Scalable Key Management for Tactical Swarms

by

Hunter C. Gatewood

B.S. in Computer Science and Engineering, MIT (2017)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by .....
Roger Khazan
Group Leader, MIT Lincoln Lab
Thesis Supervisor

Certified by .....
David A. Wilson
Technical Staff, MIT Lincoln Lab
Thesis Supervisor

Certified by .....
Ben Nahill
Technical Staff, MIT Lincoln Lab
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Scalable Key Management for Tactical Swarms

by

Hunter C. Gatewood

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Swarming is a powerful pattern allowing reliable, efficient, and economical task execution, identified as a key military technology by the Department of Defense. However, despite expectations that this evolving space will grow to encompass a diverse set of military and civil applications, minimal consideration has been given to the design or integration of schemes providing flexible and efficient secure group communication for tactical swarms. In remedy, we survey relevant literature to provide a top-down, coherent story for secure swarm and UxS communication, with emphasis on cryptographic key management. We propose a taxonomy of swarm needs, identify candidate key management schemes, extend a key management architecture via scalability considerations, and begin a prototype of the augmented architecture.

Thesis Supervisor: Roger Khazan
Title: Group Leader, MIT Lincoln Lab

Thesis Supervisor: David A. Wilson
Title: Technical Staff, MIT Lincoln Lab

Thesis Supervisor: Ben Nahill
Title: Technical Staff, MIT Lincoln Lab

Acknowledgments

A sincere thank you to everyone who supported and influenced this work, and a special thank you to the following groups and people.

To MIT Lincoln Laboratory, Division 5, which funded the research assistantship that led to this thesis.

To my thesis supervisors, Roger Khazan, David A. Wilson, and Ben Nahill, for their tireless and insightful input to this thesis, my general questions, and other work at Lincoln. Thank you especially for the time taken to review this thesis, explain ideas, and complete projects with me. A big factor in my choosing this group was your evident interest in developing your junior members—I was definitively not disappointed.

To my academic advisor, Julie Greenberg, and MIT's iconic Anne Hunter, for collectively ensuring I graduated, answering questions at odd hours, and generally being the exemplar of your roles.

To all the MIT professors, and teachers previous to MIT, that have influenced my academic career and my life in general—there are too many to name. A special shout out to Karl K. Berggren, the first MIT professor to directly, specifically tell freshman Hunter that he was capable of being successful at MIT.

To all my friends, you made MIT worth doing. To the people who supported me at my low points: Paige, Anthony, Sonja, Bianca; and to the friends who made sure my MEng semesters were full of high points: Bianca, Mariana, Louisa, Michael, Emma. To Larry, for helping me order my life. To my family—everyone else is lying when they say they have the best family in the world. And to God and the Church, my fundamental source of love, joy, and direction.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

I Introductions **21**

1 Introduction **23**

1.1 Background and motivation 23

1.1.1 Previous work 24

1.1.2 Swarms 25

1.1.3 Group key management 27

1.1.4 Usable secure communication 28

1.2 Contributions 29

1.2.1 Taxonomy of swarm requirements 29

1.2.2 Identify candidate key management schemes 30

1.2.3 Architecture for scalable secure swarm communication 30

1.3 Design overview 31

1.3.1 Provide applications a message-passing interface 31

1.3.2 Decouple group formation from linear-work secret distribution 32

1.3.3 Support runtime-configurable key management schemes 32

1.4 Thesis organization 33

2 Intended Audience **35**

II Background **36**

3 Swarms **37**

3.1 Unmanned systems 37

3.1.1 Manned versus unmanned systems . . . 38
3.1.2 Autonomy . . . 39
3.1.3 Transition toward swarms-based systems . . . 42
3.2 Swarm engineering . . . 43
3.2.1 Overview . . . 43
3.2.2 Concept of operations . . . 44
3.2.3 Example use cases . . . 45
3.3 Defining a swarm . . . 46
3.3.1 Initial attribute-based definition . . . 47
3.3.2 Simple swarms . . . 47
3.3.3 Tactical swarms . . . 48
3.3.4 Applicability of the tactical swarm model . . . 49

4 Key Management **53**
4.1 Information security . . . 53
4.2 Primitives . . . 54
4.3 Group key management . . . 55
4.3.1 Symmetric-only . . . 56
4.3.2 Asymmetric-only . . . 57
4.3.3 Hybrid . . . 58
4.3.4 General group key management . . . 60
4.4 Related work . . . 64
4.4.1 Unsuitable solutions . . . 65
4.4.2 Secure multicast . . . 66
4.4.3 Resource-constrained key management . . . 69

5 Key Management for Tactical Swarms **73**
5.1 Related work . . . 73
5.1.1 NPS swarm . . . 74
5.1.2 AFIT swarm . . . 74
5.2 Problem overview . . . 76

5.2.1 Complexity of swarm requirements . . . 77
5.2.2 Post hoc key management consideration . . . 79
5.2.3 Siloed existing solutions . . . 79
5.3 Specific challenges . . . 80
5.3.1 Dynamic group membership . . . 80
5.3.2 The swarm as a distributed system . . . 83
5.3.3 Key management and the protocol stack . . . 86
5.4 Out-of-scope challenges . . . 88

III Solutions 90

6 Solutions Overview 91

7 A Taxonomy of Swarm Requirements 93
7.1 Taxonomy overview . . . 93
7.1.1 Method . . . 94
7.2 Generated taxonomy . . . 95
7.2.1 Description by category . . . 95
7.2.2 Summary of taxonomy . . . 100
7.3 Revisiting the tactical swarm characterization . . . 101
7.4 Example usages . . . 102
7.4.1 Example usage: MARKS key management scheme . . . 103
7.4.2 Example usage: a tactical WSN . . . 104

8 Identifying Candidate Key Management Schemes 107
8.1 A tractable set of key management requirements . . . 107
8.2 Identified key management schemes . . . 109
8.2.1 Centralized direct distribution . . . 109
8.2.2 Distributed direct distribution . . . 110
8.2.3 Centralized key tree . . . 111
8.2.4 Distributed key tree with transport . . . 111

8.2.5 Distributed key tree with agreement . . . 112
8.2.6 Non-recommended schemes . . . 113
8.3 Discussion of identified schemes . . . 114

9 An Architecture for Scalable Key Management in Tactical Swarms 117
9.1 Conceptual solution . . . 118
9.1.1 Platform-level affordances . . . 118
9.1.2 Secured bootstrapping period . . . 119
9.1.3 Session-based communication . . . 121
9.1.4 Group access control . . . 126
9.1.5 Compromise detection . . . 128
9.1.6 Lower-layer keying material . . . 129
9.1.7 Delegation . . . 132
9.1.8 Passive agents . . . 133
9.1.9 Summary of the conceptual architecture . . . 134
9.1.10 Extensions to the basic architecture . . . 135
9.2 Abstract interfaces . . . 137
9.2.1 Application-ECU interface . . . 138
9.2.2 Radio-ECU interface . . . 139
9.2.3 Application-radio interface . . . 140
9.2.4 Example usage . . . 141

10 Prototyping the Architecture 145
10.1 Software architecture . . . 145
10.1.1 Supporting libraries . . . 146
10.1.2 Prototype functionality . . . 147
10.2 Abstract objects . . . 148
10.3 Performance projections . . . 149

IV Conclusions **152**

11 Conclusion and Future Work **153**

A Common Abbreviations **155**

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

1-1 Reuse of TLS as the mechanism for secure communication across applications such as HTTPS, IMAPS, and POP3S. Red indicates application data (unsecured), black indicates obscured data (secured). . . . 24

1-2 Lack of a reusable mechanism for secure communication across applications results in a coupling of the application with the security architecture. 24

1-3 Conceptual secure communication. Alice communicates message m to Bob by sending ciphertext c over an unsecured channel; e is the encrypting key and d is the decrypting key. 27

1-4 Usable, flexible schemes for secure communication are reusable, composable, and applicable to the particular use cases. 28

1-5 Flow of a message originating with the host application. The application passes message m to the cryptographic unit, which encrypts the message to produce the appropriate ciphertext c; the cryptographic unit then passes c to its radio to handle transmission to other group members. 31

1-6 Example session functionality; radios are omitted. Alice sends a message m to the session with sessionID. Since c is broadcast, all actors receive c; however, Alice's ECU encrypts m such that Bob and Carol's ECU can decrypt m, but Eve's ECU cannot. 32

4-1 The CIA triad: confidentiality, integrity, and availability, providing a basis for evaluating the effectiveness of schemes for secure communication. 54

4-2 Symmetric-only encryption scheme. There is no key distribution message K, as all application messages M are encrypted each a single time as $\mathsf{Enc}(m, \mathsf{k})$. 56

4-3 One example of an asymmetric-only encryption scheme. There is no key distribution message K, as all application messages M are encrypted once per intended recipient as $\mathsf{Enc}(m, \mathsf{pk}_i)$. 57

4-4 Simplified conception of envelope-style key distribution. In this example, the updated group key k' is distributed to each member of group N' as $\mathsf{Enc}(\mathsf{k}', \mathsf{pk}_i)$. Upon receiving the keywrap K, all application message encryption changes over from k to k', encrypting once per message via $\mathsf{Enc}(m, \mathsf{k}')$. 59

4-5 Logical key tree for a LKH. Agent u_4 holds KEKs on the path from its leaf to the root node: k_4, k_{34}, k_{14}, and k. KEKs affected when agent u_3 joins the tree: k_{34}, k_{14}, and k. 62

4-6 Modifications to the LKH upon membership updates. The c_* ciphertexts are used to distribute updated KEKs when u_3 joins the group. If u_3 was leaving the group, c_3 would be omitted. 63

4-7 Notional rekey (K) and application (M) messages in LKHs. This particular rekey message represents adding or removing u_3 in figure 4-6. The updated group key k' is recovered by successfully decrypting the root KEK. 64

4-8 Taxonomy of security considerations in secure IP multicast communication, summarized from [57]. 67

5-1 Target use case of the AFIT swarm (image from [34]). A swarm director (the Global Hawk) supervises disjoint clusters of UAVs, each with a higher-capacity manager. Mobility between clusters is supported. The group key (red "GK") is shared by the director and all cluster managers; the blue cluster keys (blue "CK") are shared by members of a particular cluster, including their cluster manager. 75

5-2 Notional selection of a minimal set of schemes aiming to collectively provide effective, application-efficient key management for a plurality of use cases. 78

5-3 Conception of a meta-architecture supporting pluggable key management schemes that minimally disturb application code. For example, if use case 4 from figure 5-2 was targeted, the application could test the comparative effectiveness of key management schemes A and B. . 80

7-1 Taxonomy of constraints and affordances for key management in tactical swarms. 96

7-2 Binary hash tree used in the MARKS key management scheme. Each resulting key k_i is used as the group key for some agreed-upon time slice. 103

9-1 Platform-level architecture decoupling application, cryptographic, and radio concerns. The ECU receives a plaintext message and session ID from the application, then forwards the duplet to the radio as the associated ciphertext, same session ID, and appropriate session epoch. The radio can then route the $(c, \texttt{sessionID}, \texttt{epoch})$ triplet, based on the appropriate membership view, which it may need to request from the ECU (and can cache for subsequent messages to the same session ID/epoch combination). 119

9-2 Conceptualization of successive subsets of sessions acting as privilege rings. In this example, the `all` session holds all members (As and Cs), while the `council` session holds only a subset of members (Cs), where `council` session members are granted additional decision privileges in the swarm. 126

9-3 Example conception of a membership view allowing the ECUs to provide sessions with multiple key domains. 137

9-4 Conceptual interface between the application and its ECU (Python-like syntax). 139

9-5 Conceptual interface between the radio and its ECU (Python-like syntax). 140

9-6 Minimal example of session-driven interactions under the proposed interfaces; here, Alice initiates a session, then transfers her role as manager to Bob. 141

9-7 Minimal example of session-driven interactions under the proposed interfaces; here, Bob defines his reactions to relevant interface events. . 142

9-8 Notional LKH message K generated by Alice's ECU to initialize the `all` session for an 8-drone Perdix swarm (cf. figure 4-7). Each set of KEKs c_* is encrypted to session member i under the appropriate ephemeral key k_i. The actual message broadcasted is perhaps $K' = \mathsf{Sign}(K, \mathsf{sk}_{\mathsf{manager}})$. Note K here requires $n \log n$ space. 143

10-1 Default topology in our prototyped implementation. Each agent is composed of three processes (app, ECU, radio) communicating over RPC. Radios communicate by sending a LAN broadcast echoed to all other agents by the switch. Mininet provides helper functions to generate and namespace the network topology, as well as simulating link QoS variables like latency, throughput, and packet loss. 148

10-2 Initial ASN.1 object definitions supporting envelope-style and tree-based forwarding schemes, simplified from [47, sec. 6]. In both cases, the manager generates an envelope-like object; in the forwarding-based scheme, however, the manager generates the envelope for a small subset of the swarm, prescribing how they should then recursively forward new envelopes to the rest of the group. 149
10-3 Some initial, simplified ASN.1 objects defined for our prototype implementation. 150

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

3.1 Comparing manned, unmanned, and swarm systems. For example, manned systems co-locate the human operator with the manned agent, where the operator acts as the controller and witness for the agent. Swarm constituents (the individual, concrete agents making up a logical swarm), conversely, do not have a co-located human operator, often act as their own controller, and require the logical swarm to act as their witness. 40

3.2 Summary of the human supervision-informed autonomy model. For example, an agent's human operator makes all high-level decisions and pilots all lower-level actions under human-in-the-loop autonomy. . . . 41

4.1 Applicability of group key management schemes discussed so far, characterized asymptotically. Computation is described as the number of asymmetric operations required per message in symmetric- and asymmetric-only schemes, and per group update in envelope-style and LKH. . . . 65

8.1 Asymptotic characterization of the identified key management schemes. Required storage overhead and whether the scheme is distributed in nature; also, per-update requirements of asymmetric work, communication rounds, and total messages. Note that D-LKH requires no more than 3 rounds, and TGDH no more than 2. 109

8.2 Rough comparison of the applicability of the recommended schemes, informed by a subset of the taxonomy categories proposed in chapter 7. Upward arrows indicate increases, downward decreases; black indicates comparative improvement, gray comparative decline. 114

Part I

Introductions

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

In this thesis, we seek to provide a bridge between swarm prototypers and usable secure communication.

Work on providing such a bridge is in-progress for unmanned systems in general, but with minimal consideration to scalability for swarms with security-critical constraints. We describe a starting point in providing a usable, flexible scheme for secure communication to researchers developing swarms and other large-scale unmanned systems.

1.1 Background and motivation

Swarms, and unmanned systems in general, provide unique constraints to secure communication schemes—they are resource-constrained, numerous, unmanned, and rapidly prototyped. Otherwise widely-used schemes for establishing secure communication, such as TLS for IP-based unicast, are often categorically unusable. Thus, while web application developers, for example, are empowered to rapidly prototype an application while trusting TLS and an established public key infrastructure to secure application traffic, researchers prototyping swarms and other unmanned systems are left to re-build, or re-purpose, application-specific security schemes for each realistic prototype (cf. figures 1-1, 1-2).

We propose that prototype swarms for security-critical applications should pos-

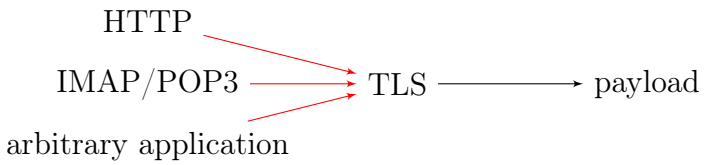

Figure 1-1: Reuse of TLS as the mechanism for secure communication across applications such as HTTPS, IMAPS, and POP3S. Red indicates application data (unsecured), black indicates obscured data (secured).

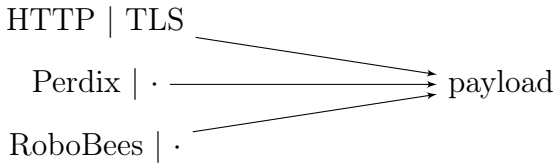

Figure 1-2: Lack of a reusable mechanism for secure communication across applications results in a coupling of the application with the security architecture.

sess a mechanism for secure communication as usable, flexible, and reusable as the mechanism web developers have in protocols like TLS or SSH. In remedying this lack, we look to map the space of swarm needs, survey the space of potential secure communication schemes, and generate a single, conceptual, pluggable API-like interface which swarm prototypers can readily adopt across use cases.

The remainder of this introduction will overview the motivation for swarm research in general, as well as touch on group key management and the challenges of key management for swarm applications.

1.1.1 Previous work

A principal challenge in secure communication for resource-constrained systems is cryptographic key management.[1] This is where we focus our effort, with emphasis on providing usable mechanisms for secure communication backed by scalable key management.

We draw on previous work in secure communications for unmanned systems in [2] and [3]. LOCKMA, the former, develops a key management architecture in the form of a software library; the SCM, the latter, provides a full, end-unit solution for secure

[1] See [1] for an overview of security challenges for swarms

communication. Both target unmanned systems, but with questionable scalability properties for larger group sizes, as we discuss.

1.1.2 Swarms

The concept of an intelligent, autonomous, intercommunicating superorganism,[2] long relegated to the realm of science fiction,[3] has evolved into an active area of research and prototyping. Drawing from real-world examples of natural organisms that operate in a swarm-like manner such as fish, birds, or insects, swarm engineering [4] aims to perform existing tasks with higher reliability, efficiency, and lower cost than possible via traditional means [5].

Swarming techniques, capabilities, and implementations are evolving in exciting ways in the literature, but with limited consideration given to schemes for securing swarm communication. Many prototypical implementations for civil applications (perhaps rightly) send messages completely in the clear (e.g. [6]), while prototypes for military applications often opt for simple-to-implement solutions with limited functionality and scalability (e.g. [7]).

We focus on swarms as a guiding example from the more general set of large-scale unmanned systems (UxS), desiring scalable forms of key management and secure communication to accommodate non-trivial group sizes.

Motivation

In a 2014 keynote speech on United States national defense efforts [8], Secretary of Defense Chuck Hagel described the need for a "third offset strategy" to cope with the proliferation of advanced weaponry and modernization of adversarial forces, emphasizing "innovative thinking" and the "development of new operational concepts." Responding to this challenge, outlines such as [9] often include increased investment in and reliance on unmanned systems as a key factor in building the next generation of armed forces, citing projections of decreased warfighter deaths, lower operational

[2] `https://en.wikipedia.org/wiki/Superorganism`

[3] E.g. Michael Crichton's *Prey* or Orson Scott Card's *Ender's Game*

costs, and more agile power projection.

For swarming specifically, the Department of Defense's unmanned systems integrated roadmap [10] describes swarming capabilities as a "key technology," while news sources are quick to push swarms as e.g. the "riskiest, most exciting things" coming out of the Pentagon [11]. Swarming capabilities are an emerging military technology, concretely evidenced by research efforts such as the 50-drone NPS swarms [12] from the Naval Postgraduate School or the 100-drone Perdix swarms [13] from MIT's Lincoln Laboratory. But efforts to provide realistic secure communication for these research prototypes are lacking—either in total, as for the Perdix swarm, or in usability, as for the NPS swarm, which we further discuss.

Military integration

The DoD's unmanned systems integrated roadmap [10] lists four "overarching themes" relevant to the value and evaluation of "agile and flexible technology," specifically

- interoperability,
- autonomy,
- secure networking, and
- human-machine collaboration.

Swarms touch all four of these areas—autonomy is required to effectively operate swarms of non-trivial size, with varying degrees of human-swarm collaboration; the nature of swarms immediately leads to the formation of a network with associated security requirements; and interoperability is desired not least in the use of a cross-application mechanism for secure communication.

