-drone attestation (N=10, K=2)

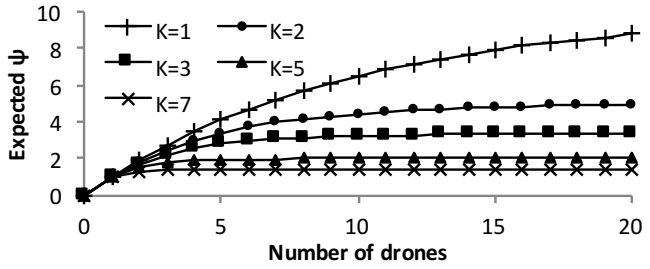


Figure 4.11.: Overhead of multi-drone attestation with different Ks (N=10)

col is used (N=10, K=2). When the number of group members is 1, both protocols impose a cost of 2 Ω on the ground station since K is 2. However, when the number of members is 2, the multi-drone attestation protocol imposes a cost of 3.6 Ω, while the single drone attestation imposes a cost of 4 Ω. If the number of members is larger than 11, the overhead of the multi-drone attestation slowly increases, while the overhead of the single drone attestation linearly increases. These results confirm that the multi-drone attestation protocol is scalable since its maximum overhead is N Ω.

Performance vs. security: We now analyze the tradeoff between performance and security in terms of K in the multi-drone attestation. Given N and K, if K is close to N, it is highly likely that two drones share the same DRs. However, if K is 1, that is, each drone has only 1 DR, the probability that two drones have a same DR is small, i.e., $1/N$.

With respect to security, as K becomes smaller, the probability that an attacker can decrypt collected data in DR's decreases. Imagine that the attacker captures two

drones and wants to decrypt the collected data in the first drone using the DRs in the second drone. If the collected data are encrypted in the $DR1'$ of the first drone and the second drone has the $DR1$, the attacker can obtain the collected data in the first drone by executing $DR1' \oplus DR1$. In other words, the success probability that the attacker can decrypt the collected data in one DR in the first drone using the second drone is K/N.

However, with respect to performance, as K becomes smaller, the computational overhead on the ground station increases since the size of each DR becomes larger. Notice that the size of each DR when $K = 1$ is α times larger than the size of each DR when $K = \alpha(\leq N)$. Fig. 4.11 shows the computational overhead of the ground station when the multi-drone attestation protocol with different Ks is used. In the figure, 1 ψ denotes the overhead when the number of drones is 1. When K is 1, as the number of drones increases, the overhead increases more rapidly than when K is larger than 1. In real applications, the tradeoff must be taken into account.

4.7 Security Analysis

In this section, we analyze the security of our protocols against various attacks specific to our protocols.

4.7.1 Generic Attacks on Code Attestation

Past work has shown various attacks on software-attestation techniques. Castelluccia et al. [21] introduced a *rootkit-based attack* using return-oriented programming (ROP). They demonstrated that ROP can be used to hide malware in an embedded system, and prevent its detection by the attestation procedure. They also introduced a *compression attack* by which an attacker can compress the original program in program memory and obtain enough free space to store and run a malicious program. Both attacks succeed in hiding malware by utilizing free space in data memory. In the *rootkit-based attack*, ROP programs must be copied to the data memory. The

compression attack requires a memory space in the data memory to run a compression routine. For this reason, both attacks cannot be launched when our attestation technique is used since the data memory of the drone is filled up with look-up tables. If the attacker terminates the drone's main program, i.e., DCP, to obtain a free space in the data memory, he/she will destroy the look-up tables. Therefore, the attacker will not be able to encrypt/decrypt messages exchanged with the ground station and, thus, the ground station can easily detect that the drone is compromised.

Yang et al. [19] introduced a *memory collusion attack*. All drones share the same main program. Therefore, n colluding drones can divide the original main program by n and each stores only $1/n$ of the program. The saved storage is used for malware. During attestation, they cooperatively compute the checksum on the main program by communicating with each other. Although we assume that it is hard for an attacker to perform memory collusion attacks since the drones are in the communication range of the ground station during attestation, the following properties of the multi-drone attestation protocol also mitigate the success likelihood of such attacks. First, in the multi-drone attestation protocol, all the drones in a group can directly communicate with each other. Second, the temporary IDs for the drones are randomly assigned and thus attestation sequences are random. Third, the direction of messages during the attestation is one-way in the ascending order of the drones' ID. Therefore, any suspicious communications between colluding drones can be detected by other benign drones or the ground station. For example, assume that drone 1 and drone 2 are compromised and collude, but drone 3 is benign. Since they are all within their communication ranges and messages for attestation are one-way (e.g., drone 1 $\rightarrow$ drone 2), drone 2 cannot send a message to drone 1 without being detected by drone 3.

Other attacks aiming at subverting attestation techniques, such as *memory shadowing attack* [21], *memory copy attack* [116], *proxy attack* [15] and *optimized implementation attack* [15] and *multiple colluding attack* [15], are not applicable to our

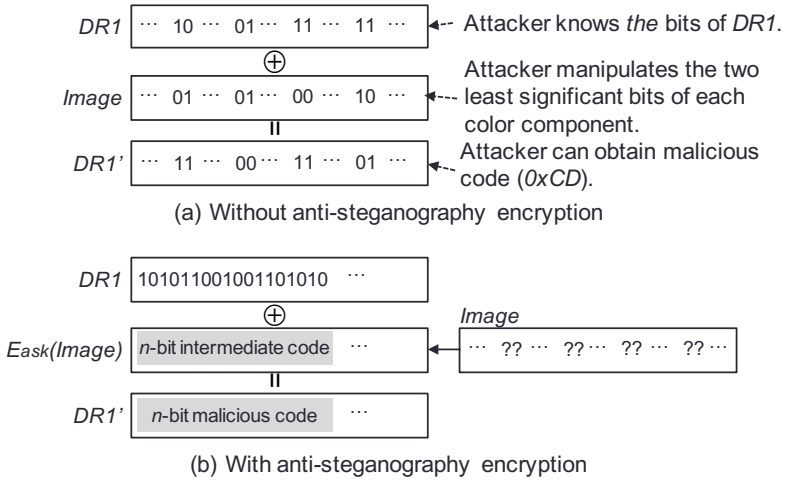


Figure 4.12.: Steganographic attacks and prevention

attestation technique since our attestation technique fills up free memory spaces with random numbers and does not rely on strict time measurements.

4.7.2 Preventing Malware in Data Repositories

In our attestation protocol, the drone sends DR'_1, DR'_2, ..., DR'_κ, which are the ciphertext of collected data, to the ground station. Then, the ground station decrypts DR'_1, DR'_2, ..., DR'_κ and obtains the collected data. If an attacker simply overwrites DR'_1 $(1 \leq i \leq \kappa)$ with his/her malicious code, the ground station can easily detect that the drone is compromised since the decrypted data of the malicious code are meaningless random bits. Without the use of $E_{ask}(P)$ during encryption described in Sec. 4.6.4, the attacker can insert his/her malicious code into an image using steganographic techniques [124]. For example, as shown in Fig. 4.12 (a), suppose that the attacker knows all the bits in DR_1 and obtains an image. Then, the attacker can generate an instruction (0xCD) by modifying the two least significant bits of each color component in the image without being detected. Likewise, the attacker may be able to generate malicious code of any size in DR's. Moreover, the attacker can insert any data like the look-up tables into an image, and thus can free space in the

data memory. The ground station may detect malware in the image using steganalytic techniques [125–127]. However, their false positive rate fp is high and highly depends on the sensitivity value sv [128] (e.g., fp=8% when sv=1, and fp=17% when sv=1.6). Also, in our multi-drone attestation protocol, each drone must run the steganalytic techniques since it must serve as a drone verifier.

However, if the drone encrypts an image using $E_{ask}(P)$, as shown in Fig. 4.12 (b), it is hard for the attacker to generate malicious code. Assume that the attacker can manipulate an image and wants obtain n-bit malicious code. To generate the n-bit malicious code, the attacker must first generate the n-bit intermediate code and XOR it with DR_1. Due to the diffusion property of AES-CFB-LB, the only way for the attacker to obtain the n-bit intermediate code is to exhaustively search it by changing the least significant bit of each color component in the image. For each try, the probability that the attacker can obtain the n-bit intermediate code is $1/2^n$. If the attacker can manipulate m-bits in the image, the success probability P_s to obtain the n-bit intermediate code is 1-$(1-\frac{1}{2^n})^{2^m}$ (e.g., P_s=0.0039 when n=40, m=32). Considering that m cannot be large due to the large search space, while the size of malware (n) is much larger than m, it is hard for the attacker to subvert our attestation protocol by utilizing steganographic techniques.

4.7.3 Protection of Collected Data

The main goal of our data protection scheme is to protect data collected before the drone is compromised. The drone uses DRs like keys in the one-time pad in order to encrypt data. Therefore, an attacker cannot decrypt the data even if he/she can obtain any information in the drone's memory including ask. However, in our protocol, ask is periodically updated and also used to encrypted data. Thus, even if the attacker had known all the bits of DRs before the data were collected, he/she cannot decrypt the collected data unless the attacker knows ask that was used to

encrypt the data. It is hard for the attacker to keep track of *ask* since malware in the drone can be detected by our attestation protocol.

4.8 Experimental Results

4.8.1 Implementation

We utilized the AR.Drone 2.0 [83] and the Raspberry Pi2 [129] as mobile data collectors. Note that our target platform is a "drone" which has a much simpler kernel than general-purpose linux devices like Raspberry Pi. The operating system of AR.Drone is Linux 2.6.32. AR.Drone has 115.4 KB RAM and a 1 GHz 32-bit ARM Cortex-A8. The size of program memory is 200 MB and the free space of the program memory is 180 MB. The operating system of Raspberry Pi is Raspbian 4.1.7. Its data memory is a 925.9 MB RAM and its CPU is a 900 MHz 32-bit quad-core ARM Cortex-A7. Its program memory is 1.9 GB and the free space of the program memory is about 1.85 GB. Both platforms have a Wi-Fi b/g/n interface.

Since the program memory is filled up with DRs, virtual memory is disabled in both platforms. After the main program starts, we use the $free$ command to check the available space in the data memory. Then, we call the *malloc* function to allocate memory spaces for look-up tables until the *malloc* function returns $fail$. To scan the program memory, the checksum function reads the disk image files.

We have also implemented the ground station using a Linux PC running 64-bit GNU Linux kernel version 3.5.0 with 1.8 GHz Intel Core i5 and 4 GB memory. It has a Wi-Fi b/g/n/ac interface. We utilized OpenSSL as a crypto-library.

4.8.2 Setup

DR generation: The ground station generates pseudo-random numbers for DRs and sends them to the drones. We measured the time required to generate and send DRs to the drones. We found that the speed of the pseudo-random number generation

was higher in one order of magnitude than the transmission speed. For instance, the ground station can generate random numbers at the speed of 141 MB/s. However, the measured transmission speed through the Wi-Fi interface is only 6.21 MB/s. Transferring the total DRs to AR.Drone and Raspberry Pi requires 29.3 seconds and 617.2 seconds, respectively.

Look-up table generation: The look-up tables are generated by the ground station and then transferred to drones. The free data memory space of AR.Drone2.0 is 18.4 KB and the free data memory space of Raspberry Pi 2 is 351 KB. For AR.Drone2.0, MF-SPACE first generates 222 (=4+218) AES keys. Then, it generates 4 L_8 tables using the first 4 AES keys. Then, the total size of the look-up tables and the remaining AES keys is 18.4 KB (= 4×3,840 B + 218×16 B). For Raspberry Pi 2, MF-SPACE generates 263 (=93+170) AES keys. Then, it generates 93 L_8 tables using the first 93 AES keys. Then, the total size of the look-up tables and the remaining AES keys is 351 KB (= 93×3,840 B+170×16 B).

We utilized AES-128 as the F function in MF-SPACE. The ground station took 0.12 ms and 2.66 ms to generate the tables for AR.Drone2.0 and Raspberry Pi 2, respectively.

4.8.3 Encryption/Decryption Performance

MF-SPACE: We have measured the en-/decryption performance on AR.Drone 2.0 and Raspberry Pi 2 when they use MF-SPACE. The en-/decryption speed of AR.Drone2.0 and Raspberry Pi 2 is 59.73 KB/s and 53.76 KB/s. Considering that the MF-SPACE cipher is used when drones encrypt telemetry data or decrypt control messages from the ground station, we believe that the en-/decryption speed is enough for the drones.

Data Repositories: If a drone collects data p, p must be encrypted using AES-CFB-LB and the output is XORed with DRs. The encryption speed on AR.Drone2.0 and Raspberry Pi 2 is 67.2 MB/s and 59.2 MB/s, respectively. After receiving the encrypted data from drones, the ground station decrypts the data. The decryption

speed on the ground station is 352 MB/s. The drone might be required to record high definition (HD) or 4K videos and encrypt them. Considering that the bitrate of HD 1080p ranges between 8 MB/s and 12 MB/s and the bitrate of 4K Digital Cinema is about 33 MB/s, the speed of encryption/decryption using DRs far exceeds the requirement.

4.8.4 Attestation

Single drone attestation protocol: We measured the time required to calculate the checksum function in the single drone attestation protocol. Since the checksum function scans the entire program memory, Raspberry Pi takes much longer time than AR.Drone. For calculating a checksum result, AR.Drone takes 35.31 seconds, while the Raspberry Pi takes 362.9 seconds. However, considering that the attestations are not executed frequently, we believe that the overhead of the checksum execution is not high for drones.

Multi-drone attestation protocol: To assess the performance of the multi-drone attestation protocol (mp) when the number of drones is large, we created virtual drones which emulate the AR.Drone2.0. We set N to 10 and K to 2. We measured the time required for the ground station to compute checksums. Fig. 4.13 shows the execution time of the ground station when the ground station runs the single drone attestation protocol (sp) and when the ground station runs mp. When the ground station runs mp, the computation time does not exceed 8.1 seconds, while the computational overhead of the ground station running sp linearly increases with the number of drones to be attested. Notice that, in mp, each drone, which is typically slower than the ground station, must be a verifier. In order to minimize the overall execution time, we must choose between the two attestation protocols.

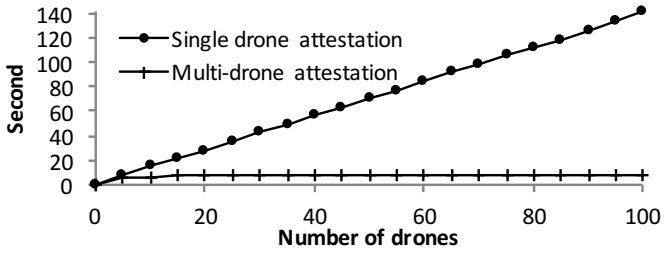


Figure 4.13.: Measured time in the ground station to calculate checksums

4.9 Summary

We propose a security solution for mobile data collectors. The solution integrates techniques for software-based attestation, data encryption and secret key protection. We introduced the notion of data repositories, which achieve two security objectives: 1) software attestation and 2) encryption of collected data. Second, we propose MF-SPACE that generates look-up tables that achieve three security objectives: 1) encryption of communications, 2) secret key leakage prevention and 3) prevention misuse of free data memory by attackers.

5 A SECURE SHUFFLING MECHANISM FOR WHITE-BOX ATTACK-RESISTANT DRONES

5.1 Introduction

Unmanned Aerial Vehicles (UAVs), also known as drones, are vehicles that are remotely controlled or autonomously navigate using sensors and path planning algorithms. Recent advances in control software technologies, along with advances in embedded systems and sensor technologies, have led to a burgeoning drone industry. Advances in systems, tools, and techniques for drones have increased the spectrum of drone applications. For instance, vision positioning and geo-fencing increase the physical safety of drone usage. Path planning and target-tracking (i.e., follow-me) algorithms increase usability. Furthermore, modern drones are now being equipped with GPUs to support advanced software capabilities such as machine learning [130], 3D modeling from 2D images [131] and autonomous navigation [132].

With the help of such technologies, drones have deeply penetrated into our daily life. They are used for a variety of purposes. They have been utilized to patrol national borders [133]. Wineries are using drones to produce better wine [134]. Amazon [135] has demonstrated that their drones can deliver packages to customers within 30 minutes.

In many applications, drones often send data, which is often privacy-sensitive or critical for decision making, to data users. For example, photos taken by agricultural drones can be used for crop health monitoring. Farmers can decide to spray pesticides on specific areas based on the results of the visual analysis. Inspection drones can be used to take pictures to check for structural cracks after an earthquake. In some envisioned applications, such as search and rescue, a group of drones may communicate with each other to cooperatively find casualties in a disaster area.

