DroneKey: A Drone-Aided Group-Key Generation Scheme for Large-Scale IoT Networks

Dianqi Han
dqhan@asu.edu
Arizona State University
Tempe, Arizona, USA

Ang Li
anglee@asu.edu
Arizona State University
Tempe, Arizona, USA

Jiawei Li
jwli@asu.edu
Arizona State University
Tempe, Arizona, USA

Yan Zhang
yanzhangyz@asu.edu
Arizona State University
Tempe, Arizona, USA

Tao Li
tli6@iupui.edu
Indiana University–Purdue University Indianapolis
Indianapolis, Indiana, USA

Yanchao Zhang
yczhang@asu.edu
Arizona State University
Tempe, Arizona, USA

ABSTRACT

The Internet of Things (IoT) networks are finding massive applications in mission-critical contexts. A group key is needed to encrypt and authenticate broadcast/multicast messages commonly seen in large-scale wireless networks. In this paper, we propose DroneKey, a novel drone-aided PHY-based Group-Key Generation (GKG) scheme for large-scale IoT networks. In DroneKey, a drone is dispatched to fly along random 3D trajectories and keep broadcasting standard wireless signals to refresh the group keys in the whole network. Every IoT device receives the broadcast signals from which to extract the Channel State Information (CSI) stream which captures the dynamic variations of the individual wireless channel between the IoT device and the drone. DroneKey explores a deep-learning approach to extract the hidden correlation among the CSI streams to establish a common group key. We thoroughly evaluate DroneKey with a prototype in both indoor and outdoor environments. We show that DroneKey can achieve a high key-generation rate of 89.5 bit/sec for 10 devices in contrast to 40 bit/sec in the state-of-art prior work. In addition, DroneKey is much more scalable and can support 100 devices in contrast to 10 nodes in the state-of-art prior work with comparable key-generate rates.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**.

KEYWORDS

Group-key generation, large-scale IoT network, drone

ACM Reference Format:
Dianqi Han, Ang Li, Jiawei Li, Yan Zhang, Tao Li, and Yanchao Zhang. 2021. DroneKey: A Drone-Aided Group-Key Generation Scheme for Large-Scale IoT Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3460120.3484789

1 INTRODUCTION

The Internet of Things (IoT) networks are finding massive applications in mission-critical contexts, such as critical infrastructure monitoring, border control and protection, military reconnaissance, and surveillance. For example, an IoT network containing thousands of devices is being used to improve the management effectiveness of the Pendjari National Park over 2,755 km^2 [25]. Intelligent factories, such as the Tesla factory, use IoT networks with thousands of or more devices in a large factory area to facilitate manufacturing process workflows [15, 34]. Moreover, the military is actively exploring IoT networks for battlefield reconnaissance and border control [13, 26]. These IoT networks are all large-scale in terms of both the coverage area and device count. In this paper, we consider such a large mission-critical IoT network formed by many distributed groups, each comprising many densely deployed nearby IoT devices. These devices communicate over the open wireless channel and frequently exchange broadcast/multicast messages with group peers. So it is necessary to explore a unique group key to secure broadcast/multicast messages in each group. This paper focuses on investigating sound schemes to establish/update the group key of each individual group in such a large-scale IoT network.

Designing group-key generation (GKG) schemes for large-scale mission-critical IoT networks faces some essential challenges. First, each group may contain several tens to hundreds of IoT nodes in a densely deployed network. So the GKG scheme should be highly *scalable* to an arbitrary group size. Second, the IoT network can be totally unattended in remote non-accessible areas, last very long time, and transmit time-sensitive information. So the GKG scheme should be *fast* in quickly updating all the group keys, which can translate into the requirement for a high group-key generation rate. Third, but not the last, the IoT devices are mostly likely to be battery-powered. So the GKG scheme should be *efficient* with low communication and computation overhead.

Key generation based on wireless physical-layer (PHY) channel characteristics has received tremendous attention as alternative methods to cryptographic techniques. Most PHY-based methods such as [17, 21, 23] target pairwise-key generation between two wireless devices. In contrast to cryptographic techniques based

	group size	key-generation rate	randomness test	evaluation in real environments
DroneKey	10	89.5 bit/sec	passed	yes
	50	50.0 bit/sec		
	100	36.2 bit/sec		
Liu et al.[22]	10	40 bit/sec	passed	yes
Wei et al.[39]	3	80 bit/sec	not provided	yes
Thai et al.[35]	4	12 bit/sec	not provided	no
Xu et al. [42]	4	9.4 bit/sec	not provided	no

Table 1: The comparison between DroneKey and representative prior work.

on computational hardness assumptions, these methods rely on *channel reciprocity* which refers to that two wireless devices can observe highly correlated variations of the wireless channel between them. Such correlated channel variations can be used as a common randomness factor for extracting a pairwise key between the two devices. As long as the eavesdropper is at least half wavelength away from legitimate devices, it cannot observe the same channel variations for inferring the pairwise key [28, 37].

PHY-based GKG has also been studied as alternative methods to cryptographic group-key generation. The most intuitive solution is to generate many PHY-based pairwise keys to spread a group key across the group. The number of involved pairwise keys must be greater than the number of devices and can be as large as hundreds in our context. To defeat sophisticated attacks, each group key in a large-scale mission-critical IoT network must be frequently updated, so are the involved massive pairwise keys. The resulting computation and communication overhead is very high, making this solution inefficient and impractical. Efficient PHY-based GKG is challenging because the wireless channel variations between any two devices cannot be measured by any other device more than half-wavelength away from both devices due to channel reciprocity. It is nearly impossible to find a common wireless channel for many distributed devices to extract a group key. Some recent schemes [16, 22, 35] try different ways to spread the measurements of selected channels to the entire group of devices. These schemes all require each device to transmit at least one probe packet, and all the packet transmissions must be finished within the short channel-coherence time which refers to the time duration in which the channel condition is considered non-varying. So these schemes are not scalable to a large group. For example, the largest group size reported in [22] is only 10. In addition, the group-key generation rate highly depends on the channel randomness and is often not satisfactory. The technique in [22] is the only one we beware that has passed the NIST randomness test and been evaluated in real environments. Its group-key generation rate is about 40 bit/sec according to their experimental setup with only 10 devices.

In this paper, we propose **DroneKey**, a novel drone-aided PHY-based GKG scheme for large-scale mission-critical IoT networks, which is highly scalable, fast, and efficient. DroneKey explores drones which are increasingly popular and widely expected to be prevalent equipment in mission-critical IoT systems [29]. Whenever group-key establishment/rekeying is needed, one or a few drones are dispatched to fly over the IoT network area and perform random 3D movements while broadcasting wireless signals to each group. Most drones have embedded WiFi transceivers, which can be used for signal broadcasting. For a drone without one, a lightweight battery-powered WiFi router attached to the drone can perform the broadcasting task [19]. So our scheme is practical in terms of hardware requirements. A group of devices can receive the same broadcast signals and each extract a Channel State Information (CSI) stream. A device's CSI stream characterizes the dynamic variations of the unique wireless channel between the device and drone, which is mainly induced by fast drone movement. Although the CSI streams of different devices in a group depend on their respective channels with the drone, they are all correlated with the drone's trajectory and thus indirectly correlated with each other. DroneKey aims to quickly establish/refresh the group key of each individual group by mining this hidden CSI correlation.

The design of DroneKey faces two critical challenges. First, the relation among different CSI streams is highly complex and affected by many factors such as device locations, hardware features, channel shadowing and fading, and multipath signal propagation. So it is challenging to extract the hidden CSI correlation for establishing a group key in a distributed fashion. Second, since the drone's 3D trajectory is the dominating randomness factor for the group key, a powerful adversary may video-tape the drone movement to reconstruct the drone trajectory and then the group key. This trajectory-reconstruction threat is unique to DroneKey.

DroneKey adopts a deep-learning approach with an obfuscation function to address the above challenges. Within a group, one device is designated as the *group head* which can be chosen based on any sophisticated cluster-head selection algorithm in multi-hop wireless networks [18, 36]. All the non-head devices in a group are called *peer* devices. Although some existing correlation measurement algorithms can be used to measure the correlation between the head's and a peer's CSI streams, they cannot extract a common secret key because their output is just a single number between -1 and 1 indicating the correlation [3, 44]. Recent studies show that the Deep Neural Network (DNN) can capture the correlation between two signals in a more sophisticated manner. For example, Wu *et al.* explore DNN to capture the correlation between the gait observations of two wearables on the same body but at different locations to infer one gait observation from the other [41]. So we are motivated to adopt DNN in the DroneKey design.

DroneKey involves a one-time DNN training process in each group during the network-initialization phase. In particular, after IoT devices are deployed, one or a few drones are dispatched to traverse the network along random 3D trajectories while broadcasting wireless signals that can be received by all devices in each group. Each group head trains a unique DNN for each of its peer devices based on a confidential *obfuscation function*, its own CSI stream, and the CSI stream submitted by the peer device. The group head then sends the trained DNNs to the corresponding peer devices over a secure channel (Section 3.2).[1] Since the DNN of each peer device is closely tied to its relative location to the group head, different peer devices have diverse DNNs. But these DNNs are trained in a special way that each device in the same group can obtain the same output as the group key by feeding its CSI stream into its DNN. Since the obfuscation function is confidential, DroneKey guarantees that a passive eavesdropper cannot infer the group key from reconstructed CSI streams through the advanced trajectory-reconstruction attack. The training and distribution of DNNs are conducted only once during network initialization and not needed in each subsequent GKG instance. Whenever a new group key is needed, a drone is dispatched to perform 3D random movement while broadcasting wireless signals to IoT devices. Each device in a group just autonomously feeds the fresh CSI streams extracted from new drone signals into their respective DNNs to obtain a new group key without any interaction with each other.

Although DroneKey involves deep learning, the computational load for each IoT device is still lightweight. In particular, the one-time DNN training process during network initialization can be offloaded to a remote server if needed. In each subsequent GKG instance, each group head only needs to perform one matrix multiplication and a simple quantification operation to obtain the group key, and each peer device needs to conduct one DNN forward computation and a similar quantification operation to obtain the group key. Therefore, DroneKey is a feasible solution for large-scale IoT networks, which may contain many resource-constrained devices.

We prototype DroneKey and thoroughly evaluate its performance in both indoor and outdoor settings. The experimental results show that DroneKey can achieve key-generation rates above 75 bit/sec in all the evaluated scenarios, and all the generated keys can pass the NIST randomness test. Table 1 shows the brief comparisons of DroneKey with the prior work. DroneKey outperforms the state-of-the-art PHY-based scheme [21] by more than 100% in terms of the key-generation rate for networks of the same size and can achieve a compatible key-generation rate for a large-scale IoT network 10 times larger than the networks considered in previous studies. In addition, we estimate that DroneKey can update all the 256-bit group keys in a large IoT network of 20,000 devices over a 1 km×1 km area within 42 minutes with just one drone and 10 minutes with five drones. We also theoretically show that DroneKey is robust to the RF eavesdropping attack and also the drone-trajectory-reconstruction attack.