Civil integration

Civil swarm applications span corporate applications such as Amazon's Kiva warehouse robots [14] or aerial delivery drones [15], novel applications in fields like human-computer interfaces [6], and even didactic applications [16].

In most civil use cases, secure communication is desirable at the right trade-off cost—in corporate scenarios, secure communication is essential to prevent rogue actors and corporate or state-level espionage. However, civil applications often have less stringent security and usability requirements, as known terrain, straightforward supervision, well-connected backhauls, and the rare need for stealth allow for more relaxed solution requirements.

Thus, while civil use cases are not the primary target for this thesis, a well-developed solution for military use cases could be made available for civil adoption upon acceptable maturity.

1.1.3 Group key management

We briefly introduce some concepts and vocabulary relevant throughout the thesis.

Conceptually, cryptographically secured communication aims to allow at least two parties to communicate over an unsecured channel, as in figure 1-3.[4]

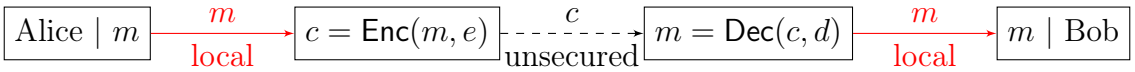

Figure 1-3: Conceptual secure communication. Alice communicates message m to Bob by sending ciphertext c over an unsecured channel; e is the encrypting key and d is the decrypting key.

Alice and Bob are abstract actors representing e.g. two communicating drones in an aerial swarm. Alice wishes to send a message m to Bob via an unsecured channel—this channel could be "stealthed"[5] or in the clear, but with the invariant that successful interception of the transmitted message c should not reveal useful information about m.

In generating the ciphertext c from the plaintext message m, Alice *encrypts* the message under a chosen key e. To recover the original message m from the received c, Bob *decrypts* the message under an applicable key d.

[4] See [17, chp. 1.4] for an introduction
[5] See [18] for an overview of TRANSEC

An immediate question arises—the relationship between the encrypting and decrypting keys, e and d. In *symmetric* encryption, Alice and Bob use the same key k, where $\mathsf{k} = e = d$. In *asymmetric* encryption, $e \neq d$, but the keys are related to each other such that anyone can send Bob a message encrypted under e which only the owner of d can decrypt.

One could imagine that group communication would be most straightforward under symmetric encryption, where all group members hold a copy of the shared group key k. But, how does the group recover from the compromise of a single member, and loss of k, to an adversary? Thus, general key management schemes make intelligent use of both forms of encryption, utilizing the advantages of each. A principal incarnation of this "hybrid" scheme involves asymmetric distribution of a shared symmetric group key—this is *key distribution*, to which we devote the majority of our attention.

1.1.4 Usable secure communication

Secure communication should be two things, principally: usable and effective. Schemes for secure communication make some set of claims about the provided security: a well-chosen set results in a *usable* scheme,[6] while correctly providing each element results in an *effective* scheme. We focus on usability along three axes, where usable schemes should be reusable, composable, and non-trivially applicable (figure 1-4).

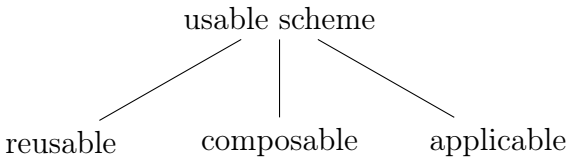

Figure 1-4: Usable, flexible schemes for secure communication are reusable, composable, and applicable to the particular use cases.

Take TLS [19], the Transport Layer Security protocol, as an example scheme for secure communication. Along with the accompanying public key infrastructure, TLS is

[6]Here and elsewhere, we use *usable* in the sense of flexibility, where usable schemes minimize their impositions on applications and allow flexibility in the set of use cases they support

- *reusable* across applications (e.g. HTTPS, IMAPS [20]) and endpoint constraints (via handshake-based agreements),
- *composable* as one component of an application-defined security architecture, and
- widely *applicable* as one of the de facto protocols for Internet-based secure communication.

TLS's applicability is not universal, however, as, for example, secure multicast is not supported. Nevertheless, any scheme looking to provide secure communication for a new set of use cases should seek to emulate the usability of similar protocols.

1.2 Contributions

Our central contribution is an extension of the concepts espoused by the SCM cryptographic unit [21, 3] with emphasis on pluggable, variously scalable key distribution options as the foundation for a flexible, widely-applicable scheme for secure multicast in military-grade swarms. Supporting this contribution, we develop a playbook for understanding which key distribution options are applicable to various use cases: a proposed taxonomy of swarm requirements allows effective characterization of swarm key management needs, and an identification of candidate key management schemes provides an initial set of schemes to meet various security and scalability needs.

1.2.1 Taxonomy of swarm requirements

The first step in developing a general solution for securing swarm communication is understanding the associated spectrum of needs.

To this end, we assemble and examine a collection of internal and external use cases to generate a taxonomy (following the method described in [22]) characterizing the corresponding space of swarm requirements from a key management and secure communication point of view.

We then characterize the assembled use cases, via this taxonomy, into more general swarm categories; this categorization process should provide straightforward context to swarm researchers searching for projects with comparable key management needs. Subsequent swarm projects should be empowered to clearly and concisely express their secure communication needs, while key management architects should be better able to identify which swarm use cases their projects aim to service. Clear associations should be possible when paired with existing group key management taxonomies, e.g. [23], and the challenge of selecting a proper pluggable option in our conceptual key management scheme should additionally be ameliorated.

1.2.2 Identify candidate key management schemes

Our surveyed swarms lean toward key management schemes that preclude dynamic group management—that is, group membership is static or additive-only. Recent efforts in LOCKMA and the SCM remedy this constraint by supporting dynamic group membership, but with minimal scalability considerations in the utilized key distribution scheme. We identify a set of key distribution schemes from relevant literatures to extend or replace the non-scalable schemes used in these influencing architectures.

1.2.3 Architecture for scalable secure swarm communication

Using the generated taxonomy of swarm requirements, we motivate the need for a new architecture for secure communication, targeting security-critical swarm applications. We then describe our proposed architecture, adopted from the SCM work, to fill the missing role across swarm prototyping efforts, serving as an application-layer key management architecture scaling to swarm-like groups of unmanned systems.

1.3 Design overview

Iterating on existing work from the designs of LOCKMA and the SCM, we describe an extension to the SCM's conceptual framework which allows scalability across variable use cases.

For reusability purposes, we conceptually separate[7] the application, cryptographic, and transceiving functionality of each member of the swarm [21]. This results in each single group member taking the form described in figure 1-5. Note how this mirrors the conceptual form of encrypted communication in figure 1-3, allowing a decoupling of application functionality from cryptographic concerns.

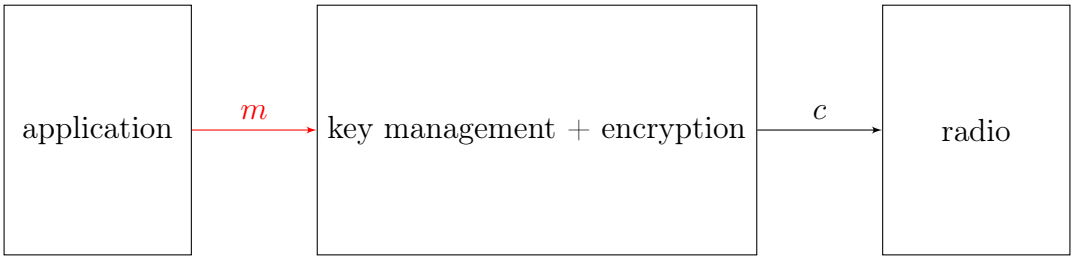

Figure 1-5: Flow of a message originating with the host application. The application passes message m to the cryptographic unit, which encrypts the message to produce the appropriate ciphertext c; the cryptographic unit then passes c to its radio to handle transmission to other group members.

1.3.1 Provide applications a message-passing interface

As in the SCM work, we adopt the concept of a *session*, a higher-level abstraction allowing a subset of a swarm to securely pass messages to all other members in that subset, as in figure 1-6 (in this example, Alice is sending to a session with members $\{\text{Alice}, \text{Bob}, \text{Carol}\}$).

By combining separation of application-ECU-radio[8] concerns with adoption of the session abstraction, an application is freed to specify only to whom a message should be sent, leaving the ECU to determine how each message should be encrypted

[7]The separation is more than conceptual in the SCM, but conceptual separation is the minimum sufficient separation

[8]ECU: end cryptographic unit, the component responsible for key management and encryption functionality

or decrypted, and the radio component to determine how each message should be delivered.

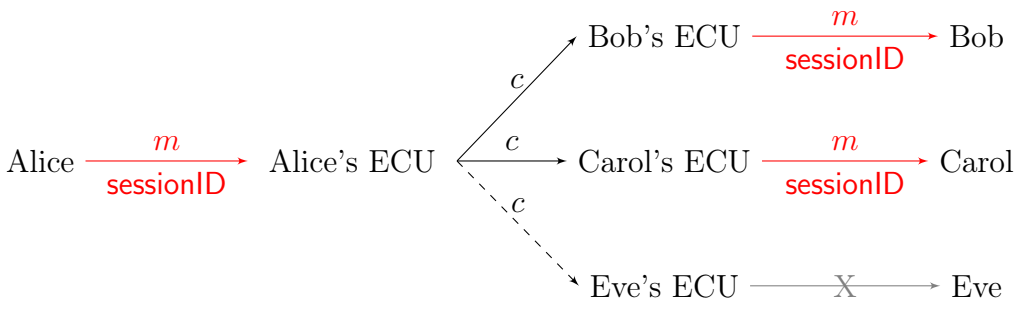

Figure 1-6: Example session functionality; radios are omitted. Alice sends a message m to the session with sessionID. Since c is broadcast, all actors receive c; however, Alice's ECU encrypts m such that Bob and Carol's ECU can decrypt m, but Eve's ECU cannot.

1.3.2 Decouple group formation from linear-work secret distribution

Without yet delving into the specific mechanism for cryptographically enforcing session membership, the SCM scheme involves on-demand distribution of a shared group key—possession of the key for a particular session grants access the session, and creating a session requires non-trivial work linear in the size of the group.

Thus, the SCM approach, which we later describe as a form of *centralized direct distribution*, couples group membership with ownership of only a single shared secret. As we discuss, this does not scale to swarm-like group sizes, and precludes application-informed choice of alternatives to this approach to key distribution.

1.3.3 Support runtime-configurable key management schemes

In remedy to the coupling of group membership with possession of a shared secret, we provide the means for a runtime-configurable, per-session choice of key management scheme. Since the application and ECU functionality are separated, application code can continue to use the session abstraction across choices of key management schemes, leaving the ECU to properly implement the application-informed choice.

This allows a single conceptual interface, which we describe, to be used across swarm applications, swapping only the key management options communicated to the ECU on session creation.

1.4 Thesis organization

We now move to a word on the intended audience of this thesis in chapter 2, concluding introductions. Subsequent chapters provide relevant background and describe proposed solutions.

For background, we overview swarms in general in chapter 3, with particular attention paid to our definition of a "tactical swarm" for security-critical military applications. We then provide a short introduction to group key management and briefly summarize useful learnings from relevant literatures in chapter 4. We conclude the background information with a discussion of the challenges of group key management for tactical swarms in chapter 5.

For solutions, we first overview our contributions in chapter 6. We then present a novel taxonomy of swarm key management requirements in chapter 7, recommend key management schemes appropriate for tactical swarms in chapter 8, describe a general architecture for secure swarm communication in chapter 9, and describe our initial, in-progress implementation of the architecture in chapter 10. Finally, we present brief conclusions and future work in chapter 11.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Intended Audience

This thesis targets two principal audiences: researchers developing prototypes of swarms and other groups of unmanned systems, and security researchers developing key management architectures aimed toward said systems.

We note the two audiences will likely have diverse security and cryptography backgrounds, so we have, where applicable, sought to provide overly clear descriptions of cryptographic topics with references for further reading.

For security researchers, we hope they find the description of swarms from a key management perspective to be an interesting introduction to an area potentially underserved by current security efforts. For swarm and unmanned system prototypers, we hope they find the generated taxonomy useful and the push for a de facto mechanism for secure communication engaging. For our team at MIT Lincoln Lab, we hope it is a productive step toward the eventual goal of providing a more scalable mechanism for secure communication in groups of unmanned systems.

Finally, see appendix A for a listing of commonly used abbreviations found in this document, as well as [24] for a glossary of information assurance definitions.

Part II

Background

Chapter 3

Swarms

We begin by discussing the unmanned systems in general, and their unique challenges compared to traditional manned systems. We then transition toward swarm-based systems, briefly touching on some challenges posed by swarm autonomy.

3.1 Unmanned systems

Unmanned systems comprise one of a core set of key military technologies [9], with some estimates expecting 10,000 UASs serving the US military by the end of the 2020s [25]. With this shift in military technologies, effective development and usage requires a shift in expectations compared to traditional manned systems.

We define four terms to clarify subsequent discussion; clarity is constructive in discussions of compromise detection and reporting in swarm systems.

- *Agent.* The system taking action on the world. Agents can be nested, so a swarm and each of its constituent robots can all be agents.

- *Operator.* The human responsible for the agent. The operator's role can vary from direct piloting to non-interactive mission specification.

- *Controller.* The logical entity responsible for making decisions at the agent. An agent's controller can be the agent itself, its operator, or another agent.

- *Witness.* The logical entity responsible for determining when an agent has been compromised. An agent's witness can be its operator, another agent, or some other application-specific entity, but an agent cannot be its own witness.

To clarify the concept of nested agents, we note that a swarm itself, the "logical swarm," is considered an agent, while its constituent, concrete members, are also considered agents.

As one example, consider a single drone piloted remotely by a human. In this case, the drone is the agent, and the remote (human) operator acts as the controller (directly piloting the drone) and the witness (directly attesting to the drone's status).

In another example, consider a drone swarm, making the distinction between the single logical swarm (an abstract agent) and its constituent drones (concrete agents). For the swarm as a whole, the entirety of the swarm is collectively viewed as a single agent. Some remote individual, responsible for directing the swarm's actions, is the operator. Assuming the swarm is not acting autonomously, this operator is also the swarm's controller. Finally, the operator is also assumed to act as the swarm's witness.

3.1.1 Manned versus unmanned systems

While manned systems rely on a human operator to handle decision making, unmanned systems and swarms increasingly shed their reliance on their operator. See table 3.1 for a summary.

Traditional manned systems co-locate one or more operators with each agent to provide full, direct control of the system. The human operator charged with piloting the agent, making communication decisions, directing interactions with other agents, and appropriately disposing of the agent upon compromise—thus also acting as the agent's witness. The requirement of a co-located operator ensures agents are generally not resource-constrained. This traditional model places minimal strain on any form of key management architecture: human operators are implicitly trusted, perhaps unconditionally; loss of an agent or operator is not an expected, standard outcome;

and compromise of an agent is known explicitly or implicitly.

Unmanned systems, by contrast, allow remote control of the agent by the operator, removing the operator's requirement for co-location with the system. Easing this requirement allows agents to be smaller, faster, cheaper, and, importantly, disposable. While manned systems place a principal concern on the safety of its operator, unmanned systems can perform dangerous or otherwise "one-way" missions. However, smaller, cheaper agents result in resource constraints, providing limited computation, battery life, and size and weight allowances.[1] Additionally, the lack of a co-located human operator means all key management functions must occur without direct human intervention.

Swarms of unmanned systems, finally, change the ratio of operators to agents. While manned and unmanned systems generally have a linear ratio of human operators per agent (e.g., two pilots per aircraft or one remote operator per drone), the ratio for swarms is necessarily sublinear (see [26] for a survey). Thus, while manned and unmanned systems usually have an $\mathcal{O}(1)$ ratio of operators to agents, many swarms opt for an $\mathcal{O}(\log n)$ or $\mathcal{O}(n)$ ratio—meaning a swarm of n agents requires around $\log n$ operators or around 1 operator, respectively.

Unmanned systems and logical swarms can be autonomous, where the human operator defers control responsibilities to the agent itself, leading to the "✓/ ✗" in table 3.1.

This increase in complexity, from the operator point of view, means each agent can no longer have a dedicated operator act as its witness. Thus, the logical "swarm agent" must act as the witness for each of its constituent group members, and a human operator can act as the witness to the swarm as a whole.

3.1.2 Autonomy

Focusing on unmanned systems, a convenient autonomy taxonomy [27, 10] classifies systems based on their degree of human supervision, reproduced below and summa-

[1] SWaP constraints: size, weight, and power; we care additionally about the resulting constraint to computational capacity

Agent type	Operator is co-located with the Agent	Operator is the Controller	Operator is the Witness
Manned	✓	✓	✓
Unmanned	✗	✓/ ✗	✓
Logical swarm	✗	✓/ ✗	✓
Swarm constituent	✗	✗	✗

Table 3.1: Comparing manned, unmanned, and swarm systems. For example, manned systems co-locate the human operator with the manned agent, where the operator acts as the controller and witness for the agent. Swarm constituents (the individual, concrete agents making up a logical swarm), conversely, do not have a co-located human operator, often act as their own controller, and require the logical swarm to act as their witness.

rized in table 3.2.

We first delineate the terms decision and action: a *decision* is any high-level choice or command, at the level that one human warfighter could deliver to another; an *action* is any maneuver that could be described as "piloting" or "directly operating" the agent. While these definitions do not unambiguously sort items as decision or action, it is useful for categorical discussion.

- *Human-in-the-loop.* Remote pilot control: all decisions and actions are taken by the human. The operator and controller are co-located.

- *Human-on-the-loop.* Supervised control: decisions and actions can be taken without human input, but human input throughout a mission is accepted and integrated. The logical controller can be passed between operator and agent throughout a mission. One form of on-the-loop has the operator making all decisions, leaving the agent to take the resulting actions.

- *Human-out-of-the-loop.* Full autonomy: all agent decisions and actions are taken by the agent without human input. The controller is co-located with the agent, reducing the operator's role to mission-level specification of goals and constraints.

Autonomy classification	Operator makes the Decisions	Actions
In-the-loop	✓	✓
On-the-loop	✓/ ✗	✓/ ✗
Out-of-the-loop	✗	✗

Table 3.2: Summary of the human supervision-informed autonomy model. For example, an agent's human operator makes all high-level decisions and pilots all lower-level actions under human-in-the-loop autonomy.

Autonomy across agent types

Traditional unmanned system architectures utilize the human-in-the-loop form of the controller-agent pattern, where the human operator uses a local controller (e.g. a "remote control" or OCU[2] to fully direct an unmanned agent.[3] A straightforward extension of this pattern allows a single controller to serve multiple agents and their corresponding human operators.[4] The resulting star topology has clear failure modes: a human is co-located with the controller, and that single logical controller provides all directives; loss of contact with the controller causes the agent to enter a standby mode.[5]

On-the-loop autonomous unmanned systems, contrastingly, allow logical control to be passed between the human-operated controller and the agent itself. The operator still acts as the witness, maintaining responsibility for determining when the agent has been compromised.

Additional complexity is introduced in swarms of unmanned systems by separating the operator and the witness—the human operator acts as the witness for the swarm as a whole, but the logical swarm must act as the principal witness for each of its constituent agents. This follows naturally from the desire to have e.g. a single operator direct an entire swarm, as the operator is cognitively incapable of witnessing to and reporting the compromise of a non-trivial subset of the swarm. However, the operator

[2]OCU: operator control unit

[3]In this case, it would be appropriate to call either the OCU or the operator the controller, but the OCU holds ultimate/final control over the agent's actions

[4]E.g. a single ground control terminal (controller) allowing multiple drone operators to remotely pilot their respective drones (agents)

[5]E.g. a drone attempts to remain airborne while disconnected from its operator

should be empowered to provide a manual override and mark a specific member of a swarm as compromised or not.