However, such widespread usage of drones implies that they can become the target of malicious attacks. Like many networked computing devices, drones can be victims of traditional attacks, referred to as black-box attacks in what follows, such as eavesdropping, manipulation, replay attacks and man-in-the-middle attacks. Furthermore, attackers can launch stronger attacks, called white-box attacks. Indeed, researchers have already discovered vulnerabilities in consumer drones and demonstrated how to hijack them [25–29]. For instance, Maldrone [26] is the first drone malware that can be installed on drones while they are flying and allows attackers to take control of the drones. In addition, the attackers may be able to launch firmware modification attacks [30] by utilizing reverse engineering tools [31, 32] since most drones available on the market support firmware upgrades. Moreover, since many drones are based on open-sourced software [33–35], the attackers may be able to exploit known vulnerabilities in such software. Once the attackers succeed in installing malware on the drones, their computing environments become untrusted. As a result, the attackers can steal secret keys from the drones and deceive data users into making incorrect decisions.

The concept of white-box cryptography (WBC) [36] was introduced in 2002 to protect software implementations of cryptographic algorithms in untrusted environments that are not equipped with hardware-assisted security mechanisms, such as the Trusted Platform Module [14] and the ARM TrustZone [37]. By untrusted environment, we refer to an environment in which the attacker has complete control of the device. Although WBC is originally intended for digital rights management (DRM), its applications are expanding to mobile devices and IoT devices [38, 39]. Recently, WBC has attracted attention from industry [40] since it does not require specialized hardware. In fact, even a small increase in per device cost leads to a significant increase in overall production costs of high-volume drone/robot manufacturing. In addition, white-box cryptography can be utilized on legacy systems, and can be upgraded by software updates.

In the context of the white-box attack model, i.e., in untrusted execution environments, traditional symmetric key-based cryptographic primitives with short secret keys, such as AES and HMAC, do not guarantee confidentiality and integrity. A white-box attacker can see and manipulate the internal state of the memory of a victim drone by obtaining a root privilege and installing malware. More seriously, the malware may be able to locate a short secret key, such as an 128-bit AES key or a private key for public-key cryptography, and send it to the remote attacker.

To protect the confidentiality of secret keys in such a white-box environment, several white-box cryptography solutions [36, 41–44] have been proposed. Such solutions hide a short key by converting it into one or more large look-up tables in order to make it hard for an attacker to extract the short secret key from the look-up table(s).

Although the existing white-box cryptography solutions provide a certain level of security against white-box attacks[1], none of them provide a method to securely change the look-up table after it is initialized. Therefore, once a white-box attacker succeeds in extracting a part of the look-up table from a drone, the attacker is able to permanently use the extracted partial table to decrypt/encrypt ciphertexts/plaintexts until the user is able to change the look-up table. Also, the attacker can decrypt all the past communications since the table is static. However, changing the look-up table in the white-box environments is not easy since the attacker can see the internal memory state by launching white-box attacks, while the look-up table is being changed.

In this chapter, we propose a look-up table shuffling mechanism that provides white-box cryptography with dynamics. Only legitimate users who share a look-up table can shuffle their look-up tables correctly, and thus the look-up tables of the users are synchronized. If an attacker does not know the entire look-up table, the shuffling mechanism makes it hard for the attacker to determine the positions of entries in the look-up table, and thus makes it infeasible to decrypt (or encrypt) ciphertexts (or plaintexts).

[1]In fact, most existing solutions except [43, 44] fail to achieve a practical level of white-box security although they provide a competitive level of black-box security. In the white-box environments, an attacker can extract a short secret key in executable work-steps [45–49].

Another challenge is to make the encryption/decryption operations of white-box cryptography fast enough to satisfy the real-time requirements of drone applications. In many drone applications, a control station (or a ground station) sends control commands related to physical safety. Such messages should be processed quickly since drones need to have enough time to avoid the accidents. Sometimes, drones need to send a large amount of collected data to the control station and the encryption of the data must not be a bottleneck. Recent architectures proposed for the Internet of Drones (IoD) [136] clearly show the high volume of sensitive real-time communications involved in the management of drone airspaces. However, white-box block ciphers are usually much slower than the traditional block ciphers since white-box block ciphers require hundreds of table look-ups and they replace many computationally efficient operations, such as XORs, with table look-ups. To address this requirement, we focus on the fact that many recent drones [130, 131] are equipped with Graphics Processing Unit (GPU)-enabled system-on-chip (SoC) and thus we can exploit the GPU to execute white-box cryptography algorithms in parallel. However, the GPU-utilization incurs the cost of moving the data from the main memory to the GPU memory and vice versa. In this chapter, we perform experiments about the use of CPU and GPU, and report results about the encryption/decryption performance according to block sizes when the CPU or the GPU is utilized. Based on the experiments, we derive guidelines about the GPU-utilization to maximize the performance of the encryption/decryption algorithms. The contributions of the chapter are summarized as follows:

- We propose a look-up table shuffling mechanism for dynamic white-box cryptography in the context of drone applications and provide its security analysis. Due to the dynamics, even if a remote attacker is able to see any part of the drone's memory through malware, it is hard for the attacker to determine the positions of the table entries, and thus to decrypt/encrypt ciphertexts/plaintexts. To the best of our knowledge, our proposal is the first to address the problem of making a white-box block cipher dynamic within the white-box environment. Also, we are

the first to use the dynamic white-box block cipher for secure communications for drones.

- We show the practical applicability of the white-box block cipher with our shuffling mechanism by implementing it on Nvidia Tegra K1, which is a GPU-enabled SoC used in several modern drones. We show that the encryption/decryption performance is significantly boosted by GPU-acceleration when the block size of a message is large. Our shuffling mechanism does not require additional memory space in drones and is executed by drones within a reasonable time.

Note that although our target devices are drones, our defense scheme can be utilized for other types of network-enabled mobile devices which are periodically located in secure locations, but may pass through insecure areas for limited time periods.

5.2 Related Work

5.2.1 White-box Cryptography

Chow et al. first introduced the concept of white-box cryptography in 2002 and presented two implementations of white-box block ciphers based on DES [41] and AES [36]. Their white-box implementations transform a traditional block cipher into a series of key-embedding table look-ups (see Fig.5.1). The secret key is embedded into the look-up tables by scrambling the key with random values. To protect the first and last round tables, external encodings are applied. However, as pointed out by Bogdanov and Isobe [43], the external encodings require a trusted execution environment in the device, which restricts the applicability of white-box ciphers. Since Billet et al. presented an algebraic cryptanalysis technique [45] that breaks the Chow et al.'s implementations in 2^{30} work-steps, several variants [117–119, 137] of the white-box AES/DES implementation were proposed. However, they all have been cryptanalyzed and broken in practical work-steps [46–49, 120].

Recently, instead of implementing a white-box cipher based on existing block ciphers such as AES/DES, block ciphers dedicated to white-box cryptography have been

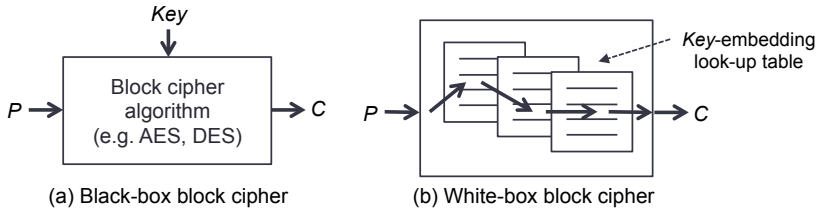


Figure 5.1.: Comparison of block ciphers

introduced [42, 43]. Biryukov at el. proposed the first dedicated block cipher based on the ASASA Structure (A:affline, S:non-linear) without external encodings [42]. Unfortunately, it has also been cryptanalyzed [138, 139].

Bogdanov and Isobe [43] introduced a construction of a dedicated white-box block cipher, called the SPACE cipher. The SPACE cipher utilizes a traditional block cipher, such as AES or Triple-DES, to generate a look-up table. Its security relies on the black-box security of the block cipher. In other words, as long as the block cipher used for the look-up table generation is secure in the black-box model, a white-box attacker cannot recover the secret key used to generate the look-up table even if he/she can see the entire look-up table. Therefore, any key recovery attacks like the Differential Computational Analysis attack [140] are not applicable to the SPACE cipher. Another property of the SPACE cipher is that it does not require any external operations, such as external encodings [36]. In addition, the construction of the SPACE cipher is variable according to the desired space hardness [43] and the resource availability of the device. The details of the SPACE cipher are described in Sec. 5.3.3. Recently, Bogdanov et al. proposed another dedicated white-box block cipher, called the SPNbox cipher [44], based on a substitution-permutation network and showed its performance on an Intel Skylake CPU and an ARMv8 CPU. They showed that the SPNbox cipher can be faster than the SPACE cipher since the SPNbox cipher provides parallelization opportunities. However, they do not show the performance on a GPU. Although the SPACE and SPNbox cipher make it hard for an attacker to

extract a short secret key from the look-up table, their security level decreases as the amount of the look-up table that has been leaked increases since the table is static.

All white-box block ciphers that have introduced so far only focus on how to protect the system from the key extraction attack and the code lifting attack. None of them provide any method to securely change the code (or look-up table(s)) after it is initialized. Therefore, once an attacker succeeds in extracting a part of the code, he/she can permanently take advantage of the extracted partial code. Also, the attacker can use the extracted code to decrypt the communications recorded in the past since the code is static (no forward secrecy[2] is supported.). It is however crucial to address this problem especially in drone applications since drones are vulnerable to white-box attacks and messages they exchanged have pressing safety and privacy requirements.

The shuffling mechanism proposed in this chapter is the first to address the problem of making a white-box block cipher dynamic. Also, we are the first to show its feasibility in a real embedded device.

5.2.2 GPU Utilization for Cryptography

Over the years, many cryptographic algorithms have been implemented and benchmarked using GPUs [141, 142]. Singla et al. [143] have been the first to implement a fast authentication scheme using a GPU for vehicular networks. They have shown that the performance of cryptographic algorithms can increase by executing parallelizable tasks on the GPU in parallel. Although we utilize the same basic ideas about GPU-utilization, to the best of our knowledge, no previous work investigated the use of GPU for white-box cryptography.

[2] *Forward secrecy* protects past communications against *future* compromises of secret keys or passwords.

5.3 Background

5.3.1 White-box Attacks

In the traditional black-box attack model, the assumption is that the attacker can access only inputs and outputs of a cryptographic algorithm. Through the known (or chosen) plaintext (or ciphertext) or adaptively-chosen plaintext/ciphertext, the attacker aims at recovering the secret key or distinguishing ciphertext from random data.

On the other hand, in the white-box attack model, the end devices are not trusted. The assumption is that attackers have full control over the execution environment of a target device, meaning that they are able to examine inputs, outputs, and intermediate values of the crypto-algorithm executions. Also, the attacker has a detailed knowledge of the cryptographic algorithms in the system and is able to modify them. If a system is implemented under the black-box attack model, a secret key or key-related information in the device memory can be located and extracted.

The primary goal of the white-box attacks is to extract the cryptographic key in the system. The attacker may be able to utilize a simple debugger or dynamic binary instrumentation tools, such as Pin [144] and Valgrind [145], to directly observe the cryptographic keying material at the time of use. Kerins and Kursawe have introduced another type of white-box attack, called *S-box blanking attack* [146]. Commonly used symmetric key algorithms, like AES, utilize lookup tables, such as S-boxes, in order to achieve confusion. Instead of finding cryptographic keys directly in a binary, they utilize the fact that the location of a S-box is easily identified since the values of the S-box are known. If the attacker is allowed to change all the values of a S-box with zeros, the final round of the execution reveals the final round key.

5.3.2 Design Goal of White-box Block Cipher

A white-box block cipher has a three-fold design goal.

- **Protection against key extraction:** Given the implementation of a white-box block cipher and its internal status, it must be computationally hard for the attacker to extract the short secret key embedded in the white-box block cipher. In other words, the attacker cannot obtain the short secret key that was used to generate the look-up table by analyzing the look-up table.
- **Protection against code lifting:** A white-box attacker may attempt to extract (lift) the whole implementation code of a white-box block cipher or its look-up tables instead of extracting the short secret key. If the attacker succeeds in *code lifting*, he/she can decrypt/encrypt any ciphertext/plaintext using the code itself as a large key. Code lifting cannot be prevented if the attacker has full control over the execution environment and sees all internal values. Thus, several new notions for code lifting security have been introduced, such as *incompressibility*, *weak white-box security* [42], and *space hardness* [43]. If a white-box block cipher is *incompressible* or *hard* in terms of memory *space*, it is computationally hard to construct a functionally equivalent implementation with a memory space that is smaller than the whole memory space required for the original white-box cipher implementation.

  As mentioned in [43], a security-critical system may have a limited communication channel bandwidth to the Internet. In such a case, the incompressibility makes it hard for malware, such as Trojans, to transmit the lifted code or the look-up tables outside of the system without being detected since their size is very large.
- **Performance:** Like for traditional block ciphers, high performance of encryption/decryption operations of white-box block ciphers is critical for many applications.

5.3.3 Details of the SPACE Cipher

The SPACE cipher [43] is an l-line target-heavy generalized Feistel network (see Fig. 5.2). It encrypts an n-bit plaintext using a k-bit secret key and outputs an n-bit

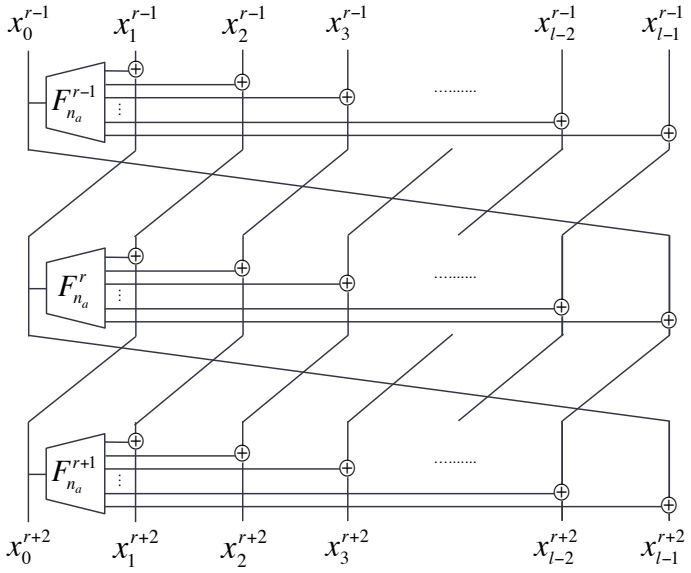


Figure 5.2.: The Feistel structure of the SPACE cipher

ciphertext. $X^r = \{x_0^r, x_1^r, ..., x_{l-1}^r\}$ denotes the n-bit state of round r and the size of each line, i.e., x_i^r, is $n_a(= n/l)$-bit. If R is the number of total rounds, X^0 is the plaintext and X^R is the ciphertext. At each round the state is updated as follows:

$$X^{r+1} = \left(F_{n_a}^r(x_0^r) \oplus (x_1^r \parallel x_2^r \parallel ...x_{l-1}^r)\right) \parallel x_0^r,$$

where $\parallel$ denotes the concatenation and $F_{n_a}^r(x)$ is a function whose input size is n_a-bit and output size is $n_b(= n - n_a)$-bit, i.e., $F_{n_a}^r(x) : \{0,1\}^{n_a} \rightarrow \{0,1\}^{n_b}$. $F_{n_a}^r(x)$ is defined as

$$F_{n_a}^r(x) = \hat{F}_{n_a}^r(x) \oplus \alpha = \Big(msb_{n_b}\big(E_K(C_0 \parallel x)\big)\Big) \oplus \alpha,$$

where E_K is a block cipher with an n-bit block and a k-bit key K, and $msb_{n_b}(x)$ is a function that returns the most significant n_b-bits of x. C_0 is n_b-bit binary zero value and α is a round constant. E_K can be any block cipher such as AES-128.

