Secure Broadcast Protocol for Unmanned Aerial Vehicle Swarms

Hongpeng Guo*, Tianyuan Liu*, King-Shan Lui†, Claudiu Danilov‡, Klara Nahrstedt*
*University of Illinois at Urbana Champaign, †The University of Hong Kong, ‡Boeing
Email: {hg5, tliu60, klara}@illinois.edu, kslui@eee.hku.com, claudiu.b.danilov@boeing.com

Abstract*—The technology advancement has made Unmanned Aerial Vehicle (UAV) swarm a promising method to achieve complicated missions that a single UAV cannot support.** ***Leader-followers formation **is a widely used swarm management scenario where a leader drone frequently broadcasts controlling messages to all follower drones to achieve collaboratively a common mission. However, managing such a UAV swarm, especially when the member drones dynamically join and leave the swarm, introduces significant security challenges and performance overhead.**

In this work, we propose a Swarm Broadcast Protocol (SBP) to facilitate the security protection of *leader-followers formation* **based UAV swarms. SBP contains a security key management scheme that manages a broadcast key among the swarm for leader to broadcast encrypted messages to followers. When swarm membership changes, the broadcast key will be updated and synchronized among the swarm to maintain both backward and forward secrecy. The overhead of SBP is small that only constant computational overhead is needed for both swarm leader and followers to achieve key synchronization when a new drone joins regardless of the current swarm size. This feature would highly reduce the overhead when there are many individual drone joining events. Through experiments on network emulator, we show that SBP achieves lowest bandwidth overhead and CPU utilization to handle multiple swarm membership changing events, comparing with two public-key-based swarm management protocol baselines.**

I. Introduction

The Unmanned Aerial Vehicles (UAVs), also known as drones, are aircrafts that fly without humans on board. UAVs are controlled either by remote controllers or on-board computer systems autonomously [1, 2]. Advances in technologies allow drones to carry sensing and communication units on board. They thus can interact with each other within a certain proximity, and fly as a group to accomplish a common mission. Such UAV groups are usually referred to as swarms [3]. Comparing to a single drone, UAV swarms are more powerful in many so-called Dull, Dirty, Dangerous (DDD) domains [4], such as rescue, surveillance [5], bridging wireless networks [6] and military operations, where a single drone cannot support every facet of the mission.

In order to collaborate consistently, swarms must be able to manage themselves in the sky with limited human intervention. For example, a swarm of UAVs is controlled by computer systems autonomously, or piloted by a single controller as a whole. *Leader-followers* mechanism is a widely used and investigated swarm management strategy [7, 8]. One UAV is a leader that, through autopiloting or remote control, determines the direction of the swarm and broadcasts commands to other swarm members (followers) periodically. The swarm members follow the commands and execute the tasks.

During the execution of a global task, it is quite common for drones to join and leave the current swarm due to battery limitation [1]. For example, a batch of drones may leave the swarm at a charging point to refill, while another group of UAVs join the swarm to continue executing the swarm mission. Geography restrictions may also be a reason resulting in dynamic swarm changes. The Federal Aviation Agency (FAA) has established a series of restricted fly zones at lower height over privacy-sensitive locations [9], where only drones satisfying regulated specs but not other drones could fly over. When flying across such restricted areas, unqualified drones must leave the swarm, and some qualified new drones will join the swarm to maintain the swarm functionality.

Despite all the benefits and potentials of UAV swarms, securing the in-swarm communication is a challenging problem, especially when the swarm membership is changing rapidly. Theoretically, eavesdropping and man-in-the-middle attacks can be prevented by encryption and signatures. However, if the keys used are not secure, the security primitives cannot be achieved. To avoid disclosing communication to previous or future swarm members requires that the encryption key must be immediately renewed and synchronized among the whole swarm. In this work, we consider the scenario where a swarm of UAVs work collaboratively on a mission along a trajectory. New drones could join the swarm anytime during the mission window individually. Some UAVs in the swarm may leave in a batch to do refilling at charging points on the path. We conclude three major challenges to realize key updates as follows.

- Key establishment and updates must be fast. In this way, newly join members can participate in the task sooner. By changing keys fast, newly broadcast messages encrypted using the new key can be sent out without much delay.
- Key updates must be reliable to tolerant unstable network conditions. The wireless links between drones are vulnerable to environment turbulence. Delay or loss of rekeying messages may result in drones' unavailability to decrypt any subsequent communication and being out of control.

We gratefully acknowledge the support of Boeing grant SSOW-BRT-Hl 117-0003. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

978-1-7281-6607-0/20/$31.00 ©2020 IEEE

- The key management scheme must be light-weight and efficient. The UAVs only embed very limited CPU and network resources. Expensive rekeying computation or bandwidth requirement would lead to drones communication failure when the swarm size scales up.

In this paper, we present the design and implementation of the Swarm Broadcast Protocol (SBP), which securely broadcasts the real-time control messages from the swarm leader to all the followers. At the beginning, SBP generates a swarm broadcast key and makes the key synchronized among the whole swarm. When membership changes, the leader will re-calculate a new symmetric broadcast key using SBP and broadcast a rekeying message to all valid members. SBP is efficient and lightweight in handling frequent membership changes. The computational overhead and message overhead for a drone joining remains constant regardless of the current swarm size, which outperforms many state-of-art broadcast encryption methods [10–13] with overhead linearly to the logarithm of swarm size. SBP is also resilient to unstable network environment of long delay and packet loss, and ensures that every packet sent through SBP can be successfully decrypted by any swarm member even several recently sent packets are lost. We outline three major contributions of the paper as below:

- We present the Swarm Broadcast Protocol which secures the broadcast communication of UAV swarms in Section IV-A, IV-B, and IV-C. The rekeying processing time for drone joining remains constant regardless of the swarm size, which makes SBP very scalable.
- We design a resilience handling mechanism in Section IV-D to maintain the resiliency of SBP in unstable network environments.
- We validate and compare the SBP with two baseline public key based protocols to show that SBP saves at least 40% network bandwidth and 57% CPU processing time when comparing with the two baselines V.

II. Related Works

To the best of our knowledge, secure broadcast protocol to facilitate control of UAV swarms with highly dynamic memberships has not been studied in the literature. Nevertheless, many works that study the following three problems independently can be identified. They are (1) General security and privacy issues related to UAVs, (2) Efficient broadcast encryption protocols for teleconference and real-time information services, and (3) Group or broadcast key management schemes that are resilient to unstable network environments.