The rest of the paper is organized as follows. Section 2 reviews the background knowledge of CSI and discusses the feasibility of DroneKey. Section 3 presents the system overview and the adversary model. Sections 4, 5, and 6 illustrate the design details of DroneKey. Section 7 theoretically evaluate DroneKey's security. Section 8 experimentally evaluates DroneKey. Section 9 discusses the related work. Section 10 concludes this paper.

[1]Note that we only assume the availability of this secure channel during the network-initialization phase, which cannot be used to distribute new group keys during network operations (Section 3.2).

2 BACKGROUND AND FEASIBILITY STUDY

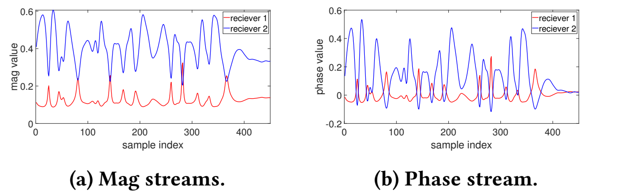
2.1 Background of CSI

The PHY wireless channel characteristic at a specific frequency can be represented by the Channel Frequency Response (CFR). Given a transmitted signal whose frequency-domain representation is $X(f,t)$, the received signal can be represented as $Y(f,t) = H(f,t)\times X(f,t)+N(f,t)$, where $H(f,t)$ is the CFR at frequency f measured at time t, and $N(f,t)$ is the noise. $H(f,t)$ is a complex value and can be represented as $H(f,t) = a(f,t)e^{2\pi\phi(f,t)j}$, where $a(f,t)$ and $\phi(f,t)$ denote the magnitude attenuation and phase shift values, respectively. For brevity, we shall abbreviate magnitude attenuation and phase shift to mag and phase, respectively.

CSI measures the CFRs of a wireless channel at the carrier frequency or multiple subcarrier frequencies. Researchers have proposed many data-aided CSI-estimation schemes that work with all prevalent wireless techniques [8, 24]. In those schemes, a predefined pilot signal which is known to the receiver is transmitted, and the receiver estimates the CSI from its received signal. By adopting the existing generic CSI-estimation schemes, DroneKey can work with all prevalent wireless techniques for IoT systems.

2.2 Feasibility Study

DroneKey depends on the premise that the CSI streams extracted from the same broadcast signals but by different receivers are correlated. We use a preliminary experiment to verify this feasibility. Our experiment uses an N210 USRP, which is attached to a DJI Matrice 100 drone, as the transmitter and two other B210 USRPs placed three meters apart as two receivers. All the experiments in this paper use this DJI drone, so we omit the drone model hereafter. We fly the drone back and forth between two receivers while the attached USRP keeps broadcasting WiFi packets. The two receivers keep extracting CSI from each received packet. The two resulting CSI streams are shown in Fig. 1. For a clear illustration, we process the raw streams with methods demonstrated in Section 4 and only show the processed streams of the 6th subcarrier here. There are obvious correlations between the two mag streams and also the two phase streams. So it is quite feasible to extract a common key from the two CSI streams.



(a) Mag streams. **(b) Phase stream.**

Figure 1: The correlated CSI streams.

In this paper, DroneKey assumes wireless techniques based on orthogonal frequency-division multiplexing (OFDM) which is implemented in all WiFi standards post 802.11b. DroneKey can be easily extended to all other prevalent wireless techniques because they all support the extraction of CSI. The CSI of OFDM-based wireless networks contains information about multiple subcarriers at different frequencies. In our scenario, the drone movement is the dominating reason for the CSI changes, and the CSI streams of different subcarriers at the same device are highly correlated. So DroneKey only uses one subcarrier for group-key generation to ensure sufficient key randomness. Hereafter, the term "CSI" refers to the CFR of the selected subcarrier, and each CSI sample is represented with a mag value and a phase value.

3 SYSTEM OVERVIEW AND ADVERSARY MODEL

3.1 System Model

We consider a large-scale IoT network containing thousands of static IoT devices deployed over a large area for a mission-critical operation. The network is divided into many distributed groups with each containing several tens to hundreds of nearby devices. Once the network is deployed, it can be fully unattended during the long network lifetime.

A drone denoted by $\mathcal{D}$ is periodically dispatched to traverse the entire network along planned routes to refresh the group key of each device group in the network. Our subsequent illustrations focus on one group of devices as shown in Fig. 2. DroneKey can support an arbitrary number of devices in each group. Without loss of generality, we assume that the IoT network adopts OFDM-based WiFi for group communications on the 2.4 GHz band as in our experimental evaluations, but DroneKey can be easily extended to any other wireless technique (e.g., Bluetooth, Zigbee, and Lora) because CSI measurement is universally supported.

One device in the group is selected as the group head with any sophisticated cluster-head selection algorithm in multi-hop wireless networks, and the rest devices in the group are called peer devices. The group head is denoted by $\mathcal{H}$, and the ith peer device is denoted by $\mathcal{P}_i$. $\mathcal{H}$ needs to be within the communication range of all the peer devices, but the peer devices do not need. In addition, we assume that the drone has a large communication range that covers the entire group.

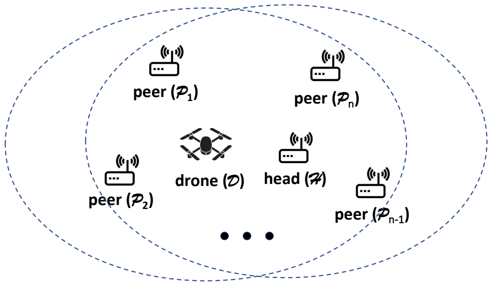


Figure 2: The system model of DroneKey.

3.2 DroneKey Workflow

DroneKey consists of a one-time initialization stage and subsequent group-key generation stages, as shown in Fig. 3. The black solid arrows indicate the timelines of the corresponding devices, and the dashed arrows represent the data exchanges between devices. The blue color indicates that a data transmission is secured with a pairwise secret key, and the red color means that a data transmission is in plain text. The workflows of all the peer devices are the same. So we use the first peer device $\mathcal{P}_1$ as the example to demonstrate our scheme hereafter.

In the initialization stage, $\mathcal{D}$ flies randomly within a predefined area while continuously broadcasting WiFi packets. $\mathcal{P}_1$ and $\mathcal{H}$ each extracts a CSI stream from the broadcast signal and processes the stream with methods detailed in Section 4.2. We denote these two processed CSI streams of $\mathcal{P}_1$ and $\mathcal{H}$ by C_1^R and C_H^R, respectively. After obtaining C_H^R, $\mathcal{H}$ acquires C_1^R from $\mathcal{P}_1$, generates a training dataset from C_1^R and C_H^R, and finally trains a DNN. The training process involves an obfuscation function that can enhance DroneKey's security. This DNN is for the group-key generation at $\mathcal{P}_1$, and we denote it by G_1. More details regarding the dataset generation and DNN training are given in Section 5. Moreover, $\mathcal{H}$ determines the numbers of quantification bins from C_H^R and C_1^R, which is critical for the group-key generation stage; and the details are demonstrated in Section 6.2. Finally, $\mathcal{H}$ sends G_1 and the quantification-bin numbers to $\mathcal{P}_1$. The transmissions of C_1^R, G_1, and the quantification-bin number are secured with the pairwise key that can be established with any cryptographic or PHY-based method [9, 10, 21].

There are two remarks to make for network initialization. First, the training and distribution of DNNs is a one-time process. Second, we only assume the security of pairwise keys in the very short initialization phase to obviate the need for secure pairwise-key update schemes. This means that such pairwise keys are unavailable in subsequent network operations for securely delivering a new group key randomly selected by the group head to its peer devices.

In each subsequent key-generation stage, $\mathcal{P}_1$ acquires the group key K by generating its own primitive key and adjusting it according to the Error Correction Code (ECC) broadcast by $\mathcal{H}$. Particularly, $\mathcal{D}$ broadcasts WiFi packets while moving randomly, and $\mathcal{P}_1$ and $\mathcal{H}$ obtain their processed CSI streams from the broadcast signals. We denote these two CSI streams of $\mathcal{P}_1$ and $\mathcal{H}$ by C_1^G and C_H^G, respectively. Then $\mathcal{H}$ generates K from C_H^G and broadcasts the ECC of K. Next, $\mathcal{P}_1$ uses C_1^G as the inputs to G_1 and generates its primitive group key K_1 from G_1's output. Due to the impacts of the ambient noise, hardware flaws, and DNN estimation errors, K_1 may not be identical to K. So $\mathcal{P}_1$ adjusts K_1 according to the ECC broadcast by $\mathcal{H}$ in the final reconciliation step. In the key-generation stage, $\mathcal{H}$ and all the peer devices must extract their CSI streams simultaneously so that their CSI streams are correlated and can be used to generate a common secret key. Since we use the indexes of WiFi packets to synchronize different devices' CSI streams, $\mathcal{H}$ and the peer devices do not need to have synchronous clocks, which is also an advantage of our scheme over prior work. DroneKey can renew the group key as needed by repeating the group-key generation process.

3.3 Adversary Model

Like all prior PHY-based GKG schemes [16, 22, 35], we focus on establishing/updating group keys among randomly deployed wireless devices to secure wireless broadcast/multicast messages. Without loss of generality, we assume that the first peer device $\mathcal{P}_1$ is the attacker's target device, and the attacker—denoted by $\mathcal{A}$—aims to obtain the group key K. We assume a very powerful $\mathcal{A}$ with the following capabilities. First, $\mathcal{A}$ can deploy wireless monitors close to $\mathcal{H}$ and $\mathcal{P}_1$, with which $\mathcal{A}$ can overhear all the wireless signals and

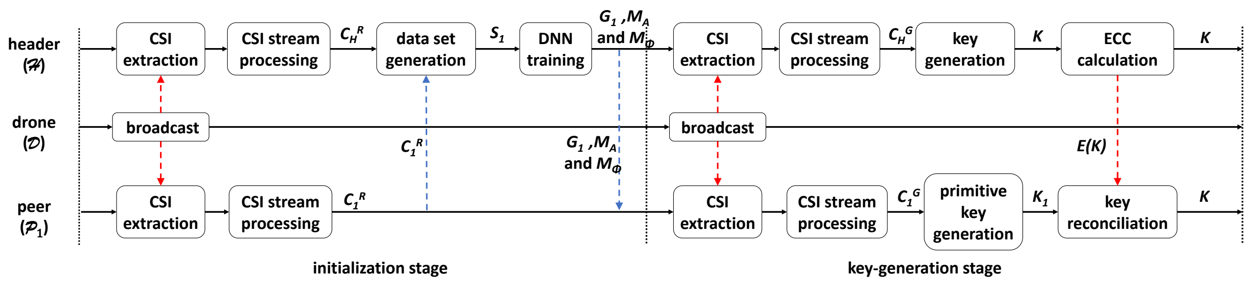


Figure 3: The workflow of DroneKey.

may be able to obtain similar copies of $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams. Second, $\mathcal{A}$ is aware of $\mathcal{H}$'s and $\mathcal{P}_1$'s locations and can obtain the trajectories of $\mathcal{D}$ with a spy camera.