In the white-box environment, $\hat{F}_{n_a}^r(x)$ is implemented by table look-ups. Since the size of a look-up table depends on n_a, the SPACE cipher can be implemented in different sizes. Let SPACE(n_a, R) be one of the SPACE cipher variants. When n and k are 128 bit long, Bogdanov et al. suggest four variants: SPACE(8, 300), SPACE(16,

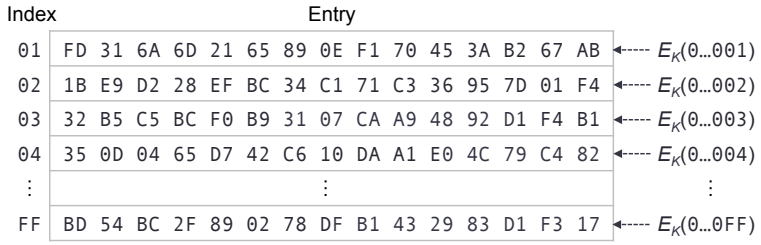


Figure 5.3.: Look-up table example of SPACE(8, *)

128), SPACE(24, 128), and SPACE(32, 128). All those variants have similar security levels, but their look-up table sizes are 3.83 KB, 918 KB, 218 MB, and 51.5 GB, respectively. The look-up table of SPACE(n_a, R) consists of 2^{n_a} entries and the size of each entry is n_b bit. For example, as shown in Fig. 5.3, in case of SPACE(8, 300), the number of entries is 256($= 2^8$) and the size of each entry is 120($= n - n_a = 128 - 8$) bits. Thus, the look-up table size is 3,840($= 256 \times 120/8$) B.

To quantitatively evaluate the difficulty of code lifting attacks, the authors introduce a security notion, called (M, Z)-*space hardness*.

Definition 4 ((M, Z)-space hardness) [43] *The implementation of a block cipher is (M, Z)-space hard if it is infeasible to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability of more than 2^{-Z} given any code (table) of size less than M.*

(M, Z)-space hardness allows one to estimate the code (or table) size M to be isolated in the white-box environment in order to decrypt/encrypt any ciphertext/plaintext with a success probability larger than 2^{-Z}. In other words, if an attacker wants to decrypt a ciphertext with a success probability larger than 2^{-Z}, he/she must extract the code of a size larger than M. If the attacker succeeds in extracting the entire code, the success probability becomes 1, i.e., $Z = 0$. For instance, SPACE(16, 128) has stronger space hardness than SPACE(8, 300) since the look-up table size of SPACE(16, 128) is much larger than the table size of SPACE(8, 300). Assume that device A runs SPACE(8, 300) and device B runs SPACE(16, 128). If

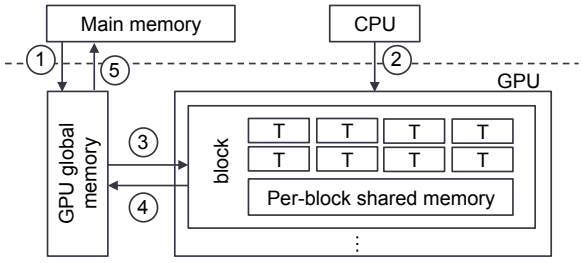


Figure 5.4.: CUDA processing flow ('T' means a 'thread'.)

an attacker wants to have a same success probability on both devices, the amount of the table-related information to be extracted from B is 239(= 918/3.84) times larger than the amount of the table-related information to be extracted from A. However, SPACE(8, 300) may be more adequate for an embedded device with a small amount of memory.

5.3.4 GPU for General Purpose Processing

GPUs have been utilized as general purpose processing units especially for parallelizable computing-intensive tasks. NVIDIA introduced the Compute Unified Device Architecture (CUDA) [147] which is a general purpose parallel computing platform and programming interface model. A programmer can define CUDA-C functions, called kernels, which are executed in parallel by CUDA threads. As shown in Fig. 5.4, the CUDA processing flow is as follows: (1) data in the main memory are copied into the GPU global memory; (2) the CPU instructs the process to execute on the GPU and then all threads in a block execute a same function; (3) during the execution, data in the global memory are read by the threads; (4-5) the computation outputs are written back into main memory through the GPU global memory.

CUDA threads may access data from multiple memory spaces. Each thread has private local memory. The global memory can be accessed by all threads even when they are in different blocks. The access time to the per-block shared memory is expected to be much smaller than the access time to the global memory. However, it

takes time for threads to copy data from the global memory to the per-block shared memory.

As discussed in Sec. 5.2.2, many cryptographic schemes have been implemented using a GPU, and mainly focus on leveraging GPU parallelism. The unique challenge of using a GPU for our scheme is to optimize the encryption performance by selecting an appropriate look-up table size and placing the look-up table into the shared memory or the global according to the size.

In drones, GPUs are mainly used for image processing acceleration [148, 149]. In fact, vision positioning, target tracking and collision avoidance have become an integral part of drone software. For example, the Parrot Kalamos [131] drones utilize a GPU to process stereoscopic images to create a 3D model of whatever their two front cameras see in real time.

In addition, drones have been increasingly equipped with GPUs to provide novel software capabilities. The Kespry drone [130] equipped with a GPU can deal with unforeseen situations using deep learning technologies. For instance, the Kespry drone is able to track targets in different work sites under different lighting/weather conditions. The GPU allows the Kespry drone to run deep learning algorithms and improve the asset tracking capability. Hossain et al. have shown that a GPU in a drone significantly enhances the path finding algorithm. The GPU helps in solving the Traveling Salesman Problem, and thus the drone can quickly decide its optimal path [132].

5.3.5 Shuffling Algorithm

There are two approaches for shuffling. The first approach is to randomly choose two elements, exchange their positions, and repeat it enough so that the positions of all elements change. Although this approach is simple, it requires $\Theta(N \log N)$ shuffles to change all the positions according to the coupon collector's problem [150]. Here, N is the number of elements.

The second approach is to store temporary data for efficiency. For instance, the Knuth shuffling algorithm [151] inserts all elements into a set and continuously determines the next element by randomly selecting an element from the set until no elements remain. Therefore, this approach requires $N/2$ shuffles. However, during shuffling, temporary data, i.e., unselected elements in the set and the current shuffling sequence (i.e., selected elements), must be stored in memory.

5.4 Attack Model and Security Goals

In this section, we first define our attack models based on the location of a drone, and then specify the design goals of our defense scheme.

5.4.1 Attacks in Secure Areas

When a drone is off duty, we assume that it is located in a secure area, such as a landing spot or a charging station, we assume that white-box attacks cannot be launched. After checking if malware exists in the drone or checking the drone's software integrity [152, 153], the drone can safely share secret values with the control station. However, black-box attacks can be launched. For example, an attacker may try to decrypt ciphertexts after eavesdropping on them, or send fake commands to the drone under the black-box attack model.

5.4.2 Attacks in Insecure Areas

After moving out of the secure area, the drone starts to receive control commands from the control station and send data (e.g., photos or videos) collected on duty to the control station.

We assume that when the drone is moving, a white-box attacker can launch white-box attacks as well as black-box attacks. The primary goal of the attacker is to steal secret information stored in the drone, such as a secret key or a look-up table, using

white-box attacks in order to decrypt/encrypt ciphertexts/plaintexts. Specifically, we assume that the attacker has knowledge about the software vulnerabilities in the drone and can install malware which can see the internal state of the drone's memory. As a result, the malware can gather the following secret information in the drone and send it to the remote attacker.

- **Short secret key:** If the drone has short secret keys (e.g., an AES key or a private key for public-key cryptography), the malware transfers them to the attacker.
- **Look-up table for a white-box block cipher:** if the drone has a large look-up table, the malware tries to transfer as much as possible of the look-up table. We assume that the white-box block cipher algorithm used for the drone is known to the attacker.
- **Shuffling-related data:** If the drone applies a shuffling mechanism to the look-up table, the malware also transfers shuffling-related information stored in the drone's memory, such as a seed value for a pseudo-random number generator and any temporary shuffling data as mentioned in Sec. 5.3.5. If the shuffling mechanism utilizes a pseudo-random number generator, like the one by Blum, Blum, and Shub [154] or a cryptographic hash function, the malware just needs to send the seed value to the remote attacker.

Once the attacker succeeds in stealing the secret information, the attacker may remove the malware from the drone to prevent it from being detected by the drone's operator later on. Using the secret information, the attacker can send fake commands to the drone, and fake data, such as manipulated photos or videos, to the control station. Also, the attacker can obtain any privacy-sensitive information by decrypting ciphertexts exchanged between the drone and the control station.

Even though the attacker can see any part of the drone's memory using the malware, we assume that the remote attacker cannot download the entire look-up table in the drone due to the following four reasons: First, the excessive use of a white-box attack tool like a debugger significantly slows down the target program, and thus has a serious impact of the real-time operations of the drone. Hence, the drone can

be regarded as compromised. Second, the uplink bandwidth that is available to the malware can be limited since the drone may be equipped with a network interface with a limited uplink bandwidth, or intentionally use up its uplink bandwidth by sending dummy data. The dummy data would have a lower priority than photos not to disturb sending the photos. Third, the operation times of drones are generally short compared to other mobile devices like smart phones because they are battery-powered and move on their own. Fourth, the drone can select the size of the look-up table after considering its memory size and its upload speed so that malware cannot upload the entire look-up table to the remote attacker given its operation time. For example, if the look-up table is 1000 KB, the drone's operation time is 30 mins and the upload speed that malware can use without being detected is 500 bytes per second, the malware cannot complete the transmission of the table while the drone is operating since sending the entire table takes approximately 33 mins.

We do not consider the case in which malware itself generate fake data (e.g., manipulated photos) and send its collected data(e.g. photos or videos) to the control station since creating realistic fake data requires computationally heavy software with human intervention like a photo editing program.

Also, we do not consider attacks that affect the availability of drones, such as physical capture attacks, denial-of-service attacks and malware making the drones unusable. Such attacks can be easily detected since the control station continuously checks the status of the drones, e.g., by pinging them and monitoring their trajectories. Therefore, the drones exhibiting abnormal behaviors can be regarded as compromised.

5.4.3 Security Goals

Our defense scheme has two security objectives depending on the type of attacks.

- **Security against black-box attacks:** Black-box attacks are weaker than white-box attacks since a black-box attacker cannot see the internal state of the drone,

and thus has no information about the secret key in the drone. However, they are more covert than white-box attacks since no malware is involved. Messages exchanged between the drone and the control station must be encrypted by an encryption scheme like a block cipher so that the black-box attacker cannot obtain any information about the encrypted messages. Also, it must be hard for the black-box attacker to create valid ciphertexts of messages (e.g., fake commands) that the attacker chooses.

- **Security against white-box attacks:** White-box attackers can see any internal state of the drone through malware. Therefore, the drone must not store any kind of short secret keys since the malware can find/send them to the remote attacker. To support encryption, the drone must have a large look-up table for white-box attack-resistant block cipher. The large size of the look-up table makes it hard for the malware to send the entire table to the remote attacker.

  However, if the look-up table is static, the attacker with some portion of the table can use it to decrypt/encrypt ciphertexts/plaintexts until the drone comes back to a secure area and generates a new look-up table. If the attacker succeeds in obtaining almost all of the table, he/she can decrypt/encrypt ciphertexts/plaintexts with a high probability as explained in Sec. 5.3.3.

  Therefore, a mechanism that shuffles the table is required. The shuffling mechanism must make it hard for the attacker to locate each entry of the table even if the attacker can perform white-box attacks and know some portion of the table. The shuffling mechanism must not store a shuffling sequence in memory. Also, it must be hard for the attacker with the knowledge of a seed to generate the whole shuffling sequence. Thus, the generation of pseudo-random numbers must be associated with the look-up table that are not easily extracted by the attacker.

5.5 Forward-secure Dynamic SPACE Cipher

In this section, we first introduce the formal descriptions of our forward-secure dynamic white-box block cipher based on SPACE. Then, we describe our shuffling mechanism that makes the white-box block cipher dynamic with a cost analysis in terms of execution time and storage space. Note that our shuffling mechanism is the first solution that provides a white-box block cipher with dynamics in the white-box attack environment.

We formalize our dynamic SPACE encryption scheme that evolves its look-up table. *The dynamic SPACE encryption scheme* D-SPACE consists of four algorithms, i.e., D-SPACE=(D-SPACE.key, D-SPACE.enc, D-SPACE.dec, D-SPACE.update) and an integer $n \geq 1$.

1) D-SPACE.key: This algorithm sets security parameters and obtains the initial look-up table (state) LT_0. This algorithm is described in Sec. 5.5.1.

2) D-SPACE.update: The operation of D-SPACE is divided into time periods $i = 0, 1, 2, ..., n$. In time period i, parties use a look-up table LT_i. A look-up table consists of the look-up table entries *Entries* and their locations Loc_i. The look-up table at period i is obtained from the look-up table at the period $i-1$ ($i \geq 1$) and a shuffling seed K_i via the deterministic update algorithm: $LT_i \leftarrow \mathsf{D\text{-}SPACE.update}(LT_{i-1}, K_i)$. After the update is completed, Loc_{i-1} and K_i are erased so that an attacker who breaks into the system cannot obtain them. Note that *Entries* are not updated. How to share a shuffling seed K_i between the parties is described in Sec. 5.5.2, and how to update the locations of the look-up table entries is described in Sec. 5.5.3.

3) D-SPACE.enc: Within the period i, the parties can encrypt a message M via $\langle C, i \rangle \leftarrow \mathsf{D\text{-}SPACE.enc}(LT_i, M)$. This algorithm is the same as the encryption algorithm of the SPACE cipher [44].

4) D-SPACE.dec: The parties can decrypt $\langle C, i \rangle$), i.e., $M \leftarrow \mathsf{D\text{-}SPACE.dec}(LT_i, C)$. This algorithm is the same as the decryption algorithm of the SPACE cipher [44].

Table 5.1. List of Notations

$\mathcal{SK}$	A secret key that is shared by the drone and the control station
$E_{\mathcal{SK}}$	A block cipher used to generate a look-up table embedding $\mathcal{SK}$
id_s	The identity of the control station
id_v	The identity of the drone
MAX_{RTT}	The maximum network round-trip time between the control station and the drone
$\mathcal{E}$	The time required by the drone to execute the entire shuffling procedures
K_i	A shuffling seed shared by the drone and the control station at a time period i
LT_i	Look-up table at a time period i
$SHF()$	SPACE cipher embedding $\mathcal{SK}$ with our shuffling function.
γ	The number of space rounds that completes one shuffling round

5.5.1 Setup

We assume that two entities, i.e., a drone and a control station, agree on the elliptic curve parameters $\{F_q, E/F_q, G_q, P\}$, where q is a k-bit prime and P is the generator of G_q. We assume that the drone receives the certificate of the control station. For the digital signature scheme, a traditional digital signature, such as Elliptic Curve Digital Signature Algorithm (ECDSA), is used for our mechanism. The drone and the control station share a secret key ($\mathcal{SK}$) through a secure channel when the drone is in a secure area where white-box attacks cannot be launched. The drone generates a look-up table LT_0 for the SPACE cipher[3] using a block cipher $E_{\mathcal{SK}}$, such as AES-128, and then completely deletes $\mathcal{SK}$ from memory by overwriting it with a garbage value. The control station does not need to generate the look-up table. Instead, it can utilizes $\mathcal{SK}$ instead of the look-up table for encryption/decryption. Table 5.1 shows the notations used in our scheme.

5.5.2 Preparation for Shuffling

To shuffle the look-up table, the drone and the control station must first authenticate each other and share a shuffling seed K_i. We utilize the authenticated Elliptic Curve Diffie-Hellman (ECDH) key agreement protocol to securely establish K_i against black-box attackers. The control station periodically initiates the shuffling mechanism at regular time intervals by executing the following steps:

- At a time period i, the control station chooses a random number $a \in \mathbb{Z}_q^*$.
- It computes $P_a = aP$ and $h_1 = H(id_s \parallel P_a \parallel i)$, where $\parallel$ denotes the concatenation operation. $H()$ is a cryptographic hash function, such as SHA2, and id_s is the identity of the control station.
- It signs h_1 and outputs the signature s.

[3]Although our scheme is based on the SPACE cipher [43], it can be applicable to any ciphers based on Generalized Feistel Networks (GFN) and table-ups.

Then, the control station sends id_s, P_a, i, and s to the drone, and records the current time $\mathcal{T}_s$. After receiving them, the drone executes the following steps:

- It checks whether i indicates the current time period. If not, it outputs an incorrect time period error and aborts the execution.
- It computes $h_1' = H(id_s \parallel P_a \parallel i)$.
- It checks whether the signature s is valid by providing h_1' as input to the verification procedure. If s is not valid, it outputs a signature verification error and aborts the execution.
- It chooses a random number $b \in \mathbb{Z}_q^*$.
- It computes $P_b = b \cdot P$ and $K_i = b \cdot P_a$ $(= ab \cdot P)$.
- It runs the shuffling function SHF with K_i as input and outputs a value σ i.e., $\sigma \leftarrow SHF(K_i)$, where SHF is the SPACE cipher embedding $\mathcal{SK}$ with a shuffling functionality. The details are described in Sec. 5.5.3.
- It computes $h_2 = H(id_v \parallel \sigma \parallel P_a \parallel P_b)$, where id_v is the identity of the drone.