Security and privacy problems are the major concerns for the pervasive deployment of UAV systems [14–16]. Various security challenges have been identified and discussed in the literature, such as eavesdropping, hijacking, Denial-of-Services, location forgery and GPS spoofing [17–19]. The work in [15] specifies different types of security issues and vulnerabilities of UAV wireless communication and suggests general defense methods. But they did not provide solution to efficiently secure the UAV swarm communication. Kamesh et al. proposed the scenario of dynamic swarm collaboration of drones owned by different authorities in work [1], but left out as an open problem how to secure the control communication of such swarms.

Broadcast encryption protocols are widely studied in network systems and cyberphysical systems, aiming to leverage the efficiency of broadcast communication while maintaining the confidentiality of broadcast contents [10–13, 20, 21]. By leveraging a tree-like key management schemes, the works [10–13] achieve computational and bandwidth overhead logarithm to the swarm size. However, in the scenario when swarm membership rapidly changes and drones joining dominates the dynamic events, a constant joining overhead protocol will be needed. The works [10–13] also require different rekeying messages being constructed and sent to different members, which leads to much overhead on the number of rekeying messages upon membership changes. A protocol which rekeys all swarm members with a single broadcast message will be more preferred.

Various methods have been developed to facilitate group or broadcast key management in unstable network conditions. [22] and [23] present reliable multicast protocols to ensure packet delivery, but the computational and bandwidth overhead are too large to be used by UAV swarms. Perrig et al. present an ELK [21] protocol which is resilient in unstable networks, and introduces very little overhead to the receiver side. However, this protocol assumes the sender has computational power strong enough as a server to handle significant workload, which is not feasible for the UAV swarm leader to adopt.

III. System Overview

A. System Model

The system consists a swarm of drones and an administrator which will configure the security primitives for all the drones. We denote the set of drones as $\mathcal{U} = \{u_0, u_1, u_2, \dots\}$. Let $\mathcal{M}_0$ be the initial swarm that contains some drones in $\mathcal{U}$. A *membership change* event is either a *single join* event or a *batch leave* event. When multiple drones join the swarm, their joining will be handled one by one. Membership change events happen in sequence and we label them using e_i, where $i > 0$ and e_i happened before e_j if $i < j$. A new swarm is formed after each membership change event. We let the set of drones in the swarm after event e_i be $\mathcal{M}_i$. Clearly, $\mathcal{M}_i \subseteq \mathcal{U}$ for all i.

Our secure broadcast scheme establishes a secure broadcast key for each $\mathcal{M}_t$. The key is generated based on a certain sequence among the drones in $\mathcal{M}_t$. Let m_i^t be the index of the ith drone in $\mathcal{M}_t$. That is,

$$\mathcal{M}_t = \{u_{m_i^t} | 0 \leq i < |\mathcal{M}_t|\}$$

For simplicity, we assume the leader remains in the swarm always and it is always the first drone in each sequence.

Example 1. *Suppose that $\mathcal{M}_0$, the initial swarm, contains drones u_0, u_2, u_5, and u_{10}. Let the sequence, denoted as M_0,*

be $[0, 5, 2, 10]$. *Then,* $m_0^0 = 0$, $m_1^0 = 5$, $m_2^0 = 2$, *and* $m_3^0 = 10$.

The index order of the sequence represents the order of drones joining the swarm. When a new drone u_j joins the swarm in the single join event e_t, swarm $\mathcal{M}_t$ is formed by including the new drone u_j in $\mathcal{M}_{t-1}$. That is, $\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{u_j\}$. The sequence M_t is formed by appending j to the end of M_{t-1}.

Example 2. *If* u_7 *joins the initial swarm in Example 1,* $\mathcal{M}_1$ *becomes* $\{u_0, u_2, u_5, u_7, u_{10}\}$ *and* $M_1 = [0, 5, 2, 10, 7]$.

When a batch of drones leaves the current swarm in event e_t, $\mathcal{M}_t$ becomes a subset of $\mathcal{M}_{t-1}$. The sequence M_t is formed by removing the elements representing the leaving drones from M_{t-1}.

Example 3. *If* u_2 *and* u_{10} *are leaving* $\mathcal{M}_1$ *in Example 2,* $\mathcal{M}_1$ *becomes* $\{u_0, u_5, u_7\}$ *and* M_2 *becomes* $[0, 5, 7]$.

B. Broadcast Key Generation

For each swarm $\mathcal{M}_t$, the broadcast key is generated according to the sequence M_t using a Diffie-Hellman chain (DH chain) mechanism. We first review the basic Diffie-Hellman mechanism that has been used in many Internet protocols to establish symmetric keys among a pair of communicating parties.

Alice and Bob want to talk to each other. They have agreed earlier on using g and p for DH key establishment. Alice keeps a secret of her own sk_a, and Bob keeps his secret sk_b. Alice sends Bob $T_a = g^{sk_a} \bmod p$ and Bob sends Alice $T_b = g^{sk_b} \bmod p$. After receiving $T_a = g^{sk_a} \bmod p$ from Alice, Bob computes $(T_a)^{sk_b} \bmod p$. Similarly, Alice computes $(T_b)^{sk_a} \bmod p$. Because,

$$(T_a)^{sk_b} \bmod p = (T_b)^{sk_a} \bmod p = g^{sk_a sk_b} \bmod p,$$

Alice and Bob have established a shared key among themselves. Although eavesdroppers can obtain T_a and T_b, they cannot compute the shared secret. This Diffie-Hellman mechanism has been widely used in many standard security protocols.

In securing communication among a group of drones, we use the Diffie-Hellman chain mechanism as shown in Figure 1 to establish a shared group key. Before any key establishment, all drones agree on g and p to be used. Let the sequence of drones in a certain swarm be u_0, u_1, ..., u_k, where u_0 is the leader. We further assume sk_i is the secret of u_i and denote $g^{sk_i} \bmod p$ be T_i. Let I_0 be sk_0. $I_i = T_i^{I_{i-1}} \bmod p$. I_k will be the shared broadcast key K.