Compromise detection of autonomous agents

Detecting compromise of autonomous agents is a non-trivial problem: consider a drone performing a twelve-hour stealth mission—how does the witness determine whether the drone had been temporarily captured during its mission? Due to this complexity, it seems necessary to require application-specific solutions to effectively witness to autonomous agents.

Swarm communication requirements

Looking back to the requirement of a sublinear ratio of swarm members to swarm operators, we note that effective swarms therefore require at least on-the-loop autonomy, as any group of in-the-loop cannot effectively act as a swarm past trivial group sizes. Ideally, a swarm would require only a single operator (as in [12]), which we assume without loss of generality.

Given that swarms then exhibit on-the-loop autonomy (e.g. the Perdix [13] or NPS swarms [12]), the swarm requires some capacity for group-wide decision making. This naturally requires real-time, secured communication across the swarm, which applicable secure communication schemes must necessarily support.

3.1.3 Transition toward swarms-based systems

Finally, we briefly motivate the transition toward swarm-based systems [28] over current unmanned systems, summarizing from [29] (and e.g. [30]).

- *Effective.* Multiple agents cooperate to perform tasks from collecting sensor data to engaging an adversary, where additional numbers, angles, and locations provide distinct application-specific advantages.
- *Reliable.* A swarm has n quasi-redundant constituent members, allowing manufacturer error or adversary action against a subset of the swarm to occur without

terminating a mission.

- *Flexible.* Members of a swarm are free to make individual decisions, e.g. carrying out solo missions as the swarm operator requires.

- *Efficient.* Consider n drones collaboratively mapping an area, compared to a single, more powerful drone. Additionally, swarms with a single up-link/down-link to their controller make economical use of limited bandwidth resources [31, chp. 9].

- *Inexpensive.* Since effectiveness is achieved via collaboration, individual swarm members can be less powerful and therefore cost less—the NPS drone swarm targets $1000 per drone, while Perdix drones are designed to be single-use.

3.2 Swarm engineering

Swarm engineering [32, 33] is defined in [4] as "an emerging discipline that aims at defining systematic and well founded procedures for modeling, designing, realizing, verifying, validating, operating, and maintaining a swarm robotics system." Accordingly, swarm robotics systems exhibit the following characteristics, summarized from [4], where each constituent robot is

- *autonomous*, capable of making decisions and/or taking actions without input from a human operator;

- *localized*, such that access to centralized control or global knowledge is limited or nonexistent; and

- *collaborative*, cooperating with other swarm members to complete tasks.

3.2.1 Overview

For clarity, we use three generic terms in describing swarm behavior: *mission*, *task*, and *action*. Human participation at each level of abstraction can range from full

control to no involvement (in-the-loop, on-the-loop, out-of-the-loop). We demonstrate these using an example mission: a fixed-wing UAV swarm chaperoning a ground-based supply convoy.

At the highest level of abstraction, a swarm acts as an atomic unit to complete a mission. A *mission* is a logical collection of tasks, lasting a situationally-appropriate duration; the task set can be designated pre-mission (during the planning phase) and/or in-mission (dynamically). Missions generally require pre-mission planning with human involvement. Mission goals can be more abstract than tasks and with less quantifiable outcomes (e.g. "chaperone the convoy" versus "patrol this area"). **Example mission:** the swarm is given the high-level objective of chaperoning the ground-based convoy; such a mission will likely require a dynamic task set.

Within a mission, a single *task* is a specific actionable for the entire swarm to accomplish. Duration can be specified or continue until completion. Examples include patrolling or mapping an area, finding or following a target, transporting items, performing evasive maneuvers, going silent, etc. (see [5] for a few generic examples). **Example mission:** the swarm may dynamically alternate between tasks such as following the convoy, aiding in transportation of essential items, and mapping surrounding areas.

Looking now at individual members of a swarm, an *action* is a particular operation performed by one member of a swarm for the purpose of cooperatively completing the current swarm-wide task. Tasks are much finer granularity and application-specific. **Example mission:** during an area-mapping task, a particular UAV may map its assigned portion of the area; at a finer granularity, during a follow-the-convoy task, a particular UAV may attempt to maintain a buffer of distance between itself and its nearest neighbors.

3.2.2 Concept of operations

We adopt a relevant, lightly-generalized form of the fundamental phases of swarm operation identified in [12].

- *Fabrication.* Decoupled from any particular mission, fabrication entails general assembly of the mission-independent UxS platform (single agent). Long-term, identifying information is loaded at this point, such as identity key pairs and associated certificates.

- *Pre-mission.* Preparation of agents by human technicians, including health checks, mission and software loading, and other application-specific bootstrapping.

- *Launch.* Deployment of the agents; can occur serially or in parallel, from one or more static or mobile launch locations.

- *In-mission.* Agents carry out some pre-defined mission; we coalesce the ingress, swarm ready, swarm active, and egress phases into this phase for brevity.

- *Mission termination.* Agents terminate their participation in the current mission; participation can end by landing, as in the NPS swarm, or by performing some form of self-destruction, as in the Perdix swarm.

- *Recovery.* Optional phase allowing human technicians to power down, inspect, and repair successfully reclaimed agents.

3.2.3 Example use cases

To make more concrete the preceding discussion of swarms in the abstract, and to demonstrate their wide applicability, we now present an array of swarm use cases.[6] We push the limits of what could be considered a swarm, including multi-robot systems requiring secure, real-time communication.

Military

Military use cases center around offensive and defensive warfare, auxiliary task completion, and intelligence gathering (ISR), including "continuous border patrol, bat-

[6]See [25] for a survey of UAV types, relevant for our principal use case of aerial drone swarms

tlespace surveillance, mapping routes for troop movement, real-time information distribution to mobile military units, and extending communications via an airborne network" [34]. All use cases have security-critical requirements.

- Aerial warfare and ISR: Perdix [14].
- Sea-based warfare and ISR: surface [35] or subsurface.
- GPS-less localization services [36, 37].
- On-demand networks: FANETs [38] and general ad hoc networks [39].

Civil

Civil use cases are more varied, from warehouse automation to user interfaces. They often require less stringent security guarantees.

- Warehouse automation: Amazon's Kiva robots [14].
- Last-mile shipping: Amazon's Prime Air delivery service [15].
- Continuous urban monitoring: crack detection in asphalt pavement [40].
- Construction: terrestrial or otherwise [41].
- Internet backhaul: satellite-based Internet [42].
- User interfaces: Zooids [6].
- Research-oriented swarms: Kilobot [16].

3.3 Defining a swarm

We now flesh out our description of a swarm, moving from "a collection of unmanned robots" to a more concrete characterization, with an eye to key management concerns and constraints.[7]

[7]See [29, 43] for a more in-depth discussion of different types of multi-UAV architectures; for our purposes, we refer to any collaborating, explicitly communicating group of unmanned robots as a swarm

3.3.1 Initial attribute-based definition

In the interest of providing useful secure communication schemes, we detail the relevant constraints stemming from the nature of swarm-based systems. This definition is initial, in that we use it only as a starting point, deferring to a more complete, taxonomy-informed definition in chapter 7. It is also attribute-based, defining a swarm by identifying common characteristics shared across specific swarms.

We delineate two broad classifications of swarms: simple and tactical. While simple swarms are more likely to be found in civil applications, we view tactical swarms as encompassing the expected set of constraints posed by military applications.

3.3.2 Simple swarms

We characterization a *simple swarm* with the following generic attributes.

Attribute 1. *Group members: homogeneous group of unmanned, mutually-trusting nodes.*

Attribute 2. *Group membership: static.*

Attribute 3. *Broadcast medium: single-hop broadcast.*

Attribute 4. *Node constraints: resource-constrained.*

Attribute 5. *Topology: fully-connected.*

Attribute 6. *Tasks types: long-running tasks without access to centralized control.*

While pared down and still less than perfectly concrete (e.g. "resource-constrained," "long-running," and group size are under-specified and context-dependent), this characterization serves as a base point of reference in evaluating schemes for secure communication.

Considering just simple swarms, current in-use key management schemes[8] are usable, performant, and often scale acceptably well. However, this simple swarm characterization is an insufficient approximation to real-world swarm architectures, particularly in military applications.

[8] E.g. pre-placing a symmetric group key, described in chapter 4

3.3.3 Tactical swarms

In the simple swarm case, trust is static and total. A particular node is either granted permanent group membership—decided pre-mission and for the duration of the mission—or denied. That is, group membership has no temporal component.

This lack of temporality makes for brittle mission parameters and specifications, precluding unplanned mission-time operations such as dynamically supplementing new group members or removing specific members. This causes issue by ignoring the real-world expectation of node capture, providing no story for revoking group membership.

Thus, while the simple swarm characterization is sufficient for many civil applications and prototype environments, we now describe a *tactical swarm* with a more realistic set of attributes encompassing the expected realities of dynamic swarms in untrusted environments.

Attribute 7. *(*) Group members: heterogeneous group of unmanned, mutually-cooperating*[9] *nodes*

Attribute 8. *(*) Group membership: dynamic.*

Attribute 9. *(*) Broadcast medium: single-hop broadcast or multi-hop routed.*

Attribute 10. *Secret protection: compromise and eventual exfiltration of private keying material of nonzero percentage of nodes.*

Attribute 11. *Node constraints: allow passive nodes.*

Attribute 12. *Task types: low-latency and high-throughput applications.*

This attribute set more closely characterizes the constraints faced by swarms in tactical environments. Whether in civilian or military environments, allowing for dynamic group membership and expected compromise of individual nodes enables or facilitates a larger and more useful set of applications.

[9] We use "cooperating" to describe trust that is subject to non-uniform temporality. *Non-uniform*: nodes are not equally/fully trusted. *Temporality*: nodes operate correctly when given group access, but with the requirement that their group membership be subject to proactive revocation/reinstatement.

Note that this characterization[10] supersedes the simple swarm attributes in the group members (Attribute 1), group membership (Attribute 2), and broadcast medium (Attribute 3) properties.

3.3.4 Applicability of the tactical swarm model

We provide three examples motivating aspects of the tactical swarm characterization.

Civilian example: Amazon's Kiva robots

Consider first a civilian environment. Amazon's Kiva robots [14] act as physical resource allocators, moving inventory pods around Amazon's fulfillment warehouses. A typical installation can have as many as 1000 robots operating at a time. We overview relevant expected swarm constraints given Amazon's setup.

1. For operational agility, robot introduction, decommission, or exchange between warehouses should not require full or partial downtime (Attribute 8).

2. Given that Amazon is a global corporation, it is incentivized to protect against, and properly handle, loss or theft of individual robots as well as intentionally malicious behaviors (Attribute 10).

3. For flexibility of implementation, the ability to have passive traffic-monitoring receivers (Attribute 11) is desired.

4. Real-time command and control messages have low-latency requirements, while reporting of real-time sensor data has moderate high-throughput requirements (Attribute 12).

Military example: FANETs

We now consider an example from a military environment. FANETs, or Flying Ad hoc NETworks [38], provide network connectivity to a large theater with improved

[10]Specifically, attributes denoted with (*)

scalability, survivability, detectability, and cost-efficiency, compared to traditional single-UAV solutions. We overview relevant expected swarm constraints given the goal of providing network connectivity via distributed low-power drones.

1. Since individual swarm members provide a content-agnostic network connection, drones from allied forces should be free to seamlessly participate in providing the ad hoc network, while the initiating force retains ultimate control over group membership decisions (Attribute 7).

2. Arbitrarily extending provided coverage range requires unplanned, mission-time group addition (Attribute 8).

3. In achieving the operational goal of extending connectivity beyond the range of a single node, packet forwarding within the swarm necessitates multi-hop routing (Attribute 9).

4. The assumed-hostile nature of military environments necessitates handling of hijacked nodes and respective theft of private keying material (Attribute 10).

5. Whether by design or tactical stealth requirements, select nodes must retain the long-term capacity to receive data from the network without actively transmitting (Attribute 11).

6. Intra-swarm command and control motivates the low-latency constraint, while support for media applications, such as forwarding of surveillance video over the network, motivates high-throughput constraint (Attribute 12).

Military example: autonomous, stealthed ISR

Finally, consider a swarm of long-duration, fixed-wing UAVs performing autonomous, stealthed ISR. Minimal to no connection to a centralized controller is expected, and SWaP constraints are amplified to maximize mission duration. We overview a set of expected constraints for this swarm.

1. Swarm members which fail to make a swarm-designated rendezvous point should be cryptographically removed from the group to prevent damage resulting from compromise of the shared secret (Attribute 8, Attribute 10).

2. Since the autonomous swarm acts as its own controller, it must have some form of group-wide decision promulgation; the default mechanism is broadcasting into the shared medium[11] (Attribute 9).

3. Consider a scenario where the stealthed swarm is capable of receiving a brief, planned controller payload via satellite link—the controller here desires to establish a shared group key with the swarm (or a representative member of the swarm) without receiving transmissions from the swarm (Attribute 11).

4. With hours of no backhaul connectivity, the swarm itself must collect and process relevant sensor data, perhaps from mounted cameras, all while maintaining low-latency communication to coordinate swarm-wide actions (Attribute 12).

Note that the above representative use cases do not necessarily fully exhibit simple swarm or tactical swarm attributes as, for example, the FANET and Kiva swarms are generally stationary and would expect to have undisturbed access to a centralized controller (no need for Attribute 6). The provided examples do, however, motivate aspects of the tactical swarm characterization.

[11] Various low probability of detection [18] communication mechanisms could also be employed, turning the single-hop broadcast into a multi-hop routed swarm

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Key Management

We now examine key management as the principally challenging facet of secure communication in swarm contexts.

Key management consists of the "secure generation, storage, distribution, and destruction" of cryptographic keys [44]. We focus on schemes for efficient distribution and update, accounting for constraints in secure storage techniques. We begin by overviewing general information security and introducing abstract cryptographic primitives, then moving the group key management and a survey of related work.

4.1 Information security

In looking to provide effective information security (see [17, chap 1.2]), the classic CIA triad of confidentiality, integrity,[1] and availability provide a clearer basis for evaluating the efficacy of a system's security (figure 4-1).

Data *confidentiality* involves preventing full or partial information disclosure to adversaries or other unintended parties; data *integrity* aims to prevent unnoticed mutation of stored or transmitted data; and *availability* look to make services and data as temporally accessible as possible, even in the face of adversarial effort.

In realistic schemes, edge case scenarios can require applications to make trade-

[1] For brevity, we'll include authentication, among other security desires, under the umbrella term of integrity

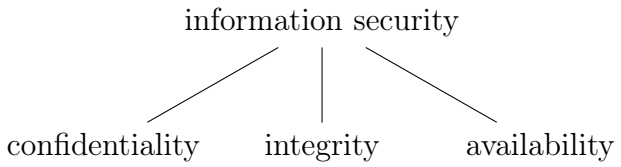

Figure 4-1: The CIA triad: confidentiality, integrity, and availability, providing a basis for evaluating the effectiveness of schemes for secure communication.

offs between information security objectives. And while one could expect that e.g. confidentiality should be the principal objective, military applications often opt for the capacity to make the decision in-mission. Consider an isolated warfighter in the field attempting to use a secure radio for which his current cryptographic key is out of date—should he be allowed to make an unsecured broadcast for assistance (availability), or should the radio categorically prevent such a broadcast (confidentiality)?

4.2 Primitives

In designing schemes for secure communication, developers compose various cryptographic primitives to provide intended functionality. For the most part, we take these primitives as black-box functions, not delving into the specific of any cryptosystems except when applicable.

First, a listing of notation and primitives.

- N a group of agents
- $|N|$ number of agents in group N, also denoted n
- k symmetric key
- pk public key
- sk private (secret) key
- $\mathsf{Enc}(m, \mathsf{k})$ encrypt plaintext m under symmetric key k
- $\mathsf{Dec}(c, \mathsf{k})$ decrypt ciphertext c under symmetric key k

- $\mathsf{Enc}(m, \mathsf{pk})$ encrypt plaintext m under asymmetric public key pk
- $\mathsf{Dec}(c, \mathsf{sk})$ decrypt ciphertext c under asymmetric private key sk
- $\mathsf{Sign}(m, \mathsf{sk})$ sign some text m with private key sk
- $\mathsf{Verify}(c, \mathsf{pk})$ verify some signed text c with public key pk
- K an encrypted key distribution message
- M an encrypted application message

We describe symmetric and asymmetric encryption below; in general, encryption and decryption act as inverse operations (given the appropriate keys), such that $m = \mathsf{Dec}(\mathsf{Enc}(m, \mathsf{k}), \mathsf{k})$.

Sign and verify are also linked. The sign/verify construct returns `True` in symmetric settings when the same key is used to sign and verify, i.e. $\mathsf{Verify}(\mathsf{Sign}(m, \mathsf{k}), \mathsf{k}') \rightarrow$ `True` exactly when $\mathsf{k} = \mathsf{k}'$; it returns `True` in asymmetric settings when sk and pk come from the same keypair,[2] i.e. $\mathsf{Verify}(\mathsf{Sign}(m, \mathsf{sk}), \mathsf{pk}) \rightarrow$ `True` exactly when sk is the private key associated with pk.

4.3 Group key management

Group key management provides key management for a logical group of agents, where communication occurs between all members of the group (see [45, 23] for surveys). Group key management schemes fall into three categories: *symmetric-only*, *asymmetric-only*, and *hybrid*, which composes a useful synergy of symmetric and asymmetric operations.

Key distribution involves distributing keying material to some or all group members. Distribution is performed pre-mission (offline/out-of-band) in symmetric schemes,[3]

[2] We note that, in asymmetric settings, private and public keys are logically combined as a *keypair* owned by a agent, e.g. this keypair for Alice: $\{\mathsf{sk}_{\text{Alice}}, \mathsf{pk}_{\text{Alice}}\}$

[3] In practice, a symmetric long-term key may be loaded pre-mission, where ephemeral session keys can then be distributed in-mission via the long-term symmetric key

is unnecessary in asymmetric schemes (assuming agreement on the utilized cryptosystem[4]), and can generally occur during any mission phase under hybrid schemes.

4.3.1 Symmetric-only

Symmetric-only schemes for secure communication require a single shared secret—the group key k—be held by all group members.[5] All outgoing messages m_{send} are encrypted with the group key as $c_{\text{send}} = \mathsf{Enc}(m_{\text{send}}, \mathsf{k})$; all incoming messages c_{recv} are decrypted with the group key as $m_{\text{recv}} = \mathsf{Dec}(c_{\text{recv}}, \mathsf{k})$. Thus, message recovery is performed via the same-key inverse operation as $m = \mathsf{Dec}(\mathsf{Enc}(m, \mathsf{k}), \mathsf{k})$. See figure 4-2.

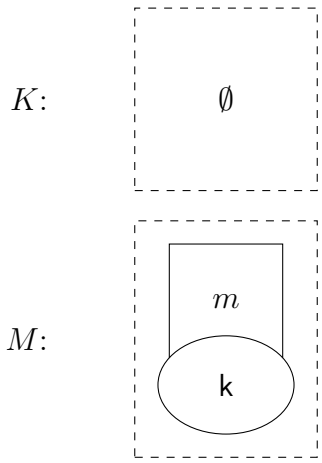

Figure 4-2: Symmetric-only encryption scheme. There is no key distribution message K, as all application messages M are encrypted each a single time as $\mathsf{Enc}(m, \mathsf{k})$.