K_i is completely erased by replacing it with a garbage value in memory right after the first space round of SHF. Erasing K_i makes it harder for the attacker to know the shuffling procedure since this reduces the chance that the malware finds K_i. However, this is not an essential procedure for our defense scheme. Our defense scheme is secure even if K_i is leaked by the attacker (see Sec. 5.6.2).

The drone sends id_v, P_b and h_2 to the control station. After receiving them, the control station records the current time $\mathcal{T}_e$ and executes the following steps:

- It checks if $\mathcal{T}_e$-$\mathcal{T}_s < MAX_{RTT} + \mathcal{E}$, where MAX_{RTT} is the maximum network round-trip time between the control station and the drone. $\mathcal{E}$ is the time required by the drone to execute the entire shuffling procedures. If not, it aborts the process and sends *failure*.

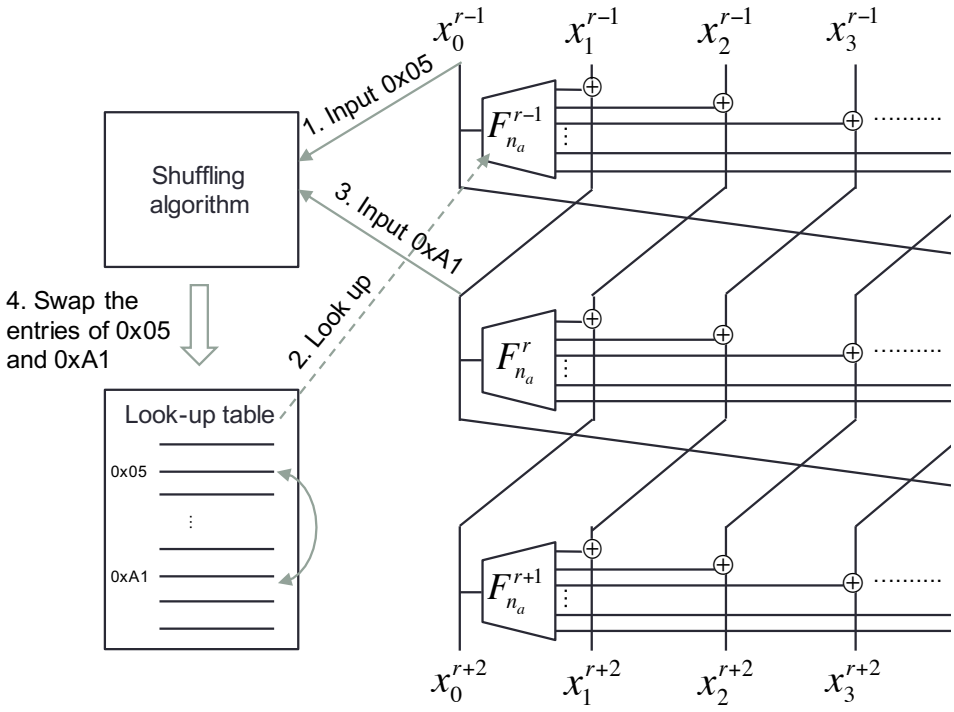


Figure 5.5.: Interaction between our shuffling algorithm and the SPACE cipher

- It computes $K_i = a \cdot P_b \ (= ab \cdot P)$.
- It runs the shuffling function SHF with K_i as input and outputs a value σ' i.e., $\sigma' \leftarrow SHF(K_i)$.
- It computes $h_2' = H(id_v \parallel \sigma' \parallel P_a \parallel P_b)$.
- It checks whether h_2 is equal to h_2'. If they are equal, it sends *success*. Otherwise, it sends *failure* and restores the original look-up table.

5.5.3 Shuffling

Design Motivation

In general, a shuffling algorithm generates a random permutation of a finite sequence based on a pseudo-random number generator. Random permutations that

are generated by the pseudo-random number generator are determined by only seeds. Therefore, if a white-box attacker can steal the seeds, he/she can re-generate the permutations. A secure shuffling algorithm against white-box attackers must refer to the entire look-up table so that a white-box attacker cannot generate the sequence without the knowledge of the entire look-up table even if he/she has the knowledge of K_i.

Objectives

SHF has two objectives. First, SHF is used to generate an authentication code σ like HMAC as we discribed in Sec. 5.5.2. Only when the look-up tables at both entities are identical, the outputs of $SHF(m)$ at both entities will be same. Therefore, in our mechanism, a control station can authenticate legitimate drones. An attacker without the knowledge of the entire look-up table of a drone cannot be authenticated by the control station.

Second, SHF shuffles the look-up table in order to change the location of every table entry. We utilize the fact that, only when the control station and the drone share an identical look-up table and an identical input, i.e, $K_i(=ab \cdot P)$, for SHF, their table look-up sequences will be same.

Shuffling Algorithm

Let x_i^r be the value of the $(i+1)$-th line at $(r+1)$-th space round. Then, the sequence of the table look-ups is as follows: $x_0^0 \rightarrow x_0^1 \rightarrow x_0^2 \cdots \rightarrow x_0^\gamma$. The order of the sequence is an important factor for our shuffling algorithm. After both entities share K_i, they execute the shuffling function (SHF) with K_i as input. If the input K_i is larger then n-bit, it is truncated to n-bit. As shown in Fig. 5.5, SHF is a dynamic version of the SPACE cipher with γ *space rounds* interacting with our shuffling algorithm.

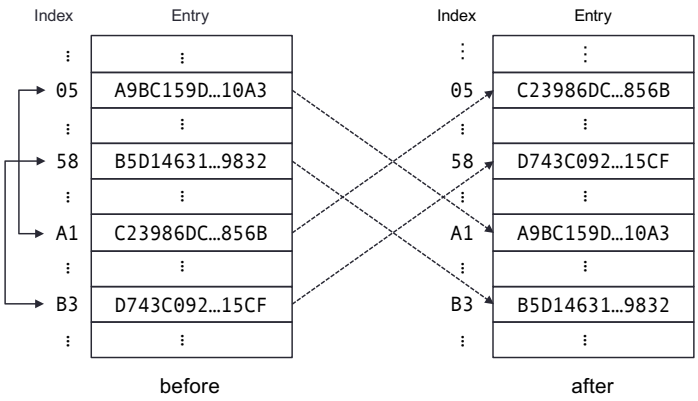


Figure 5.6.: Positions of the entries when the table look-up sequence of the drone and the control station is 05→A1→58→05→B3→ $\cdots$.

Definition 5 (Space round) *One* space round *is one execution of* $\left(F^r_{n_a}(x^r_0) \oplus (x^r_1 \parallel x^r_2 \parallel ...x^r_{l-1})\right) \parallel x^r_0$ *in the SPACE cipher.*

For example, if n_a is 8, $SHF(m)$ behaves like the SPACE(8, γ) cipher which takes an n-bit input plaintext m and outputs an n-bit ciphertext by executing γ space rounds. Note that SHF is not identical to the original SPACE cipher since the look-up table is dynamically changed throughout γ space rounds. After γ space rounds, one *shuffle round* is completed.

Definition 6 (Shuffle round) *One* shuffle round *is γ space rounds. After γ space rounds, the positions of all entries in the look-up table are swapped at least once.*

Given a sequence of table look-ups, $x^0_0 \rightarrow x^1_0 \rightarrow x^2_0 \cdots \rightarrow x^\gamma_0$, the entry at x^i_0 and the entry at x^j_0 $(j > i)$ are swapped in the process of SHF unless both entries have been swapped before. In order to reduce the number of memory accesses and to save shuffling time, we do not move entries more than once. As shown by Algorithm 4, the algorithm prepares a boolean array $u = \{u[0], u[1], ..., u[2^{n_a} - 1]\}$ and sets each entry in the array to *false*. The array u is used to check if an index x^i_0 was previously presented or not. At every space round, the algorithm computes $F^{r-1}_{n_a}(x_0) \oplus x_1$, outputs the next table index (x_0), and checks if x_0 has been presented before (line

Algorithm 4 Shuffling Algorithm

1: $cnt \leftarrow 0$
2: $u[0], ..., u[2^{n_a} - 1] \leftarrow false$
3: $\hat{e} \leftarrow empty$
4: $i \leftarrow -1$
5: $r \leftarrow 0$
6: $x_0 \leftarrow x_0^0$ ▷ The first n_a-bit of K_i
7: $e_0 \leftarrow e[x_0^0]$ ▷ e[x]: the entry value at x
8:
9: **while** $cnt < 2^{n_a}$ **do** ▷ 1 loop is a space round
10: **if** $u[x_0] = false$ **then**
11: $u[x_0] \leftarrow true$
12: **if** $\hat{e} = empty$ **then**
13: $\hat{e} \leftarrow e_0$
14: $i \leftarrow x_0$
15: **else**
16: $e[i] \leftarrow e_0$ ▷ swap
17: $e[x_0] \leftarrow \hat{e}$
18: $\hat{e} \leftarrow empty$
19: $cnt \leftarrow cnt + 2$
20: **end if**
21: **end if**
22: $r \leftarrow r + 1$
23: $e_0 = \hat{F}_{n_a}^{r-1}(x_0) \oplus \alpha$
24: $x_0 \leftarrow e_o \oplus x_1$ ▷ $0 \leq x_0 \leq 2^{n_a} - 1$
25: **end while**

10). If not, it checks if a temporary variable $\hat{e}$ is empty (line 12). If $\hat{e}$ is empty, x_0 and the entry value at x_0 (i.e., e_0) are saved at some temporary variables, i.e., i and

$\hat{e}$, respectively. If $\hat{e}$ was already occupied, two entries ($\hat{e}$ and $e[x_0]$) are swapped (line 16-17). As illustrated in Fig. 5.6, given a sequence like 05→A1→58→05→B3→..., the entry at 05 and the entry at A1 are swapped after the second space round, and the entry at 58 and the entry at B3 are swapped after the fifth space round. Note that, since $\hat{F}^r_{n_a}(x)$ is implemented by table look-ups in the drone, the drone has a look-up table and actually shuffles the table. However, the control station utilizes $E_{\mathcal{SK}}$ for $\hat{F}^r_{n_a}(x)$. Therefore, it needs to have a mapping table which stores the current shuffling status. For example, if the entry originally located at 3 is moved to 7 after shuffling in the drone, the control station must run $\hat{F}^r_{n_a}(7)$ instead of $\hat{F}^r_{n_a}(3)$ when the input is 3.

Shuffling Cost

1) Expected space rounds for one shuffle round: To protect against an adversary who knows all entry values of the look-up table and their positions, the positions of all entries must be changed at least once. Since the input of SHF, i.e. K_i, is a random number, the sequence of the table look-ups is also random. Obtaining the expected number of space rounds to complete one shuffle round is equal to the well-known coupon collector's problem [150].

Let D be the number of draws to collect all N coupons, and let d_i be the number of draws to collect the i-th coupon ($1 \leq i \leq N$) after $i-1$ coupons have been collected. Then, D and d_i are considered as random variables. The probability of collecting a

new coupon given $i-1$ coupons is $p_i = \frac{N-(i-1)}{N}$. Since d_i has a geometric distribution with expectation $1/p_i$, the expectation of D is as follows:

$$\begin{aligned} E(D) &= E(d_1) + E(d_2) + E(d_3) + ... + E(d_N) \\ &= 1/p_1 + 1/p_2 + 1/p_3 + ... + 1/p_N \\ &= \frac{N}{N} + \frac{N}{N-1} + \frac{N}{N-2} + ... + \frac{N}{1} \\ &= N \times \sum_{k=1}^{N} \frac{1}{k}. \end{aligned}$$

The asymptotic growth rate of $E(D)$ is $\Theta(N \log N)$ [155]. In our shuffling mechanism, the coupons correspond to the table entries and the number of entries is 2^{n_a}. Therefore, the expected space rounds for one shuffle round is as follows,

$$E(\gamma) = 2^{n_a} \times \sum_{k=1}^{2^{n_a}} \frac{1}{k}.$$

For instance, for values of n_a equal to 8 and 16, $E(\gamma)$ is approximately 1,567 and 764,646, respectively.

2) Execution cost: Here, we consider only the memory accesses to the look-up table and the boolean array u since other temporary variables can reside on a faster cache memory space. To complete one shuffle round, the *while* loop in Algorithm 4 must be repeated γ ($\approx n_a \times 2^{n_a}$) times and 2 reads are required per repeat at lines 10 (one bit read from u) and 23 (an entry read from the look-up table). Since the lines from 11 to 20 are repeated 2^{n_a} times (2^{n_a} is the number of the look-up table entries), 2^{n_a} writes to update the array of u (line 11) and 2×2^{n_a} writes (lines 16 and 17) to swap two entries in the look-up table are required. Thus, if $\mathcal{N}$ shuffle rounds are executed, the total memory access cost is

- $\gamma \times \mathcal{N}$ bit reads from u
- $\gamma \times \mathcal{N}$ table entry reads from the look-up table
- $2^{n_a} \times \mathcal{N}$ bit writes to u.

Table 5.2. The expected number of table entry reads/writes for one shuffle round

	$E(\gamma)$	E(num. of reads)	E(num. of writes)
$n_a = 8$	1,567	1,567	512
$n_a = 16$	764,646	764,646	131,072

- $2 \times 2^{n_a} \times \mathcal{N}$ table entry writes to the look-up table.

Note that the use of u reduces the amount of bytes to be written into the look-up table from $\gamma \times \mathcal{N} \times (128 - n_a)/8$ bytes to $2 \times 2^{n_a} \times \mathcal{N} \times (128 - n_a)/8$ bytes at the cost of $\gamma \times \mathcal{N}$ bit reads from u and $2^{n_a} \times \mathcal{N}$ bit writes to u. In other words, the use of u can reduce the number of table entry writes by $n_a/2$ times and increase the overall shuffling efficiency considering that u can reside in cache memory and the cost to access u is small[4]. Table 5.2 shows the expected number of reads and writes.

The malware might attempt to leak the shuffle sequence. The shuffling mechanism with malware intervention requires at least additional $2^{n_a} \times \mathcal{N}$ reads, $2^{n_a} \times \mathcal{N}$ writes in order to copy every element of the shuffle sequence in the memory space of the malware. Therefore, the attempt to leak the sequence can be detected if $\mathcal{N}$ is sufficiently large. Selecting $\mathcal{N}$ depends on n_a and the network round-trip time (*drone* $\rightleftharpoons$ *control station*) and discussed in Sec. 5.7.3.

3) Encryption/decryption cost: After the table is shuffled, plaintexts/ciphertexts are encrypted/decrypted using the original SPACE cipher. Since only the positions of the table entries change, our shuffling mechanism does not require additional storage and computation costs for encryption/decryption compared to the SPACE cipher.

[4]According to memory bandwidth benchmarks [156], cache is 2 to 9 times faster than main memory.

5.5.4 Group Communication

A secure communication protocol between drones is required when a set of drones cooperatively performs a common mission like search and rescue. In this section, we briefly discuss how our dynamic white-box block cipher can support communication between a group of drones.

Setup: Each drone generates two look-up tables by executing the setup procedure in Sec. 5.5.1. First, each drone $\mathcal{V}^x$ generates an individual look-up table (LT_0^x) to communicate with the control station. Second, all the drones share the same look-up table (LT_0^g) to communicate with the group members.

Preparation for shuffling: At a time period i, a leader drone $\mathcal{V}$ first establishes a shuffling seed K_i with the control station using ECDH (see Sec. 5.5.2). The leader drone can be pre-assigned or each drone can become the leader in turn. The control station generates $C = E_g(CMD_{sh}||ID_{\mathcal{V}}||K_i||i)$, signs C and outputs s, where $E_g()$ is the SPACE cipher using LT_i^g, CMD_{sh} is a command for shuffling, $ID_{\mathcal{V}}$ is the ID of $\mathcal{V}$ and i is the current time period. Then, the control station broadcasts C and s to all the drones. Some drones may not be able to listen C and s because they are temporarily out of range of the control station. In such a case, they request C and s from the other group members or the control station after they are within the communication range of the control station.

Shuffling: If s is valid, $\mathcal{V}^x$ obtains K_i by decrypting C and shuffles LT_{i-1}^x/LT_{i-1}^g using the shuffling mechanism in Sec. 5.5.3 and obtains LT_i^x/LT_i^g.

Join: If the control station wants to make a new drone join the group, the control station sends the current group look-up table (LT_i^g) to the new drone.