We now briefly explain how the leader and the followers, other drones in the swarm, establish the shared key based on the information they have. More details of the protocol can be found in Section IV. The leader, u_0, has $I_0 = sk_0$. It also has the group membership information to determine the sequence of the chain. By collecting g^{sk_i} from each follower, u_0 can compute K by computing I_i one by one. For follower u_i, it has sk_i. It can compute I_i by $(g^{I_{i-1}})^{sk_i} \bmod p$. With I_i and g^{sk_j}, where $j > i$, K can be computed. To simplify our discussion, we call $g^{I_{i-1}}$ *blind keys generated by the leader*. In Section IV, we will explain how the followers acquire the required information.

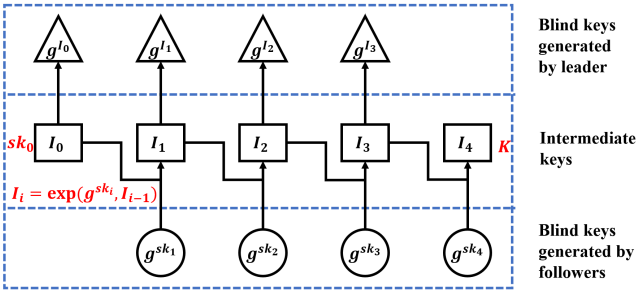


Fig. 1: Overview of Broadcast Key Generation

Notations	Description
sk_i	Secret key assigned to drone u_i by the administrator.
T_i	Blind key of u_i, can be calculated from sk_i such that $T_i = \exp(g, sk_i) \bmod p$
K_t	The broadcast key used in swarm $\mathcal{M}_t$
B_j^t	The Blind key of the j-th drone in swarm sequence M_t. That is, if u_i is the j-th drone in M_t, we have $B_j^t = T_i$.
I_j^t	The j-th intermediate key of the Diffie-Hellman chain for swarm $\mathcal{M}_t$
S_0, V_0	The signing and verification key pair of the swarm leader.
S_{ad}, V_{ad}	The signing and verification key pair of the administrator.

TABLE I: Notations used by our protocol

C. Authentication

To prevent man-in-the-middle attack, swarm access control and key update information must be authenticated properly. That is, (1) the leader must verify that a new joining drone is sent by the administrator before include it into the swarm, (2) the followers must verify that the key update information comes from the leader before they update their keys accordingly. We introduce two extra asymmetric sign-verify key pairs, (S_{ad}, V_{ad}) and (S_0, V_0) for system authentication, which are the sign-verify key pairs for administrator and swarm leader respectively.

Every drone will be provided V_{ad} and V_0 by the administrator and they can thus verify if a message is singed by administrator or swarm leader during the mission. The swarm leader will also be equipped with S_0 to sign all the key update information before sending it to the followers. Further more, the administrator will use S_{ad} to sign the join message of every follower before they take off, so that the swarm leader is able to verify if the joining drone is actually sent by the administrator when they receive a joining request. Details of the authentication process will be provided in Section IV.

IV. Swarm Broadcast Protocol

In this section, we present the design details of the Swarm Broadcast Protocol. We present the set of keys used in our protocol in Table I.

A. Initial Swarm Formation

1) Protocol Details Walk Through. Before the drones can fly and work on the mission, they must be configured by an administrator to obtain the security primitives. Drone u_i will be configured with g, p, and sk_i. g and p are the constants used in the Diffie-Hellman scheme described in Section III-B. sk_i is the secret key of u_i. Using sk_i, u_i can generate its blind key as $T_i = g^{sk_i} \bmod p$. Every drone will also be provided key V_{ad} to verify any administrator signed message during the mission.

At the beginning of the task, the administrator will identify a leader drone u_0 and form an initial swarm $\mathcal{M}_0$ at the mission trajectory starting point. The leader u_0 will be further equipped with a singing key S_0 to sign its broadcast messages in the swarm. Administrator provides every other drone V_0 to verify communications from the leader.

The administrator will inform u_0 about the membership of the initial swarm by sending it the unique swarm sequence M_0 as well as the blind keys of all the swarm followers, which are $\{T_i | i \in M_0 \backslash \{0\}\}$. In Example 1, to allow all drones in $\mathcal{M}_0$ to establish a shared key, the administrator sends u_0 a message containing

$$\mathsf{Sign}_{S_{ad}}\Big(\big[0,5,2,10\big] \;||\; \{T_5, T_2, T_{10}\}\Big)$$

With the swarm sequence and blind keys of all the followers, the leader applies the Diffie-Hellman chain mechanism to develop a broadcast encryption key K_0. To allow each follower to compute K_0 on its own, the leader distributes (1) the swarm sequence M_0, (2) all T_i of the followers, and (3) the blind keys generated by the leader. All messages sent from leader must be signed by S_0 for authentication.

To differentiate the Diffie-Hellman chains for different swarms after different events at different points of time, we augment the relevant symbols used earlier to include superscripts for representing time. Formally, for swarm $\mathcal{M}_t$ with sequence $M_t = [m_0^t, m_1^t, ..., m_{|M_t|-1}^t]$ after event e_t, the intermediate keys in the Diffie-Hellman chain are I_i^t where $0 \le i < |M_t|$. I_0^t is sk_0 for all t. The blind keys generated by the followers at t are $B_i^t = g^{sk_{m_i^t}}$ for $0 < i < |M_t|$, and that the intermediate keys can be computed as

$$I_i^t = \exp\left(B_i^t, I_{i-1}^t\right) \bmod p$$

Figure 2 presents the Diffie-Hellman chain for Example 1. After obtaining $B_1^0 = g^{sk_5} = T_5$, $B_2^0 = g^{sk_2} = T_2$ and $B_3^0 = g^{sk_{10}} = T_{10}$, the leader u_0 will calculate all the intermediate keys I_i^0's using the Diffie-Hellman chain mechanism one by one, where $1 \le i \le 3$. The last intermediate key I_3^0 along the key chain is used as the broadcast encryption key K_0.