Conceptually straightforward, this scheme fulfills the required attributes of the simple swarm while allowing low-complexity implementation, high performance [46], and low-overhead broadcast[6] requiring $\mathcal{O}(1)$ encrypt operations.

However, while suitable for simple swarms, and therefore many real-world use cases, symmetric-only schemes fail to support many attributes of the tactical swarm. Perhaps most importantly, consider what happens under compromise of an agent's

[4]Cryptosystem: a fully specified set of algorithms and relevant parameters required for cryptographic operations

[5]For brevity we focus on the case where a single secret is shared between the entire swarm and ignore decentralized schemes which partition the group into separate key domains

[6]Broadcasting $\mathsf{Enc}(m, \mathsf{k})$ allows all holders of k to access m as $\mathsf{Dec}(\mathsf{Enc}(m, \mathsf{k}), \mathsf{k})$

private keying material (Attribute 10): an adversary gains access to the group key k and is now able to decrypt all intercepted messages; this loss of confidentiality and availability persist to mission termination, as symmetric-only schemes provide no mechanism for subsequent cryptographically secure update of group membership.[7]

4.3.2 Asymmetric-only

Asymmetric-only schemes require a public-private keypair per agent, which for agent i we denote $(\mathsf{pk}_i, \mathsf{sk}_i)$. Each agent holds its private key secret, while freely disseminating its public key. Encryption of a message m passed from agent i to agent j is performed with the receiver's public key as $c_j \coloneqq \mathsf{Enc}(m, \mathsf{pk}_j)$; decryption of the respective received ciphertext is performed with the receiver's private key as $m = \mathsf{Dec}(c_j, \mathsf{sk}_j)$. Thus, message recovery is performed via same-keypair inverse operations as $m = \mathsf{Dec}(\mathsf{Enc}(m, \mathsf{pk}_i), \mathsf{sk}_i)$. See figure 4-3.

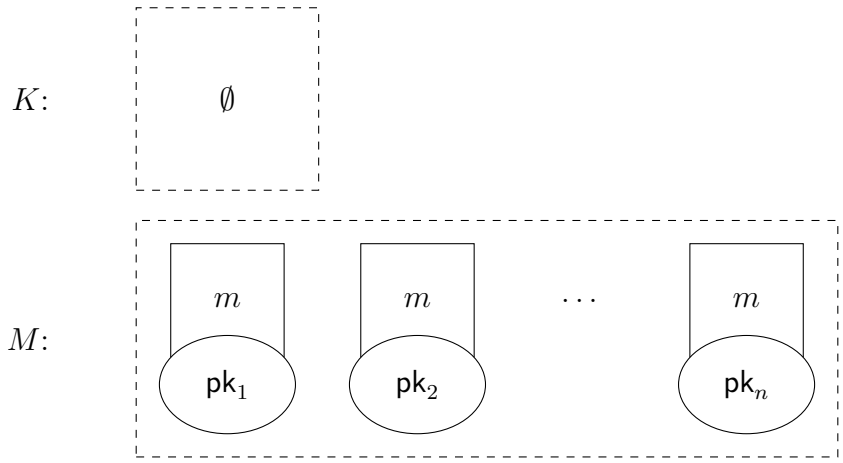

Figure 4-3: One example of an asymmetric-only encryption scheme. There is no key distribution message K, as all application messages M are encrypted once per intended recipient as $\mathsf{Enc}(m, \mathsf{pk}_i)$.

The strength of asymmetric schemes lies in removing the requirement that all group members be provisioned pre-mission with the shared group key, allowing more flexible, ad hoc communication and group membership. However, this benefit comes

[7] E.g. an "update group key" message could be broadcast containing the new group key as $\mathsf{Enc}(\mathsf{k}', \mathsf{k})$, where k is either the previous group key or the group's long-term key, but the eavesdropping adversary holding k would be able to recover k' as $\mathsf{Dec}(\mathsf{Enc}(\mathsf{k}', \mathsf{k}), \mathsf{k})$

at a cost—aside from performance considerations of asymmetric versus symmetric algorithms [17, chap 1.8.4], asymmetric-only schemes fail to support asymptotically-viable broadcast encryption (Attribute 9). That is, broadcasting a particular message m to a group of size n requires n ciphertexts $(c_1, c_2, \ldots, c_n)$ generated by $\mathcal{O}(n)$ asymmetric operations.[8] This makes asymmetric-only schemes unsuitable for even simple swarms due to the non-scalable mechanism for encryption of broadcast messages.

4.3.3 Hybrid

Symmetric-only and asymmetric-only categories each include limitations inherent to the cryptographic mechanism. A hybrid approach, then, seeks to merge the best of each into a performant, usable scheme for secure communication.

The general manifestation involves use of a symmetric group key k for traffic encryption, as in symmetric-only schemes, but to selectively update k—both the key itself and which agents hold the key—via asymmetric channels. Ideally, this key update should be performed as efficiently as possible.

One challenge (or opportunity) for hybrid schemes is in the space of possible compositions of symmetric and asymmetric schemes. That is, a diverse set of hybrid schemes are possible, with accompanying performance and usability trade-offs. One straightforward manifestation, following the "envelope" style[9] via key encapsulation, asymmetrically encrypts the updated group key to each desired group member, as follows (cf. figure 4-4).

1. Choose new key k' and updated group membership N'
2. Encapsulate k' for each member i of N' as $c_i = \mathsf{Enc}(\mathsf{k}', \mathsf{pk}_i)$
3. Collect all c_i a into a single message[10] $K \coloneqq (c_1, c_2, \ldots, c_{n'})$
4. Broadcast K

[8] Contrast with broadcast under a shared group key, requiring a single ciphertext c generated by $\mathcal{O}(1)$ symmetric operations

[9] As in the Cryptographic Message Syntax (CMS) [47]

[10] Assuming a broadcast medium; for more sparse mesh-based topologies, individual messages may be more economical

Each member of N' can recover the group key via $\mathsf{k}' = \mathsf{Dec}(c_i, \mathsf{sk}_i)$. Normal communication then proceeds as in the symmetric-only scheme, encrypting and decrypting messages under the new k'. This results in $\mathcal{O}(n)$ work during the key distribution phase, but allows $\mathcal{O}(1)$ work to encrypt normal application traffic.

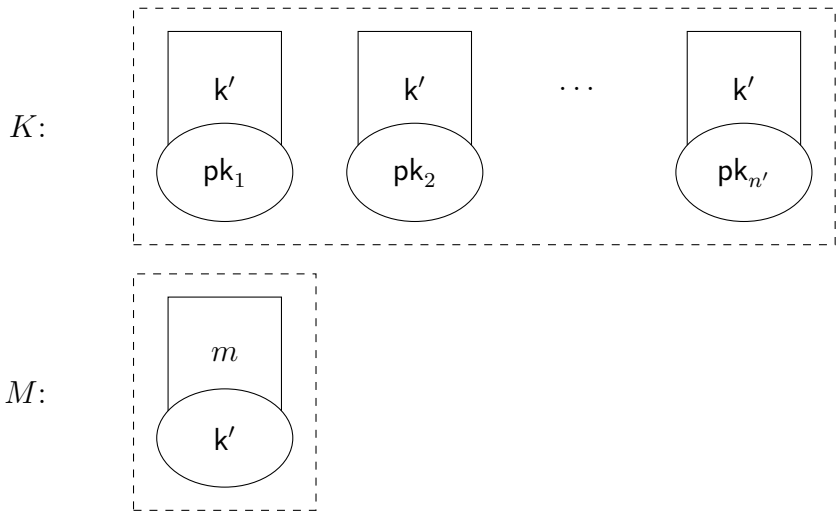

Figure 4-4: Simplified conception of envelope-style key distribution. In this example, the updated group key k' is distributed to each member of group N' as $\mathsf{Enc}(\mathsf{k}', \mathsf{pk}_i)$. Upon receiving the keywrap K, all application message encryption changes over from k to k', encrypting once per message via $\mathsf{Enc}(m, \mathsf{k}')$.

Limitations of hybrid approaches

While this envelope-style hybrid scheme is more usable in tactical swarm applications,[11] it still suffers from $\mathcal{O}(n')$ work per key distribution phase, which stresses the single resource-constrained group controller (Attribute 4) performing the rekey operation for a potentially large swarm (Attribute 1).

One straightforward solution to linear key distribution work, viable in many use cases, is to accept the undesirable asymptotic work of key distribution and restrict group controllers to a subset of high-capability group members (Attribute 7).[12] Unfortunately, this makes usability impositions by precluding swarm homogeneity and hindering swarm mobility and autonomy—that is, the swarm must remain localized

[11] Supporting dynamic group membership, membership revocation, etc.

[12] E.g., a separate ground-control terminal with low size, weight, and power constraints acting as the controller for a localized swarm of resource-constrained aerial drones

near the high-capability controller, which often is less mobile or durable than other swarm agents (conflicting with Attribute 6).

Instead, methods for ameliorating the key distribution asymptotic are desired. Applicable schemes can be found in relevant literature and are discussed in chapter 9.

4.3.4 General group key management

Most usable group key management schemes are hybrid schemes. And since most hybrid schemes take the approach of using a distribution phase to promulgate a shared symmetric group key, we first examine the desired operations and goals of these schemes (see [48, sec. 3] for a more rigorous treatment).

Group membership operations

While many protocols simply distribute a shared group key (i.e., the group is immutable), a number of asymptotically favorable schemes support *group updates* (i.e., the group is mutable). These updates result in a *rekey*, or the ultimate updating of the shared group key, often at a cost less than $\mathcal{O}(n)$. Thus, while traditional protocols lack flexible group membership manipulation,[13] many usable group key management schemes provide mechanisms for a sufficiency of the following group membership updates.

- *Join.* Add a single member to a group.
- *Leave.* Remove a single member from a group.
- *Merge.* Merge multiple groups into a single group.
- *Partition.* Split a single group into multiple groups.

[13] E.g., compromise recovery in the Group Key Management Protocol [49], and in the SCM, requires creation of an entirely new group

After each group membership update, the shared group key must be updated. Thus, for mutable groups, it is natural to refer to a group's current shared key via its *epoch*, a small number incremented upon each group membership update.

Confidentiality goals

In supporting these four group operations, shared group keys must be secured from agents which were not part of the group during the particular group key's epoch. This leads to five desirable properties.

- *Group key secrecy.* Non-group members cannot[14] discover any group key.
- *Forward group secrecy.* A former group member cannot discover future group keys.
- *Backward group secrecy.* A current or future group member cannot discover past group keys.
- *Key independence.* Knowing a particular group key provides no information about group keys from other epochs.
- *Collusion resistance.* No colluding set of current or former group members can collaboratively discover a superset of the group keys they otherwise have access to.

These properties are not mutually exclusive, but are useful for delineating specific concerns with various key management schemes. Key independence, for example, can form a basis for these attributes: when group keys are properly independent, no colluding set of agents can discover keys from epochs for which no agent in the colluding set was granted group access.

Additionally, many concrete key management schemes make security trade-offs to gain efficiency. For example, traditional flat tables [45, sec. 4.6] admit collusion by a sufficiently large subset of agents, to the reward of improved rekey message size and computation.

[14] By "cannot" we informally refer to the computational infeasibility of the action

Example scheme: key trees

Key trees[15] are group key management schemes which construct logical, hierarchical associations between keys enabling efficient updates to a shared group key [45, sec. 4.2]. Variants exist, with the general technique storing state at each group member that scales logarithmically with group size; in return, the computation to update the shared group key also scales logarithmically with group size.

In logical key hierarchies (LKHs), a class of key trees, one group member is appointed as *group manager*, acting as a centralized key distribution center for the rest of the group. The manager maintains a full logical tree of key encryption keys (KEKs), and each other group member holds all keys in the path from its parent leaf to the root node (see figure 4-5). Thus, for balanced trees, the group manager holds $\mathcal{O}(n)$ KEKs, and all other group members hold $\mathcal{O}(\log n)$ KEKs.

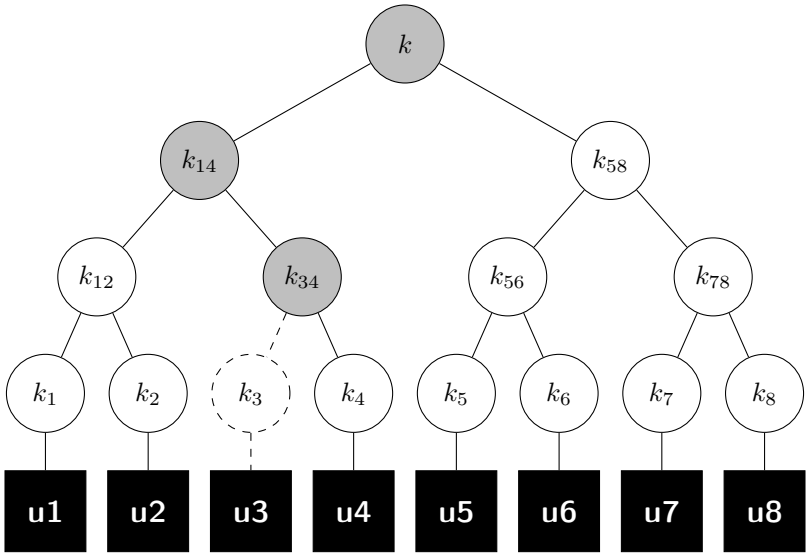

Figure 4-5: Logical key tree for a LKH. Agent u_4 holds KEKs on the path from its leaf to the root node: k_4, k_{34}, k_{14}, and k. KEKs affected when agent u_3 joins the tree: k_{34}, k_{14}, and k.

In figure 4-5, the manager selects k as the shared group key and places it as the new root KEK in the logical tree. It then generates all other required keys and distributes them appropriately.

[15] Also known as *key graphs* [50] or *key hierarchies* [51]

Join. An agent joining the group is assigned a leaf and placed into the tree by the manager; all KEKs in the path from the new agent's leaf to the root of the tree now need to be updated via a rekey message. In figure 4-5, for example, adding u_3 requires updating $\{k_{34}, k_{14}, k\}$. Each updated KEK is encrypted with the KEKs from each of its child nodes, as in figure 4-6; for example, agent u_4 recovers first $k'_{34} = \mathsf{Dec}(c_4, k_4)$, following recursively up the tree until it recovers $k' = \mathsf{Dec}(c_{14}, k'_{14})$.

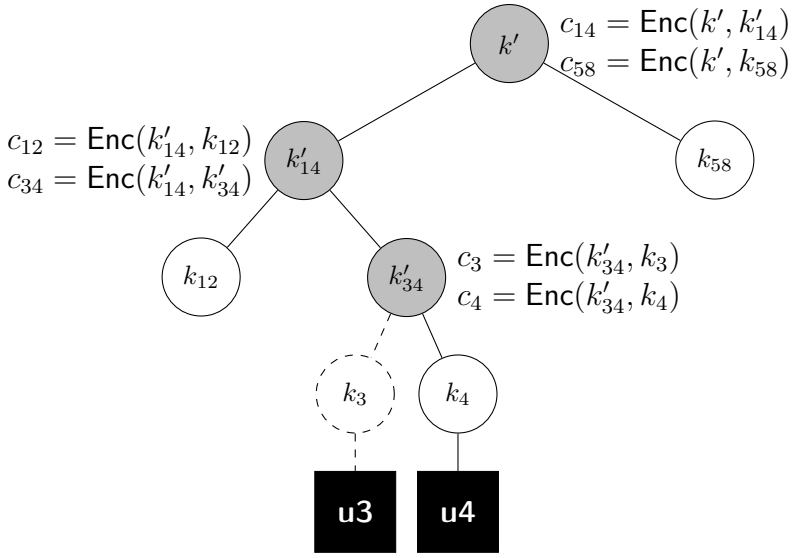

Figure 4-6: Modifications to the LKH upon membership updates. The c_* ciphertexts are used to distribute updated KEKs when u_3 joins the group. If u_3 was leaving the group, c_3 would be omitted.

Leave. The leave operation is similar to the join operation, as all KEKs in the path from the root to the removed agent's leaf need to be updated. The only difference is the removed agent does not receive the KEK of its parent node—in figure 4-6, if u_3 was instead being removed, the update messages would be identical except for the omission of c_3.

Merge, partition. LKHs as presented in [50, 51] do not support multi-member group operations, instead requiring a group update to be performed once per joining/leaving agent.

Note that the rekey message, conceptualized in figure 4-7, needs to exhibit integrity assurances (as do other messages). One way of achieving integrity and authenticity is collecting the rekey fragments into a single message K, then signing and broadcasting

the message. Each recipient then verifies the signature on K before applying the group update. In figure 4-6, we can set $K = (c_{14}, c_{58}, c_{12}, c_{34}, c_3, c_4)$, $K' = \mathsf{Sign}(K, \mathsf{sk}_{\text{manager}})$, and broadcast K'.

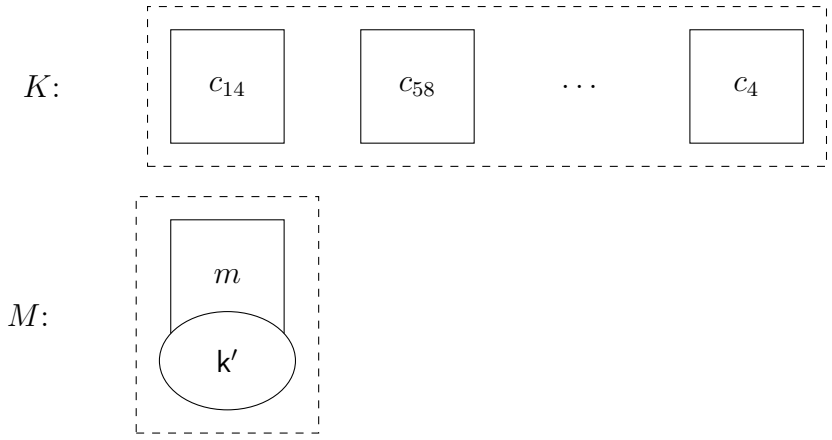

Figure 4-7: Notional rekey (K) and application (M) messages in LKHs. This particular rekey message represents adding or removing u_3 in figure 4-6. The updated group key k' is recovered by successfully decrypting the root KEK.

Comparing schemes

Key trees, finally, provide a key management scheme that appears to serve acceptably well for tactical swarms. We discuss this, other group key management options, and other challenges to scalability in chapter 5; we discuss identified candidate key management schemes in chapter 8. For now, see table 4.1 for a comparison of the applicability and associated costs of the schemes presented so far.

4.4 Related work

The problem of providing a shared group key to a large, intercommunicating group is studied in multiple areas. We first describe a number of well-established yet unsuitable solutions, then overview secure multicast and resource-constrained key management.

Scheme	Computation at the Manager	Computation at the Member	Storage at the Manager	Storage at the Member	Usable in swarms: Simple	Usable in swarms: Tactical
Symmetric	1	1	1	1	✓	✗
Asymmetric	n	1	n	1	✗	✗
Envelope-style	n	1	n	1	✓	✓/ ✗[1]
LKH	$\log n$	$\log n$	n	$\log n$	✓	✓

[1] Envelope-style key distribution schemes for tactical swarms, as seen in e.g. LOCKMA and the SCM, do not scale to tactical swarms of reasonable sizes

Table 4.1: Applicability of group key management schemes discussed so far, characterized asymptotically. Computation is described as the number of asymmetric operations required per message in symmetric- and asymmetric-only schemes, and per group update in envelope-style and LKH.