As demonstrated later, the one-time initialization stage is as short as several minutes, so it is very hard for the attacker to compromise DroneKey in this short time window. In addition, the deployment of peer devices usually involves human effort, and the initialization of a peer device is executed right after it is deployed. It is reasonable to assume human aid available in the initialization stage. Therefore, human-involved authentication schemes such as the code-based Bluetooth-like pairing can be adopted to defeat attacks against the initialization stage. Note that the security of the short network-initialization phase has been assumed in the extensive literature such as random key predistribution schemes for sensor networks [11]. We also assume that $\mathcal{A}$ cannot compromise $\mathcal{H}$ or $\mathcal{P}_1$, which is the same with all the existing GKG studies. Otherwise, $\mathcal{A}$ can directly obtain the group key. However, we have minimal assumptions about the security of $\mathcal{D}$. In particular, $\mathcal{A}$ can compromise $\mathcal{D}$ and control $\mathcal{D}$'s trajectory and the broadcast signal. $\mathcal{A}$ can also impersonate $\mathcal{D}$ with a malicious drone. Under these assumptions, we consider three specific attacks as follows.

- *Malicious-drone attack.* $\mathcal{A}$ compromises $\mathcal{D}$ and fully controls $\mathcal{D}$'s trajectory and the broadcast signal. $\mathcal{A}$ can also use a malicious drone to mimic $\mathcal{D}$. By manipulating the drone's trajectory and broadcast signal, $\mathcal{A}$ hopes to infer some information about the group key K.
- *Eavesdropping attack.* $\mathcal{A}$ eavesdrops on the WiFi channel used by DroneKey and tries to infer the group key K from the overheard signals.
- *Reproduction attack (or drone-trajectory-reconstruction attack).* $\mathcal{A}$ sets up an environment similar to DroneKey's and repeat $\mathcal{P}_1$'s initialization and key-generation processes to reproduce K. Specifically, $\mathcal{A}$ flies a drone in the arranged environment in the same trajectory as $\mathcal{D}$'s and measures CSI streams at the locations corresponding to those of $\mathcal{H}$ and $\mathcal{P}_1$. With the obtained CSI streams, $\mathcal{A}$ generates dataset, trains DNN, and generates a secret key as DroneKey does, hoping that the produced key is identical to K.

DroneKey cannot work when the jamming attack is launched. However, $\mathcal{A}$ cannot obtain K via the jamming attack, and our scheme still works once the jamming signals are not present. Therefore, we do not consider the jamming attack in this paper.

4 CSI EXTRACTION AND PROCESSING

This section demonstrates the details of CSI extraction and processing. $\mathcal{H}$ and $\mathcal{P}_1$ use the same method to extract and process their CSI streams in both stages of DroneKey. Without loss of generality, we choose the 6th WiFi subcarrier for key generation, and the CSI sample only contains the mag and phase values of this subcarrier.

4.1 CSI Extraction

Among many existing CSI estimation schemes, DroneKey adopts the least-square equalizer for its low computational complexity [8, 24]. Specifically, $\mathcal{D}$ broadcasts successive WiFi packets containing only packet indexes. The preamble of each packet contains two copies of a predefined training sample, denoted as X_T. Given a received packet, the corresponding CSI sample is denoted by c and estimated as

$$c = \frac{Y_1 + Y_2}{2X_T}, \tag{1}$$

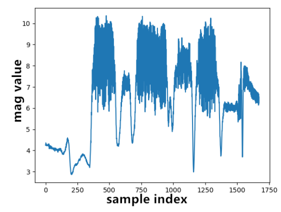
where Y_1 and Y_2 are the two received training samples. We use the index of this packet as c's index, which can be used to synchronize different devices' CSI streams. The CSI sample rate λ, i.e., the number of CSI samples extracted within one second, equals the packet transmission rate. The network works on the 2.4 GHz band and can transmit around 140 continuous WiFi packets within one second. So λ is around 140 sample/sec.

4.2 CSI Stream Processing

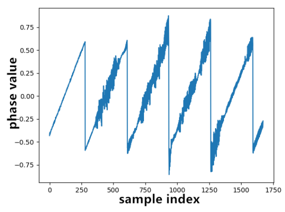
$\mathcal{H}$'s and $\mathcal{P}_1$'s raw CSI streams are heavily distorted due to the ambient noise and hardware flaws. Figs. 4a and 4b show $\mathcal{P}_1$'s raw mag and phase streams extracted in one experiment, in which $\mathcal{D}$ first keeps static and then moves randomly. The mag and phase streams are both distorted by high-frequency noise, and there is a periodically changing offset in the phase stream. Moreover, some CSI samples corresponding to heavily interpreted WiFi packets are missing, so the mag and phase streams are not continuous in time. DroneKey processes the raw mag and phase streams with the following three steps.

Missing-sample estimation. We detect missing CSI samples based on sample indexes, estimate a missing sample with the uninterrupted samples around it, and insert the estimated samples to the original stream.

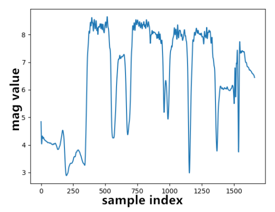
Phase calibration. After obtaining the continuous CSI stream, we calibrate the phase stream to remove the phase offset. The phase



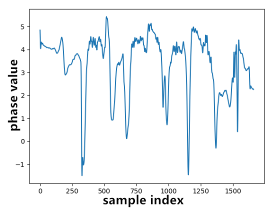
(a) Raw mag stream.



(b) Raw phase stream.



(c) Processed mag stream.



(d) Processed phase stream.

Figure 4: Raw and processed mag and phase streams.

offset arises from the carrier-frequency offsets and asynchronous clocks of the transmitter (i.e., drone $\mathcal{D}$) and receiver. As shown in Fig. 4b, the phase offset periodically changes with time. Therefore, we use the phase samples extracted when the transmitter is static as a profile to calibrate the phase stream. Specifically, we first estimate the phase-offset changing cycle, denoted as T, from the profile and then calibrate the phase stream as

$$\phi_n = \phi_n - \phi_{(n \bmod T)}, \tag{2}$$

where ϕ_n is the nth phase sample in the stream.

For the convenience of phase calibration, we let $\mathcal{D}$ keep static for a period before moving randomly when DroneKey conducts CSI extraction. Our experimental results show that T is always less than two seconds. So we set the static period of $\mathcal{D}$ as two seconds. The profile samples are extracted when $\mathcal{D}$ is static and contain little information useful for group-key generation. So we remove the profile samples from the CSI stream after the phase calibration.

Low-pass filter. Finally, we remove the high-frequency noise in the mag and phase streams with low-pass filters. The phase stream is more sensitive to noise and environmental changes than the mag stream. So the mag and phase filters use different cutoff frequencies. We denote the mag and phase cutoff frequencies by f_A and f_ϕ, respectively, which are obtained through experiments. In particular, we use an N210 USRP attached to the drone as the transmitter and two B210 USRPs placed together as two receivers. The distance between the two receivers is less than three centimeters. We fly the drone back and forth to the receivers randomly for 20 seconds while the drone keeps broadcasting WiFi packets. Each receiver extracts one CSI stream and processes it with missing-sample estimation and phase calibration. We denote these two processed CSI streams by $< A_1, \Phi_1 >$ and $< A_2, \Phi_2 >$, where A_i and Φ_i are the mag and phase streams of the ith receiver, respectively. We first determine f_A with A_1 and A_2. The CSI sampling rate is 140 sample/sec. According to the Nyquist–Shannon sampling theorem [31], the highest frequency of the signals contained in the mag stream is 70 Hz. So we test 70 values ranging from 1 Hz to 70 Hz with a step of 1 Hz for f_A. We use each tested value as f_A to filter A_1 and A_2 and calculate the correlation between the two filtered mag streams. The distance between the two receivers is less than half wavelength of the WiFi signal, and their CSI streams should be highly correlated without the noise's impact. So a good cutoff frequency should achieve a high correlation value. However, a low cutoff frequency may filter out too much information useful for key generation and thus reduce the key-generation rate. Therefore, we adopt 15 Hz as f_A because it is the maximum one among the values that achieve correlations above 0.9. Similarly, we determine f_Φ with Φ_1 and Φ_2 and adopt 20 Hz as f_Φ. The processed mag and phase streams are shown in Figs. 4c and 4d, respectively, which are continuous in time, smooth, free of noise, and can thus be used for key generation.

5 KEY-GENERATION DNN

In the peer device $\mathcal{P}_1$'s initialization stage, the group head $\mathcal{H}$ trains a DNN for $\mathcal{P}_1$ after obtaining $\mathcal{P}_1$'s CSI stream C_1^R. Specifically, $\mathcal{H}$ first generates a GKG dataset S_1 from C_1^R and $\mathcal{H}$'s own CSI stream C_H^R. It then trains $\mathcal{P}_1$'s DNN G_1 with S_1. Apart from generating the dataset and training the DNN, $\mathcal{H}$ also determines the number of quantification bins, which is critical for subsequent key-generation stages. However, the determination process is closely related to the details of the key-generation stage, so we defer its details to Section 6.2. This section first demonstrates the generation of S_1 and then details the training process of G_1.

5.1 GKG Dataset Generation

We first discuss G_1's function, which determines how the GKG dataset S_1 is generated. In the key-generation stage, the group key K is generated from $\mathcal{H}$'s CSI stream C_H^G. Device $\mathcal{P}_1$ uses its CSI stream C_1^G as G_1's input and generates a primitive group key K_1 from G_1's output. Therefore, G_1 should be able to estimate some information related to C_H^G from C_1^G, so K_1 can be similar to and can be adjusted to K in the final reconciliation step. To achieve this goal, a training sample for G_1 should contain two elements generated from $\mathcal{P}_1$'s and $\mathcal{H}$'s CSI streams, respectively. In the training process, the element related to $\mathcal{P}_1$ is used as G_1's input, and the element related to $\mathcal{H}$ is the target output. For consistence with machine learning concepts, we term the elements related to $\mathcal{P}_1$ and $\mathcal{H}$ as the feature and label, respectively.

Now we illustrate the generation of one training sample, and S_1 can be obtained by repeating this process. The training sample contains a feature and a label, which are two vectors denoted by V^f and V^l, respectively. We first randomly select a one-second segment, i.e., 140 continuous CSI samples, from C_1^R and represent it by $\tilde{C}_1^R$. We use the mag and phase values of the selected samples as the elements of V^f which can be represented as $[a_{(1,1)}, a_{(1,2)}, ..., a_{(1,140)}, \phi_{(1,1)}, \phi_{(1,2)}, ..., \phi_{(1,140)}]$, where $a_{(1,n)}$ and $\phi_{(1,n)}$ are the mag and phase values of the nth CSI sample in $\tilde{C}_1^R$, respectively. Then we select a one-second segment which is extracted simultaneously with $\tilde{C}_1^R$ from C_H^R. We use the sample indexes to synchronize C_H^R and C_1^R, and $\tilde{C}_1^R$ can be easily obtained. We denote the selected segment of C_H^R by $\tilde{C}_H^R$ and generate V^l from $\tilde{C}_H^R$ through a more complex process.