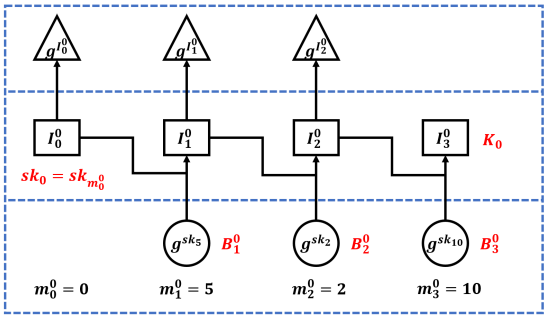


Fig. 2: Diffie-Hellman Chain Generated for Example 1

After K_0 is established, the leader will send the key information to the followers to calculate K_0 on their own. In Example 1, the key information broadcast by u_0 should be,

$$\mathsf{Sign}_{S_0}\Big(\big[0,5,2,10\big] \;||\; \{T_5, T_2, T_{10}\} \;||\; \{g^{I_0^0}, g^{I_1^0}, g^{I_2^0}\}\Big)$$

As a follower of the swarm, a drone u_i is able to calculate the broadcast key using the key information and its own secret key sk_i. In Example 1, u_2 will first check the swarm sequence M_0 to obtain its position in the key chain. As 2 is the third element in the swarm sequence, $m_2^0 = 2$. u_2 thus has to compute I_2^0 and it requires $g^{I_1^0}$, which is the second element of the leader generated blind keys. u_2 then calculates the third intermediate key as

$$I_2^0 = \exp\left(g^{I_1^0}, sk_2\right) \bmod p$$

After I_2^0 is obtained by u_2, it is easy for it to calculate the broadcast key

$$K_0 = I_3^0 = \exp\left(B_3^0, I_2^0\right) = \exp\left(T_{10}, I_2^0\right) \bmod p$$

Note that all the drones in the swarm will keep the key information in their memory to facilitate computation when drones join or leave the swarm later. Specifically, the leader will store all the intermediate keys I_i^0 for $0 \le i < |\mathcal{M}_0|$, the followers' blind keys B_i^0 for $1 \le i < |\mathcal{M}_0|$, and $g^{I_i^0}$ for $0 \le i < |\mathcal{M}_0| - 1$. A follower will keep the leader distributed key information in its memory as well as all the intermediate keys it computed. In Example 1, u_2 will keep I_2^0 and I_3^0 in its memory for later use.

After the followers have calculated the broadcast encryption key, they will be able to decrypt the encrypted control messages sent by the leader and fly to execute the mission.

2) Computation and Communication Overhead. It is easy to observe that the computational complexity for the leader to calculate the broadcast key is linear to the number of followers in the initial swarm. Specifically, the number of heavy exponential operations is $2(|\mathcal{M}_0| - 1)$, half of which calculate I_i^0s, for $1 \le i < |\mathcal{M}_0|$ and another half calculate $g^{I_j^0}$s, for $0 \le j < |\mathcal{M}_0| - 1$.

The computational overhead for a follower u_i to calculate the broadcast key from the key information varies according to its position in the swarm sequence. In the worst case when u_i is the first follower in the sequence, it needs to execute

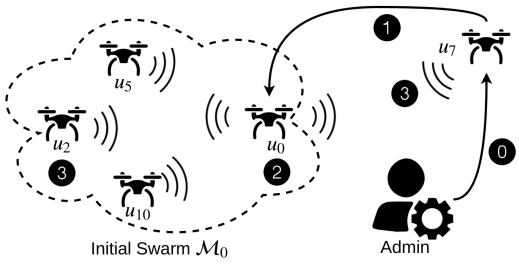


Fig. 3: Drone Join Event on Example 2

$|\mathcal{M}_0| - 1$ exponential operations to derive K_0. On the other hand, if u_i is the last drone in the sequence, it can directly calculate K_0 by a single exponentiation.

Overall, we show that the computational complexity for a drone in the swarm initialization phase is $\mathcal{O}(|\mathcal{M}_0|)$. As the swarm initialization happens only once before the drones fly to the sky, this overhead will not influence the swarm performance.

The communication overhead for swarm initialization depends on the length of key information sent from the leader to followers, which contains (1) M_0, (2), B_i^0 for $1 \leq i < |\mathcal{M}_0|$, and (3) $g^{I_j^0}$ for $0 \leq j < |\mathcal{M}_0| - 1$. We interpret the message length as the number of cryptography keys being transmitted, which is $2(|\mathcal{M}_0| - 1)$. We can thus conclude the communication overhead for swarm initialization is $\mathcal{O}(|\mathcal{M}_0|)$. Note that the key information distributed to all followers is the same. That is, no matter how large the swarm is, a single broadcast message is sufficient for all followers to establish the same shared key.

B. Join Event

Suppose event e_t is a join event where drone u_j joins the swarm. $M_t = [m_0^t, m_1^t, ..., m_{|M_t|-1}^t] = [m_0^{t-1}, m_1^{t-1}, ..., m_{|M_{t-1}|-1}^{t-1}, j]$. The Diffie-Hellman chain of M_t can be constructed from the one of M_{t-1} by appending one more pass. Given $I_{|M_{t-1}|-1}^{t-1} = K_{t-1}$ which is the shared key of swarm $\mathcal{M}_{t-1}$, the shared key of $\mathcal{M}_t = I_{|M_t|-1}^t$ can be computed by $\exp(g^{sk_j}, K_{t-1}) \bmod p$. This operation takes only constant time and is independent of the size of the current swarm.

1) Protocol Details Walk Through. When the swarm is flying along the mission trajectory, a new drone could join the swarm at any time in the mission window. As shown in figure 3, the join event is triggered by the joining drone, denoted as u_j, to send a join message containing j and T_j to the swarm leader (step). Refer to example 2, when u_7 joins the swarm, it will send a join message containing $\mathsf{Sign}_{S_{ad}}(7 \mid\mid T_7)$ to the swarm leader u_0. Note that the join message is singed by the administrator before the follower drone takes off (step), so that the leader can verify a valid joiner upon join event. After receiving the join message and authentication, the swarm leader should update the broadcast key for the whole swarm to maintain backward secrecy.

Figure 4 presents the Diffie-Hellman chain for Example 2. As the join event of u_7 will only append a new pass to

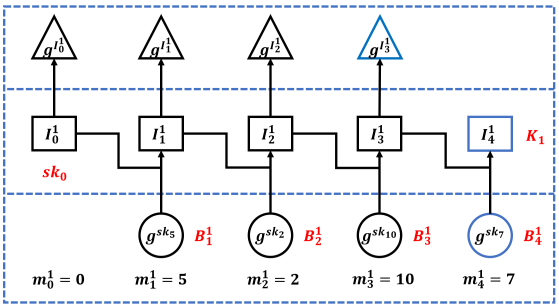


Fig. 4: Diffie-Hellman Chain Generated for Example 2

the original Diffie-Hellman chain structure, all the existing intermediate and blind keys will remain the same after the join event has happened, that is, $I_i^1 = I_i^0$ for $0 \leq i \leq 3$, $g^{I_i^1} = g^{I_i^0}$ for $0 \leq i \leq 2$ and $B_i^1 = B_i^0$ for $1 \leq i \leq 3$. After obtaining $B_4^1 = g^{sk_7}$, u_0 will only process two exponential operations to update the key information, which are, $K_1 = I_4^1 = \exp(B_4^1, I_3^1) \bmod p$ and $g^{I_3^1} = \exp(g, I_3^1) \bmod p$.