4.4.1 Unsuitable solutions

First, though, we clarify why a number of existing, varyingly-applicable solutions would not adequately meet the requirements imposed by tactical swarms.

TLS. The security of the Transport Layer Security [19] architecture supports ad hoc key generation between communicating parties, but is coupled to transport over TCP, which is inherently unicast.

DTLS. Datagram Transport Layer Security [52] removes the dependency on TCP, but is still coupled to a unicast communication model by its inclusion of the one-to-one handshake phase for session key agreement.

Bluetooth mesh. Bluetooth mesh [53] is closer to a fully usable solution, enabling a large, dynamic group of resource-constrained nodes to communicate securely. However, as Bluetooth's use case does not require timely updates to shared group keys, its key update procedure [54, sec. 3.10.4] targets infrequent leave and join operations and does not scale well for large groups. It also targets low-security information assurances, as key derivations originate from user-input pin numbers. Finally, it does not support pluggable key management/distribution options.

Static group key. Preloading a static, mission-tied symmetric group key (as in [7]), and performing application-level encryption of all messages, provides a simple solution meeting the majority of tactical swarm requirements. However, dynamic group membership is impossible under a static group key, and compromise recovery

relies entirely on the tamper resistance of each agent's cryptoprocessor.[16]

LOCKMA. The Lincoln Open Cryptographic Key Management Architecture [2] targets unmanned systems and supports dynamic group membership, but its non-scalable key update mechanism precludes full effectiveness with tactical swarms.

4.4.2 Secure multicast

Secure multicast refers to the capacity for a single encrypted message to be readable by a non-trivial subset of possible recipients. While this is straightforwardly achievable via a single shared group key, other desired attributes, such as source authentication, require more involved constructs [57]. The session concept we work with is a direct abstraction of a secure multicast; much of the work in secure multicast literature, however, has focused on IP networks, which often have different requirements than tactical swarms.

Multicast group descriptions vary over a number of axes. As a precursor to our full swarm taxonomy from a key management perspective, we outline a relevant subset of multicast qualities [57], summarized in figure 4-8.

- *Multicast group characteristics.* Looking at the individual agents and the group as a whole, as well as the constraints imposed by the particular application, a set of characteristics emerge from group size to group updates.
  - *Group size.* Expected and/or maximum group size.
  - *Member resource constraints.* SWaP constraints of individual agents, with emphasis on capacity for transmission and cryptographic computation.
  - *Group updates.* Presence, rate, and timeline of any group update operations.
  - *Senders.* Number and type of senders, and whether the set of senders is static or dynamic.
  - *Traffic.* Volume and type of application traffic.

[16] See [55, 56] for a cautionary note on such practices

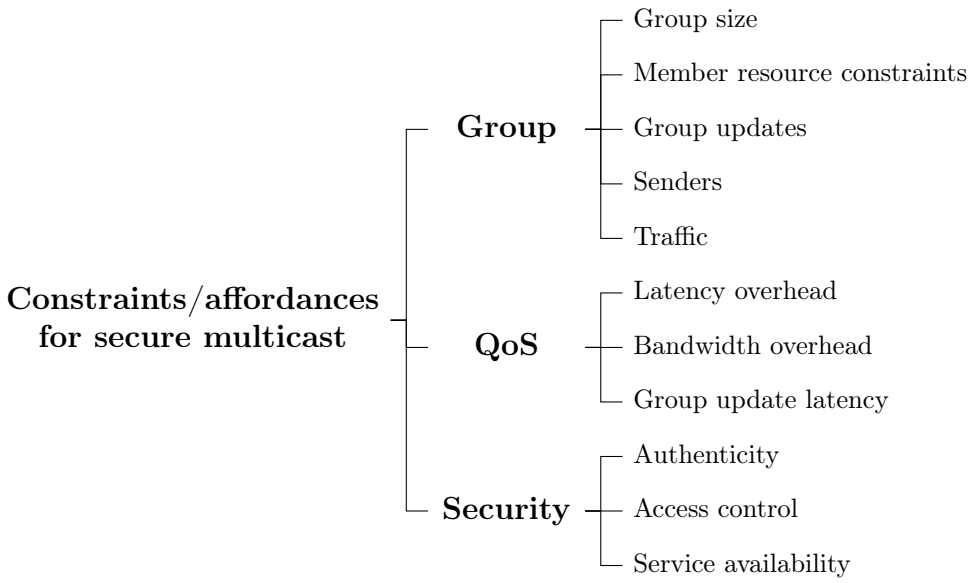

Figure 4-8: Taxonomy of security considerations in secure IP multicast communication, summarized from [57].

- *QoS requirements.* The quality-of-service (performance) requirements imposed by the particular communication-based application running over the group.
  - *Latency overhead.* Percent latency increase caused by per-message operations.
  - *Bandwidth overhead.* Percent bandwidth decrease caused by per-message operations, extra per-message bits, and all key management-related messages.
  - *Group update latency.* Expected and/or maximum time to commit a requested group update.
- *Security requirements.* Security-specific considerations, which don't fit nicely into group-specific characteristics or general performance requirements.
  - *Authenticity.* Group vs. source authentication: whether message recipients need to verify a message originated from the group as a whole or a specific member of the group.
  - *Access control.* The policy for how agents are granted access to the group,

whether the policy is static or application-informed, and which group member(s) are responsible for enacting the policy.

– *Service availability.* Availability at various protocol layers should not be reduced due to adversarial misuse of introduced key management constructs.

Secure IP multicast

Secure IP multicast takes IP multicast routing [58, 59, 60] as an established primitive and seeks to secure multicasts by distributing and efficiently updating a symmetric key shared among members of the multicast group [45, 61]. IP multicast focuses specifically on IP-based networks and Internet-level use cases, resulting in excessively distributed applications compared to the tactical swarm characterization. In [61], for example, the key management scheme is benchmarked for a network with 280,000 routers, and its claim of constant-time overhead for group update operations relies on the spatial clustering of the multicast group members (i.e., precluding effectiveness in ad hoc networks).

Nevertheless, many key management schemes designed with sufficient abstractions (i.e., decoupled from the underlying IP multicast substrate) can be adopted for our tactical swarm purposes.

Mobile ad hoc networks

An ad hoc network is a "collection of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration" [62]. Mobile ad hoc networks (MANET) [62, 63], then, look to provide the full functionality of traditional, wired networks on top of an ad hoc group of mobile nodes, with minimal to no guarantee of the capabilities or allegiances of said nodes.

This focus on extreme generality caused a lack of concrete security solutions for generalized mobile ad hoc networking [62, chp. 12] (see [64] for a survey of key management schemes). For example, allowing truly ad hoc group formation precludes meaningful per-agent guarantees, such as the requirement that secrets, including any shared group keys, be stored in tamper-resistant hardware.

Thankfully, except in perhaps exceptional cases, tactical swarms do not exhibit truly ad hoc relationships among group members or separate swarms, and can often make at least some guarantee on the capabilities of group members. This is perhaps especially true for military applications, a priori trust relationships (e.g. access control and other certificate-based decisions) ultimately stem from a single, DoD-enforced point. This allows us to apply a restricted, concrete subset of MANET research, especially regarding group key management for mobile, physically-vulnerable [65] nodes.

One related description of this narrowed conception of a MANET is the dynamic peer group (DPG) [48]. DPGs are conceived as groups of hundreds (rather than thousands or millions) of members, with emphasis on many-to-many communication patterns (as opposed to many IP multicast-based solutions focusing often on one or few senders). DPGs then are a better approximation to tactical swarms.

Finally, MANET research has also evolved a number of new, promising areas of active research focusing on more concrete use cases such as wireless sensor networks [66], which we describe below.

4.4.3 Resource-constrained key management

Key management for resource-constrained groups focuses on providing primitives and overarching schemes for efficient establishment of secure channels among SWaP-constrained nodes in optionally-mobile networks. We focus on the overarching schemes, though real-world implementations (in the wild or as a pluggable component of our architecture) must also make use of efficient primitives (e.g. [67, 68, 69]).

Wireless sensor networks

We mention wireless sensor networks (WSNs) specifically as the canonical modern effort in providing effective key management to resource-constrained groups. WSNs differ from tactical swarms in focusing more on securing unicast messages in ad hoc

networks[17] and often consisting of more resource-constrained nodes.

WSNs are also assumed to be unreachable for manual configuration—and, conversely, often within reach of potential adversaries.[18] This results in efforts seeking to overcome node capture, node tampering, and denial of service attacks [71].

Thus, the main applicability of WSNs to tactical swarms is in the need to secure multiple protocol layers, and protect excessively resource-constrained nodes against active, often physical, attacks. See [72] for an overview of security in wireless sensor networks.

Key management schemes for WSNs [73] generally attempt to provide information security guarantees without the use of asymmetric cryptography (though some do allow an inexpensive subset of asymmetric operations). This results in many schemes defaulting to a form of pre-distribution of symmetric keys or keying material. This is valuable for tactical swarms resembling WSNs, though this is not the common case.

We now describe a small handful of security frameworks for WSNs, though we note that none prescribe a key management scheme applicable for tactical swarms.

SPINS [74], the Secure Protocols for Sensor Networks, provides a small suite of cryptographic functionality aimed at WSNs. Included in the suite is SNEP, the Secure Network Encryption Protocol, which provides a specific low-overhead format for securing unicast communication in WSNs; SNEP assumes a shared "master" group key is held at all nodes in the network. The suit also describes a construct for lightweight authenticated streaming broadcast named μTESLA.

TinySec [75] is a specified and implemented scheme providing confidential and source-authenticated link-layer communication, aiming to supersede SNEP. Its design serves as one architecture for securing communication below the application layer. It does not, however, specify mechanisms for generating or updating relevant group keys.

TinyPK [76] extends TinySec, assuming the network-wide group key model, by providing (a) a mechanism for non-resource-constrained external parties to securely

[17] Though secure multicast is explored as well, e.g. base station broadcasts to local sensors

[18] E.g., [70] demonstrated that a MICA2 mote sensor, without with tamper-resistant hardware, can be compromised in under a minute

gain access to the WSN's shared group key, and (b) a Diffie-Hellman-based demonstration of the additive *join* and *merge* group operations.[19]

ZigBee, among similar standardization efforts, seeks to "define the network layer specifications" for various network topologies, including peer-to-peer, and "provide a framework for application programming in the application layer" [77]. While ZigBee does allow for multiple possible key management schemes, it does not specify any particular key management solution.[20]

[19] TinyPK's "data access" by the external party causes latency overhead on the order of 10 seconds for its target sensor; its Diffie-Hellman-based group operations have latency overhead on the order of 1 minute

[20] Additionally, since it builds on the IEEE 802.15.4 standard (which defines the physical and MAC layers for low cost, low rate personal area networks [77]), ZigBee is coupled to low bandwidth applications; while this is acceptable for WSNs, it precludes effective integration with tactical swarms

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Key Management for Tactical Swarms

Following the preceding chapters on swarms and key management, we now examine challenges to effective key management specifically for tactical swarms. For context, we first present existing efforts, then move to an overview and enumerate specific challenges.

5.1 Related work

Few published efforts have focused on securing communication in physical, non-simulated drone swarms. A notable exception is the effort to secure the NPS[1] swarm via a pre-placed key scheme [7]. For simulated swarms, a simulated swarm at the AFIT[2] [34] was secured using a hierarchical, multi-key-domain scheme. Other, one-off efforts to secure swarm communication include securing messages in the simulated Yokoyama swarm [78] via asymmetric-only methods.

We examine the NPS and AFIT swarms as the prominent examples of providing secure communication for military swarms. We note that these examples represent ad hoc efforts toward securing communication for specific swarms (as in figure 1-2), while our aim is a generalized security architecture that can then be customized to meet the key management needs of individual swarms (as in figure 1-1).

[1]NPS: Naval Postgraduate School
[2]AFIT: Air Force Institute of Technology

5.1.1 NPS swarm

The NPS swarm use case [12] is well represented by the simple swarm characterization: a swarm of around 100 homogeneous, fixed-wing UAVs, communicating in broadcast-like fashion with the entirety of the swarm. Further, group membership is static and immutable. In this restricted use case, symmetric-only schemes are usable and efficient.

Thus, the scheme presented in [7], which places a shared group key with each agent during the pre-mission phase, is suitable for its application. However, for most extensions of the NPS swarm's functionality (e.g. any application desiring dynamic group membership), the symmetric-only scheme is functionally insufficient. That is, coupling the NPS swarm to a particular key management scheme makes strong usability impositions by precluding a non-trivial set of possible missions.

This vision of a more usable key management scheme is briefly mentioned in [7], stating that "asymmetric cryptography... is a potential option for initially keying or for inflight rekeying."

5.1.2 AFIT swarm

The AFIT swarm use case [34] targets scalable rekeying of a specific use case (see figure 5-1) by simulating a hierarchical, multi-domain group keying scheme based on an architecture designed for LEO satellites [79]. Further work was conducted in [80] to compare per-domain rekeying mechanisms.

The AFIT work marks a positive step toward scalable key management schemes for tactical swarms. The keying mechanism specifically addresses scalability issues by providing multiple group keys based on spatially clustered agents. However, as figure 5-1 depicts, the AFIT use case expects, and is coupled to, a specific application (a two-tiered hierarchical structure of heterogeneous agents). Additionally, the architecture expects to be run over a specific, secured variant of IP multicast, further coupling the key management design to the application and communication protocol. This results in severe usability impositions when looking to adopt the AFIT architecture for use

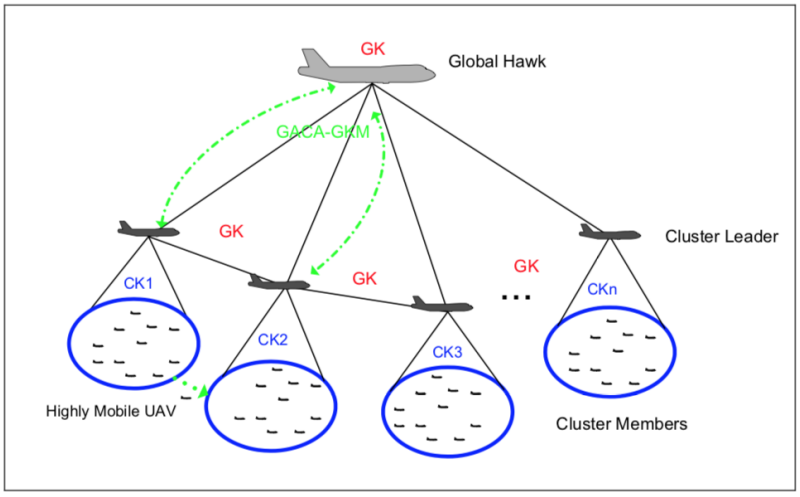

Figure 5-1: Target use case of the AFIT swarm (image from [34]). A swarm director (the Global Hawk) supervises disjoint clusters of UAVs, each with a higher-capacity manager. Mobility between clusters is supported. The group key (red "GK") is shared by the director and all cluster managers; the blue cluster keys (blue "CK") are shared by members of a particular cluster, including their cluster manager.

with general tactical swarms.

Abstracting out the key management scheme specifically, however, we can evaluate the potential of hierarchical clustering as an efficient key distribution mechanism. While this abstraction does allow a more efficient rekey than a flat envelope-style rekey across all agents, it tightly couples the ability to communicate across key domains to the liveness and full operation of the cluster managers—if a cluster manager dies, that cluster's agents cannot perform extra-cluster communication *at all* until a new manager is selected and receives the tier-one group key from the director. This temporary outage is inherent to the multi-domain scheme, requiring re-encryption of the inter-cluster messages at cluster managers.

For an example of cross-domain message passing, consider passing a message m in figure 5-1 from cluster CK_1 to cluster CK_3. The interim c_G-related steps could be eliminated if manager 1 knew the cluster keys of all other clusters, as is common in other decentralized schemes [23].

- Member of cluster 1 generates $c_1 = \mathsf{Enc}(m, CK_1)$ and sends to its manager
- Manager of cluster 1 decrypts $m = \mathsf{Dec}(c_1, CK_1)$

- Manager of cluster 1 re-encrypts $c_G = \mathsf{Enc}(m, GK)$ and sends to manager of cluster 3
- Manager of cluster 3 decrypts $m = \mathsf{Dec}(c_G, GK)$
- Manager of cluster 3 re-encrypts $c_3 = \mathsf{Enc}(m, CK_3)$ and sends to the appropriate cluster member(s)
- Each applicable member of cluster 3 decrypts $m = \mathsf{Dec}(c_3, CK_3)$

Periodic inter-cluster communication outages, caused by reliance on cluster managers for cross-cluster translation, is further exacerbated with each level added to the hierarchical scheme. Additionally, since managers act as gateways for their respective clusters, all inter-cluster messages must first be forwarded through the relevant, necessarily minimally-constrained managers, further imposing on the flexibility (e.g. network flow, group member makeup, manager/cluster relative locations) afforded to the application design and mission-time operations.

Thus, the AFIT scheme is appropriate for this and similar applications, where heterogeneous swarms can be hierarchically and spatially organized around more powerful, long-lived manager agents. Unfortunately, this is not the expected general case for tactical swarms and, regardless, the space of tactical swarm use cases precludes a key management scheme tethered to a particular application construction. Finally, as with similar group key management schemes, there seems to be no gradation or non-uniformity to group access—a problem we address via our session abstraction.

5.2 Problem overview

The specific challenges faced by swarms, and exacerbated in tactical swarms, stem principally from the complexity of individual swarms and the breadth of the field in general. The complexity of effective security is set among an array of challenges to successful autonomous swarming [28, 4], while key management is a single facet of swarm security. The emergence of the field is further hampered, given this complexity,

by more traditional attitudes toward providing general solutions to securing swarm communication.

5.2.1 Complexity of swarm requirements

The scope of swarm requirements produces a wide array of key management requirements, with various often conflicting needs—or, as stated in [57], "the scenarios are so diverse that there is little hope for a unified security solution that accommodates all scenarios."

No silver bullet

Consider, as one example, whether to use asymmetric cryptography for key distribution or rely on pre-placed symmetric-only schemes. For swarms of minimally-constrained agents, such as a swarm of ground vehicles, the overhead of asymmetrically distributing rekey packages could be effectively negligible and therefore a worthwhile trade-off for the benefit of effective, on-demand group rekeying. Contrastingly, for swarms of maximally-constrained agents, such as a futuristic swarm of RoboBees (see [25]), key management must follow pre-placed schemes established in the WSN literature, foregoing asymmetric cryptographic primitives due to the ultra-resource-constrained nature of individual agents. Thus, while the two described use cases could both rightly be considered to be swarms, the sets of acceptable solutions to each are perhaps disjoint.

It is generally assumed, therefore, to be out of the question that a single, unified key management scheme could provide for the needs of the vast majority of (tactical) swarm applications. This is assumed the case in the influencing literatures as well (secure IP multicast, WSN, and others).

We are left then with searching for a minimal set of schemes that would collectively cover all but an acceptably negligible set of use cases, as in figure 5-2.