$\tilde{C}_H^R$ cannot be directly used as the label for two reasons. First, it is hard for G_1 to accurately estimate the fine-grained channel information contained in $\tilde{C}_H^R$. Second, directly using $\tilde{C}_H^R$ as the label is not secure. If we use $\tilde{C}_H^R$ as the label, G_1's output is the estimation of $\mathcal{H}$'s CSI stream. As demonstrated later in Section 6, $\mathcal{P}_1$ obtains its primitive group key K_1 by quantifying G_1's output. Accordingly, we must generate the group key by quantifying $\mathcal{H}$'s CSI stream. In this case, a powerful attacker who has obtained a similar copy of $\mathcal{H}$'s CSI stream can easily infer the group key.

To address the aforementioned concerns, we generate V^l from $\tilde{C}_H^R$ through down-sampling and obfuscation. Specifically, we first down-sample $\tilde{C}_H^R$ with a ratio of 1 to 10, and the down-sampled segment can be represented as a vector $V_d = [a_{(H,5)}, a_{(H,15)}, ..., a_{(H,135)}, \phi_{(H,5)}, \phi_{(H,15)}, ..., \phi_{(H,135)}]$, where $a_{H,n}$ and $\phi_{H,n}$ are the mag and phase values of the nth CSI sample in $\tilde{C}_H^R$, respectively. Then we apply an obfuscation function Λ to V_d to obtain the label vector $V^l = \Lambda(V_d)$. The obfuscation function can be represented as

$$\begin{aligned}\Lambda(V_d) &= V_d O \\ &= V_d \begin{bmatrix} O_A & 0 \\ 0 & O_\Phi \end{bmatrix} \\ &= \begin{bmatrix} A_H O_A & \Phi_H O_\Phi \end{bmatrix}\end{aligned} \tag{3}$$

Here, O is a 28×28 matrix and termed as an obfuscation matrix. O_A and O_Φ are both 14×14 matrices. A_H and Φ_H are the first and second half parts of V_d, respectively. The generated label V^l is a 1×28 vector. We term the whole process converting $\tilde{C}_H^R$ to V^l as label generation and denote it by $\mathcal{Z}(\tilde{C}_H^R, O)$. $\mathcal{H}$ uses the same obfuscation matrix to generate the group key-training datasets for all the peer devices.

We have two requirements for Λ. First, the information contained in each element of V_d must be inherited by $\Lambda(V_d)$. To satisfy this requirement, O_A and O_Φ must be full-rank matrices. Second, for the convenience of key quantification demonstrated later, the ranges of all the elements in $A_H O_A$ must be the same, and $\Phi_H O_\Phi$ should fulfill the same requirement. Since the movement of $\mathcal{D}$ is random, the elements in A_H are independent and identically distributed variables with the minimum value Min_A and the maximum value Max_A. The elements in A_Φ are also independent and identically distributed variables with the minimum value Min_Φ and the maximum value Max_Φ. The second requirement can thus be satisfied by requiring that the sum of O_A's elements and the sum of O_Φ's elements in the same column both equal one. Besides, we require all the elements in O to be non-negative. With the carefully designed O_A and O_Φ, the elements in $A_H O_A$ are within the range $[\text{Min}_A, \text{Max}_A]$, and $\Phi_H O_\Phi$'s elements are within the range $[\text{Min}_\Phi, \text{Max}_\Phi]$.

5.2 GKG DNN Training

We adopt a Convolutional Neural Network (CNN) as G_1. Due to the random movement of $\mathcal{D}$, the relations between different CSI samples in the same CSI stream are not significant, and CNN is thus more suitable for DroneKey than the Recurrent Neural Network (RNN). G_1 contains four hidden layers and one output layer, and its architecture is shown in Fig. 5. The first hidden layer is a fully connected layer containing 280 neurons. The second hidden layer is a 1D convolutional layer without padding. The kernel size and step of the second hidden layer are 1×5 and 5, respectively. The third layer is another fully connected layer containing 112 neurons, and

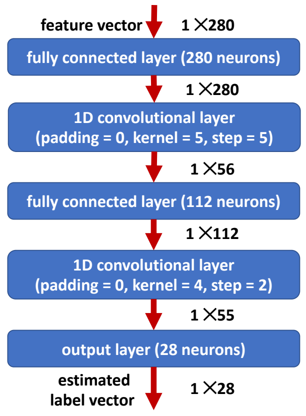

feature vector
1 ×280
fully connected layer (280 neurons)
1 ×280
1D convolutional layer
(padding = 0, kernel = 5, step = 5)
1 ×56
fully connected layer (112 neurons)
1 ×112
1D convolutional layer
(padding = 0, kernel = 4, step = 2)
1 ×55
output layer (28 neurons)
estimated
label vector
1 ×28

Figure 5: The architecture of a key-generation DNN.

the last hidden layer is a no-padding 1D convolutional layer whose kernel size and step are 1×4 and 2, respectively. The four hidden layers all use the ReLU function as their activation functions. The fully connected output layer contains 28 neurons. The dimensions of each layer's input and output are also shown in Fig.5.

In the training of G_1, we adopt the scaled Mean Square Error Loss (MSELoss) as the loss function. As demonstrated later in Section 6.2, $\mathcal{H}$ generates the group key K by quantifying the elements of a key-source vector, denoted as Θ. G_1's output, denoted as Θ_1, is the estimation of Θ. $\mathcal{P}_1$ generates its primitive group key by quantifying Θ_1's elements. Whether a GKG instance can succeed depends on Θ_1's worst element, i.e., the element with the largest estimation error. Therefore, we adopt MSELoss to balance the errors of different elements in G_1's output. Besides, the first and second half parts of Θ are generated from the mag and phase values, respectively, and they have different ranges. So does G_1's output. Therefore, we scale the loss function so that the first and second half parts of G_1's output evenly contribute to the loss value. In particular, given a training sample whose feature and label vectors are $V^f = [v_1^f, ...v_{280}^f]$ and $V^l = [v_1^l, ..., v_{28}^l]$, respectively, G_1's output is represented as $G_i(V^f) = [\hat{v}_1^l, ..., \hat{v}_{28}^l]$, and the scaled MSELoss $l(V^l, G_i(V^f))$ is calculated as

$$\begin{aligned} l(V^l, G_1(V^f)) = &\frac{1}{14}\sum_{n=1}^{14}(\frac{v_n^l - \hat{v}_n^l}{\text{Max}_A - \text{Min}_A})^2 \\ &+ \frac{1}{14}\sum_{m=15}^{28}(\frac{v_m^l - \hat{v}_m^l}{\text{Max}_\Phi - \text{Min}_\Phi})^2. \end{aligned} \tag{4}$$

Here, Min_A and Max_A are the minimum and maximum mag values of $\mathcal{H}$'s CSI samples, respectively. Min_Φ and Max_Φ are the minimum and maximum phase values of $\mathcal{H}$'s CSI samples, respectively.

The training process of G_1 consists of multiple epochs. We adopt the Stochastic Gradient Descent (SGD) optimizer to update G_1's parameters in each epoch and adopt cross-validation to avoid overfitting. In particular, we equally divide the training dataset into 10 subsets and randomly select one of them as the validation set in each epoch. Once the validation set is chosen, we iteratively use every training sample in the rest nine subsets to calculate the gradients of G_1's parameters and update these parameters accordingly.

To accelerate the training process, we use the average gradients calculated with 20 training samples to update G_1's parameters. At the end of each epoch, we calculate the averaged loss value with the training samples in the validation set and stop the training if the decrease of the loss value after this epoch is not significant, e.g., when the loss value decreases by less than five percent.

6 KEY GENERATION AND RECONCILIATION

In the key-generation stage, $\mathcal{H}$ and $\mathcal{P}_1$ first manage to obtain two related CSI streams from the same broadcast signals of $\mathcal{D}$ and then obtain the group key K through key generation and reconciliation. We denote $\mathcal{H}$'s and $\mathcal{P}_1$'s processed CSI streams by C^G_H and C^G_1, respectively. C^G_H and C^G_1 are synchronized with sample indexes. Assume that C^G_H and C^G_1 are extracted within one second such that they each contains 140 CSI samples. We demonstrate how to generate K from C^G_H and C^G_1 in this section. To generate a group key from CSI streams longer than one second, $\mathcal{H}$ and $\mathcal{P}_1$ can obtain the key by segmenting their CSI streams, generating a key fragment from each CSI segment, and finally piecing all the key fragments together in the order of time.

$\mathcal{H}$ can directly generate K from C^G_H, and $\mathcal{P}_1$ acquires K with the aid of $\mathcal{H}$. In particular, $\mathcal{H}$ first generates a group key-source vector Θ_H from C^G_H. Then $\mathcal{H}$ acquires K by quantifying Θ_H's elements. Similarly, $\mathcal{P}_1$ also first generates its group key-source vector Θ_1 and then acquires its primitive group key K_1 by quantifying Θ_1's elements. Finally, $\mathcal{P}_1$ adjusts K_1 according to the ECC broadcast by $\mathcal{H}$ and acquires K after key reconciliation. In what follows, we first illustrate the generation of key-source vectors, then how to quantify them, and finally the key reconciliation process.

6.1 Key-source Vector Generation

$\mathcal{P}_1$'s group key-source vector Θ_1 must be similar to that of $\mathcal{H}$'s (i.e., Θ_H) so that the difference between the extracted K_1 and K is subtle and can be mitigated by ECC. To fulfill this requirement, we use $\mathcal{Z}(C^G_H, O)$ as Θ_H and use $G_1(C^G_1)$ as Θ_1. Here, $\mathcal{Z}$ and O are the label-generation process and obfuscation matrix explained in Section 5.1, respectively. $G_1(C^G_1)$ is the output of G_1 with C^G_1 as the input. As introduced in Section 5.2, $G_1(C^G_1)$ is the estimation of $\mathcal{Z}(C^G_H, O)$, so they are very close. In addition, Θ_1 and Θ_H are both 1×28 vectors.