As K_1 is established, the leader distributes the key information to all followers, including the new drone u_7, to update the broadcast key to be K_1 on their own (step). Suppose u_j joins the swarm in event e_t, leader sends out

$$\mathsf{Sign}_{S_0}\left(j \;\mid\mid\; T_j \;\mid\mid\; g^{I_{|M_t-2|}^t}\right)$$

to all followers. In Example 2, the key information being distributed is

$$\mathsf{Sign}_{S_0}\left(7 \;\mid\mid\; T_7 \;\mid\mid\; g^{I_3^1}\right)$$

An existing follower can update the broadcast key using T_j. In Example 2, u_2 is a swarm member before u_7 joining the swarm, and knows K_0, the original broadcast key. Then u_2 can calculate the new broadcast key as

$$K_1 = \exp(T_7, K_0) \mod p$$

For the new joining drone u_7, it will calculate the broadcast key as

$$K_1 = \exp(g^{I_3^1}, sk_7) \mod p$$

With the above process, all the old and new swarm members will use K_1 as the shared broadcast key for subsequent communication encryption. The followers will finally acknowledge the swarm leader after their keys are updated (step).

2) Computation and Communication Overhead. As we mentioned in Section IV-B1, the number of exponential operations for the swarm leader to add a new drone to the swarm is always 2 no matter how large the swarm size is. For a follower, regardless it is the new drone or an existing one, the computational overhead to calculate the new shared key is always 1 cryptographic exponentiation. Therefore, we show that the computational overhead for a join event is $\mathcal{O}(1)$, which is independent of the swarm size.

The communication overhead for a join event depends on the length of key information sent from the leader to followers

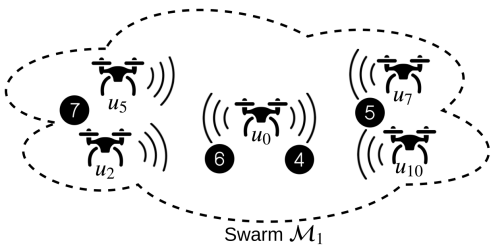


Fig. 5: Drone Leave Event on Example 3

to update the shared broadcast key. The key information for a drone joining only contains two cryptographic keys, which are (1) the blind key of the joining drone T_j, and (2) the blind version of the broadcast key before u_j joins the swarm. Therefore, we show that the communication overhead for a join event is $\mathcal{O}(1)$.

C. Batch Leave Event

Now we consider a batch leave event e_t where one or more drones leave the swarm. $\mathcal{M}_t \subset \mathcal{M}_{t-1}$ and $M_t \subset M_{t-1}$. Note also that $\{B_i^t | 0 < i < |M_t|\} \subset \{B_i^{t-1} | 0 < i < |M_{t-1}|\}$. Therefore, the leader can compute the shared key for $\mathcal{M}_t$ by going through the Diffie-Hellman chain without acquiring any new information.

1) Protocol Details Walk Through. There are several scenarios where one or more drones leave the swarm. First, a drone may fail in the middle of the journey. In this case, the leader should exclude it from the swarm and establish a different broadcast key after detecting such failure. A drone, on the other hand, can initiate leaving the swarm. The drone should send a message to the leader and the leader can act accordingly.

In this paper, we also consider situations that multiple drones leave the swarm at the same time. While the swarm is flying along the mission path, the leader will collect the real-time battery readings from the followers periodically. When the swarm is approaching a charging point, the leader will choose and inform a subset of drones to leave the swarm to refill at the charging points. Let e_t be a batch leaving event. We denote $\mathcal{L}_t$ as the set of drones leaving the swarm $\mathcal{M}_{t-1}$ at event e_t, such that $\mathcal{L}_t = \mathcal{M}_{t-1} - \mathcal{M}_t$.

This batch leave event is triggered by swarm leader u_0 sending of a leave message to the drones in $\mathcal{L}_t$ (step). The leave message can contain instruction to inform what a leaving drone should do. After receiving acknowledgments from the leaving drones (step), the leader must update the broadcast key and synchronize it with the remaining swarm members as soon as the batch leave event happens.

Refer to the batch leave event in example 3. As u_2 and u_{10} leave the swarm, the swarm contains only three drones after the batch leaving, that is $M_2 = [0, 5, 7]$. Because drone leaving breaks the original structure of the DH chain, u_0 has to reestablish the key chain following the same way as in Section IV-A. Note that although the key chain is reconstructed, some previous results can be used to reduce computation. In the key chain generated in example 3, I_0^2 and I_1^2 are the same as I_0^1 and I_1^1, respectively. The only exponential computation needed in the key update process is

$$K_2 = I_2^2 = \exp\left(g^{sk_7}, I_1^2\right) \bmod p$$

In general, on a batch leave event e_t, let k be the position index of the first leaving drone along swarm sequence M_{t-1}, $1 \leq k$. We can observe that the first k passes of the DH chain remain the same on the leave event. Reestablishing the DH chain only requires calculating the remaining $|\mathcal{M}_t| - k$ intermediate keys and $|\mathcal{M}_t| - k - 1$ blind intermediate keys.

When the new key is generated, the leader would distribute the key information for every staying follower of the swarm $\mathcal{M}_t$ to calculate the new broadcast key K_t (step). The key information is

$$\mathsf{Sign}_{S_0}\Big(\left[m_{k-1}^t, \ldots m_{|\mathcal{M}_t|-1}^t\right] \;||\; \left\{B_k^t, \ldots, B_{|\mathcal{M}_t|-1}^t\right\} \;||\; \left\{g^{I_{k-1}^t}, \ldots, g^{I_{|\mathcal{M}_t|-2}^t}\right\}\Big)$$

For the first $k-1$ followers in M_t, I_{k-1}^t is a common knowledge to them. They can derive all intermediate keys after I_{k-1}^t one by one and thus obtain K_t. For the last $\mathcal{M}_t - k$ followers, they will derive K_t using the same method described in Section IV-A using their secret keys.