Leave: If the control station wants to make a drone $\mathcal{V}^y$ leave the group, the control station prevents $\mathcal{V}^y$ from synchronizing T_i^g with the other drones. That is, the control station individually establishes a shuffling seed K_i with all the drones except $\mathcal{V}^y$. Then, the control station generates $C_x = E_x(CMD_{sh}||ID_{\mathcal{V}}^x||K_i||i)$ for $\mathcal{V}^x$ ($\neq \mathcal{V}^y$),

where $E_x()$ is the SPACE cipher using the individual look-up table LT_i^x for $\mathcal{V}^x$. Then, the control station sends C^x to $\mathcal{V}^x$.

5.6 Security Analysis

5.6.1 Black-box attacks

Our dynamic block cipher utilizes the SPACE cipher [43] as the encryption/decryption algorithm and the security level of the SPACE cipher is as strong as AES. Therefore, an eavesdropper (black-box attacker) cannot obtain any information about the plaintext from its ciphertext. Also, the eavesdropper is not able to know the K_i (i.e., shuffling input value) established by the authenticated ECDH key agreement protocol unless he/she can solve the Elliptic Curve Discrete Logarithm Problem (ECDLP) or forge the ECDSA signature.

Our shuffling mechanism is built based on the security of asymmetric-key cryptography as well as symmetric-key cryptography to prevent a black-box attacker from falsely initiating our shuffling algorithm. The black-box attacker may try to impersonate a drone (or a control station) in order to shuffle the look-up table of a control station (or a drone) incorrectly. To prevent the attacker from impersonating a control station, our shuffling mechanism adopts ECDSA. Since only a legitimate control station with its private key can generate a valid signature, a drone can safely initiate the shuffling algorithm only after it verifies the signature. In addition, the increasing current time period t prevents s and P_a from being replayed. It is important to prevent the false initiation for drone since the original look-up table cannot be recovered after it is shuffled. A drone might record the shuffle state information in memory in order to recover the original look-up table. However, such memory recording of the shuffle state information may introduce a new security risk. If such information is leaked by a white-box attack, the attacker is able to shuffle his/her look-up table without needing to know K_i, all the entry values of the look-up table, and their positions.

To prevent the attacker from impersonating a drone, our shuffling mechanism utilizes the white-box block cipher, i.e., SHF as a keyed-hash message authentication code. In SHF, σ is generated by looking up all the entries of the look-up table where a secret key ($\mathcal{SK}$) is embedded. Since $\mathcal{SK}$ is shared only between the drone and the control station, the control station can authenticate the drone. Therefore, an attacker, who does not know $\mathcal{SK}$ or the entire information about the look-up table, cannot generate a valid σ. After the control station receives $h_2(= H(id_v \parallel \sigma \parallel P_a \parallel P_b))$, id_v and P_b, it also computes σ' and $h_2'(= H(id_v \parallel \sigma' \parallel P_a \parallel P_b))$. Then, it checks whether σ, id_v, P_a and P_b are valid by comparing h_2' and h_2. If h_2' is equal to h_2, the control station can authenticate the drone. Otherwise, the mapping table reverts to the original state. In case of the control station, reverting is not a high cost as it is for a drone since the control station can hold the previous mapping table during the shuffling mechanism and use it in the event that h_2 turns out to be different from h_2'.

5.6.2 White-box attacks

In the white-box attack model, any data stored in the drone can be leaked by the attacker. In this section, we consider an attacker who breaks in the drone system at a time period i and steals the current look-up table LT_i. Based on LT_i, the attacker wants to decrypt communications encrypted at past periods, and also wants to encrypt/decrypt future communications. We first discuss the decryption success probability when the attacker knows only a subset of LT_i. Second, we introduce a stronger attacker who knows the entire LT_i. We discuss how our shuffling mechanism makes it hard for such attackers to successfully decrypt the ciphertext, and thus provides the SPACE cipher with forward security. Finally, we prove that our shuffling mechanism makes it hard for the break-in attacker to decrypt future communications if he/she does not have the knowledge of the entire look-up table. Finally we summarize the security analysis of our techniques with respect to white-box attacks.

Decryption/encryption success probability when a subset of look-up table entries is leaked

In the SPACE cipher [43] without shuffling, the probability that, given a random ciphertext, the corresponding plaintext can be decrypted with $t(\leq 2^{n_a})$ entries of the look-up table is $(t/2^{n_a})^R$, where R is the number of space rounds. Therefore, if an attacker knows all entries, he/she can always decrypt a ciphertext successfully regardless of R.

However, if the positions of all entries are changed by our shuffling mechanism, the attacker must locate entries on correct positions before he/she tries to decrypt a ciphertext. We consider an attacker who knows a subset of look-up table and formally define such an attacker as follows:

Definition 7 ($BreakInAttacker_i$) *$BreakInAttacker_i$ is an attacker who breaks in a system at a time i and knows a subset of look-up table LT_i. That is, $BreakInAttacker_i$ knows $t(\leq 2^{n_a})$ entries of the look-up table. However, $BreakInAttacker_i$ does not know their positions at a time period j $(j < i)$.*

Theorem 3 *The decryption success probability $P(S)$ with which $BreakInAttacker_i$ can decrypt any ciphertext recorded at a time period j $(j < i)$ is*

$$P(S) = \sum_{i=0}^{t} \Big(\frac{N_{T-i} \times i^R}{i! \times (T-i)! \times T^R} \Big),$$

where $t \leq T-1$ $(=2^{n_a}-1)$, S is the event that an attacker successfully decrypts (or encrypts) a ciphertext (or plaintext) and $N_{T-i} = (T-i-1) \times (N_{T-i-1} + N_{T-i-2})$. $N_j (j \geq 1)$ can be recursively calculated based on $N_1 = 0$ and $N_2 = 1$.

Proof Assume that the attacker knows t entry values, but does not know their positions. Then, the success probability that, given a random ciphertext, the corresponding plaintext can be computed using t entries is as follows:

$$P(S) = \sum_{i=0}^{t} \big(P(C = i) \times (i/2^{n_a})^R \big), \tag{5.1}$$

To successfully decrypt/encrypt a ciphertext/plaintext, the attacker first has to correctly locate i entries and then look up only the i entries throughout R space rounds. $P(C = i)$ can be obtained as follows:

(1) The probability that the attacker correctly locates all T entries is

$$P(C = T) = 1/T!.$$

(2) The probability that i entries are correctly located, but $T - i$ entries are located at the wrong positions is

$$P(C = i) = \frac{\binom{T}{i} \times N_{T-i}}{T!} = \frac{N_{T-i}}{i! \times (T-i)!},$$

where, N_j is the number of possible cases in which, given j entries, all j entries are located at wrong positions.

Therefore, $\binom{T}{i} \times N_{T-i}$ is the total number of cases in which i entries are correctly located, and $T - i$ entries, that is, the rest, are wrongly located. If i is $T - 1$, $P(C = T - 1)$ is 0 since N_1 is 0. In other words, it is impossible that only one entry is located at a wrong position, while $T - 1$ entries are correctly located.

To calculate $P(C = i)$ $(0 \leq i \leq T - 1)$, N_j $(1 \leq j \leq T)$ should be calculated. Since N_j is recursively obtained, we need to know N_1, N_2 and N_3 first. If there is only one entry, then it cannot happen that the entry is located at a wrong position. If there are two entries, there is only one case in which the two entries are located at wrong positions. Such case occurs when the first entry is located at the second position and the second entry is located at the first position. We denote this case as {2, 1}. Similarly, if there are three entries, there are only two cases in which all three entries are located at wrong positions. These cases are denoted as {2, 3, 1} and {3, 1, 2}. Therefore, we can obtain

$$N_1 = 0, N_2 = 1, N_3 = 2.$$

Now, we can calculate N_j using N_{j-2} and N_{j-3}. For instance, when 4 entries are given, the 4th entry must be located at one of three possible positions, i.e., {4, *, *,

}, {, 4, *, *} and {*, *, 4, *}. Suppose that the 4th entry is located at the 1st position, i.e., {4, *, *, *}. Then, one among the 1st, 2nd, and 3rd entries must be located at the 4th position. If the 1st entry is located at the 4th position, i.e., {4, *, *, 1}, then the 2nd and 3rd entries must be located as {4, 3, 2, 1} and this is the only case like N_2.

The 2nd or 3rd entry can be located at the 4th position. If the 2nd entry is located at the 4th position, i.e., {4, *, *, 2}, then the 1st and 3rd entries must be located as {4, **3**, **1**, 2} and this is the only case like N_2. Likewise, if the 3rd entry is located at the 4th position, i.e., {4, *, *, 3}, then the 1st and 3rd entries must be located as {4, **1**, **2**, 3} and this is the only case like N_2. Therefore, $N_4 = 3(N_2 + 2N_2) = 9$. Similarly, when 5 entries are given, the 5th entry must be located at one of four possible positions, i.e., {5, *, *, *, *}, {*, 5, *, *, *}, {*, *, 5, *, *} and {*, *, *, 5, *}. Suppose that the 5th entry is located at the 1st position, i.e., {5, *, *, *, *}. Then, one among the 1st, 2nd, 3rd, and 4th entries must be located at the 5th position. If the 1st entry is located at the 5th position, i.e., {5, *, *, *, 1}, then the 2nd, 3rd and 4th entries must be located as {5, **3**, **4**, **2**, 1} or {5, **4**, **2**, **3**, 1}. The number of these cases is equal to N_3.

The 2nd, 3rd or 4th entry can be located at the 5th position. If the 2nd entry is located at the 5th position, i.e., {5, *, *, *, 2}, then the 1st entry can be located anywhere, while the 3rd and 4th can be located the 4th and 3rd position, respectively. If the 1st entry is located at the 2nd position, i.e., {5, 1, *, *, 2}, then there is only one possible location for the 3rd and 4th entries, i.e., {5, 1, **4**, **3**, 2}, same as N_2. However, if the 1st entry must be located at the 3rd or 4th position, we can handle the 1st entry as the 2nd entry that must be located at the 3rd or 4th position. Therefore, there are two possible cases concerning where 1st, 3rd and 4th entries can be located, i.e., {5, **4**, **1**, **3**, 2} and {5, **3**, **4**, **1**, 2}, same as N_3.

$$\begin{aligned} N_5 &= 4 \times \big(N_3 + 3(N_2 + N_3)\big) = 4 \times \big(N_3 + N_4\big) \\ &= 4 \times \big(2 + 9\big) = 44. \end{aligned}$$

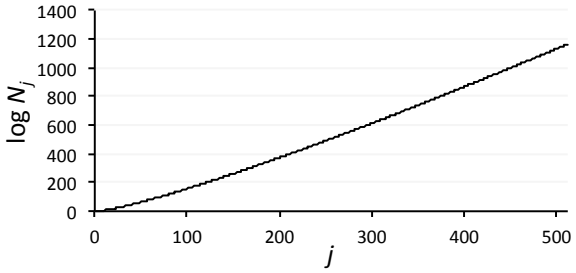


Figure 5.7.: Number of possible cases in which all the entries are located at wrong positions for different values of the number of entries.

Likewise, we can recursively calculate the number of possible cases in which all j entries are located at wrong positions as follows:

$$N_j = (j-1) \times (N_{j-2} + N_{j-1}) \tag{5.2}$$

The asymptotic growth rate of N_j is $O(j!)$. We have plotted Equation 5.2 in Fig. 5.7 the values of N_j $(1 \leq j \leq 512)$. The final form of Equation 5.1 is

$$\begin{aligned} P(S) &= \sum_{i=0}^{t} \left(P(C=i) \times (i/2^{n_a})^R \right) \\ &= \sum_{i=0}^{t} \left(\frac{N_{T-i} \times i^R}{i! \times (T-i)! \times T^R} \right), \end{aligned}$$

where $N_{T-i} = (T-i-1) \times (N_{T-i-2} + N_{T-i-1})$. ■

Decryption/encryption success probability when entire look-up table is leaked

Now we consider a stronger attacker who knows the entire look-up table LT_i at a time period i. If such an attacker cannot decrypt messages encrypted at a time period j $(< i)$, our encryption scheme is forward secure. In what follows, we formally describe how our shuffling mechanism provides the SPACE cipher with forward security.

Experiment construction: Messages encrypted using LT_j must not be decrypted even though the adversary knows LT_i $(i > j)$. Let E be an adversary algorithm

and $E^{\mathsf{D\text{-}SPACE.enc}(LT_i,\cdot)}$ (find, h) be E in the find state, taking the current history h and returning $(d, (m_0, m_1, j), h)$, where h is an updated history and $d \in \{\mathsf{find}, \mathsf{guess}\}$. Consider the following experiment:

Experiment $\mathbf{Exp}^{fsind-cpa}_{\mathsf{D\text{-}SPACE}}(E)$

$LT_0 \xleftarrow{\$} \mathsf{D\text{-}SPACE.key}$; $i \leftarrow 0$; $h \leftarrow \epsilon$

`Repeat`

$i \leftarrow i + 1$; $LT_i \leftarrow \mathsf{D\text{-}SPACE.update}(LT_{i-1}, K_{i-1})$

$(d, (m_0, m_1, j), h) \xleftarrow{\$} E^{\mathsf{D\text{-}SPACE.enc}(LT_i,\cdot)}(\mathsf{find}, h)$

`Until` $(d = \mathsf{guess})$ or $(i = n)$

$c \xleftarrow{\$} \{0, 1\}$

`If` $j \geq i$ `then return` c

`Else`

$C \xleftarrow{\$} \mathsf{D\text{-}SPACE.enc}(LT_j, m_c)$

$g \xleftarrow{\$} E(\mathsf{guess}, LT_i, C, h)$

`If` $g = c$ `then return` 1 `else return` 0

Adversary E first executes the find period where it accesses an oracle for the encryption algorithm using the current look-up table. At the end of a period, it decides to breaks in the system. That is, if its output d is guess, a pair (m_0, m_1) of equal length messages and the period j are provided. One of the two messages, m_c, is chosen at random and encrypted under LT_j to yield a challenge ciphertext C. E is then given the look-up table LT_i (from the break-in) and C, and wins if it guesses g. The forward-

secure indistinguishability under chosen-plaintext attack (*fsind-cpa-advantage*) of E in attacking D-SPACE is formulated as follows:

$$\mathbf{Adv}^{fsind-cpa}_{\text{D-SPACE}}(E) = 2 \cdot \Pr\left[\mathbf{Exp}^{fsind-cpa}_{\text{D-SPACE}}(E) = 1\right] - 1$$

The *fsind-cpa-advantage* of D-SPACE is the maximum over all adversaries E that have time-complexity at most t and make at most q queries in each period and can be formulated as follows:

$$\mathbf{Adv}^{fsind-cpa}_{\text{D-SPACE}}(q,t) = \max_{E}\{\mathbf{Adv}^{fsind-cpa}_{\text{D-SPACE}}(E)\}.$$

Bellare et al. [157] proved that the advantage of a forward secure key-evolving symmetric encryption scheme is negligible if the advantage of its base symmetric encryption scheme and the advantage of its forward secure pseudo-random bit generator are negligible. Applying this theorem to our scheme, we can obtain as follows:

$$\mathbf{Adv}^{fsind-cpa}_{\text{D-SPACE}}(q,t) \leq \mathbf{Adv}^{fsind-cpa}_{\text{GEN}}(q,t_1) + n\mathbf{Adv}^{ind-cpa}_{\text{SPACE}}(q,t_2)$$

where $t1 = t2 = 2t + O(n+b)$, b is the block length and GEN is our shuffling mechanism which determines new locations of the table entries. $\mathbf{Adv}^{fsind-cpa}_{\text{D-SPACE}}(q,t)$ is negligible if $\mathbf{Adv}^{fsind-cpa}_{\text{GEN}}(q,t_1)$ and $\mathbf{Adv}^{ind-cpa}_{\text{SPACE}}(q,t_2)$ are negligible. Under the assumption that the advantage of the original SPACE cipher is negligible [43], the advantage of D-SPACE is also negligible since $\mathbf{Adv}^{fsind-cpa}_{\text{GEN}}(q,t_1)$ is negligible due to the following two reasons: First, when the block length of D-SPACE is n-bit and SPACE(n_a, R) is used, GEN, i.e., our shuffling mechanism takes the first n-bit of $K_i(=ab \cdot P)$ as an input for a period i. Note that the randomness of K_i, which is the shared key based on the ephemeral private/public key pair for the period i, results from a and b which are selected uniformly at random in $\mathbb{Z}^*_q$. An adversary cannot infer any information about K_i from K_j $(i \neq j)$. Second, under the assumption that the adversary performs brute-force attacks to find the correct locations of the look-up table entries, the search space of shuffling sequences SP_s, i.e, $2^{n_a}!$, which is larger than the number of possible inputs SP_i, i.e., 2^n. For example, when SPACE(8, R) is used, SP_s is $256!(\approx \sqrt{2\pi e}(\frac{256}{e})^{256}$ according to the Stirling's approximation), while

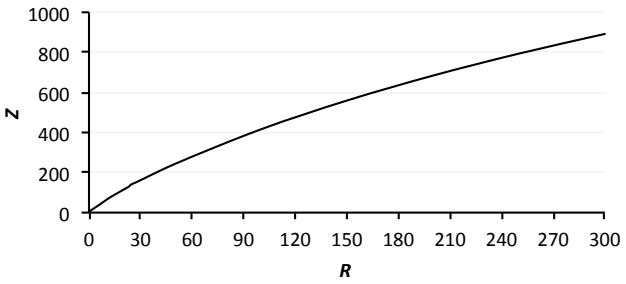


Figure 5.8.: The upper bound of a success probability when n_a is 8. Here, $Z = -\log_2 \hat{P}(S)$

SP_i is 2^{128}. Therefore, the shuffling mechanism is forward secure, in turn, provides the SPACE cipher with forward security.