6.2 Key-source Vector Quantification

$\mathcal{H}$ obtains K by quantifying the elements of Θ_H, and $\mathcal{P}_1$ obtains K_1 by quantifying Θ_1's elements. We use Θ_H's first element, denoted by $\theta_{(H,1)}$, as an example to illustrate the element quantification. We denote the the minimum and maximum values of $\theta_{(H,1)}$ by Min_{θ_1} and Max_{θ_1}, respectively, and divide the range $[\text{Min}_{\theta_1}, \text{Max}_{\theta_1}]$ to multiple bins. These bins are represented with a vector $B_1 = [b_{(1,0)}, b_{(1,1)}, b_{(1,2)}, ..., b_{(1,M_1)}]$, where $b_{(1,m-1)}$ and $b_{(1,m)}$ are the upper and lower bounds of the mth bin, respectively. $b_{(1,0)}$ equals Min_{θ_1}, and $b_{(1,M)}$ equals Max_{θ_1}. M_1 is the number of quantification bins for $\theta_{(H,1)}$. In practice, Min_{θ_1} and Max_{θ_1} are estimated, and $\theta_{(H,1)}$ may be outside the range $[\text{Min}_{\theta_1}, \text{Max}_{\theta_1}]$. Therefore, we quantify $\theta_{(H,1)}$ as

$$Q(\theta_{(H,1)}) = \begin{cases} 0, & \text{if } \theta_{(H,1)} < \text{Min}_{\theta_1}, \\ m, & \text{if } b_{(1,m)} < \theta_{(H,1)} \le b_{(1,m+1)}, \\ M-1, & \text{if } \theta_{(H,1)} > \text{Max}_{\theta_1}. \end{cases} \quad (5)$$

$\mathcal{H}$ can extract $\log_2(M_1)$ key bits from $Q(\theta_{(H,1)})$ with the gray coding technique [43]. Since Θ_1's first element $\theta_{(1,1)}$ is the estimation of $\theta_{(H,1)}$, we also quantify $\theta_{(1,1)}$ with B_1.

Now we demonstrate how to obtain B_1. We first look into the distribution of $\theta_{(H,1)}$. Since Θ_H and the label vectors in S_1 are all generated from $\mathcal{H}$'s CSI streams with the same process, we use the distribution of the label vector's first element v^l_1 in S_1 as the estimated distribution of $\theta_{(H,1)}$. Let $F_{\theta_{(H,1)}}(x) = P(\theta_{(H,1)} \le x)$ denote $\theta_{(H,1)}$'s Cumulative Distribution Function (CDF). We use the minimum and maximum values of v^l_1 as Min_{θ_1} and Max_{θ_1}, respectively. So $b_{(1,0)}$ and $b_{(1,M)}$ are obtained. Given the number M_1 of the quantification bins, the rest elements of B_1 are determined as $F_{\theta_1}(b_{(1,m)}) = m/M_1$. As demonstrated in Section 5.1, the first 14 elements of Θ_H are related to the mag values of C^G_H with the same ranges. So we adopt the same number of quantification bins for the first 14 elements of Θ_H, which is denoted by M_A. For the same reason, the number of quantification bins for the last 14 elements of Θ_H is the same, which is denoted by M_Φ. Since $\mathcal{H}$ uses M_A and M_Φ to quantify its key-source vector, the peer devices in the network all use M_A and M_Φ for key-source vector quantification.

M_A and M_Φ significantly affect the group key-generation rate. For example, we can extract $\log_2(M_A)$ valid key bits from $\theta_{(H,1)}$, and a large M_A results in a high key-generation rate. However, with the increase of M_A, the key-mismatch rate between $\mathcal{H}$ and $\mathcal{P}_1$ also increases. Specifically, $\theta_{(1,1)}$ is the estimation of $\theta_{(H,1)}$ with a deviation $\delta_{(1,1)}$. With the increase of M_A, the sizes of quantification bins decrease, the possibility that $\theta_{(1,1)}$ and $\theta_{(H,1)}$ fall to different bins increases, and the number of mismatched key bits increases. If the mismatched key bits cannot be corrected by the ECC, the group-key generation instance fails.

We determine the optimal values of M_A and M_Φ for $\mathcal{P}_1$ based on the success rate of $\mathcal{P}_1$'s group-key generation and obtain the specific values with experiments. For convenience of presentation, we term the key bits extracted from the first 14 elements of Θ_H and Θ_1 as mag bits and term the key bits extracted from the rest elements of Θ_H and Θ_1 as phase bits. A group key-generation instance is successful for $\mathcal{P}_1$ only if the mismatched key bits between K_1 and K can be corrected by the ECC, requiring that mismatched mag and phase bits be corrected. We determine M_A based on the mag-part success rate, i.e., the possibility that the mismatched mag bits are correctable. Specifically, we test multiple values for M_A and estimate the mag-part success rate for each tested value with S_1. Among the tested values whose mag-part success rates are above a predefined threshold τ_s, we select the maximum one as M_A. The value of M_Φ can be obtained with a similar experiment using the same threshold value τ_s. The determined values of M_A and M_Φ guarantee that $\mathcal{P}_1$ can succeed with a probability above τ_s^2 in a group key-generation instance. The details of the experiments and the numerical results in different scenarios are given in Section 8.

The values of M_A and M_Φ determined with different peer devices' CSI streams may be different. After all the peer devices being initialized, DroneKey adopts the minimum one among the many obtained values of M_A as the final M_A value and determines the final M_Φ value similarly.

6.3 Reconciliation

We use the final reconciliation step to mitigate the deviations between $\mathcal{H}$'s and $\mathcal{P}$'s secret keys. Specifically, $\mathcal{H}$ calculates K's ECC, denoted by $E(K)$, and broadcast it. Then $\mathcal{P}_1$ adjusts K_1 according to the received ECC. DroneKey adopts the BCH(31,15) code [7] in the evaluation, which allows three mismatched key bits for every 16 key bits.

7 SECURITY ANALYSIS

7.1 Security against Malicious-drone Attack

In DroneKey, $\mathcal{D}$ merely keeps broadcasting wireless signals to serve as a source for correlated channel randomness and has no other interaction with either the group head or peer devices. This means that the attacker $\mathcal{A}$ gets no information about the group key K from the compromised $\mathcal{D}$.

$\mathcal{A}$ may also manipulate the drone's trajectory by either compromising $\mathcal{D}$ or mimicking $\mathcal{D}$ with a malicious drone, hoping to manipulate K. As long as the predefined signal is broadcast, the correlation among devices' CSI streams exists, and a group key can be obtained by the head and all peer devices. Although the generated group key is manipulated, $\mathcal{A}$ cannot infer the group key without knowing the key-generation DNN and the obfuscation matrix.

In addition, $\mathcal{A}$ may manipulate the broadcast signal. In this case, the correlation among devices' CSI streams is broken. The number of unmatched bits between $\mathcal{H}$ and $\mathcal{P}_1$ increases and cannot be corrected with the ECC. The GKG instance fails, and all the devices stay with the old group key until a legitimate drone starts a new GKG instance. The old group key is unknown to $\mathcal{A}$, so DroneKey is not compromised either.

7.2 Security against Eavesdropping and Reproduction Attacks

Before analyzing DroneKey's security against eavesdropping and reproduction attacks, we first discuss how to launch these two attacks effectively. We assume that $\mathcal{A}$ is aware of DroneKey's workflow and the locations of $\mathcal{H}$ and $\mathcal{P}_1$, so $\mathcal{A}$ can obtain similar copies of $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams through eavesdropping or reproduction attacks. Moreover, we assume that $\mathcal{A}$ has inferred M_A and M_Φ, which are the numbers of quantification bins demonstrated in Section 6.2, through the reproduction attack. Since the group key K is generated from $\mathcal{H}$'s CSI stream C_H^G, $\mathcal{A}$ tries to infer K from the obtained copy of C_H^G. In addition, we assume that $\mathcal{A}$ is lucky enough to obtain an identical copy of C_H^G. However, $\mathcal{A}$ has no information about the obfuscation matrix and can only try random matrices.

We aim to give a lower bound for the number of attempts after which $\mathcal{A}$ can certainly find a matrix, denoted by $\hat{O}$, that is close enough to O and can thus be used as the replacement of O to generate group keys. With the same format as O, $\hat{O}$ can be represented as $\hat{O} = \begin{bmatrix} \hat{O}_A & 0 \\ 0 & \hat{O}_\Phi \end{bmatrix}$. Since each element of the key-source vector is the dot product of V_d^G and the corresponding column of O, $\mathcal{A}$ can find $\hat{O}$ by searching for all the columns, i.e., the columns of $\hat{O}_A$ and $\hat{O}_\Phi$. Here, V_d^G is the vector obtained by down-sampling C_H^G and is represented as $[a^G_{(H,5)}, a^G_{(H,15)}, ..., a^G_{(H,135)}, \phi^G_{(H,5)}, \phi^G_{(H,15)}, ..., \phi^G_{(H,135)}]$, where $a^G_{H,n}$ and $\phi^G_{H,n}$ are the mag and phase values of the nth CSI sample in C_H^G, respectively.

We first investigate how many attempts are needed for the attacker to find $\hat{O}_A$'s first column, which can be denoted by $\hat{O}_{(A,1)} = [o_1, o_2, ..., o_{14}]'$. Also, we represent the mag values of V_d^G as $V_A = [a^G_{(H,5)}, a^G_{(H,15)}, ..., a^G_{(H,135)}]$. $\mathcal{A}$ uses $Q(V_A\hat{O}_{(A,1)})$ to estimate $Q(\theta_{(H,1)})$ and generates key bits from $Q(V_A\hat{O}_{(A,1)})$ with gray coding. Here, Q denote the quantification function, and $\theta_{(H,1)}$ is the first element of the group key-source vector Θ_H. We can know $\theta_{(H,1)} = V_A O_{(A,1)}$, where $O_{(A,1)}$ is the first column of O_A. We denote the key bits generated from $\theta_{(H,1)}$ and $V_d^G O_{(A,1)}$ as κ and $\hat{\kappa}$, respectively. The number of mismatched bits between κ and $\hat{\kappa}$ must be less than $\frac{3 * \text{len}(\kappa)}{16}$ so that $\hat{\kappa}$ can be corrected to κ according to its ECC. Here, len(κ) is the number of key bits in κ. Our experiments show that len(κ) is always less than 32, so κ and $\hat{\kappa}$ differ by at most one bit. In gray coding, two integers' codes differ in one bit only if they are adjacent, so we can get $Q(\theta_{(H,1)}) - 1 \leq Q(V_A\hat{O}_{(A,1)}) \leq Q(\theta_{(H,1)}) + 1$. For simplicity, we assume that the quantification bins for $Q(\theta_{(H,1)})$ are of the same size. According to Eq. (5), $\hat{O}_{(A,1)}$ must satisfy the following requirement

$$V_A\hat{O}_{(A,1)} - V_A O_{(A,1)} \leq \frac{\text{Max}_A - \text{Min}_A}{M_A}, \tag{6}$$

where Min$_A$ and Max$_A$ denote the minimum and maximum mag values of $\mathcal{H}$'s CSI samples, respectively. We denote $\hat{O}_{(A,1)} - O_{(A,1)}$ as μ, then we can know

$$\begin{aligned} V_A\mu &= V_A \cdot \mu' \\ &= |V_A||\mu'| \cos\omega \\ &\leq \frac{\text{Max}_A - \text{Min}_A}{M_A}, \end{aligned} \tag{7}$$

where μ' is the transpose of μ and ω is the angle between vectors V_A and μ'. Since the drone $\mathcal{D}$'s movement is quite random, the V_A's direction and the angle ω are random. V_A's elements are non-negative, and our experimental results show that M_A is always more than eight. So we reformulate the requirement as

$$\begin{aligned} |(\hat{O}_{(A,1)} - O_{(A,1)})'| &\leq \frac{\text{Max}_A - \text{Min}_A}{M_A * |V_A|} \\ &\leq \frac{\text{Max}_A - \text{Min}_A}{M_A * \min(|V_A|)} \\ &= \frac{1}{\sqrt{14}} * \frac{\text{Max}_A - \text{Min}_A}{M_A * \text{Min}_A} \\ &= \frac{1}{8\sqrt{14}} * \frac{\text{Max}_A - \text{Min}_A}{\text{Min}_A}. \end{aligned} \tag{8}$$