In Example 3, because u_2 is the first leaving drone from M_1 and that $m_2^1 = 2$, we can get $k = 2$. The key information being distributed is

$$\mathsf{Sign}_{S_0}\Big(\left[5, 7\right] \;||\; \left\{B_2^2 = T_7\right\} \;||\; \left\{g^{I_1^2}\right\}\Big)$$

For follower u_5, it knows I_1^2 and then it can obtain $K_2 = \exp\left(g^{sk_7}, I_1^2\right) \bmod p$. For follower u_7, it can calculate the new broadcast key $K_2 = \exp\left(g^{I_1^2}, sk_7\right) \bmod p$. All staying swarm members thus can communicate securely using K_2. The followers will finally keep the leader informed that their keys are updated (step).

2) Computation and Communication Overhead. As we mentioned in Section IV-C1, the number of exponential operations for the swarm leader to handle batch leave depends on the position index k of the first leaving drone along the swarm sequence. In general, the number of exponential operations for the leader of swarm $\mathcal{M}_t$ after a batch leave event e_t is $\max\left(0,\ 2(|\mathcal{M}| - k) - 1\right)$ for $1 \leq k \leq |\mathcal{M}|$. Overall, the worst case computational overhead for swarm leader is $\mathcal{O}(|\mathcal{M}|)$. Similarly, the worst case computational overhead for followers to update the key is also $\mathcal{O}(|\mathcal{M}|)$.

The number of keys contained by the key information is also proportional to k and the swarm size, which is $\max\left(0,\ 2(|\mathcal{M}| - k) - 1\right)$ for $1 \leq k \leq |\mathcal{M}|$. The worst case communication overhead for batch leave events is $\mathcal{O}(|\mathcal{M}|)$.

D. Resilience Handling

UAV swarm is a very dynamic structure even if the group membership does not change, because the distance and communication channel quality between two UAVs can change very frequently. That is, the wireless links among drones can

be very unreliable that packets may be lost [1]. It is thus very important for a UAV protocol to be resilient in unstable network circumstances.

Our protocol distributes key information through a single broadcast message in each re-keying. To distinguish the keys used after different events, the leader also inserts a version number in the message. A follower who receives the message should acknowledge the leader (step and). The leader then knows whether all followers have received the new key information successfully. If not all followers have received the message, the key information will be appended in a regular broadcast message that is encrypted using the new key. Note that the same key information will be appended no matter how many followers and which followers have not received the key information. Followers who have not received the new key information earlier can still decrypt the message using the key information sent with the encrypted message.

For example, suppose u_5 and u_{10} did not receive the new key information sent for the join event of u_7 in Example 2. As leader u_0 did not get the acknowledgement from u_5 and u_{10}, when it sends out a new broadcast message, while it uses the new key K_1 to encrypt, it also sends out the key information. Then, when u_{10} receives the new message, it knows how to develop the new key and decrypt the message.

It is possible that a follower still has not received the new key information before another event occurs. That is, in Example 3, by the time u_2 and u_{10} leave, u_5 still has not received the key information for K_1. In this case, the leader can decide whether it still wants u_5 to stay. If so, the leader can append all necessary key information to a message. If the leader has decided to drop u_5, it can distribute a new key without including u_5 as a member.

It is worth noting that as our protocol uses broadcast messages to relay key information, the overhead in handling message loss is very minimal when compared with other protocols that are not designed specifically for UAVs. Protocols that require different messages to be constructed for different followers would have very high overhead when the network is unstable.

V. Simulations & Experiments

In this section, we first show the fast execution time and short rekeying message length of the Swarm Broadcast Protocol (SBP) in updating the broadcast keys among a UAV swarm. We then apply SBP in a swarm management scenario of 90 seconds with multiple UAV mobility events using CORE network emulator [24, 25] to demonstrate that SBP handles the dynamic swarm changes with pretty low bandwidth overhead and CPU utilization. Before going into the details of our experiment designs, we first introduce two baseline protocols to compare SBP with as below.

A. Two Baseline Protocols

The first baseline approach is the Public Key Broadcast Protocol (PKB). PKB assigns every drone an asymmetric key pair. The swarm leader has the knowledge of public keys of all the follower drones. PKB generates a random symmetric broadcast key to encrypt broadcast contents. The symmetric broadcast key is securely distributed to every individual follower by being encrypted using the public key of the follower. When a drone joins or leaves the swarm, a new symmetric key must be generated and synchronized to the followers using asymmetric key infrastructure to maintain forward and backward secrecy.

The second baseline method is the Public Key Unicast Protocol (PKU). Similar to PKB, every drone will be assigned an asymmetric key pair. But the leader will maintain a unique symmetric communication key with every single follower. All the communication between the swarm leader and followers happens in a P2P manner. With the PKU protocol, broadcast key synchronization is not needed upon dynamic swarm events. Nevertheless, content delivery will consume much more computation and bandwidth resources. A single message broadcast is actually multiple unicast sessions for PKU.

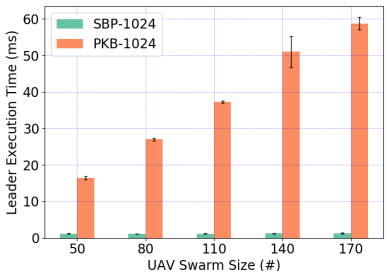
B. Key Synchronization Evaluation

As the major overhead of the key management protocols appears only when the swarm membership dynamically changes and the leader drone synchronizes the symmetric broadcast key among the swarm, we only compare the performance of SBP and PKB on dynamic swarm events. We are interested in three metrics: (1) computational overhead for key synchronization on swarm leader, (2) computational overhead on the followers, and (3) total key information length sent from the swarm leader to followers for updating the broadcast key.