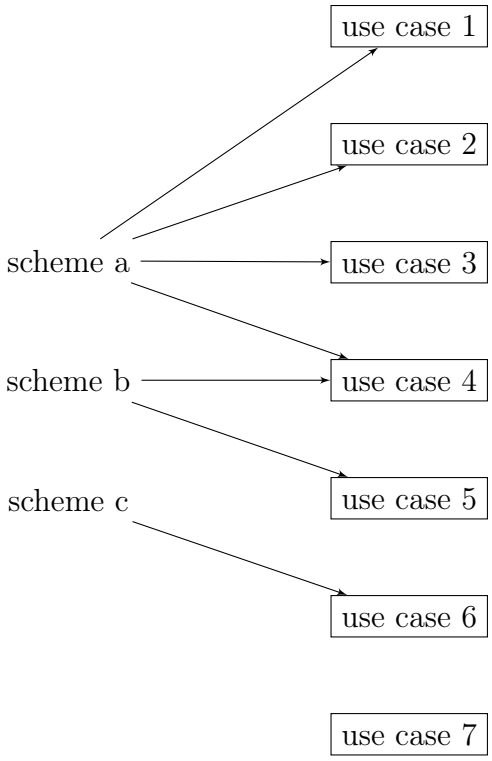

Figure 5-2: Notional selection of a minimal set of schemes aiming to collectively provide effective, application-efficient key management for a plurality of use cases.

Complexity multipliers

In searching for a set of candidate key management schemes, whether described or implemented, a few aspects of the swarm field serve as complexity multipliers.

Newness. Swarm engineering is an emerging field [5], with foundational contributions made around the early 1990s (e.g. [81]) and functional prototypes materializing principally in the past decade. With the only recent emergence of concrete military prototypes to develop against, much of the effort in the field thus far has focused on producing functional prototypes rather than looking ahead to securing said swarms in potentially adversarial conditions.

Breadth. As stated, the breadth of what can be considered a swarm is non-trivial. From platform (e.g. autonomous ground vehicles versus flapping micro-robots) to swarm type (e.g. 10 Reaper drones versus ten thousand relocatable wireless sensors) to application (e.g. wildfire containment versus autonomous ISR), the space and combinations of potential constraints is vast. No single swarm solution, and no single

secure communication scheme, can serve all combinations equally well.

Magnitude. While the breadth of constraints increases complexity, the magnitude (degree) of each constraint is also often pushed to physical or computation limits in particular use cases. Additionally, swarms for military use cases must uphold strict security requirements and gracefully handle adversarial conditions.

5.2.2 Post hoc key management consideration

Securing swarm communication is further complicated by a development environment that often views security and key management as non-core engineering concerns to be tacked on to a product post hoc.

The NPS swarm, for example, was first developed as a research prototype in [12] seeking advances in "autonomous launch, flight, and landing." Later, in [7], a core feature of the security architecture was back-added to provide secure swarm communication.

While not especially toxic for current research-oriented prototypes, the lack of published focus on security considerations has resulted in a deficiency of experiential input. It also poses a more abstract challenge.

5.2.3 Siloed existing solutions

Published solutions targeting swarms specifically are often rigid in their conception (e.g. NPS) and/or coupled to their specific application (e.g. AFIT). Additionally, these, and more generic key management schemes, are siloed to their particular application or literature by the lack of straightforward way to integrate them with existing and future swarm prototypes.

The AFIT swarm is an example of a siloed security solution, developed for a specific application rather than as an independent, pluggable mode of operation in a more abstract architecture; it is also an example of a solution not readily decoupled from its intended application or communication protocol.

What we desire then is a mechanism allowing described and implemented schemes

for secure communication and efficient key management to (a) be developed without targeting or coupled to any exact swarm and (b) be rapidly employed by current and future prototyping efforts. Such a mechanism would benefit both sides: the security researcher would have a larger space of potential testbeds, while e.g. the avionics engineer would have access to a palette of potential solutions to trial and adopt. Looking back to figure 1-5, we can conceive that a properly decoupled application-ECU-radio abstraction would allow key management schemes to be swapped out at the ECU without altering application code, as in figure 5-3.

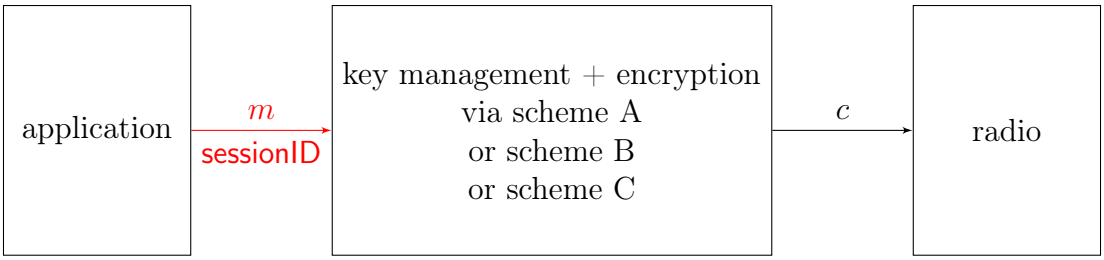

Figure 5-3: Conception of a meta-architecture supporting pluggable key management schemes that minimally disturb application code. For example, if use case 4 from figure 5-2 was targeted, the application could test the comparative effectiveness of key management schemes A and B.

5.3 Specific challenges

We now transition to a number of key challenges faced by general tactical swarms. These challenges follow from the requirements for secure, dynamic group membership in adversarial environments.

5.3.1 Dynamic group membership

The two principal aspects of dynamic group membership are (a) effective compromise recovery and (b) flexible group membership. While the latter makes a key management scheme more usable, the former is a strict security requirement for drone swarms of sufficient lifetime. Successful compromise recovery requires successful execution of the *leave* group update, cryptographically forcing the compromised agent out of the

group through group-wise keying updates that render the captured agent's on-board keying material useless.

Compromise recovery

Given that swarms, and unmanned systems in general, operate in adversarial environments with the expectation of non-trivial agent loss, strategies for handling the compromise of lost or captured agents must be a first-tier consideration. Capture agents should not be allowed to rejoin a swarm, and all keying material aboard the captured agent should be inconsequential by the time an adversary could extract said material.

The need for active compromise recovery measures (e.g. updating the shared group key) depends on expected mission length versus expected efficacy of per-agent tamper resistance measures. Tamper resistance [56, 82, 83] can involve the usage of a security coprocessor to hold keying material and execute cryptographic functions over the keying material, and/or involve precluding tampering with the agent as a whole.[3]

The decision of whether to rely solely on tamper resistance measures for group security is necessarily a per-application decision. Some developers will desire the flexibility of on-demand, cryptographically-supported compromise recovery, at the expense of non-traditional key management. For those applications leaning toward adopting full reliance on tamper resistance, on the other hand, we echo the notes of [55], cautioning against undue dependence on tamper resistance. Placing the entirety of group security on the effective tamper resistance of the individual agents is especially problematic in military swarm applications, where

- prototyping efforts are nascent,
- emergence of new certified solutions is often protracted,
- adversaries are expected to have nation-state-level resources, and

[3]We note that the security architecture from [21, 3] which we adopt (figure 5-3) straightforwardly supports the secure coprocessor paradigm (that is, the ECU acts as the secure coprocessor), while also not precluding any full-agent tamper-resistance mechanisms.

- incentives for low cost and low SWaP imposition per unit are maximized.

In DoD scenarios, a zero-day exploit of a developed swarm's tamper resistance solutions would render the swarm entirely unsecured across all future missions.

We note that agent compromise can occur in two ways: *extraction* of keying material and *hijack* of the agent. For agents with a secure ECU handling storage and operations involving keying material, extraction of keying material requires an adversary to successfully break a tamper-resistant ECU. While this is a non-trivial challenge, nation-state-level resources paired with advances in side channel attacks [84, 85] mean tamper "resistance" is perhaps the loftiest goal of such tactics. For attempts at hijacking an agent, a tamper-resistant ECU need not even be broken—successful hijacking of application code is sufficient to exfiltrate data or influence swarm decisions (defenses addressed in e.g. [83]).

Thus, key management schemes foregoing a compromise recovery mechanism rely on the inexact, quantitatively unspecifiable security provided by tamper resistance mechanisms. And while this is acceptable for current prototypes with shorter mission times, or for general non-security-critical missions, the resilient, responsive, and scalable global surveillance and strike network envisioned in e.g. [9] seems to advocate for a more flexible, quantifiable secure solution.

That said, static-only group membership is effective for many prototypes in current development—for example, Perdix missions [13] last less than an hour with localized deployment from a secure, manned aircraft, and NPS missions similarly last less than an hour [12], with a secure, localized ground deployment.

For these current prototypes, short mission times preclude the need for effective compromise recovery, and secure, localized deployments allow time to manually preload the static group key. However, the mission's strong oversight and non-combat deployment conditions are perhaps unrealistic.

Fully dynamic group membership

In the abstract conception of a swarm, support for fully dynamic group membership is assumed. As a specific example, consider a UAV swarm broadcasting ISR video

to US ground forces. If the video feed would like to be temporarily shared with allied forces, a mechanism for cryptographically-backed group updates is necessary, performing the *join* then, later, *leave* operations for the allied receivers.

When complexity is minimal and SWaP impositions are minimized, then, the capacity for ad hoc group formation and fully dynamic group membership updates can be a worthwhile functionality for general tactical swarms. Moreover, a generalized key management architecture should not categorically preclude use cases requiring dynamic group membership.

5.3.2 The swarm as a distributed system

Swarms necessarily take on attributes of a distributed system, with all the accompanying complexity. That is, the agents of a swarm act in what can be modeled as an asynchronous environment, meaning there is no bound on the delivery time of messages sent between agents [86]. Some key management schemes attempt to avoid the need for distributed systems semantics by incorporating mechanisms such as group update batching (as in [87]) or publishing a public "bulletin" to promulgate serialized group view information (as in [88]); neither of these approaches are particularly suitable for tactical swarm use cases.

Consider first a networked swarm (tactical swarm Attribute 9). In this case, where message passing across a network can be deterred by crashed or malicious nodes, no hard bound can be placed on the expected time for a message to be successfully passed between two nodes in the network. However, even in single-hop, non-networked swarms, a hard bound is still impossible. Consider, for example, a single swarm member which drifts away from the rest of the swarm and out of radio range; after a few minutes, it returns to radio range—this unbounded time away from the swarm results in unbounded time to transmit a message to or from the wandering member. Thus, the wireless, mobile nature of swarms results in an unavoidably asynchronous message passing environment.[4]

[4]Of course, for simplicity, prototype developers can simplify their assumptions for the purpose of dealing with an assumed-synchronous environment, but full-featured, war-ready swarms may not

Having moved then into the arena of distributed systems, we note the mutable, security-related elements which must be agreed upon across the swarm (i.e., a *consensus* [89] must be reached).

- *Group view.* This includes current information on which agents are part of the group, which agent manages the group, any shared keying material (e.g. the shared group key), and any mutable reference to protocols for updating the group view. Lack of consensus about the group view at best could result in loss of availability, while at worst could jeopardize confidentiality or even allow adversaries to gain access to join or direct the group.

- *Member compromise detection.* By detection, we mean the swarm must reach consensus on whether an individual member should be deemed compromised.

- *Blacklists, whitelists, CRLs.*[5] Agreement on these lists constitutes the group access control policy—that is, which agents would be allowed to join the group.

Given the requirement for cross-swarm agreement on at least the above elements, two potential solutions can result from our application-ECU-radio architecture: either the ECU can assume responsibility for its own view of group and group attributes, or the application itself can be held responsible. In this latter conception, the ECU would provide methods for the application to inform the ECU when, for example, the group view was updated.

It may be tempting to lean toward the ECU-driven approach, for the purpose of fully containing all ECU-related concerns to the ECU proper and insulating the application completely from key management-related efforts. However, this approach is inevitably undesirable as the ECU must (a) understand what constitutes agent compromise versus agent disconnectivity, (b) expand past its default manifesto to support distributed systems semantics, and (c) adopt specific consensus algorithms

be able to make such simplifying assumptions when system correctness and availability are mission-critical characteristics

[5]We separate CRLs (certificate revocation lists) from blacklists to indicate that blacklists may be swarm-local lists, while CRLs would be "official" revocation lists signed by relevant extra-swarm authorities

and semantics. In (a), determining what constitutes agent compromise is inherently an application-informed action;[6] in (b), the inclusion of support for distributed systems semantics breaks the modularity of the application-ECU-radio architecture; in (c), the adoption of any specific algorithms or semantics imposes on the application, which may have orthogonal mechanisms for reaching group view consensus, or which function perfectly well with lightweight consensus schemes.

Thus, an application-driven approach seems more suitable, where the ECU provides methods for the application to relay group-related information. This results in a more modular, flexible, and less complex ECU. We note, finally, that perhaps distribution of CRL-like materials could be handled without application involvement, since updates to these can be serialized by e.g. the member-compromise inputs (which would be assumed to have been serialized by the application).

All said, the likeness of (tactical) swarms to distributed systems poses a non-trivial challenge to application developers, and exposing a proper interface between application and ECU must be handled with care.

Compromise detection and autonomy

Given that we adopt the application-driven architecture for dealing with consensus across the swarm, we are left with two needs: (a) the swarm must detect compromise of any constituent members, and (b) some other agent (or human operator) must be responsible for detecting the compromise of the swarm as a whole (the logical swarm agent).

Detecting compromise of constituent members can be accomplished with established mechanisms for reaching consensus on the failure status of distributed nodes (e.g. via a "heartbeat" message) [90], with the need for application-specific policy on what constitutes assumed compromise.

Detecting the compromise of the swarm as a whole, on the other hand, is less straightforward. For single-use swarms, or swarms remaining in range of allied ob-

[6]Consider a swarm which intentionally sends one of its members away from the swarm for a few minutes—and whether the ECU should unconditionally assume that agent's compromise, or the application code should be free the inform the ECU whether the agent has yet been compromised

servation, witnessing to the non-compromise of the swarm can be achieved by active observation by the swarm's human operator. This is perhaps the expected case. For fully autonomous, long-running swarms, however, certainty of the non-compromise of a swarm that was, for example, fully non-observable for two hours is non-straightforward. Whether to trust such a swarm outright, or how to provide otherwise partially effective witnessing, is necessarily left as a per-application exercise.

5.3.3 Key management and the protocol stack

While above we considered aspects of the application-ECU interface, we now examine challenges for the radio and its radio-ECU interface.

In the mission-critical, adversarial scenarios expected for tactical swarms, security requirements must be considered at every layer of the communication stack. We consider layers from the OSI model [91] relevant for communication in an untrusted broadcast medium from a security perspective. Successful security at each layer nominally allows the following communication patterns despite adversarial environment.

- *Physical layer.* Single sender on a single-hop broadcast.
- *Medium access layer.*[7] Multiple senders on a single-hop broadcast.
- *Network layer.* Multiple senders on a multi-hop broadcast.[8]
- *Application layer.* General purpose communication, or exclusion, given the abstraction of stable and secure broadcast and/or routing.

While information security is relevant at all layers of the stack, needs vary across layers. The CIA triad allow clearer understanding of the needs at each layer.

Confidentiality. Data confidentiality can be achieved at the application layer alone via end-to-end encryption—e.g., shared group keys can secure broadcast messages or point-to-point communication within a trusted group. Confidentiality of metadata

[7] Traditionally referred to as the "data link layer" in the OSI model, which focuses on wired links
[8] Note that the network layer is irrelevant for single-hop broadcast use cases

from the network and medium access layers, such as routing or access coordination, can be handled at the corresponding layer.[9]

Integrity. Integrity needs are similar to confidentiality needs: data integrity is handled at the application layer, while integrity of metadata from network and medium access layers is handled at the corresponding layer. Critically, loss of integrity at each layer can cause loss of availability at higher layers.

Availability. Availability is relevant at all layers, but is especially delicate from a security perspective below the application layer.

- *Availability at physical layer.* Preventing DoS attacks on the physical medium, such as jamming attacks [92, 93].

- *Availability at medium access layer.* Preventing denial of service (DoS) attacks on the medium access protocol or on particular group members.[10]

- *Availability at network layer.* Secure routing in mobile ad hoc networks such as preventing malicious route advertisements.

- *Availability at application layer.* Application-level key management over the abstraction of stable and secure routing.

While pre-shared secrets can make secure communication at all levels relatively trivial, such symmetric-only schemes result in brittle design-time constraints and low usability—that is, the key management scheme imposes on application flexibility. Thankfully, availability without pre-shared secrets is possible at the physical layer via secured bootstrapping process,[11] while asymmetric methods can generally be used to achieve availability at the medium access [97, 98, 75], network [64, 99], and application layers [2].

[9]We will target applications where metadata from lower layers, such as routing or medium access coordination, hold lower classification requirements

[10]For the medium access layer, we consider only attacks on the protocol made by otherwise-behaving nodes, since misbehaving nodes (e.g. not properly waiting its turn in slot-oriented medium access schemes) are effectively performing a physical layer DoS.

[11]Some authors suggest uncoordinated spread spectrum [94, 95, 96] can eliminate the need for this bootstrapping process

If the ECU (with input from the application) is ultimately responsible for key management for the group at the application layer, it seems suitable that it should play a role in securing lower layers as well. That is, in security-critical environments where jamming attacks would be expected and only trusted group members can be granted the physical-layer keying material, trusting the radio to handle such operations seems like another break in our modular design—where, for example, the radio would have to fully re-implement group membership semantics for determining group views below the application layer. Instead, the radio should receive from the ECU e.g. the relevant physical-layer key, which the radio can then use to defeat physical layer DoS attacks. In this way, the application-ECU tandem retains full group management concerns, while the radio can bootstrap its anti-DoS efforts off the ECU. We describe our specific solution to lower-layer security in chapter 9.

5.4 Out-of-scope challenges

We focus on key management as one facet of an effective security architecture. For context, we touch on a few notable problems faced by any overall security architecture. See [1] for further challenges.

Resource starvation. If the swarm harvests energy from the environment, adversaries can look to hamper the energy source en masse (consider a smoke-based attack against solar-dependent UAVs). For battery-powered swarms, adversaries can look for inexpensive methods to distract and drain autonomous swarms.

Validation of sensor data. Even the best-designed key management and tamper resistance mechanisms cannot protect against faulty data entering through on-board sensors. Attacks include hampering normal swarm operation (e.g. GPS spoofing attacks [100]) or misguiding learning-based systems by persuading algorithms into false beliefs.

Intrusion detection. Some swarm applications may hold a zero-intrusion policy, adopting strict node compromise policies to preclude reasonable probability of untrustworthy agents re-entering the swarm; for other applications, an acceptable per-

centage of adversarial group members may be permissible. For this latter category, proactive intrusion detection aids in the removal of clandestine adversaries [101].

Secure routing. For networked swarms, such as the FANET use case, placing non-total trust (e.g. in allied UAVs) in network nodes can allow for a larger, more redundant network. If such half-trusted agents are allowed into the network, then secure routing protocols [102] are needed to ensure nodes exhibit proper routing behavior.

Part III

Solutions

Chapter 6

Solutions Overview

We now move from formulating the problem of efficient, flexible key management in tactical swarms toward prescribing a general solution.

The first step in formulating a solution is identifying which key management schemes from the literature support tactical swarm requirements. Toward this end, we propose a taxonomy of swarm needs, intended to clarify and delineate the ways in which swarms constrain the set of applicable key management schemes. Second, with an understanding of the form that candidate key management schemes must take, we recommend and characterize a set of promising schemes from relevant literatures, including respective use cases where each scheme would be most applicable. Third, we flesh out the generic, interface-driven secure communication architecture we have been working toward (cf. figure 5-3), into which the recommended key management schemes can be integrated. Finally, we describe our initial, in-progress implementation of the prescribed generic architecture, along with a subset of the recommended key management schemes.

Our specific contributions are as follows.