Decryption/encryption success probability According to Theorem 3, the success probability that $BreakInAttacker_i$ decrypts a ciphertext is as follows:

$$P(S) = \sum_{i=0}^{T} \Big(\frac{N_{T-i} \times i^R}{i! \times (T-i)! \times T^R} \Big).$$

When SPACE(8, 300) or SPACE(16, 128) is used with our shuffling mechanism the above attacker's advantage is negligible since $P(S)$ is approximately 2^{-892} or 2^{-627}, respectively. Fig. 5.8 shows $P(S)$ according to the number of space rounds (R) when SPACE(8, *) is used with our shuffling mechanism. Without our shuffling mechanism, SPACE(8, *) requires 1,684($=-256/\log_2 0.9$) space rounds to limit the attacker's success probability to 2^{-256} when 90 percent of the table is leaked. When 99 percent of the table is leaked, 17,655($=-256/\log_2 0.99$) space rounds are required to achieve the same success probability. However, with our shuffling mechanism, only 54 space rounds are required to achieve the success probability of 2^{-258}.

Future communication protection

Our shuffling mechanism also makes it hard for the break-in attacker to decrypt/encrypt future communications if he/she does not have the knowledge of the

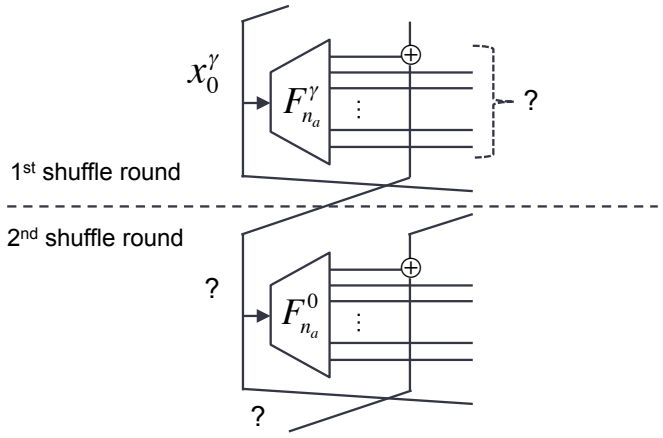


Figure 5.9.: The last space round in the 1st shuffle round and the first round in the 2nd shuffle round

entire look-up table entries. In our shuffling mechanism, K_i is established by the ECDH key agreement protocol and resides in the memory space for a very short time (see our experimental results in Sec. 5.7.3). Due to the hardness of the discrete logarithm problem and ECDH problem, an attacker $\mathcal{A}$ cannot establish K_i by eavesdropping communications. Therefore, it is hard for $\mathcal{A}$ to determine K_i. However, it is not impossible for $\mathcal{A}$ to obtain K_i by launching a white-box attack when $K(= b \cdot P_a)$ or b resides in the memory space. Then, we can think of four possible attacker's capabilities: (C1) $\mathcal{A}$ knows K_i and the whole look-up table entries; (C2) $\mathcal{A}$ does not know K_i, but knows the whole look-up table entries; (C3) $\mathcal{A}$ does not know K_i, but knows the partial look-up table entries; (C4) $\mathcal{A}$ knows K_i and the partial look-up table entries.

If $\mathcal{A}$ has the C1 capability, our shuffling mechanism does not guarantee security since $\mathcal{A}$ has the complete knowledge for SHF. If $\mathcal{A}$ does not know K_i, i.e. $\mathcal{A}$ has the C2 or C3 capability, our shuffling mechanism with *one* shuffle round guarantees security as we mentioned in the previous section. However, if $\mathcal{A}$ has the C4 capability, our shuffling mechanism requires *two* shuffle rounds in order to guarantee security.

Theorem 4 *When an attacker knows K_i and the partial look-up table, it is hard for the attacker to successfully decrypt/encrypt a ciphertext/plaintext after* two *shuffle rounds.*

Proof The worst case of the C4 capability is when an attacker $\mathcal{A}$ knows all entry values and their positions except for one entry value in the look-up table, i.e., when $T-1$ entries are leaked. Then, $\mathcal{A}$ can run $SHF(K_i)$ using the knowledge of the look-up table. In the first shuffle round of SHF, the best case for $\mathcal{A}$ is as follows. Given the look-up sequence, $x_0^0 \rightarrow x_0^1 \rightarrow x_0^2 \cdots \rightarrow x_0^\gamma$, if $\mathcal{A}$ knows all entry values except the entry value at x_0^γ, then $\mathcal{A}$ can correctly swap all entries even though $\mathcal{A}$ does not know the entry value at x_0^γ. However, $\mathcal{A}$ cannot correctly swap the entries from the second shuffling round, as shown in Fig. 5.9, since $\mathcal{A}$ does not know the first index (x_0^0) for the first space round in the second shuffle round. Thus, all entries are safely shuffled throughout the second shuffle round. As a result, if $\mathcal{A}$ knows K_i and all entry values except one entry value, it is hard for $\mathcal{A}$ to successfully decrypt/encrypt a ciphertext/plaintext after two shuffling rounds since $\mathcal{A}$ does not know any positions of entries. ■

Note that the number of shuffle rounds is usually more than two in order to detect malware's intervention as discussed in Sec. 5.5.3. In Sec. 5.7.3, we provide our recommendations for the number of shuffle rounds according to n_a.

Summary of security analysis for white-box attacks

Our dynamic block cipher satisfies the security requirements with respect to the white-box attack model as follows.

First, no short secret key is used by a drone to encrypt/decrypt messages. Instead, the drone utilizes a large look-up table constructed by the SPACE cipher algorithm. Therefore, the attacker cannot extract the short secret key that was used to generate the table.

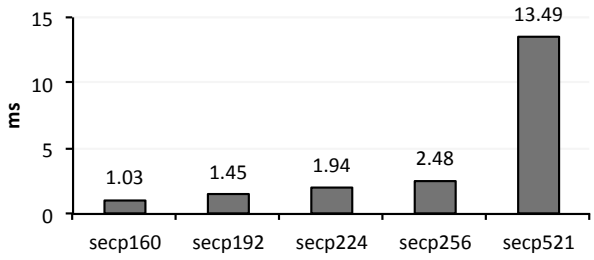


Figure 5.10.: Time required for a signature verification

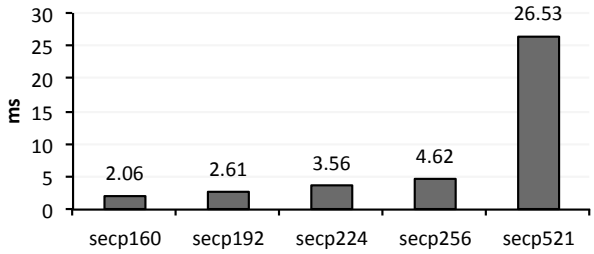


Figure 5.11.: Time required to compute bP and bP_a

Second, we have shown that our shuffling mechanism makes it hard for the break-in attacker to successfully en-/decrypt any past/future plaintext/ciphertext. Although our dynamic block cipher does not cover the case in which the attacker knows the whole look-up table and K_i, we believe that it is hard for the attacker to obtain both information because of the large size of the table and the short residence time of K_i.

Third, our shuffling mechanism does not store a shuffling sequence in memory. In addition, since a shuffling sequence generated by our shuffling mechanism is fully associated with the look-up table, the attacker cannot generate the shuffling sequence by simply extracting a seed K_i.

Finally, even if the malware can steal a shuffling sequence, it is hard to steal it without be detected since stealing shuffling sequence requires additional memory access, which increases the execution time of our shuffling mechanism. With a sufficiently large number of shuffle rounds, the control station can easily detect such malicious behaviors by measuring the response time.

5.7 Evaluation

In this section, we present the performance of our shuffling mechanism and the GPU-accelerated SPACE cipher on a device with a GPU-enabled SoC. Also, we present the energy consumption of a drone (DJI phantom 3) that runs the SPACE cipher with our shuffling mechanism.

5.7.1 Experimental Setup

We utilized the Nvidia Jetson TK1 developer kit [79] equipping the Tegra K1 SoC, which is used for Parrot's Kalamos [131]. Its operating system is Ubuntu Linux (ver. 3.10.40) and its main memory is DDR3L of size 8 GB. The Tegra K1 SoC consists of a ARM Cortex-A15 CPU (2.3 GHz) and Nvidia Kepler GPU (0.85 GHz) with 192 CUDA Cores. We chose this kit since the GPU in the Tegra K1 SoC is the only mobile GPU to support Nvidia CUDA. Its CUDA runtime version is 6.5 and capability version is 3.2. The maximum number of threads per block is 1,024. The size of the global memory is 1,892 MB and the size of the shared memory is 49,152 B. We utilized the MIRACL crypto-library [78] (ver. 7.1) which is an open source SDK for elliptic curve cryptography (ECC). Since the SPACE cipher requires a block cipher for $\hat{F}^r_{n_a}(x)$, we adopted the implementation source code of AES-128 from mbedTLS [78] which is an open source crypto-library and is designed to fit on embedded devices. All the schemes for evaluations in what follows have been implemented in the C language and all the execution time measurements reported are the averages after 100 iterations.

Two SPACE ciphers, i.e., SPACE(8, 300) and SPACE(16, 128), have been chosen for our implementation since their look-up table sizes are 3.83 ($= 2^8 \times 15$) KB and 918 ($= 2^{16} \times 14$) KB, respectively, and thus adequate for an embedded device with a limited memory space. We exclude SPACE(24, 128) and SPACE(32, 128) since their look-up table sizes are 218 MB and 51.5 GB, respectively, which are too large for the memory of our target systems (drones). Even if the systems uses virtual memory,

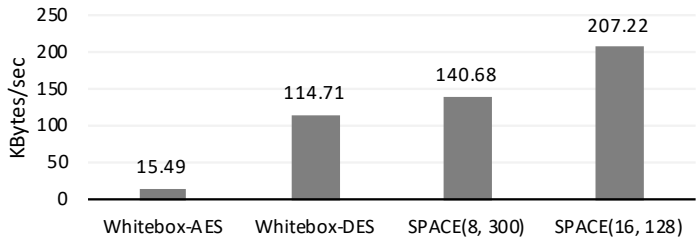


Figure 5.12.: Encryption speeds of white-box encryption algorithms

they will suffer from thrashing which may cause serious operational faults in drones because the look-up table entries will frequently move between physical memory and the HDD (or SSD).

5.7.2 Performance Comparison of White-box Encryption Schemes

We have chosen the SPACE cipher as the base encryption algorithm of our scheme since the other existing white-box encryption algorithms have been cryptanalyzed as discussed in Sec. 5.2.1. However, performance is also a critical factor since many drone applications require efficiency. We have measured the encryption speed of three representative white-box encryption schemes, i.e., 1) white-box DES (WB-DES) [41], 2) white-box AES (WB-AES) [36][5] and 3) the SPACE cipher [44] on the CPU of the Nvidia Jetson TK1 kit. Note that the numbers of the table look-ups (and XORs) for WB-AES, WB-DES, SPACE(8, 300) and SPACE(16, 128) are 3008, 384, 300 and 128, respectively. As shown in Fig. 5.12, the SPACE cipher outperforms the others. This result confirms that the encryption performance is highly dependent on the number of table look-ups and, consequently, the SPACE cipher is the best choice for drone applications in terms of performance as well as security.

[5]We utilized the implementation of white-box AES in https://github.com/jeffsaremi/wbaes and the implementation of white-box DES in https://github.com/mimoo/whiteboxDES.

5.7.3 Shuffling Mechanism Execution Times

Look-up table generation: We measured the time required to generate a look-up table at the system setup step. When n_a is 8, a drone must execute AES-128 256 times to generate the table of size 3.83 KB and the time required for the look-up table generation is only 0.152 ms. When n_a is 16, AES-128 must be executed 65,536 times to generate the table of size 918 KB and it takes 25.29 ms. Considering that look-up tables are generated very infrequently, such times are not a large overhead for a drone.

Signature verification and ECDH: At the beginning of our shuffling mechanism, the control station sends a digital signature and P_a. After receiving them, the drone verifies the signature and computes bP and bP_a. We measured the time required to verify a signature, and compute bP and bP_a. As elliptic curve parameters, we utilized secp160, secp192, secp224, secp256, and secp521. Their key sizes are 160, 192, 224, 256, and 521 bits, respectively. Fig. 5.10 shows the time required to verify a signature and Fig. 5.11 shows the time required to compute bP and bP_a. In both executions, considering that b resides in memory during the ECDH execution and an attacker may launch a white-box attack, using a 521-bit key is not a good choice since it takes too long time. In addition, the time difference between a 224-bit key and a 256-bit key is much smaller than the difference between a 256-bit key and a 521-bit key, which implies that a 256-bit key may be a reasonable choice since it provides it provides better security (128-bit symmetric-key security) than a 128-bit key (112-bit symmetric-key security) with a very small time increase.

Shuffling Time: We measured the time required to shuffle a look-up table when different shuffle rounds were applied. Also, we implemented a malware emulator, which leaks a shuffle sequence. The emulator reads each element of the shuffle sequence during shuffling and writes the element in a buffer in order to transfer the sequence to a remote attacker after all the shuffling procedures are completed. Fig. 5.13 and Fig. 5.14 show the shuffling mechanism execution time when n_a is 8

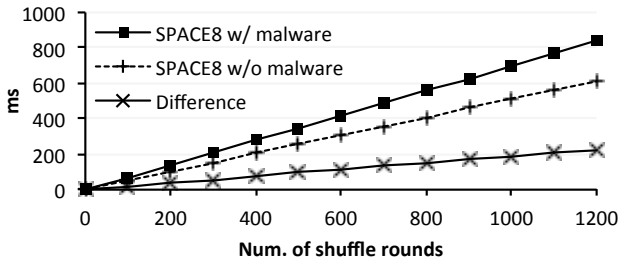


Figure 5.13.: Shuffling mechanism execution time when n_a is 8

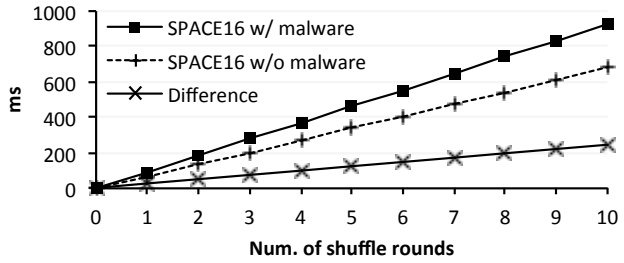


Figure 5.14.: Shuffling mechanism execution time when n_a is 16

and 16, respectively. The number of shuffle rounds must be large enough to detect malware intervention. Considering that the maximum network round-trip time (RTT) of real-time data in LTE is 150ms [158, 159], we can set the number of shuffle rounds to be 800 and 7 when n_a is 8 and 16, respectively. For instance, when n_a is 8 and the number of shuffle rounds is 800, shuffling without the malware requires 408.1ms, while shuffling with the malware requires 559.3ms. Assume that secp256 is used and the RTT range is [30, 150]ms. If there is no malware intervention, the control station $\mathcal{C}$ can receive the shuffling results from the drone within 565.2(=4.62+2.48+408.1+150)ms[6]. If there exists malware intervention, $\mathcal{C}$ would receive the shuffling results after 596.4(=4.62+2.48+559.3+30)ms. Thus, if $\mathcal{C}$ receives the shuffling results between 565.2ms and 596.4ms, $\mathcal{C}$ may retry the shuffling mech-

[6]For the sake of simplicity, we ignore the computation time for other operations like hash computations and delays in the operating system.