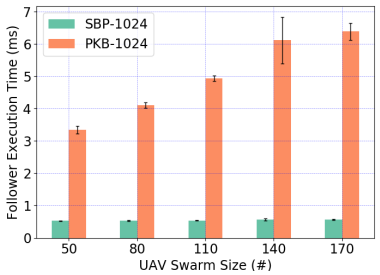
We initiate a UAV swarm containing M drones, of which one is the swarm leader and the others are followers. At the beginning of the experiment, the broadcast key is synchronized among the swarm. To evaluate the performance of the broadcast protocols, we introduce two types of dynamic events, which are single drone join event and a batch of K drones leaving. In our simulation, We choose RSA as the asymmetric encryption algorithm used in PKB and set the same key size for SBP and PKB to maintain comparable security strength. For either protocol, the symmetric broadcast key size is set to be 256-bit to work with AES-256 encryption method. We further choose 1024-bit RSA key pairs for authentication on both two protocols.

We measure the computational time and message length when two different events occur. In both events, the key size is 1024 bits for both SBP and PKB. For Event I, we set 5 initial swarm sizes as 50, 80, 110, 140 and 170 to measure the results when a new drone joins these swarms. For Event II, the initial 5 swarm sizes are the same as those of event I, but we measure the results when a batch of 10 drones leave the swarm.

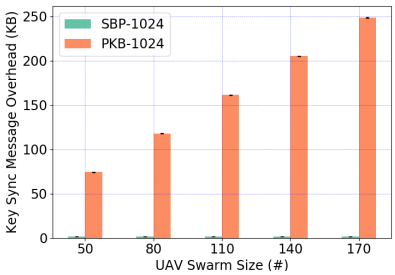
In Fig. 6a and Fig. 6b, we show the computational overhead for broadcast key synchronization of swarm leader and follower, respectively, when a new drone joins the existing swarm. It can be observed that the SBP introduces only constant execution overhead for both leader and follower drones, while the processing time for a PKB leader to update



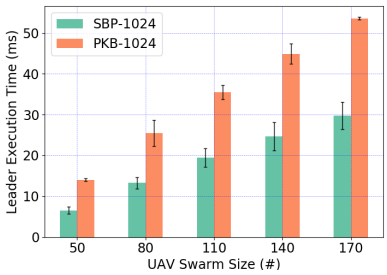
(a) Leader computational overhead (I)



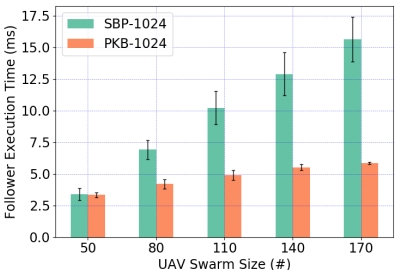
(b) Follower computational overhead (I)



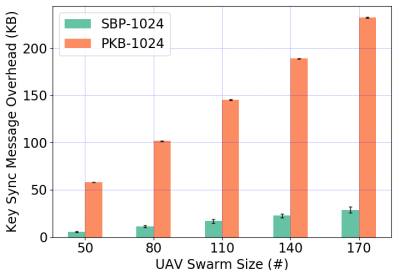
(c) Message length for key synchronization (I)



(d) Leader computational overhead (II)



(e) Follower computational overhead (II)



(f) Message length for key synchronization (II)

Fig. 6: Computational and Message Overheads for Key Synchronization

the broadcast key grows linearly with the swarm size and is significantly longer than SBP. In Fig. 6c, we compare the key information length sent from leader to followers to update the broadcast key when drone joining happens. As the figure shows, SBP will generate rekeying message of almost constant length regardless of the swarm size, while PKB will generate much longer rekeying message, whose length increases linearly with the swarm size.

In Fig. 6d and Fig. 6e, we show the computational overhead of the swarm leader and a follower, respectively, when drone leaving happens. It can be observed that the performance of SBP is better than PKB for the swarm leader in all five initial sizes. A SBP follower may spend more time than PKB to update the broadcast key on drones leaving. Nevertheless, the average execution time for a SBP follower is much short than the leader and is less than 20ms even the swarm size is 170. The execution overhead for a SBP follower is manageable with swarm size scaling up. Fig. 6f suggests that SBP can generate much shorter key information to synchronize the broadcast key than PKB under the batch leave scenario.

C. Swarm Management Scenario Evaluation

In this experiment, we implement a UAV swarm using the CORE network emulator [24, 25] and introduce multiple dynamic events to the swarm during a time window of 90 seconds. We compare SBP with two baseline protocols, PKB and PKU, to demonstrate that our Swarm Broadcast Protocol is efficient and lightweight in managing a dynamic UAV swarm.

Time (s)	15	[25, 45]	55	[65, 85]
Dynamic Events	5 join	1 join/ 2s	10 leave	1 join/ 2s

TABLE II: Swarm dynamic events

The swarm contains 10 drones in the beginning with one of them being the swarm leader. The leader sends an encrypted 8 KB control message to every swarm follower every second to realize the swarm control. We measure the network data and CPU utilization for the swarm leader in 90 seconds to evaluate the overhead caused by the swarm management process. We set the key size of all three protocols to be 1024-bit in this experiment.

During the 90s swarm management, we inject multiple dynamic events as shown in Table II. During the first 15 seconds, the swarm remains stable and the broadcast key is synchronized among all the UAVs. At time 15s, five drones join the swarm at the same time, and the swarm size becomes 15. During interval [25s, 45s], a new drone joins the swarm every two seconds. At the 55s moment, ten drones leave the swarm. The last dynamic events happen between 65s to 85s, when ten drones join the swarm one by one every two seconds. We present the bandwidth and CPU utilization measurement results of the swarm leader in Fig. 7.