Now we talk about $\mathcal{A}$'s strategy searching for $\hat{O}_{(A,1)}$. Without any information about $O_{(A,1)}$, $\mathcal{A}$'s best strategy is brute-force searching with a grain of $\frac{1}{\gamma}$, where γ is a positive integer. Specifically, knowing that the sum of $\hat{O}_{(A,1)}$'s elements equals one, $\mathcal{A}$ tries all the vectors like $[\frac{n_1}{\gamma}, \frac{n_2}{\gamma}, ..., \frac{n_{14}}{\gamma}]'$, where n_i is called the grain-amount of ith element. All the grain-amounts are non-negative integers, and $\sum_{i=1}^{14} n_i = \gamma$. Given a grain $\frac{1}{\gamma}$, the number of vectors in the searching space can be calculated as

$$C(\gamma + 13, 13) = \frac{(\gamma + 13)!}{\gamma! * 13!}, \tag{9}$$

where γ must be sufficiently small so that at least one of the vectors in the searching space is close enough to $O_{(A,1)}$ and meets the requirement in Eq. (8). Among all the vectors in the searching space, we denote the one that is closest to $O_{(A,1)}$ as V_c. The largest possible value of $|(V_c - O_{(A,1)})'|$ is $\frac{\sqrt{14}}{2\gamma}$. Therefore, we formulate the requirement for γ as

$$\frac{\sqrt{14}}{2\gamma} \le \frac{1}{8\sqrt{14}} * \frac{\text{Max}_A - \text{Min}_A}{\text{Min}_A}, \tag{10}$$

i.e.,

$$\gamma \ge \frac{56 * \text{Min}_A}{\text{Max}_A - \text{Min}_A}. \tag{11}$$

In our experiments, the value of $\frac{\text{Min}_A}{\text{Max}_A - \text{Min}_A}$ is always greater than 0.3, so γ must be greater than 17. According to Eq. (9), the searching space contains more than 1×10^8 vectors. Therefore, $\mathcal{A}$ needs to try at least 14×10^8 vectors to obtain a replacement matrix for O_A and try even more vectors to obtain a replacement matrix for O, which is extremely hard. In practice, there is a nontrivial deviation between C_H^G and $\mathcal{A}$'s copy, which makes it even harder to obtain a replacement obfuscation matrix. Therefore, DroneKey is robust to the eavesdropping and reproduction attacks.

8 EVALUATION

We evaluate DroneKey in this section. In what follows, We first introduce our prototype implementation and then measure DroneKey's key-generation rates in different scenarios. We also evaluate the randomness and entropy of the generated keys with the standard NIST runs test and the system overhead of our scheme. Finally, we give a theoretical estimation of the time consumed to generate and update the group keys for a large-scale network based on our experimental results.

8.1 Implementation

We implement the system in Fig. 2 with three USRPs and a DJI Matrice 100 drone. Specifically, We attach one N210 USRP to the drone and use them together as $\mathcal{D}$. We use one B210 USRP as the group head $\mathcal{H}$ and another B210 USRP as the peer device $\mathcal{P}_1$. The N210 USRP is connected to a laptop with an Ethernet cable, and the two B210 USRPs are connected to a desktop which has two Intel 4.2 GHz i5 processors where all the computations are executed. We implement our CSI-estimation tool on GNU Radio [2] by modifying the open source code of the Wime project [6] and implement the group key-generation DNN with PyTorch [1]. Our prototype is for the purpose of evaluation. In practice, $\mathcal{D}$ can use its embedded WiFi transceiver or an attached lightweight battery-powered WiFi router, instead of the USRP, for signal broadcasting.

8.2 Performance Metrics

We use three performance metrics, including the key-generation rate, the group-success rate, and the randomness of the generated keys. We define the key-generation rate as the number of key bits generated from the CSI samples collected within one sec. This definition is also adopted in previous studies [4, 22, 32, 35]. We define the group-success rate as the possibility that all the peer devices in the group can obtain a common key identical to the key generated by $\mathcal{H}$. Correspondingly, we term the possibility that a specific peer device can obtain the key generated by $\mathcal{H}$ as the peer device's individual-success rate. Without loss of generality, we assume that $\mathcal{P}_1$ is the peer device whose individual-success rate is the lowest. As recommend by NIST [27], we use the runs test for randomness checking and also measure the entropy of generated key bits.

We conduct experiments with different configurations to evaluate DroneKey's performance in various scenarios. These experiments have the basic procedure and only differ in specific settings. To avoid redundancy, we use a basic experiment to demonstrate the common experimental procedure and then present the results for specific additional settings.

8.2.1 *Basic Experiment.*

We conduct the basic experiment in a regular one-story residential house. $\mathcal{P}_1$ is placed 3 m away from $\mathcal{H}$. Since $\mathcal{D}$'s movement is limited by the cable connecting USRP to the laptop, we fly the drone within a 2 m×2 m×2 m cubic area whose center is around 4 m away from both $\mathcal{P}_1$ and $\mathcal{H}$. Hereafter, we refer to the center of $\mathcal{D}$'s moving area as $\mathcal{D}$'s location.

We first extract $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams for 5 min and obtain around 40,000 CSI samples for each of them. Then we generate 10,000 training samples for the training dataset S_1 and train the group key-generation DNN G_1.

We also determine the quantification bins' numbers, i.e., M_A and M_Φ, from S_1 through massive virtual key-generation instances. For each virtual instance, we randomly select two synchronized CSI segments, termed as a CSI-segment pair, with each containing 140 CSI samples from $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams. By repeating this process, we obtain 2,000 CSI-segment pairs. Then the 20 integers between 1 and 20 are used as M_A in turn to generate group keys from the 2,000 CSI-segment pairs. We consider a virtual instance mag-part successful if the mag bits generated from $\mathcal{P}_1$'s segment can be corrected to those generated from $\mathcal{H}$'s segment with BCH(31,15). Finally, we choose M_A as the highest value whose mag-part success rate is above a threshold τ_s. Similarly, we use the same threshold to determine M_Φ, and $\mathcal{P}_1$'s individual-success rate is thus τ_s^2. Since group-success rate decreases exponentially with the decrease of the individual-success rate, we adopt 99.5% as τ_s so that DroneKey can achieve an individual-success rate above 99% for $\mathcal{P}_1$. M_A and M_Φ are thus set to 12 and 9, respectively, and the corresponding key-generation rate is around 95 bit/sec.

Finally, we conduct real key-generation instances to verify that DroneKey can indeed achieve the expected success rate and also check the randomness of the generated keys. We extract $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams for 500 sec, which can be considered 500 continuous key-generation instances. For each instance, we extract and compare $\mathcal{H}$'s and $\mathcal{P}_1$'s keys. Only three of the 500 instances fail, and $\mathcal{P}_1$'s individual-success rate is 99.4%, which is above the expectation. Then we piece together the 497 keys from the successful instances in the order of time and obtain a binary sequence containing around 47,000 bits. This binary sequence passes the runs test with a p-value of 0.69, which is much larger than the threshold 0.1, and the entropy of each key bits was 0.99. Therefore, DroneKey can generate around 95 random key bits per sec for a peer device while guaranteeing the individual-success rate of the device is above 99%.

We also measure the DNN training time in the basic experiment. A desktop which has two Intel 4.1 GHz i5 processors can finish the training within 50 sec. Considering that each peer device requires a unique DNN, the whole DNN training task for a large group can cost several hours. This issue can be addressed by offloading the training task to a remote server. A Dell 7920 Tower server with a Quadro RTX 5000 GPU can finish the DNN training within 5 sec. For a group containing 100 devices, all the peer devices' DNNs can be trained within 8 min, which is acceptable for the one-time group initialization.

We repeat the basic experiment in the outdoor environment, which is a backyard. With the same requirement for individual-success rate, DroneKey achieves a key-generation rate of 99 bit/sec, slightly better than in the indoor environment. This is because the impact of multipath signals is less significant in the outdoor environment, and the correlation between $\mathcal{H}$'s and $\mathcal{P}_1$'s CSI streams is thus more significant. The generated keys also pass the runs test with p-value of 0.73, and the entropy of each key bits is 0.99.

8.2.2 *Impact of Network Scale.*

DroneKey's performance also relates to the group size and the group-coverage area.

Impact of Group Size. Since the key generations at different peer devices are independent, the group size does not impact the individual-success rate of a specific device. However, when the group size increases, the group-success rate may decrease significantly. Therefore, we use the time consumed to generate 1,000 key bits as the metric to evaluate the impact of the group size and denote it by T_t. Although a large-scale IoT network may contain thousands of devices, the number of devices that are within the communication range of a drone is much smaller. In this evaluation, we assume that the group size is at most 100, which is much larger than the maximum group size of 10 reported in previous studies. We also consider the outdoor environment which is common for large-scale IoT network. We adopt the individual-success rate obtained in the aforementioned experiment as the minimum individual-success rate of any peer devices and can thus estimate the time consumption as $T_t = 1000/(99 * 0.99^N)$ sec, where N is the group size. The corresponding values of T_t for N equaling 10, 50, and 100 are 11.17 sec, 20.01 sec, and 27.6 sec, respectively. With the group size increasing by 400% and 900%, the time consumption only increases by 79.05% and 147.09%, respectively. For a dense network containing 100 devices, DroneKey can still generate 1,000 key bits within 30 sec. Therefore, DroneKey is scalable with the group size.

Impact of Group-coverage Area. To cover as many devices as possible, $\mathcal{H}$ should be close to the center of the group, and so should $\mathcal{D}$. Therefore, we measure the group-coverage area with the distance between $\mathcal{P}_1$ and $\mathcal{D}$. We first evaluate the group-coverage's impact in the indoor environment. In particular, we fix the distance between $\mathcal{H}$ and $\mathcal{D}$ as 3 m and increase the distance between $\mathcal{P}_1$ and $\mathcal{D}$ from 3 m to 7 m with a step of 0.5 m. Due to the constraints in the environment, we are unable to evaluate longer distance settings. For each location of $\mathcal{P}_1$, we measure the corresponding key-generation rate and show the results in Fig. 6. Then we repeat the experiment in the outdoor environment and test 45 distance settings ranging from 3 m to 25 m with a step of 0.5 m. The results are also shown in Fig. 6. All the generated keys have passed the randomness check with p-values above 0.65. The results show that the impact of the group-coverage area is not significant. For a peer device 25 m away from $\mathcal{D}$, DroneKey can still achieve a key-generation rate above 80 bit/sec. Therefore, DroneKey is scalable in terms of the group-coverage area.