- A novel taxonomy of swarm requirements
- An extension of the SCM's architecture for secure communication, aiming to support scalable applications and pluggable key management schemes

- A recommendation of leading candidate key management schemes for inclusion as pluggable options within the proposed secure communication architecture
- An initial, in-progress implementation of the prescribed architecture and accompanying key management schemes

Chapter 7

A Taxonomy of Swarm Requirements

In looking to provide effective key management for tactical swarms, the first step is understanding swarms from a key management perspective. The constraints and affordances swarms provide are similar to those of related systems (§4.4), but do not fit neatly into any single existing literature.[1]

To remedy this lack of a singular, clear view of challenges to effective key management for swarms, we develop a taxonomy of constraints and affordances posed by swarms[2] to key management solutions.

7.1 Taxonomy overview

We look to generate a unified taxonomy enumerating the set of possible swarm attributes which would impose on the choice of key management scheme. With this hierarchically-organized enumeration, we can then specifically describe various swarm characterizations from a key management perspective. Ideally, concise understanding of the key management needs of these characterizations (such as the needs of tactical swarms) should allow straightforward selection of serviceable key management schemes.

[1] Like WSNs and mesh networking, secure communication for tactical swarms can be seen as a practical, less-general continuation of early MANET research [66], with influence from secure multicasting research

[2] And, more generally, by UxSs

In generating this taxonomy, we take influence from the taxonomy of multicast security considerations in [57] (summarized in §4.4), the UAS threat model in [103], and the survey of security challenges for swarm robotics in [1].

We note that, in previous chapters, we presented the characterization of tactical swarms ex nihilo, aiming to motivate this work. While generation of a conceptual taxonomy cannot be described as rigorous, we do look to the proposed taxonomy to provide background and a more general set of characterizations (including a tactical swarm characterization), subsuming efforts in previous chapters.

7.1.1 Method

We loosely follow the method for taxonomy development prescribed in [22]. That is, we alternatively identify new sets of key management schemes (in e.g. [45]) and new sets of swarm use cases (cf. §3.2.3). For each set of key management schemes, we determine what swarm affordances the scheme requires; for each set of swarm use cases, we determine what key management constraints the swarm would impose. Affordances and constraints are organized hierarchically by conceptual similarity, resulting in an incomplete taxonomy.

We continue this process, iteratively identifying and understanding key management schemes and swarm use cases, updating the taxonomy fragment. The objective ending condition is met when a particular iteration produces no update to the provisional taxonomy. At this point, every category in the taxonomy should partially characterize the key management requirements of at least one swarm use case. The subjective ending condition is met when the taxonomy is deemed concise, robust, and explanatory.

Finally, we aim to produce a useful taxonomy. While determining a taxonomy's usefulness a priori is perhaps non-trivial,[3] we can at least say the proposed taxonomy does allow additional insight into what constitutes a tactical swarm, and what constraints tactical swarms place on candidate key management schemes.

[3]Indeed, [22] asserts "determining sufficient conditions for usefulness is difficult and evaluating usefulness may come down to seeing if others use it"

7.2 Generated taxonomy

See figure 7-1 for the proposed taxonomy. The enumeration spans six top-level categories of concern, organized to reflect influence from figure 4-8.

7.2.1 Description by category

We now explain the proposed taxonomy by explaining the constraints and affordances included in each top-level category. We provide a discrete set of options for some attributes as a non-rigorous point of context.

Platform

The platform can impose on key management schemes by providing constrained resources, requiring a non-trivial initialization period, and extended lifetimes.

- *Resources.* The principal resources required for cryptographic operations are efficient computation and non-constrained storage.
  - *Computation.* Which cryptographic operations can be performed (e.g., whether the platform's ECU is capable of performing asymmetric operations), and the comparative duration of such operations.[4]
  - *Storage.* Whether sufficient storage is present to hold swarm-wide data like certificates, group-wide keying material, etc.
  - *Lifetime.* Expected/maximum lifetime of the platform: 10m, 1hr, 10hr, indefinite.
- *Initialization.*
  - *Restrictions.* Restrictions to full functionality of the platform during the initialization period (e.g., perhaps communication between platforms is not possible due to how the platform is stored).
  - *Duration.* Length of the initialization period: 10s, 1m, 5m.

[4]See [25, sec. 3.4] for a discussion of computational capacity in UAVs

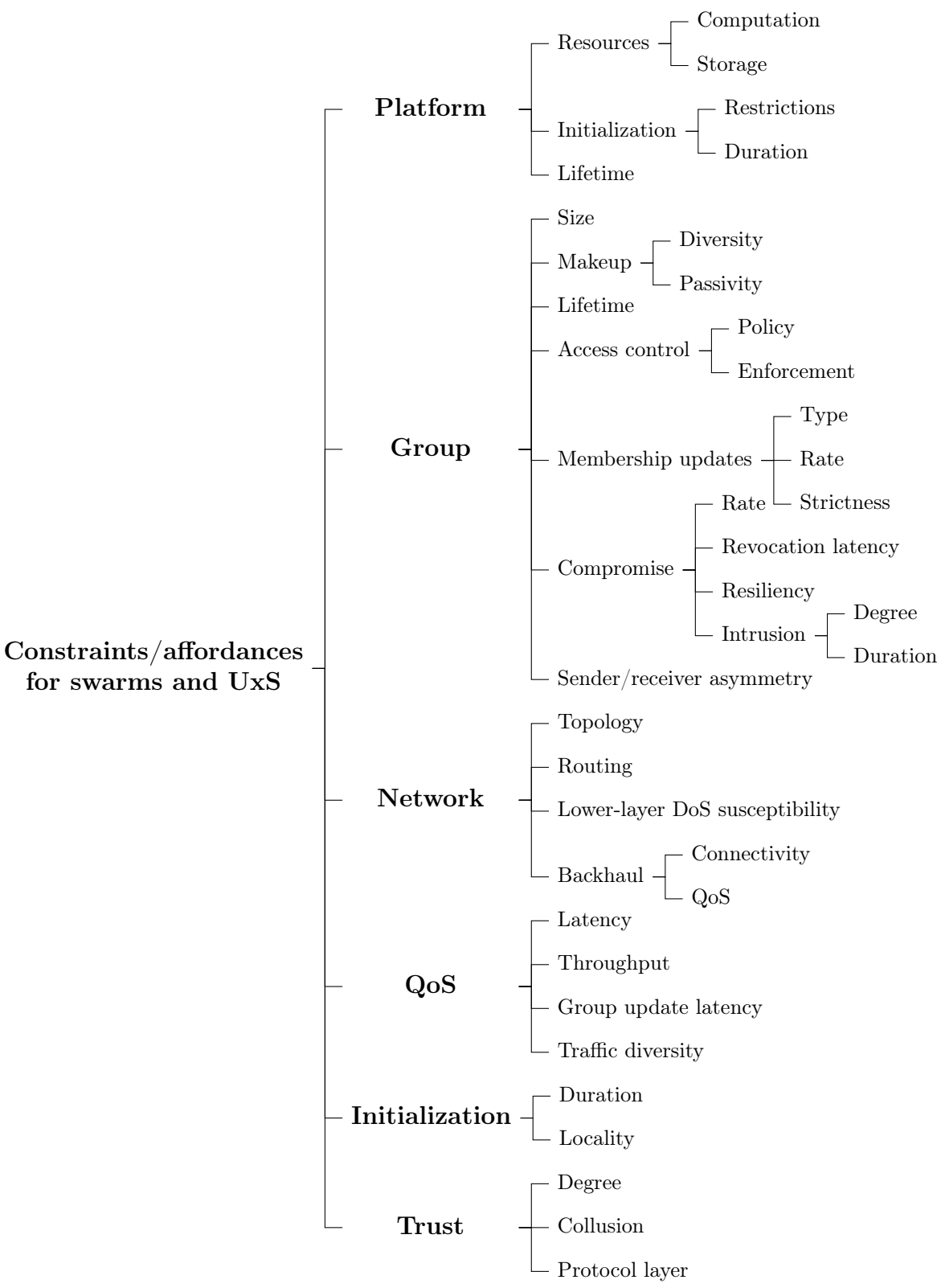

Figure 7-1: Taxonomy of constraints and affordances for key management in tactical swarms.

Group

The largest set of constraints and affordances come with the swarm's system-level group attributes. Its size, makeup, lifetime, updates, and other factors impose on effective key management in variously requiring scalability, prompt group updates, fine-grained access control, and other affordances.

- *Size.* Expected/maximum number of group members: 10, 100, 1000.
- *Makeup.* Describing the platforms that constitute the group.
  - *Diversity.* Whether swarm platforms form a homogeneous or heterogeneous set, and the range of platform-level constraints in a heterogeneous swarm.
  - *Passivity.* Presence and number of semi-passive and/or fully-passive group members.
- *Lifetime.* Expected/maximum mission length: 10m, 1hr, 10hr, indefinite.
- *Access control.* How and where access to the group should be granted.
  - *Policy.* The need for static versus dynamic access control, and binary or fine-grained access.
  - *Enforcement.* Which group members are allotted the capacity to add/remove members, and whether that set is static or dynamic.
- *Membership updates.* Nature of updates to the group view.
  - *Type.* Type of updates allowed: none,[5] known a priori, additive-only, fully dynamic.
  - *Rate.* Rate at which membership updates are requested: per 1s, 10s, 100s.
  - *Strictness.* Cross-update confidentiality guarantees: perfect, lenient.[6]

[5] I.e., static group membership

[6] E.g., whether the group key needs to be changed for additive group membership updates

- *Compromise.* Rate of and group-wide reaction to compromise of individual platforms.
  - *Rate.* Rate of lost platforms: at least one loss per 1m, 10m, 1hr, no loss.
  - *Revocation latency.* Tolerance to lazy/slow group updates; i.e., how long a platform's tamper resistance is expected to protect its secrets and keying material: negligible, 10m, 1hr, 3hr, indefinite.
  - *Resiliency.* Degree to which compromise of a (perhaps majority) subset of the group affects the group's capacity to carry out its mission.
  - *Intrusion.* The application's tolerance to compromised platforms participating in the group, in degree of the intrusion tolerated (none, negligible, minority, non-total) and duration of a particular compromised platform's access to the group.[7]
- *Sender/receiver asymmetry.* Whether access to the group should grant full send and receive capabilities; some use cases may require e.g. send-only or read-only group access (at the application layer).

Network

Network considerations include the local network (topology, routing, sub-application DoS) and the up-link/down-link connection to the backhaul (quality of the connection and DoS capabilities).

- *Topology.* The graph of single-hop edges between platforms: fully connected, ad hoc mesh, etc.
- *Routing.* How messages are routed between platforms: single-hop, multi-hop.
- *Lower-layer DoS susceptibility.* Whether sub-application protocol layers must deal with adversarial DoS efforts, and at which layers (e.g. physical and/or network).

[7] Consider, e.g., a WSN that can tolerate the compromise and continued group access of a minority percentage of its constituent sensors

- *Backhaul.* Presence and quality of the swarm's connection to a centralized controller/processor.
  - *Connectivity.* How frequently, and for how long, the swarm has backhaul connectivity.
  - *QoS.* Quality of the backhaul link via latency, throughput, etc.

QoS

Quality of service is concerned with the application's performance requirements, on which secure communication necessarily imposes.

- *Latency.* The application's expected and worst-case latency requirements.
- *Throughput.* The application's expected and worst-case throughput requirements.
- *Traffic diversity.* Diversity of application traffic; e.g., an application may require both low latency (C2) and high throughput (streaming video).
- *Group update latency.* Application's tolerance to the duration of requested group updates.[8]

Initialization

System-level initialization can consist of actions like noting which platforms are included in the group, initializing sensor aggregation policies, promulgating mission data, etc. The desire for short system-level initialization periods imposes on key management schemes requiring a stable, secure bootstrapping period.

- *Duration.* Required or maximum length of the system-level initialization period: 10s, 1m, 5m.
- *Locality.* Spatial locality of the deployments—that is, whether the full group is initialized in a single location: centralized, distributed.

[8] Group update latency here is delineated from other group update-related timings by referring specifically to the QoS afforded to the application, rather than considerations of confidentiality, etc.

Trust

Finally, trust also plays a role in restricting key management architectures. We care about what it means to trust a platform (degree of trust), at what protocol layers trust is granted, and the application's allowance or expectation of colluding platforms.

A note on trust gradations: we describe *unconditional* trust as fully trusting the relevant platform to act correctly with regard to the confidentiality, integrity, and availability of all data; *cooperative* indicates the application can be trusted to not act maliciously with regard to integrity and availability concerns (e.g., it will not DoS the swarm), but it does not receive full confidentiality privileges (e.g., it may attempt to collude to escalate its confidentiality privileges); finally, *provisional* refers to minimal trust placed in a platform—allowing it into the group, but acknowledging that it may attempt to e.g. DoS the swarm while being a member of the swarm.[9]

- *Degree.* Characterization of what an application indicates by deeming a platform trustworthy: unconditional, cooperative, provisional.
- *Collusion.* Whether platforms will attempt to collude, and the degree to which the application can tolerate collusion.
- *Protocol layer.* Which protocol layers trust is granted at, collusion occurs at, and any granularity across layers.

7.2.2 Summary of taxonomy

The proposed taxonomy organizes salient swarm attributes into six conceptual categories, with each attribute acting as a constraint or affordance to candidate key management schemes. The categorization begins with consideration of a single platform, moving to attributes of multiple platforms (group), how the platforms communicate (network), and performance requirements of applications built on top of these communicating platforms (QoS). Two final categories are added that cross the boundaries

[9]For example, MANETs attempt to allow fully provisional trust in most or all members of the network, resulting in more complicated protocols at the network layer and above

of previous categories: initialization requirements (a group-wide constraint affecting application QoS requirements) and trust (touching e.g. group membership, network availability, and platform affordances).

A few swarm attributes not included in the taxonomy include considerations of autonomy, group structure, and mission/application. Autonomy, both the degree and duration, imposes more on the application (compromise detection) rather than utilized key management schemes. Group structure, describing the group's organization as flat, hierarchical, etc., is subsumed into descriptions of network topology or application-driven preference. Finally, the particular mission to be accomplished by the swarm does place constraints on key management schemes, but in a manner implicitly subsumed by other categories.

Our taxonomy intentionally mirrors the taxonomy of secure multicast requirements in [57], summarized in figure 4-8. Similarities include the inclusion of group and QoS categories, with largely similar attributes. Their security category was subsumed by other categories in our taxonomy; for example, their notion of service availability fits better into network considerations in our hierarchy. The categories added in our taxonomy (platform, network, initialization, and trust) reflect the ways in which swarm key management does not sit perfectly within secure IP multicast research. That is, the platform category breaks the assumption in traditional IP-based multicast that individual nodes are initialized, long-lived, well-provisioned, general purpose computers; the network category details the shift away from assuming a stable, secure, IP-based multi-hop network with backhaul connectivity as the norm; and the trust category examines how security-critical applications require a more fine-grained understanding of trust.

7.3 Revisiting the tactical swarm characterization

We now look back to the tactical swarm characterization presented in earlier chapters. Using the proposed taxonomy, we can transition our description from a collection of unrelated attributes to a taxonomy-driven characterization examining the space of

related attributes. We now describe the characterization by category.

- Platform
  - Resources: computation is constrained
- Group
  - Makeup: diversity is possibly heterogeneous, passivity is allowed
  - Membership updates: fully dynamic (application-driven)
  - Compromise: rate is nonzero
- Network
  - Topology: fully connected or ad hoc mesh
  - Routing: single-hop or multi-hop
- QoS
  - Latency, throughput: application-driven requirements
  - Traffic diversity: expected
- Trust
  - Degree: cooperative

With the tactical swarm characterization placed on top of a more general set of relevant, organized attributes, we can see attributes where tactical swarms make impositions, and also which attributes they leave unspecified—that is, which attributes can vary across specific tactical swarm use cases.

7.4 Example usages

We now detail an example usage of the proposed taxonomy, in two parts. First, an example usage by security researchers, characterizing the swarms their key management scheme can serve, followed by an example usage by swarm/UxS prototypers

looking to understand the constraints their system would impose on candidate key management schemes.

7.4.1 Example usage: MARKS key management scheme

MARKS is a centralized key management scheme (see [45, sec. 5.4]) that selects a length of time, divides it into slices a priori, and uses a binary hash tree to produce group keys for each time slice. For example, in figure 7-2, key k_0 is used for the first time slice in the period, and key k_1 is installed as the subsequent group key at a known time. The benefit of the MARKS scheme is only a single secret must be transferred to a group member per contiguous period of group membership; this comes at the cost, however, of providing additive-only group membership updates and favoring longer group update latencies.

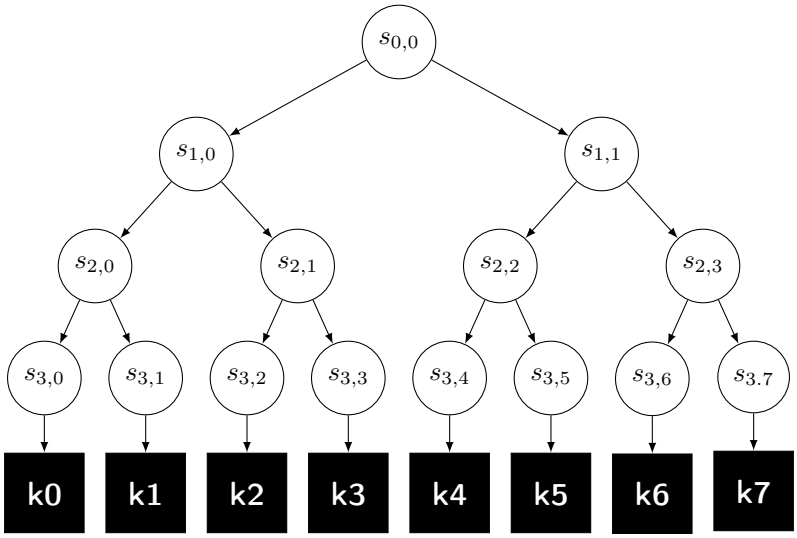

Figure 7-2: Binary hash tree used in the MARKS key management scheme. Each resulting key k_i is used as the group key for some agreed-upon time slice.

The proposed swarm taxonomy can be used to characterize the MARKS scheme as follows.

- Group
  - Membership updates: type is additive-only

  - Sender/receiver asymmetry: no default support for asymmetry

- QoS
  - Group update latency: set a priori (i.e., cannot be changed within a particular hash tree)

- Trust
  - Degree: no default support for provisional trust

Aside from these constraints, the taxonomy can also be used to describe the scheme's affordances: the scheme does not incentivize collusion, the shared group key with efficient distribution supports QoS desires, predetermined (time-driven) group membership updates result in minimal increase in group update latency, etc.

7.4.2 Example usage: a tactical WSN

We use our taxonomy to highlight the main attributes of a hypothetical WSN performing ambient ISR.

- Platform
  - Resources: computation excessively constrained (no capacity for asymmetric operations), storage excessively constrained (cannot store e.g. a symmetric key per sensor in the network)
  - Lifetime: days to months

- Group
  - Size: 100s to 1000s
  - Makeup: diversity is heterogeneous, some passive platforms[10]
  - Lifetime: indefinite (via refill)

[10] Perhaps a number of low-power actuators capable of receiving messages but foregoing transmitting due to power conservation considerations

  - Compromise: rate expected nonzero, revocation latency negligible (no tamper resistance), intrusion tolerated

- Network
  - Topology: ad hoc mesh
  - Routing: multi-hop

With this characterization of needs from a key management perspective, we hope the developers for this hypothetical tactical WSN would be equipped to more effectively convey or understand which key management schemes are applicable to their use case.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

Identifying Candidate Key Management Schemes

As the field matures and prototype efforts turn toward production-ready proof of concepts, plug-and-play secure communication solutions will become increasingly desirable. Before describing our architecture for pluggable, message-based secure communication, we first use the taxonomy generated in chapter 7 to inform a recommendation of candidate key management schemes for use with tactical swarms.