Fig. 7a presents the swarm size vs time. Fig. 7b shows the bandwidth overhead of the three protocols. It can be observed that the overhead of PKU is significantly higher than the other two methods and dominates in the figure due to its unicast nature. To better observe the difference of SBP and

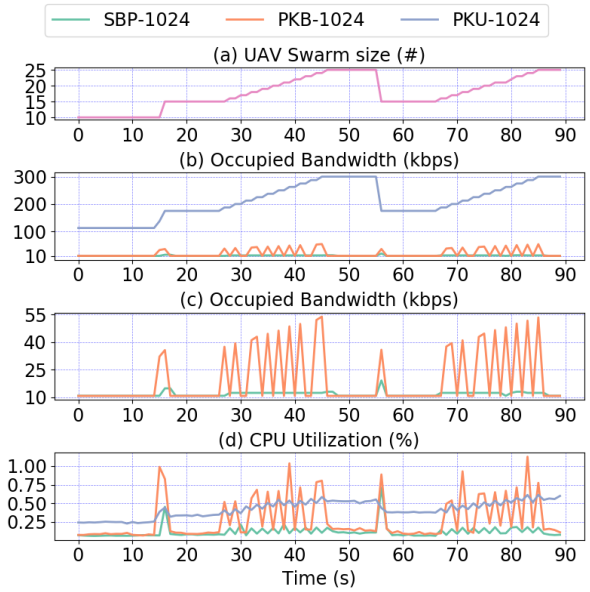


Fig. 7: Swarm Management Scenario Evaluation Results

PKB, we plot the bandwidth overhead of SBP and PKB in Fig. 7c. As Fig. 7c suggests, SBP and PKB occupy similar bandwidth when the swarm is stable, but PKB will consume much more bandwidth when drones join or leave the swarm. The bandwidth consumed by SBP, on the other hand, is almost constant and close to the 8 kbps baseline even when the swarm is highly dynamic.

Fig. 7d shows the results of CPU utilization measurements. As PKU protocol has to encrypt the control message for each follower using the pair-wise symmetric key, it has much higher CPU utilization than the other two methods. PKB maintains low CPU utilization when the swarm is stable, but occupies much more computational resources when dynamic events happen. SBP maintains pretty low CPU utilization either when swarm membership changes or when the swarm is stable.

We also apply SBP in unstable networks to show that our resilience handling method is efficient and introduces very little bandwidth overhead. We set the packet loss rate to be 0%, 10%, 20% and 30%, respectively, and measure the occupied network bandwidth of SBP through the network emulator. Through measurements, we get the average bandwidth occupation of the four network conditions to be 11.71 kbps, 12.11 kpbs, 12.31 kbps and 12.86 kbps. We show that the extra bandwidth overhead for resilience handling is pretty small (within 10%) even the packet loss rate is as high as 30%.

VI. Conclusion

In this work, we consider the broadcast security issue in UAV swarms with dynamic memberships. We designed the Swarm Broadcast Protocol, which manages the broadcast key to maintain forward and backward secrecy and is resilient to unstable wireless network conditions. Consequently, our protocol facilitates secure management of the *leader-followers formation* UAV swarms with low communication and computation overhead.

References

[1] K. Namuduri, S. Chaumette, J. H. Kim, and J. P. Sterbenz, *UAV Networks and Communications*. Cambridge University Press, 2017.

[2] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, and M. Debbah, "A tutorial on uavs for wireless networks: Applications, challenges, and open problems," *IEEE Communications Surveys & Tutorials*, 2019.

[3] H. Shakhatreh, A. Khreishah, J. Chakareski, H. B. Salameh, and I. Khalil, "On the continuous coverage problem for a swarm of uavs," in *2016 IEEE 37th Sarnoff Symposium*.

[4] D. of Defense, "Unmanned aerial vehicle roadmap 2000-2025," Department of Defense, 2001.

[5] Y. Yang, Z. Hu, K. Bian, and L. Song, "Imgsensingnet: Uav vision guided aerial-ground air quality sensing system," in *IEEE INFOCOM*, 2019.

[6] M. Moradi, K. Sundaresan, E. Chai, S. Rangarajan, and Z. M. Mao, "Skycore: Moving core to the edge for untethered and reliable uav-based lte networks," in *ACM MobiCom*, 2018.

[7] Z. Liu, X. Yu, C. Yuan, and Y. Zhang, "Leader-follower formation control of unmanned aerial vehicles with fault tolerant and collision avoidance capabilities," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE.

[8] M. Chen, F. Dai, H. Wang, and L. Lei, "Dfm: A distributed flocking model for uav swarm networks," *IEEE Access*, 2018.

[9] Faa summary of small unmanned aircraft rule (part 107). [Online]. Available: https://goo.gl/JbpgST

[10] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM transactions on networking*, 2000.

[11] W. He, Y. Huang, R. Sathyam, K. Nahrstedt, and W. C. Lee, "Smock: a scalable method of cryptographic key management for mission-critical wireless ad-hoc networks," *IEEE Transactions on Information Forensics and Security*, 2009.

[12] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *ACM CCS*, 2000.

[13] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees," in *ACM CCS*, 2018.

[14] A. Fotouhi, H. Qiang, M. Ding, M. Hassan, L. G. Giordano, A. Garcia-Rodriguez, and J. Yuan, "Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges," *IEEE Communications Surveys & Tutorials*, 2019.

[15] D. He, S. Chan, and M. Guizani, "Communication security of unmanned aerial vehicles," *IEEE Wireless Communications Magazine*, 2017.

[16] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, and M. Alam, "Cyber security threat analysis and modeling of an unmanned aerial vehicle system," *2012 IEEE Conference on Technologies for Homeland Security (HST)*.

[17] M. Hooper, Y. Tian, R. Zhou, B. Cao, A. P. Lauf, L. Watkins, W. H. Robinson, and W. Alexis, "Securing commercial wifi-based uavs from common security attacks," in *IEEE MILCOM*, 2016.

[18] E. Vattapparamban, İ. Güvenç, A. İ. Yurekli, K. Akkaya, and S. Uluağaç, "Drones for smart cities: Issues in cybersecurity, privacy, and public safety," in *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*.

[19] T. Liu, A. Hojjati, A. Bates, and K. Nahrstedt, "Alidrone: Enabling trustworthy proof-of-alibi for commercial drone compliance," *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*.

[20] A. T. Sherman and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE transactions on Software Engineering*, 2003.

[21] A. Perrig, D. Song, and D. Tygar, "Elk, a new protocol for efficient large-group key distribution," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*.

[22] X. R. Xu, A. C. Myers, H. Zhang, and R. Yavatkar, "Resilient multicast support for continuous-media applications," in *NOSSDAV'97*. IEEE, 1997.

[23] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *ACM SIGCOMM Computer Communication Review*, 1995.

[24] Core, common open research emulator, url = http://coreemu.github.io/core/, urldate = Oct, 2018.

[25] N. Suri, A. Hansson, J. Nilsson, P. Lubkowski, K. Marcus, M. Hauge, K. Lee, B. Buchin, L. Mısırhoğlu, and M. Peuhkuri, "A realistic military scenario and emulation environment for experimenting with tactical communications and heterogeneous networks," in *2016 ICMCIS*. IEEE.