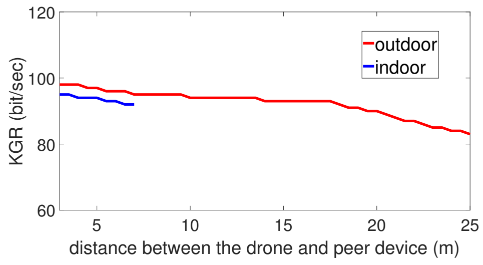


Figure 6: The impact of the group-coverage area.

8.2.3 *Impact of Environmental Factors.*

Since CSI is sensitive to environment changes, we evaluate the impact of environmental factors with experiments. Environment changes are usually caused by the relocation and movements of objects. We consider two common objects including persons and furniture for the indoor environment, as well as two common objects including persons and vehicles for the outdoor environment.

Impact of Indoor Environment Changes. We first measure the key-generation rate of DroneKey in a static indoor environment and use the result as the reference for comparison. In the experiments, $\mathcal{H}$ are placed 3 m away from $\mathcal{P}_1$, and $\mathcal{D}$ is 4 m away from both $\mathcal{H}$ and $\mathcal{P}_1$. One person stands still during the experiment, and there is a 2.5m (H)×2m(W)×1m(D) cabinet in the room. The person and the cabinet do not block the Line On Sight (LOS) channel between $\mathcal{H}$ and $\mathcal{D}$, denoted as channel H2D, or the LOS channel between $\mathcal{P}_1$ and $\mathcal{D}$, denoted as channel P2D. Then we let the person move to three new locations, denoted as locations A, B, and C. Location B blocks channel H2D; location C blocks the channel P2D, and location A does not block either channel. We measure DroneKey's key-generation rate with the person standing still in each location. Specifically, we conduct 200 GKG instances for each location and still uses the key-generation DNN obtained in the reference experiment to extract the group key. Due to the person's relocation, unmatched bits between $\mathcal{H}$ and $\mathcal{P}_1$ increase, and many GKG instances fail with the key-generation rate equal to 0 bit/sec. We calculate the average key-generation rate for each location and show the result in Table 2. After that, the person returns to the original location in the reference experiment, and we measure DroneKey's key-generation rates with the cabinet moved to locations A, B, and C. Finally, we let the person move in the area where DroneKey is deployed along two trajectories and measure DroneKey's key-generation rates accordingly. The person does not block the two LOS channels along trajectory 1 but frequently cuts off the two channels along trajectory 2. Table 2 shows the results measured in different settings. In the indoor environment, DroneKey's key-generation rate slightly decreases due to environment changes. Human movement has the most significant impact, but DroneKey can still achieve a key-generation rate of 91 bit/sec. Therefore, DroneKey is robust to indoor environment changes.

Impact of Outdoor Environment Changes. We first measure DroneKey's key-generation rate in a reference outdoor setting. Then we change some environmental factors and measure the corresponding key-generation rates. The locations of $\mathcal{H}$, $\mathcal{D}$, and $\mathcal{P}_1$ are fixed in all the experiments. $\mathcal{H}$ is placed 3 m away from $\mathcal{P}_1$, and $\mathcal{D}$ is 4 m away from both $\mathcal{H}$ and $\mathcal{P}_1$. There are one person standing still and a static vehicle in the reference setting, and neither blocks channel H2D or P2D. In the following experiments, we first let the person stand still in three different locations, denoted as A, B, and C, and measure the corresponding key-generation rates. Location B blocks channel H2D; location C blocks the channel P2D; and location A does not block either channel. Then we let the person return to the original location in the reference setting, move the vehicle to locations A, B, and C, and measure the corresponding key-generation rates. Finally, we measure DroneKey's key-generation rates with a person or a vehicle moving around. We evaluate trajectories 1 and 2 for the person and 3 and 4 for the vehicle. Trajectories 1 and 3 do not block the two LOS channels, while trajectories 2 and 4 do. Table 2 shows the key-generation rates measured in different settings. The movements and relocation of small objects such as humans have no obvious impact on DroneKey in the outdoor environment. In contrast, the movement and relocation of large objects, especially metal objects with flat surfaces such as vehicles, can slightly decrease DroneKey's key-generation rate. However, DroneKey still achieves a key-generation rate of 90 bit/sec in the worst case, which significantly outperforms the existing GKG schemes.

Indoor factors	**Key-generation rate** (bit/sec)
indoor reference	95
human locations A/B/C	95/95/94
cabinet locations A/B/C	95/92/93
human trajectories 1/2	93/91
Outdoor factors	**Key-generation rate** (bit/sec)
outdoor reference	99
human locations A/B/C	99/99/97
vehicle locations A/B/C	97/93/92
human trajectories 1/2	99/96
vehicle trajectories 3/4	95/90

Table 2: The impact of environment changes.

8.3 Overhead of DroneKey

This section evaluates DroneKey's overhead, including the memory cost, the energy consumption, and the computation and communication overhead.

Memory Cost. $\mathcal{D}$ stores the broadcast signal, which is less than 1 KB. $\mathcal{H}$ stores the obfuscation matrix O, M_A, and M_Φ. M_A and M_Φ are two float-type numbers, and O has 392 float-type none-zero elements. So the memory cost for $\mathcal{H}$ is less than 2 KB. A peer device stores a key-generation DNN, M_A, and M_Φ. The DNN contains around 86,000 parameters, each of which is a float-type number. So the memory cost for a peer device is less than 0.5 MB. The memory cost is trivial for any of the devices involved in DroneKey.

Energy Consumption. In DroneKey, $\mathcal{D}$ can return to a support station for battery charging or get battery changes after each key-generation instance. So the energy consumption is not a constraint for $\mathcal{D}$. We only evaluate the energy consumption of the head and peer devices. The energy consumption mainly results from the computation and communication, so we evaluate the computation and communication overhead of the head and peer devices instead.

Computation and Communication Overhead. Since the initialization is conducted once, we only consider the key-generation stage. In one key-generation instance, $\mathcal{H}$ broadcasts the ECC for one time, and no transmission is needed at the peer device. Therefore, the communication overhead for the whole group is one transmission per instance. Existing PHY-based GKG schemes all require each device to transmit at least one probe packet [22, 35, 39, 42]. The communication overhead of these schemes is at least n transmissions per GKG instance, where n is the number of devices in the group. So our scheme has much lower communication overhead than existing GKG schemes.

For each key-generation instance, $\mathcal{H}$ needs to perform a matrix multiplication, and a peer device needs to calculate the output of the key-generation DNN, which involves three matrix-multiplication operations. The computation overhead of DroneKey is slightly higher than that of existing PHY-based GKG schemes, which only involve quantification operations. However, the sizes of the involved matrices are not large, and the computation task can be easily handled by the processor of most IoT devices. Therefore, the computation overhead does not impact the deployment of DroneKey.

8.4 Whole-network Group-key Generation

A large-scale IoT network consists of many distribute device groups, each of which needs to refresh its group key from time to time. We can estimate the total time it takes to generate or update the group keys for the entire network based on previous results for a single group. Assume that the network covers a 1 km×1 km square region in which 20,000 IoT devices are deployed. Our goal is to generate a group key of 256 bits for each device group, which is a recommended key size of the Advanced Encryption Standard (AES). Assume that a drone can cover a circle region with a diameter of 100 m (i.e., the typical WiFi transmission range at 2.4 GHz), each corresponding to the coverage area of a group. The whole network can then be divided into 196 device groups, each containing about 100 IoT devices. We first discuss the scenario in which only one drone is available. In this case, the drone flies to each device group one by one to help generate a group key. The distance between the centers of two adjacent device groups is around 71 m, and the speed of a COTS drone can be above 15 m/sec [12]. So the drone movement between two adjacent groups can be finished within 5 sec. According to our previous experimental results, the expected time of the drone's random movement for generating 256 key bits is 7.07 sec. Since the key-generation and reconciliation steps do not involve the drone, the drone only needs to stay with each group for no more than 8 sec. The total time consumption for generating the group keys of all the groups can thus be estimated as $196 \times (5\text{ sec} + 8\text{ sec}) = 2,548$ sec, which is around 42 minutes. The communication range of the drone can be increased by using a more powerful transmitter. In addition, multiple drones can be dispatched to assist different groups in parallel. For example, if five drones are used, DroneKey can update the group keys of the entire network within 10 minutes, which is short enough for most application scenarios.

9 RELATED WORK

Recently, researchers have proposed many GKG schemes for IoT networks. In addition, a number of earlier GKG schemes proposed for wireless sensor networks can also be applied to the IoT network. Those schemes mainly fall into two categories based on cryptography and PHY information, respectively.

The schemes in the first category rely on cryptographic methods to secure the group-key distribution or agreement. Tubaishat *et al.* propose a scheme based on the multi-party Diffie–Hellman protocol [38]. In this scheme, a head device generates the group key after accumulating the rest devices' partial keys. The transmissions of both partial keys and the final group key to each device rely on pairwise key-based encryption. Public key cryptography is used to distribute the group key in [30] and [5]. Zhu *et al.* propose a scheme that first establishes pairwise keys between the head device and each other device, and then the group key can be distributed [46]. In the context of large-scale IoT networks, a large amount of pairwise keys and public-private key pairs involved in [5, 30, 38, 46] must be updated for each group-key update instance, making those schemes impractical in our considered context. In contrast, DroneKey also uses pairwise keys to secure the very short initialization stage and has no need for pairwise-key updates, so it is a more practical GKG solution for large-scale IoT networks.

Researchers have also proposed many cryptographic schemes not involving pairwise keys. Wen *et al.* propose a Bloom's matrix-based GKG scheme, in which a matrix is pre-shared among a group of devices and can be used for subsequent group-key generation [40]. Teo *et al.* explore the Burmester-Desmedt group-key agreement method to generate a common key for devices forming a circular hierarchical group [33]. Those schemes require multiple rounds of communications involving all the devices and also incur heavy computation load when the group size is large. By comparison, DroneKey is efficient in both communication and computation. In terms of the communication overhead, DroneKey only requires the group head to broadcast the ECC in the group-key generation stage. As for the computation load, the group head performs a matrix multiplication and a simple quantification operation to obtain the group key, and each peer device conducts the DNN forward computation for one time and a similar quantification operation to obtain the group key. All the involved calculations are lightweight and suitable for resource-constrained IoT devices.