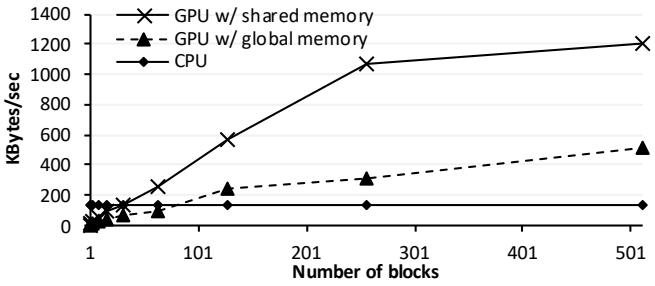


Figure 5.15.: The encryption performance when SPACE(8, 300) is used

anism with the drone η times more. If the number of retries exceeds η or $\mathcal{C}$ receives the results after 596.4ms, $\mathcal{C}$ can regard the drone as compromised.

5.7.4 Encryption Time

We measured the encryption time on the CPU and the GPU when the number of message blocks increases from 1 to 512. The size of one message block is 16 B. The CPU just performs encryptions in serial order, while each thread in the GPU performs encryptions in parallel, meaning that n threads are created for the encryptions of n blocks.

Before the GPU starts encryptions, the entire look-up table is copied to the global memory for the GPU. When n_a is 8, the entire table in the global memory can be loaded on the shared memory since the table size is 3.83 KB, while the size of the shared memory is 49.152 KB. However, when n_a is 16, the table in the global memory cannot be loaded on the shared memory since the table size is 918 KB (>49.152 KB). In such a case, the shared memory is not used. The shared memory provides a much faster access speed than the global memory. Although it takes time to load the table on the shared memory, once the table is loaded on the shared memory, each thread can look up the table in the shared memory, and thus perform encryptions much faster than when the table resides in the global memory.

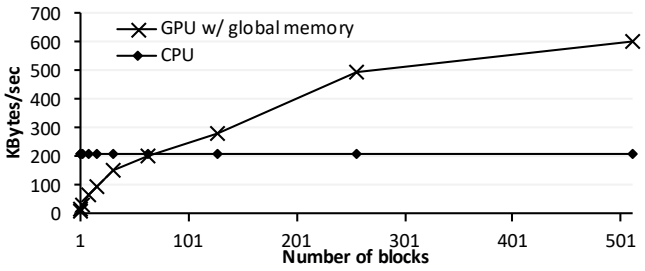


Figure 5.16.: The encryption performance when SPACE(16, 128) is used

Fig. 5.15 shows the encryption performance according to the number of blocks when SPACE(8, 300) is used. When the number of blocks is 1, the encryption performance on the CPU is 21 times better than the performance on the GPU due to two reasons: (1) the clock rate of the CPU is faster than the clock rate of the GPU and (2) it takes time to copy the look-up table and the message block to the global/shared memory for the GPU. However, as the number of blocks increases, the time required to copy the look-up table is amortized and the number of threads executed in parallel increases. If the number of blocks is larger than 32, the encryption performance of the GPU with the shared memory is better than the encryption performance of the CPU. As the number of blocks increases, the performance gap between the GPU with the shared memory and the CPU becomes larger. If the number of blocks is less than 256, the encryption performance on the GPU with the shared memory increases as the number of blocks increases since each block is encrypted by each thread in parallel. However, if the number of blocks is greater than 256, the performance on the GPU does not increase as much as the number of blocks increases due to the limited memory bandwidth.

Fig. 5.16 shows the encryption performance when SPACE(16, 128) is used. Compared to when SPACE(8, 300) is used, the encryption performance on the CPU is approximately 1.5 times better since it requires less space rounds ($128 < 300$) than SPACE(8, 300) and the unit for memory access (2 B) is larger than the unit of memory access (1 byte) in SPACE(8, 300). When SPACE(16, 128) is used, the look-up

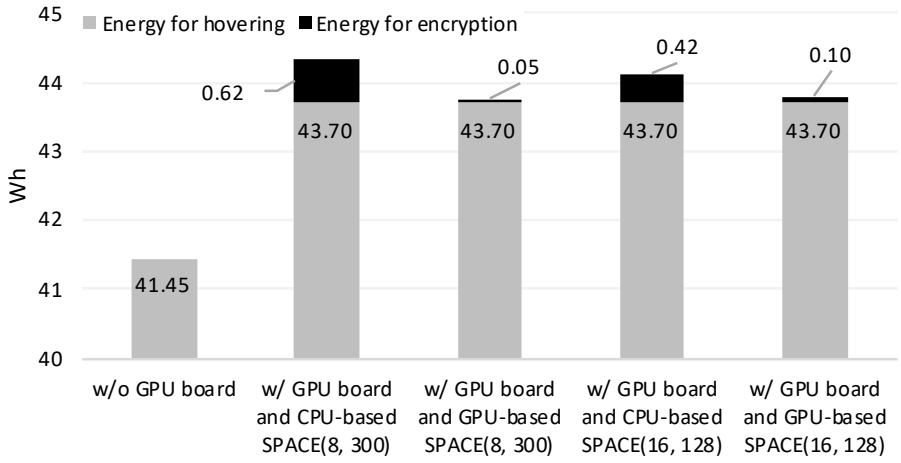


Figure 5.17.: Energy consumption for 20 minutes flight and 100MB encryption

table cannot be loaded on the shared memory. Since the threads in the GPU must access the slower memory, i.e., the global memory, the performance gap between the GPU and the CPU is not as large as the performance gap when SPACE(16, 128) is used. However, if the number of blocks is greater than 64, the encryption speed on the GPU outperforms the encryption speed on the CPU.

In summary, when SPACE(8, 300) is used, the most efficient approach is to use the GPU with the shared memory if the number of blocks is greater than 32. When SPACE(16, 128) is used, the most efficient approach is to use the GPU if the number of blocks is greater than 64.

5.7.5 Energy Consumption

We utilized a DJI Phantom 3 [160] drone to measure the energy consumption when it is hovering in the air with/without the Nvidia Jetson TK1. We developed a telemetry monitoring program using the DJI mobile SDK [161] and measured the required power for the drone to hover. Nvidia Tegra TK1 is a developer kit that contains various components for developers. However, some of them are not required for our purpose, such as an audio jack, an HDMI port, an Ethernet port and an SD

card socket. We detached the unnecessary components from the kit (we refer to it as *GPU board* in what fallows) and measured the energy consumption for 20 minutes hovering flight before and after attaching the GPU board. As shown in Fig. 5.17, without the GPU board, the energy consumption for the 20 minutes hovering flight is 41.45 Wh, while the energy consumption for the same flight with the GPU board is 43.70 Wh. The increase in the energy consumption from 41.45 Wh to 43.7 Wh is exactly proportional to the weight increase from 1,280 g to 1,369 g.

We separately measured the energy consumption of the GPU board when the GPU board executes the two SPACE ciphers, i.e., SPACE(8, 300) and SPACE(16, 128), to encrypt 100 MB data and runs our shuffling mechanism every minute. Nvidia Tegra TK1 is equipped with an R5C11 sense resistor of 0.005 ohm (R). The power consumption can be obtained by measuring the voltage drop at the resistor using a digital multi-meter. The voltage drops (V_d) when the board is running the encryption on the GPU is approximately 2.1 mV, while the voltage drops when the board is running the encryption using purely on the CPU is 3.0 mV. The power consumption can be calculated by $P{=}IV{=}(V_d/R)\times V_{DC}$, where V_{DC} is the 12V DC input of Nvidia Tegra TK1. The power required for the encryption using the GPU is 5.04 W, while the power required for the encryption using the CPU is 7.2 W. Encrypting 100 MB data using SPACE(8, 300) and SPACE(16, 128) on CPU are approximately 0.62 Wh and 0.42 Wh, respectively. However, encrypting 100 MB data using SPACE(8, 300) and SPACE(16, 128) on GPU are approximately 0.05 Wh and 0.10 Wh, respectively.

The energy consumption for the encryption is relatively small compared to the energy consumption for the flight. However, if a drone is already equipped with a GPU that is integrated with the main board of the drone, we believe that the weight increase due to the GPU can be minimized. Also, if the drone requires an white-box attack-resistant cipher and the amount of data to be en-/decrypted is large, we believe that the use of the GPU can reduce the energy consumption, and thus increase the drone operation time.

5.8 Summary

In this chapter, we propose a secure shuffling mechanism to enhance a white-box block cipher with dynamics in drone applications. Our shuffling protocol can be safely executed in the white-box environment since no short secret key is used by a drone during the protocol. We have proven that our shuffling protocol makes it hard for a white-box attacker to successfully encrypt/decrypt any plaintext/ciphertext even if the attacker has the knowledge of the entire look-up table. Through the experiments using a GPU-enabled SoC, we have shown the practicality of the white-box block cipher with our shuffling protocol and identified the most efficient usage the CPU and the GPU according to the number of blocks.

6 CONCLUSIONS

This dissertation addresses four notable challenges for secure drone applications: 1) how to efficiently enable secure communications between drones and sensors, 2) how drones securely locate sensors, 3) how to protect data collected by drones and detect drone software modifications, and 4) how to protect the confidentiality of secret keys in a white-box environment. The research contributions of this dissertation are summarized as follows:

- We propose an efficient CertificateLess Signcryption Tag Key Encapsulation Mechanism (eCLSC-TKEM) which 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 communications, 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 communications, we propose 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 which 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 GPU and show their GPU-accelerated performance.

- We introduce two kinds of known sensor position attacks: Aligned-Beacon-Position (ABP) attack and inside attack. The ABP attack can easily distort sensor position estimates by exploiting the fact that benign beacon nodes are usually aligned in a line. To protect against those attacks, we introduce two

defense schemes. First, we propose a novel beacon placement strategy to protect against ABP attacks. Second, we propose a new filtering technique that can filter out malicious location references introduced by inside attacks. Finally, we propose a localization algorithm improved with respect to accuracy and efficiency compared to the state-of-the-arts. We evaluate the impact of the two known sensor position attacks on the existing algorithms and the performance of our algorithm by simulation and test-bed experiments.

- We propose an attestation technique that fills up free memory spaces with data repositories. Data repositories consist of pseudo-random numbers that are also used to encrypt collected data. We also propose a group attestation scheme to efficiently verify the software integrity of multiple drones. Finally, to prevent secret keys from being leaked, we utilize a technique that converts short secret keys into large look-up tables. This technique prevents attackers from abusing free space in the data memory by filling up the space with the look-up tables.

- We propose a look-up table shuffling mechanism for dynamic white-box cryptography in the context of drone applications and provide its security analysis. Due to the dynamics, even if a remote attacker is able to see any part of the drones memory through malware, it is hard for the attacker to determine the positions of the table entries, and thus to decrypt/encrypt ciphertexts/plaintexts. To the best of our knowledge, our proposal is the first to address the problem of making a white-box block cipher dynamic within the white-box environment. Also, we are the first to use the dynamic white-box block cipher for secure communications for unmanned vehicles. We also show the practical applicability of the white-box block cipher with our shuffling mechanism by implementing it on Nvidia Tegra K1, which is a GPU-enabled SoC used in several modern drones. We show that the encryption/decryption performance is significantly boosted by GPU-acceleration when the block size of a message is large. Our shuffling

mechanism does not require additional memory space in drones and is executed by drones within a reasonable time.

REFERENCES

REFERENCES

[1] DEADrones. http://www.dea-drones.com, 2016.

[2] PrecisionHawk. http://www.precisionhawk.com, 2016.

[3] Jerome P Lynch and Kenneth J Loh. A summary review of wireless sensors and sensor networks for structural health monitoring. *Shock and Vibration Digest*, 38(2):91–130, 2006.

[4] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *IEEE S&P*, 2003.

[5] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *ACM CCS*, 2003.

[6] An Liu and Peng Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *IPSN*, 2008.

[7] Sk. Md. Mizanur Rahman and Khalil El-Khatib. Private key agreement and secure communication for heterogeneous sensor networks. *J. Parallel Distrib. Comput.*, 70(8):858–870, August 2010.

[8] Xing Zhang, Jingsha He, and Qian Wei. Eddk: Energy-efficient distributed deterministic key management for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2011.

[9] Kakali Chatterjee, Asok De, and Daya Gupta. An improved id-based key management scheme in wireless sensor network. In *Advances in Swarm Intelligence, LNCS*, volume 7332. Springer, 2012.

[10] Kexiong (Curtis) Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. All your GPS are belong to us: Towards stealthy manipulation of road navigation systems. In *USENIX Security 18*, pages 1527–1544, Baltimore, MD, 2018. USENIX Association.

[11] Junsung Cho, Jaegwan Yu, Sanghak Oh, Jungwoo Ryoo, JaeSeung Song, and Hyoungshick Kim. Wrong siren! a location spoofing attack on indoor positioning systems: The starbucks case study. *Comm. Mag.*, 55(3):132–137, March 2017.

[12] Nils Ole Tippenhauer, Kasper Bonne Rasmussen, Christina Pöpper, and Srdjan Čapkun. Attacks on public wlan-based positioning systems. In *MobiSys '09*, pages 29–40, New York, NY, USA, 2009. ACM.

[13] Skyhook. https://www.skyhook.com.

[14] Trusted Platform Module. http://www.trustedcomputinggroup.org/, 2017.

[15] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Scuba: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security*, 2006.

[16] Arvind Seshadri, Mark Luk, and Adrian Perrig. Sake: Software attestation for key establishment in sensor networks. In *Distributed Computing in Sensor Systems: 4th IEEE International Conference*, 2008.

[17] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: software-based attestation for embedded devices. In *IEEE S&P*, 2004.

[18] Mark Shaneck, Karthikeyan Mahadevan, Vishal Kher, and Yongdae Kim. Remote software-based attestation for wireless sensors. In *Security and Privacy in Ad-hoc and Sensor Networks*, pages 27–41. Springer, 2005.

[19] Yi Yang et al. Distributed software-based attestation for node compromise detection in sensor networks. In *IEEE SRDS*, 2007.

[20] Young-Geun Choi, Jeonil Kang, and DaeHun Nyang. Proactive code verification protocol in wireless sensor network. In *Computational Science and Its Applications ICCSA*. Springer, 2007.

[21] Claude Castelluccia et al. On the difficulty of software-based attestation of embedded devices. In *ACM CCS*, 2009.

[22] Adi Shamir and Nicko van Someren. *Playing 'Hide and Seek' with Stored Keys*, pages 118–124. Springer, 1999.

[23] J. Alex Halderman et al. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, May 2009.

[24] Tim Kerins and Klaus Kursawe. A cautionary note on weak implementations of block ciphers, 2006.

[25] Fernando Trujano, Benjamin Chan, Greg Beams, and Reece Rivera. Security analysis of dji phantom 3 standard, 2016.

[26] Rahul Sasi. Maldrone, 2015.

[27] Nils Rodday. Hacking a professional drone. *Black Hat Asia*, 2016.

[28] Oleg Petrovsky. Attack on the drones: security vulnerabilities of unmanned aerial vehicles. *Virus bulletin*, 2015.

[29] Jonathan Andersson. Attacking dsmx spread spectrum frequency hopping drone remote control with sdr. *PacSec*, 2016.

[30] Ang Cui, Michael Costello, and Salvatore J. Stolfo. When firmware modifications attack: A case study of embedded exploitation. In *20th Annual Network and Distributed System Security Symposium, NDSS*, 2013.

[31] Hex-rays, 2015.

[32] Boomerang Decompiler, 2012.

[33] Open Source Robotics Foundation. Robot operating system (ros), 2018.

[34] Paparazzi Project, 2012.

[35] ArduPilot, 2019.

[36] Stanley Chow et al. White-box cryptography and an aes implementation. In *SAC, LNCS*, volume 2595, pages 250–270. Springer, 2003.

[37] ARM. http://www.arm.com/products/processors/technologies/trustzone/, 2017.

[38] J. Y. Park, J. N. Kim, J. D. Lim, and D. G. Han. A whitebox cryptography application for mobile device security against whitebox attacks - how to apply wbc on mobile device. In *2014 International Conference on IT Convergence and Security*, pages 1–5, Oct 2014.

[39] Yang Shi, Wujing Wei, Zongjian He, and Hongfei Fan. An ultra-lightweight white-box encryption scheme for securing resource-constrained iot devices. In *ACSAC*, pages 16–29. ACM, 2016.

[40] Intertrust. whitecryption.

[41] Stanley Chow et al. A white-box des implementation for drm applications. In *DRM*, LNCS. Springer, 2003.

[42] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the asasa structure: Black-box, white-box, and public-key (extended abstract). In *Advances in Cryptology, ASIACRYPT*, volume 8873 of *LNCS*, pages 63–84. Springer, 2014.

[43] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In *ACM CCS*, pages 1058–1069, 2015.