The PHY-based GKG schemes adopt the channel variations of one or multiple channels in the network as the randomness factor, from which the group key is generated. The PHY information is first explored to establish pairwise keys between two devices, and many secure and efficient schemes have been proposed [14, 21, 28, 45]. There is also effort to achieve PHY-based GKG. The most intuitive solution is generating enough pairwise keys which can be used to distribute a group key from device to device. The scheme in [20] follows this idea. The authors propose to first establish pairwise keys between a virtual center node and each of the rest nodes in a star network or between each node and its two neighbors in a chain network. Then the pairwise keys are used to securely transmit a random group key from device to device. Their scheme is not practical for large-scale IoT networks because a huge number of pairwise keys must be established in each group-key generation instance, which can consume significant time. Researchers have also proposed to spread the measurements of selected channels to the entire group of devices. Specifically, each device broadcasts a signal which can be obtained by fusing the measurements of multiple channels [21, 35] or splitting the measurement of one channel [42]. A legitimate device can infer the measurements of selected channels from the broadcast signals, while an attacker cannot. However, these methods require that each device transmit at least one probe packet in a group-key generation instance, and all the packets must be transmitted within the short channel coherent time without interfering with each other. So these schemes are not scalable to a large group in a dense IoT network either.

10 CONCLUSION

In this paper, we propose DroneKey, a drone-aided PHY-based GKG scheme for large-scale mission-critical IoT networks. We use a randomly moving drone to introduce a common randomness factor, which can be acquired by the entire group of devices and thus be used as the common randomness source for generating group keys. In particular, the CSI streams extracted from the same broadcast signals but by different devices are all correlated to the drone's movement and thus inherently correlated with each other. We adopt the deep-learning technique to capture the correlations among the CSI streams of different devices in a group, which guarantee the consistency of their individually generated keys. DroneKey involves a single broadcast message by the group head and no other message exchange within the group, so it is highly scalable with the group size. In case that a powerful attacker may obtain the drone's trajectory and further infer the group key, we adopt an obfuscation function to enhance DroneKey's security and theoretically prove that DroneKey is robust against both eavesdropping and trajectory-reproduction attacks.

We build a prototype and evaluate DroneKey's performance in multiple scenarios. DroneKey can achieve a group-key generation rate over 85 bit/sec in most evaluated scenarios, significantly outperforming the state-of-the-art prior work. According to the experimental results, DroneKey is scalable in terms of the group size and network scale. Moreover, we estimate the time consumption of generating and updating group keys for an extremely large-scale IoT network covering a region of 1 km^2 and containing 20,000 devices. According to our estimation, the group-key update can be finished within 43 minutes, and the time consumption can be further reduced to 10 minutes by involving multiple drones that are equipped with signal amplifiers. In summary, DroneKey is a scalable, fast, and efficient GKG solution for large-scale IoT networks.

REFERENCES

[1] 2004. PyTorch. https://pytorch.org/.

[2] 2020. GNU Radio. https://www.gnuradio.org/.

[3] Herve Abdi. 2007. The Kendall Rank Correlation Coefficient. *Encyclopedia of Measurement and Statistics* (2007), 508–510.

[4] Ortal Arazi and Hairong Qi. 2005. Self-certified group key generation for ad hoc clusters in wireless sensor networks. In *IEEE INFOCOM*. Miami, FL.

[5] Xirong Bao, Jin Liu, Lihuang She, and Shi Zhang. 2014. A key management scheme based on grouping within cluster. (2014), 3455–3460.

[6] Bastian Bloessl, Michele Segata, Christoph Sommer, and Falko Dressler. 2018. Performance assessment of IEEE 802.11p with an open source SDR-based prototype. *IEEE Transactions on Mobile Computing* 17, 5 (2018), 1162–1175.

[7] Chandra Bose and Dwijendra Ray-Chaudhuri. 1960. On a class of error correcting binary group codes. *Information and control* 3, 1 (1960), 68–79.

[8] Sinem Coleri, Mustafa Ergen, Anuj Puri, and Ahmad Bahai. 2002. Channel estimation techniques based on pilot arrangement in OFDM systems. *IEEE Transactions on Broadcasting* 48, 3 (September 2002), 223–229.

[9] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *Transactions on Information Theory* 22, 6 (1976), 644–654.

[10] Wenliang Du, Jing Deng, Yunghsiang Han, Pramod Varshney, Jonathan Katz, and Aram Khalili. 2005. A pairwise key predistribution scheme for wireless sensor networks. *Transactions on Information and System Security (TISSEC)* 8, 2 (2005), 228–258.

[11] Laurent Eschenauer and Virgil Gligor. 2002. A key-management scheme for distributed sensor networks. In *ACM CCS*. Washington, DC, 41–47.

[12] Jonathan Feist. 2018. How fast can a drone fly. https://dronerush.com/.

[13] Phil Goldstein. 2016. DHS, Border protection and marines want commercial IoT devices on the battlefield. https://fedtechmagazine.com/article/2016/05/dhs-border-protection-and-marines-want-commercial-iot-devices-battlefield.

[14] John Hershey, Amer Hassan, and Rao Yarlagadda. 1995. Unconventional cryptographic keying variable management. *IEEE Transactions on Communications* 43, 1 (1995), 3–6.

[15] Texas Instruments. 2021. Engineering smarter factory automation control designs. https://www.ti.com/applications/industrial/factory-automation/overview.html.

[16] Harshan Jagadeesh, Rohit Joshi, and Manish Rao. 2021. Group secret-key generation using algebraic rings in wireless networks. *IEEE Transactions on Vehicular Technology* 70, 2 (2021), 1538–1553.

[17] Suman Jana, Sriram Premnath, Mike Clark, Sneha Kasera, Neal Patwari, and Srikanth Krishnamurthy. 2009. On the effectiveness of secret key extraction from wireless signal strength in real environments. In *ACM MobiCom*. Beijing, China.

[18] Sang Kang and Thinh Nguyen. 2012. Distance based thresholds for cluster head selection in wireless sensor networks. *IEEE Communications Letters* 16, 9 (2012), 1396–1399.

[19] KasaSmart. 2021. AC750 Wireless Travel Router. https://www.tp-link.com/us/home-networking/wifi-router/tl-wr902ac/.

[20] Guyue Li, Liangjun Hu, and Aiqun Hu. 2019. Lightweight group secret key generation leveraging non-reconciled received signal strength in mobile wireless networks. In *IEEE ICC*. Shanghai, China, 1–6.

[21] Hongbo Liu, Yang Wang, Jie Yang, and Yingying Chen. 2013. Fast and practical secret key extraction by exploiting channel response. In *IEEE INFOCOM*. Turin, Italy.

[22] Hongbo Liu, Jie Yang, Yan Wang, Yingying Chen, and Can Koksal. 2014. Group secret key generation via received signal strength: Protocols, achievable rates, and implementation. *IEEE Transactions on Mobile Computing* 13, 12 (2014), 2820–2835.

[23] Suhas Mathur, Wade Trappe, Narayan Mandayam, Chunxuan Ye, and Alex Reznik. 2008. Radio-telepathy: Extracting a secret key from an unauthenticated wireless channel. In *ACM MobiCom*. San Francisco, California.

[24] Michele Morelli and Umberto Mengali. 2001. A comparison of pilot-aided channel estimation methods for OFDM systems. *IEEE Transactions on Signal Processing* 49, 12 (December 2001), 3065–3073.

[25] Pendjari National Park. 2021. Management and Infrastructure. https://www.africanparks.org/our-work/Management-Infrastructure.

[26] IoBT Reign. 2020. IoBT: EMPOWERING READINESS TO MEET COMMANDER INTENT. https://iobt.illinois.edu/.

[27] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. 2001. A statistical test suite for random and pseudorandom number generators for cryptographic applications. https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final.

[28] Akbar Sayeed and Adrian Perrig. 2008. Secure wireless communications: Secret keys through multipath. In *ICASSP*. Las Vegas, NV.

[29] Lukas Schroth. 2020. The drone market size 2020-2025: 5 key takeaways. https://droneii.com/the-drone-market-size-2020-2025-5-key-takeaways.

[30] Seung-Hyun Seo, Jongho Won, Salmin Sultana, and Elisa Bertino. 2014. Effective key management in dynamic wireless sensor networks. *Transactions on Information Forensics and Security* 10, 2 (2014), 371–383.

[31] Claude Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review* 5, 1 (2001), 3–55.

[32] Tianyu Tang, Ting Jiang, and Weixia Zou. 2017. Group secret key generation in physical layer, protocols and achievable rates. In *IEEE ISCIT*. Cairns, Australia.

[33] Joseph Teo and Chik Tan. 2005. Energy-efficient and scalable group key agreement for large ad hoc networks. In *Proceedings of the 2nd ACM international workshop on performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. 114–121.

[34] Tesla. 2021. Tesla Factory. https://www.tesla.com/factory.

[35] Chan Thai, Jemin Lee, and Tony Quek. 2015. Secret group key generation in physical layer for mesh topology. In *IEEE GLOBECOM*. San Diego, CA.

[36] Ma Thein and Thandar Thein. 2010. An energy efficient cluster-head selection for wireless sensor networks. In *IEEE ISMS*. Liverpool, UK, 287–291.

[37] Michael Tope and John McEachen. 2001. Unconditionally secure communications over fading channels. In *IEEE MILCOM*. McLean, VA.

[38] Malik Tubaishat, Jian Yin, Biswajit Panja, and Sanjay Madria. 2004. A secure hierarchical model for sensor network. *Sigmod Record* 33, 1 (2004), 7–13.

[39] Yunchuan Wei, Changmin Zhu, and Jun Ni. 2012. Group secret key generation algorithm from wireless signal strength. In *IEEE ICICSE*. Zhengzhou, China, 239–245.

[40] Mi Wen, Yan-Fei Zheng, Wen jun Ye, Ke-Fei Chen, and Wei-Dong Qiu. 2009. A key management protocol with robust continuity for sensor networks. *Computer Standards & Interfaces* 31, 4 (2009), 642–647.

[41] Yuezhong Wu, Qi Lin, Hong Jia, Mahbub Hassan, and Wen Hu. 2020. Auto-Key: Using autoencoder to speed up gait-based key generation in body area networks. *ACM, Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (2020), 1–23.

[42] Peng Xu, Kanapathippillai Cumanan, Zhiguo Ding, Xuchu Dai, and Kin Leung. 2016. Group secret key generation in wireless networks: algorithms and rate optimization. *IEEE Transactions on Information Forensics and Security* 11, 8 (2016), 1831–1846.

[43] Chunxuan Ye, Alex Reznik, and Yogendra Shah. 2006. Extracting secrecy from jointly Gaussian random variables. In *IEEE ISIT*. Seattle, WA.

[44] Jerrold Zar. 2005. Spearman rank correlation. *Encyclopedia of Biostatistics* 7 (2005).

[45] Junqing Zhang, Alan Marshall, Roger Woods, and Trung Duong. 2016. Efficient key generation by exploiting randomness from channel responses of individual OFDM subcarriers. *IEEE Transactions on Communications* 64, 6 (2016), 2578–2588.

[46] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. 2006. LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. *Transactions on Sensor Networks* 2, 4 (2006), 500–528.