[44] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In *ASIACRYPT 2016*, pages 126–158. Springer, 2016.

[45] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Crypt-analysis of a white box aes implementation. In *Selected Areas in Cryptography*, pages 227–240. Springer, 2005.

[46] Brecht Wyseur et al. Cryptanalysis of white-box des implementations with arbitrary external encodings. In *SAC, LNCS*, volume 4876. Springer, 2007.

[47] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box aes implementation. In *INDOCRYPT*, volume 6498 of *LNCS*, pages 292–310. Springer, 2010.

[48] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the xiao–lai white-box aes implementation. In *Selected Areas in Cryptography*, pages 34–49. Springer, 2013.

[49] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box aes implementation. In *Revised Selected Papers on Selected Areas in Cryptography*, volume 8282, pages 265–285. Springer, 2014.

[50] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*. Springer, 2001.

[51] Li Zhou et al. Supporting secure communication and data collection in mobile sensor networks. In *INFOCOM*, 2006.

[52] Hui Song et al. Least privilege and privilege deprivation: Toward tolerating mobile sink compromises in wireless sensor networks. *ACM TOSN'08*, 2008.

[53] A. Rasheed et al. Secure data collection scheme in wireless sensor network with mobile sink. In *IEEE NCA*, 2008.

[54] A. Rasheed. et al. The three-tier security scheme in wireless sensor networks with mobile sinks. *IEEE TPDS*, 2012.

[55] C. Schurgers et al. Optimizing sensor networks in the energy-latency-density design space. *IEEE TMC*, 2002.

[56] W. Zhao et al. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *ACM MobiHoc*, 2004.

[57] Guomin Yang and Chik-How Tan. Strongly secure certificateless key exchange without pairing. In *ACM ASIACCS*, 2011.

[58] Haiyan Sun, Qiaoyan Wen, Hua Zhang, and Zhengping Jin. A novel pairing-free certificateless authenticated key agreement protocol with provable security. *Frontiers of Computer Science*, 7(4), 2013.

[59] S. Sharmila Deva Selvi, S. Sree Vivek, and C. Pandu Rangan. Certificateless kem and hybrid signcryption schemes revisited. In *Information Security, Practice and Experience*, pages 294–307, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[60] Seung-Hyun Seo, Jongho Won, and Elisa Bertino. pclsc-tkem: a pairing-free certificateless signcryption-tag key encapsulation mechanism for a privacy-preserving iot. *Transactions on Data Privacy*, 2016.

[61] Jongho Won, Seung-Hyun Seo, and Elisa Bertino. A secure communication protocol for drones and smart objects. In *ACM ASIACCS*, 2015.

[62] SattamS. Al-Riyami and KennethG. Paterson. Certificateless public key cryptography. In *ASIACRYPT*. Springer, 2003.

[63] K. A. Shim. A survey of public-key cryptographic primitives in wireless sensor networks. *IEEE Communications Surveys Tutorials*, 18(1):577–601, Firstquarter 2016.

[64] Debiao He et al. A pairing-free certificateless authenticated key agreement protocol. *Int. Journal of Comm. Sys.*, 2012.

[65] Manman Geng and Futai Zhang. Provably secure certificateless two-party authenticated key agreement protocol without pairing. In *CIS '09*, 2009.

[66] Fagen Li, Masaaki Shirase, and Tsuyoshi Takagi. Certificateless hybrid signcryption. In *Information Security Practice and Experience*, pages 112–123, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[67] Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In *PKC*, 2002.

[68] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In *PKC*, 2003.

[69] N. P. Smart. Efficient key encapsulation to multiple parties. In *Security in Communication Networks (SCN)*, 2004.

[70] M. Barbosa and P. Farshim. Efficient identity-based key encapsulation to multiple parties. In *IMACC*, 2005.

[71] Alexandre Pinto, Bertram Poettering, and Jacob C.N. Schuldt. Multi-recipient encryption, revisited. In *ACM ASIACCS*, 2014.

[72] Soufiene Ben Othman, Abdelbasset Trad, Hani Alzaid, and Habib Youssef. Performance evaluation of ec-elgamal encryption algorithm for wireless sensor networks. In *Wireless Mobile Communication and Healthcare*, pages 271–285, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[73] O. Ugus, D. Westhoff, R. Laue, A. Shoufan, and S. Huss. Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks. In *WESS*, 2007.

[74] Samuel Antão, Jean-Claude Bajard, and Leonel Sousa. Rns-based elliptic curve point multiplication for massive parallel architectures. *The Computer Journal*, 2011.

[75] Joppe W. Bos. Low-latency elliptic curve scalar multiplication. *Intl. Journal of Parallel Programming*, 40, 2012.

[76] Shujie Cui et al. High-speed elliptic curve cryptography on the nvidia gt200 graphics processing unit. In *ISPEC*, 2014.

[77] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE TIT*, 1985.

[78] Certivox. Miracl cryptographic sdk, 2014.

[79] Nvidia. Jetson tk1, http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html, 2015.

[80] J. Won, S. Seo, and E. Bertino. Certificateless cryptographic protocols for efficient drone-based smart city applications. *IEEE Access*, 5:3721–3749, 2017.

[81] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *ACM SenSys*, 2004.

[82] Claus-Peter Schnorr and Markus Jakobsson. Security of signed elgamal encryption. In *ASIACRYPT*, 2000.

[83] Parrot. http://ardrone2.parrot.com, 2013.

[84] Brad Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *ACM MobiCom*, 2000.

[85] Donggang Liu and Peng Ning. Location-based pairwise key establishments for static sensor networks. In *ACM SASN*, 2003.

[86] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, October 2010.

[87] Lee Rainie Janna Anderson. The internet of things will thrive by 2025.

[88] D. Singh, G. Tripathi, and A.J. Jara. A survey of internet-of-things: Future vision, architecture, challenges and services. In *IEEE World Forum on Internet of Things*, 2014.

[89] S. Capkun and J.-P. Hubaux. Secure positioning of wireless devices with application to sensor networks. In *INFOCOM*, 2005.

[90] L. Lazos, R. Poovendran, and S. Capkun. Rope: robust position estimation in wireless sensor networks. In *IPSN*, 2005.

[91] S. Capkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE JSAC*, 24, Feb 2006.

[92] Sheng Zhong, M. Jadliwala, S. Upadhyaya, and Chunming Qiao. Towards a theory of robust localization against malicious beacon nodes. In *IEEE INFOCOM*, 2008.

[93] Yingpei Zeng, Jiannong Cao, Shigeng Zhang, Shanqing Guo, and Li Xie. Pollution attack: A new attack against localization in wireless sensor networks. In *IEEE WCNC*, 2009.

[94] Zang Li, W. Trappe, Y. Zhang, and B. Nath. Robust statistical methods for securing wireless localization in sensor networks. In *IPSN*, 2005.

[95] Donggang Liu, Peng Ning, An Liu, Cliff Wang, and Wenliang Kevin Du. Attack-resistant location estimation in wireless sensor networks. *ACM TISSEC*, 11(4):22:1–22:39, July 2008.

[96] C. Wang, An Liu, and Peng Ning. Cluster-based minimum mean square estimation for secure and resilient localization in wireless sensor networks. In *WASA*, 2007.

[97] Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of MobiCom*, 2001.

[98] Telosb, http://www.memsic.com/userfiles/files/datasheets/wsn/telosb_datasheet.pdf.

[99] Micaz, http://www.memsic.com/userfiles/files/datasheets/wsn/micaz_datasheet-t.pdf.

[100] C. H. Ou and K. F. Ssu. Sensor position determination with flying anchors in three-dimensional wireless sensor networks. *IEEE Transactions on Mobile Computing*, 7(9):1084–1097, Sept 2008.

[101] Cristina M. Pinotti, Francesco Betti Sorbelli, Pericle Perazzo, and Gianluca Dini. Localization with guaranteed bound on the position error using a drone. In *MobiWac*, New York, NY, USA, 2016. ACM.

[102] Ting Zhang, Jingsha He, and Hong Yu. Secure localization in wireless sensor networks with mobile beacons. *International Journal of Distributed Sensor Networks*, 2012.

[103] D. Liu, Peng Ning, and Wenliang Du. Detecting malicious beacon nodes for secure location discovery in wireless sensor networks. In *IEEE ICDCS*, 2005.

[104] Xingfu Wang, Lei Qian, and Haiqing Jiang. Tolerant majority-colluding attacks for secure localization in wireless sensor networks. In *WiCom*, 2009.

[105] B.H. Cheng, R.E. Hudson, F. Lorenzelli, L. Vandenberghe, and K. Yao. Distributed gauss-newton method for node loclaization in wireless sensor networks. In *IEEE SPAWC*, 2005.

[106] R. Garg, A.L. Varna, and Min Wu. Gradient descent approach for secure localization in resource constrained wireless sensor networks. In *IEEE Acoustics Speech and Signal Processing*, 2010.

[107] R. Huang and G. V. Zaruba. Static path planning for mobile beacons to localize sensor networks. In *PerCom Workshops*, 2007.

[108] Atmel, http://www.atmel.com/images/doc8228.pdf.

[109] F. Anjum, S. Pandey, and P. Agrawal. Secure localization in sensor networks using transmission range variation. In *IEEE MASS*, 2005.

[110] S. Capkun, Kasper Bonne Rasmussen, M. Cagalj, and M. Srivastava. Secure location verification with hidden and mobile base stations. *Mobile Computing, IEEE Transactions on*, 7(4):470–483, April 2008.

[111] H. Lim, G. Ghinita, E. Bertino, and M. Kantarcioglu. A game-theoretic approach for high-assurance of data trustworthiness in sensor networks. In *IEEE ICDE*, pages 1192–1203, April 2012.

[112] W. Du, L. Fang, and P. Ningi. Lad: localization anomaly detection for wireless sensor networks. In *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005.

[113] A. Srinivasan, J. Teitelbaum, and Jie Wu. Drbts: Distributed reputation-based beacon trust system. In *IEEE DASC*, 2006.

[114] ublox neo-6m, http://www.u-blox.com.

[115] ns2, http://www.isi.edu/nsnam/ns/.

[116] Arvind Seshadri et al. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *ACM SOSP*, 2005.

[117] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006:468.

[118] Mohamed Karroumi. Protecting white-box aes with dual ciphers. In *ICISC*, pages 278–291. Springer, 2011.

[119] Yaying Xiao and Xuejia Lai. A secure implementation of white-box aes. In *Computer Science and its Applications*, pages 1–6, 2009.

[120] Wil Michiels et al. Cryptanalysis of a generic class of white-box implementations. In *SAC, LNCS*, volume 5381. Springer, 2009.

[121] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In *ACM CCS*, New York, NY, USA, 2015. ACM.

[122] N. Asokan et al. Seda: Scalable embedded device attestation. In *ACM CCS*, 2015.

[123] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. Darpa: Device attestation resilient to physical attacks. In *ACM WiSec*, 2016.

[124] Niels Provos and Peter Honeyman. Hide and seek: an introduction to steganography. *IEEE Security Privacy*, 1:32–44, May 2003.

[125] F. Bastien. Simple steganalysis suite, 2015.

[126] P. Forczmanski and M. Wegrzyn. Open virtual steganographic laboratory. in international conference on advanced computer systems, 2009.

[127] N. Provos and P. Honeyman. Detecting steganographic content on the internet. tech. rep., center for information technology integration university of michigan, 2001.

[128] Omed S. Khalind et al. A study on the false positive rate of stegdetect. *Digital Investigation*, 9:235 – 245, 2013.

[129] Raspberry Pi 2. https://www.raspberrypi.org/, 2015.

[130] Kespry. http://www.kespry.com, 2019.

[131] Parrot. http://www.parrot.com, 2019.

[132] R. Hossain, S. Magierowski, and G. G. Messier. Gpu enhanced path finding for an unmanned aerial vehicle. In *IEEE Intl. Parallel Distributed Processing Symposium Workshops*, pages 1285–1293, 2014.

[133] Chad C. Haddal and Jeremiah Gertler. Homeland security: Unmanned aerial vehicles and border surveillance. *Congressional Research Service*, Jul 2010. Rept. RS21698, Washington, D.C.

[134] Fortune. California vineyard using drones, 2016.

[135] Amazon prime air, http://www.amazon.com/b?node=8037720011.

[136] Mirmojtaba Gharibi, Raouf Boutaba, and Steven Lake Waslander. Internet of drones. *IEEE Access*, 4:1148–1162, 2016.

[137] Jong-Yeon Park, Okyeon Yi, and Ji-Sun Choi. Methods for practical whitebox cryptography. In *2010 International Conference on Information and Communication Technology Convergence*, pages 474–479, Nov 2010.

[138] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Decomposing the asasa block cipher construction. Cryptology ePrint Archive, Report 2015/507, 2015.

[139] Henri Gilbert, Jérôme Plût, and Joana Treger. Key-recovery attack on the asasa cryptosystem with expanding s-boxes. Cryptology ePrint Archive, Report 2015/567, 2015.

[140] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. Cryptology ePrint Archive, Report 2015/753, 2015.

[141] Qinjian Li, Chengwen Zhong, Kaiyong Zhao, Xinxin Mei, and Xiaowen Chu. Implementation and analysis of aes encryption on gpu. In *IEEE Intl. conf. on High Performance Computing and Communication*, pages 843–848, June 2012.

[142] Johannes Gilger, Johannes Barnickel, and Ulrike Meyer. Gpu-acceleration of block ciphers in the openssl cryptographic library. In *Proceedings of the 15th Intl. conf. on Information Security*, pages 338–353. Springer, 2012.

[143] Ankush Singla, Anand Mudgerikar, Ioannis Papapanagiotou, and Attila A. Yavuz. Haa: Hardware-accelerated authentication for internet of things in mission critical vehicular networks. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 1298–1304, 2015.

[144] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *ACM SIGPLAN*, pages 190–200, 2005.

[145] Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *ACM SIGPLAN*, pages 89–100, 2007.

[146] Tim Kerins and Klaus Kursawe. A cautionary note on weak implementations of block ciphers. In *In 1st Benelux Workshop on Information and System Security*, page 12. WISSec 2006, 2006.

[147] CUDA. https://developer.nvidia.com/cuda-zone, 2015.

[148] Dongwoon Jeon, Doo-Hyun Kim, Young-Guk Ha, and Vladimir Tyan. Image processing acceleration for intelligent unmanned aerial vehicle on mobile gpu. *Soft Computing*, 20:1713–1720, 2016.

[149] A. Benini, M. J. Rutherford, and K. P. Valavanis. Real-time, gpu-based pose estimation of a uav for autonomous takeoff and landing. In *IEEE ICRA*, pages 3463–3470, 2016.

[150] Gunnar Blom, Lars Holst, and Dennis Sandell. Chaper 7.5. coupon collecting i. In *Problems and Snapshots from the World of Probability*, pages 85–87. Springer, 1994.

[151] Donald Knuth. *Semi-numerical algorithms. The Art of Computer Programming.* Addison Wesley, 1969.

[152] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy*, pages 272–282, May 2004.

[153] Frederik Armknecht, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. A security framework for the analysis and design of software attestation. In *ACM CCS*, pages 1–12, New York, USA, 2013.

[154] Lenore Blum, Manuel Blum, and Michael Shub. *Comparison of Two Pseudo-Random Number Generators.* Advances in Cryptology, Springer, 1983.

[155] David Wells. harmonic series. In *The Penguin Dictionary of Curious and Interesting Numbers.* 1986.

[156] Zack Smith. Memory bandwidth benchmark, 2018.

[157] Mihir Bellare and Bennet Yee. Forward-security in private-key cryptography. In *Proceedings of the 2003 RSA Conference on The Cryptographers' Track*, CT-RSA'03, pages 1–18, 2003.

[158] M. Laner, P. Svoboda, P. Romirer-Maierhofer, N. Nikaein, F. Ricciato, and M. Rupp. A comparison between one-way delays in operating hspa and lte networks. In *Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2012.

[159] T. Blajić, D. Nogulić, and M. Družijanić. Latency improvements in 3g long term evolution. In *MIPRO CTI*, 2007.

[160] DJI. Dji phantom3, 2019.

[161] DJI. Dji developer sdk, https://developer.dji.com/, 2019.

VITA

VITA

Jongho Won was born in Seoul, Korea received the B.S.(2006) and M.S.(2008) degrees from Seoul National University, Korea. He was a research engineer of LG Electronics Advanced Research Institute for 3 years. He joined the Department of Computer Science in Purdue since 2011. He has had the opportunity to work at Microsoft Research as an Intern for the Farmbeat project. He was an intern in VMWare and is working for the VMWare IoT project. His broad research interests are in areas related to security and privacy in various wireless networks.