8.1 A tractable set of key management requirements

We begin by describing a tractable, taxonomy-informed set of requirements for candidate key management schemes. Requirements are listed in roughly decreasing order of importance, preceded by their motivating tactical swarm attributes.

- *Broadcast medium; many-to-many communication on ad hoc mesh or fully-connected topology.* The simplest solution for efficient group-wide communication is distribution of a shared group key for symmetric encryption of unicast and multicast messages, though decentralized, multi-domain solutions can be feasible on a per-application basis.[1]

[1] Unicast-dominated communication patterns, or excessively resource-constrained use cases, could also opt for mission-tied key predistribution schemes [104, 73], but these do not immediately support

- *Dynamic group membership.* The shared group key(s) must be updatable, and group keys must be mutually independent.[2]

- *Large group, SWaP constraints, low connectivity.* Updates to the shared group key must be efficient, even when performed by SWaP-constrained group members.

- *Membership update temporal tolerance.* Updates to the shared group key must be timely, as overly-long join (time till new members can receive messages) and leave (time till a compromised node can no longer decrypt messages) operations can be undesirable. Real-time requirements favor membership-driven key updates over time-driven.[3]

- *Zero collusion tolerance.* In looking to support security-critical applications handling potentially classified data, applications desiring cooperative-level trust cannot allow collusion to result in loss of confidentiality.

- *Possible receive-only members.* Key distribution schemes should support use cases where some group members have passive, receive-only radios, precluding contributory key agreement involving the passive member.

- *Potential for targeted DoS.* While DoS attacks on the broadcast medium must be considered, schemes should additionally be resistant to adversarial attacks which directly DoS, confuse, or destroy small subsets of the swarm; to this end, a rekey should require minimal communication rounds and total messages, with preference for schemes which can complete a rekey without full participation from all members.

- *Uncertain time to node failure.* Key management techniques vary in the quantity and location of architecture-relevant state. Given that tactical swarms expect non-trivial loss of group members with minimal warning, key management techniques should consider the trade-off between storing this state at a

efficient multicast

[2]See [48] for a more rigorous definition of key independence

[3]Though batching-based amortization (e.g. [87]) can be integrated as permitted by the application

single member of the swarm (centralized), or spread across many or all members (distributed).

8.2 Identified key management schemes

We enumerate several classes of key management schemes which fulfill the above requirements, proposing for each a principal technique from the class. See table 8.1 for an asymptotic characterization of the recommended schemes.

	Manager		Members				
Scheme	Work	Storage	Work	Storage	Rounds	Messages	Distr.
Centralized direct	n	n	1	1	1	1	✗
Distributed direct	1	1	1	1	$\log n$	n	✓
LKH	$\log n$	n	1	$\log n$	1	1	✗
D-LKH	$\log n$	n	$\log n$	$\log n$	1	$\log n$	✓
TGDH	$\log n$	n	$\log n$	n	1	1	✓

Table 8.1: Asymptotic characterization of the identified key management schemes. Required storage overhead and whether the scheme is distributed in nature; also, per-update requirements of asymmetric work, communication rounds, and total messages. Note that D-LKH requires no more than 3 rounds, and TGDH no more than 2.

8.2.1 Centralized direct distribution

We use *direct distribution* to refer to the direct, stateless conveying of the group key to each group member—that is, the group rekey consists of the new group key addressed to all group members.

The principal incarnation of centralized direct distribution, used in LOCKMA and the SCM, is *envelope-style distribution.* Following the envelope pattern [47] described in figure 4-4, this form of centralized direct distribution chooses a group manager to generate the group key, package it as a single message addressed (encrypted) to each group member, and broadcast the rekey message.

Centralized direct distribution is the optimal choice for many use cases. For medium-sized swarms with minimal group membership updates, high update latency tolerance, high-quality backhaul connection, and only minimally-constrained plat-

forms, the simplicity of centralized direct distribution outweighs its non-ideal scalability.[4]

Advantages. Simple, stateless, collusion-free.

Drawbacks. Non-scalable for larger swarms of constrained platforms, particularly where QoS or compromise tolerance requirements are paramount.

Target use cases. Small to medium groups of minimally-constrained members with a low rate of group updates or relaxed QoS requirements.

8.2.2 Distributed direct distribution

The distributed form of direct distribution maintains a centralized source of distribution, but enlists trusted peers to aid in confidentially promulgating the group key. A principal example from this class is *tree-based envelope forwarding*, where the group manager forwards the group key to k peers, who are each tasked with recursively forwarding the group key to their respective subset of the group. Under this scheme, each platform must perform only k distributions, at the cost of a full rekey requiring $\mathcal{O}(\log_k n)$ rounds of message passing.

Advantages. Simple, stateless, collusion-free, and more scalable than its centralized counterpart; supports excessively-constrained platforms by minimizing computation at any particular group member.

Drawbacks. Non-trivial group update latencies, especially for swarms with high per-message communication latencies.

Target use cases. Large groups of excessively-constrained members with relaxed group update latency requirements.

[4]Consider e.g. a 1000-member Kiva [14] swarm—group membership changes are relatively infrequent, a particular group remains localized to its warehouse, and nodes are largely not resource constrained. Thus, a dedicated group member (though not necessarily an actual Kiva robot) could be charged with effectively managing group membership updates via centralized direct distribution techniques.

8.2.3 Centralized key tree

As described in §4.3.4, *key trees* are stateful schemes utilizing logical hierarchies of symmetric keys to asymptotically reduce the computation needed to produce a rekey message. The principal (and original) centralized key tree is the *logical key hierarchy (LKH)* [50, 51]—it is collusion-free, supports one-round group updates, and requires only logarithmic computation at the manager.

Advantages. Non-complex, collusion-free, and scalable to arbitrarily large group sizes (restricted only by the adherence to a centralized manager). Supports constrained platforms and stronger revocation latency and QoS requirements.

Drawbacks. Key trees reduce group update latencies by storing state (the key tree) at the manager, resulting in a stateful protocol. Storing the key tree requires $\mathcal{O}(n)$ space for the tree and $\mathcal{O}(n \log n)$ for each cached rekey message, where one mechanism for combating DoS attacks involves caching multiple rekey messages at the manager.

Target use cases. Medium to very large groups of constrained members, without excessive storage constraints, requiring lower group update latencies. However, group initialization can be non-trivial in large groups without a higher-capacity manager ($\mathcal{O}(n)$ asymmetric operations to distribute the initial KEKs), so high-threat scenarios where group managers could be selectively targeted would result in reduced QoS.

8.2.4 Distributed key tree with transport

While centralized key trees store the entire key tree at the manager, distributed variants store subsets of the tree across the group, resulting in a scheme more resilient against targeted attacks. Distributed key trees can produce KEKs at each node in the tree via *transport*, where one node selects the KEK and securely transports it to other relevant nodes, or *agreement*, where relevant nodes each have some input into producing the KEK. We look first at transport-based schemes.

The attractive scheme in this class is the *distributed LKH (D-LKH)* described in [105], requiring exactly 2 communication rounds per group update (3 for group

initialization), while recovering efficiently from loss of the group manager. Additionally, only $\mathcal{O}(\log n)$ asymmetric operations are required at the manager for group initialization, rather than the $\mathcal{O}(n)$ required by the centralized variant.

Advantages. Collusion-free, fault-tolerant, acceptable group update latencies in both expectation and worst-case; more scalable than its centralized counterpart, as the manager requires asymptotically less computation and storage. Supports excessively-constrained platforms by minimizing computation at any particular group member.

Drawbacks. Not as simple as previous schemes, requires stronger group membership and message delivery semantics, and more susceptible to selective DoS during initialization.

Target use cases. Medium to very large groups of constrained to excessively-constrained members where preventing worst-case group update latencies (and group membership revocation) is paramount. Secure, stable bootstrapping period is desirable.

8.2.5 Distributed key tree with agreement

The attractive scheme in this class is *tree-based group Diffie–Hellman (TGDH)* [48], a blend of distributed key trees and Diffie-Hellman key exchange. The scheme uses modular exponentiation as a blinding function to achieve 2-round join operations requiring only 3 messages, and 1-round leave operations requiring only 1 message. The cost of this efficiency (minimal rounds, minimal messages, and message sizes logarithmic in group size) is requiring $\mathcal{O}(\log n)$ modular exponentiations at each group member per update.[5]

Advantages. Highly scalable in group size, collusion-free, flexible access control enforcement (any group member can independently perform a group membership update). Fully distributed/fault-tolerant, without resulting in excessive message count overhead (more resistant to targeted DoS attacks). Computation and storage require-

[5]Specifically, the manager performs no more than $2\log n - 2$ modular exponentiations, while all other group members perform no more than $\log n - 1$

ments are uniform across group members.

Drawbacks. The requirement for logarithmic modular exponentiations may impose on more resource-constrained platforms, resulting in decreased QoS; specifically, group update latency increases proportionally with the computational constraints of the most constrained group member, meaning TGDH may not be appropriate for heterogeneous swarms.

Target use cases. Medium to very large groups with no excessively-constrained members. Use cases requiring frequent, concurrent group updates, with desire for efficient batched updates (merges/partitions), and stronger protection against targeted DoS attacks.

8.2.6 Non-recommended schemes

For context, we include a set of schemes which were appropriate save a fatal flaw.

- *One-way function trees (OFT)* [106]. OFTs provide reduced message sizes compared to LKHs, but are insecure against certain collusion attacks [107, 108], precluding effectiveness under collaborative trust assumptions.

- *Distributed OFT (D-OFT)* [109]. Provides distributed, fault-tolerant agreement on a group key, but requires $\mathcal{O}(\log n)$ communication rounds per group membership update.

- *(Distributed) flat table (FT, D-FT)* (see [45]). General flat tables purchase efficiency at the cost of imperfect collusion resistance.

- *Collusion-resistant flat table (C-FT)* [110]. Eliminates the imperfect collusion resistance of FTs at the cost of prohibitively large ciphertext overheads for non-trivial group sizes.

- *Time-driven* schemes (see [23]). Schemes such as Kronos [111] support large group sizes by batching group update requests, performing a rekey only at time-driven intervals. While this is attractive for some use cases, it clearly results in non-trivial group update latencies.

- *Self-healing* schemes [112, 113]. Allow efficient rekey with emphasis on approximating a stateless scheme; unfortunately, the mechanism for rejoining a group after losing a rekey message plays out over several additional group updates, resulting in unacceptable QoS trade-offs; imperfect collusion resistance is also a concern.

8.3 Discussion of identified schemes

As discussed in §5.2.1, no single identified scheme functions as a silver bullet to serve every desired swarm use case. Indeed, each constrain their space of applicable use cases in some non-trivial way. However, informed by the taxonomy of swarm key management needs, we understand how these constraints differ across schemes (see table 8.2). Ideally, it is these differences which would allow this set of schemes to collectively cover the majority of realistic use cases.

	Group			QoS	Init.
Scheme	Size	Diversity	Update rate	Update latency	Duration
Centralized direct	↓	↑		↑	↑
Distributed direct	↑			↑	↑
LKH	↑	↑		↓	↑
D-LKH	↑		↑	↓	↓
TGDH	↑	↓	↑		

Table 8.2: Rough comparison of the applicability of the recommended schemes, informed by a subset of the taxonomy categories proposed in chapter 7. Upward arrows indicate increases, downward decreases; black indicates comparative improvement, gray comparative decline.

Observing table 8.2, we see how the recommended schemes compare across a subset of taxonomy categories. For example, centralized LKH allows for large group sizes and heterogeneity in group membership (constrained members can be supported by a high-capacity manager) and lower group update latencies; however, it suffers a longer initialization duration (initializing the centralized key tree) and does not improve on the update rate (centralized schemes require all membership update requests to serialize through the manager). From another perspective, support for group diversity is

higher in the centralized schemes, where a high-capacity centralized manager allows for highly constrained group members, and low for TGDH, which requires effectively uniform work across group members. Other categories from the taxonomy include e.g. platform-level computation, where TGDH makes stronger impositions on this category by requiring all platforms to perform efficient, timely modular exponentiations.

Consider finally, as two examples, Perdix drones and RoboBees. For the first, imagine a swarm of Perdix drones streaming video-based ISR to allied forces, but with differing classification levels across time (i.e., allies' access to the feed must be selectively rescinded). In this case, the distributed LKH scheme is attractive, as no Perdix drone is required to undertake $\mathcal{O}(n)$ asymmetric operations, while group update latencies have a strong, non-terrible upper bound. For the second, imagine a futuristic swarm of 1000 RoboBees is engaging in high-resolution ISR in a combat zone, with continuous refill and no requirements for fine-grained application-layer access control. In this case, the tree-based envelope forwarding example of distributed direct distribution is attractive, as it places minimal imposition on the excessively-constrained RoboBee platform and has acceptable revocation latencies, while the non-ideal QoS-related group update latencies are unimportant for the particular mission.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 9

An Architecture for Scalable Key Management in Tactical Swarms

After scoping and shaping our view of the problem faced in providing effective key management and secure communication for tactical swarms, we now transition to describing our candidate conceptual solution. The architecture looks to provide a clarifying, simplifying set of abstractions to tame operational complexity while not overly stressing application flexibility; it also serves as a unifying interface, allowing a single application-facing API to support the full set of key management schemes identified in chapter 8, swappable with minimal changes to application code.

In looking to provide a reusable, composable, and widely-applicable architecture in the face of varying swarm key management requirements, we turn to a generic, pluggable meta-architecture as outlined in figure 5-3. That is, decoupling the application, radio, and cryptographic components of each platform should allow abstract key management schemes to be swapped out at the ECU with minimal code changes at the application or radio.

Our conceptual separation of concerns, along with further architectural decisions below, will necessarily require real-world implementation to determine its true suitability, as well as to determine which decisions or assumptions should be altered. Prototyping work of this sort is under way, as described in chapter 10.

In this chapter, we begin by describing our conceptual solution, with its restriction

of supported use cases and prescription for how secure swarm communication should look. We then move to the resulting abstract interfaces allowing effective collaboration of the application, ECU, and radio, and providing a minimal example of its use.

9.1 Conceptual solution

Our proposed conceptual solution utilizes a functionally-decoupled platform architecture (the application-ECU-radio abstraction) to transition from coupled, library-based key management toward fully decoupled ECU functionality. At this simplified interface, we provide a session-based communication abstraction with updatable sessions (both membership views and metadata). We describe how concepts like physical-layer secrets,[1] compromise detection, limited delegation, and passive group members work with this platform-level decoupling of cryptographic responsibilities, ending with a summary of the architecture.

9.1.1 Platform-level affordances

First, we assume a set of platform-level affordances as the building blocks for the system-level architecture: cryptographic decoupling, identity, and tamper resistance. This is one way in which tactical swarms differ from generic MANETs—the individual platform is well understood and can make specific guarantees, allowing a reduction in system-level complexity.

Cryptographic decoupling. We assume the platform follows the platform-level architecture prescribed in previous chapters (reproduced in figure 9-1, where session epochs are descried below). Specifically, the cryptographic functionality should be abstracted out to a simpler, message-based interface, with an intermediary ECU providing for required cryptographic functionality.

Unique identity. Each platform has a globally unique identity, with accompanying identity and signing certificates, loaded to the ECU during the *fabrication* phase of operations. This identity is not tied to any particular mission.

[1] I.e., TRANSEC keying material

Tamper resistance. Every rekey operation takes some amount of time. While we deem the multi-hour requirements of static-only schemes to be inflexible, a form of at least mild tamper resistance is required to allow for rekey operations to allow local blacklists to update, nullifying the platform's identity-based secrets.[2]

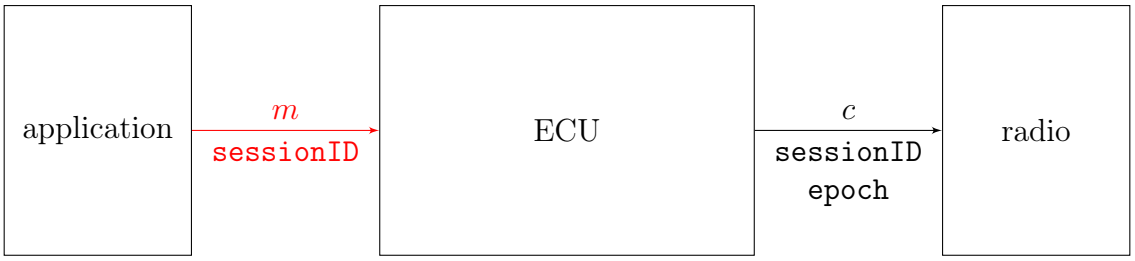

Figure 9-1: Platform-level architecture decoupling application, cryptographic, and radio concerns. The ECU receives a plaintext message and session ID from the application, then forwards the duplet to the radio as the associated ciphertext, same session ID, and appropriate session epoch. The radio can then route the $(c, \texttt{sessionID}, \texttt{epoch})$ triplet, based on the appropriate membership view, which it may need to request from the ECU (and can cache for subsequent messages to the same session ID/epoch combination).

9.1.2 Secured bootstrapping period

For simplicity of operations, we restrict our model to include an initial secured bootstrapping period of some desired length. The bootstrapping period allows time for promulgation of physical layer secrets, as well as efficient group initialization. While some applications can choose to forego this affordance via application-specific trade-offs, dealing with assumed-adversarial environments is easier with the assumption of an initial period with stable group membership and secured communication channels.

Concerning length, the duration of the bootstrapping period is unspecified, though minimization is desirable. For context, we (non-rigorously) view 5 minutes as a reasonable upper bound, 1 minute as agreeing with existing bootstrapping period lengths, and 10 seconds as an ideal goal.

[2]Some applications may choose to forego any tamper resistance whatsoever, e.g. many WSN applications. We focus on use cases where full confidentiality and zero intrusion are desired, as this is the general case for swarm and UxS-like systems.

The bootstrapping period is assumed to occur during the *pre-mission* phase (see §3.2.2). However, for sufficiently short periods, bootstrapping can occur during a well-prescribed *launch* phase, allowing more application flexibility.

The need for a bootstrapping phase is demonstrated at the platform level and the system level. For the platform, mission-specific data and secrets must be loaded and initialized, such as application code, initial blacklists and whitelists, any frequency hopping secrets, initialization of the ECU's interfaces, configuration of the radio, etc. For the system, secured and efficient mission-specific agreement and validation can occur such as clock synchronization, group-wide certificate validation, and any promulgation of the physical layer secrets. Note that it is this system-level bootstrapping that can, in some cases, be transitioned to the launch phase of operations to allow flexible group formation.

Looking specifically to physical layer security, the assured existence of a bootstrapping period provides a straightforward solution to providing availability at the physical layer (see §5.3.3). That is, the one-time promulgation a shared secret across the group is straightforward within a secured bootstrapping period, where confidentiality and full availability is assumed.

For applications looking to move the system-level bootstrapping actions from the pre-mission to launch phase, application-specific mechanisms must be implemented to ensure effective first-pass promulgation of physical-layer secrets. Example solutions include application-specific low-power, directed cleartext unicasting of the TRANSEC keying material, or perhaps mechanisms pulling from uncoordinated spread spectrum techniques [94, 95, 96].

Finally, considering the flexibility of supporting centralized versus distributed pre-mission and launch phases—agents should not be required to be launched from a single location. Fortunately, the only requirement in distributed launches is promulgation of the physical-layer key. Also, where immediate physical-layer DoS is not a primary concern, all system-level bootstrapping can occur entirely during the *launch* or *in-mission* phases.

9.1.3 Session-based communication

We use the notion of a session with mutable attributes to logically and cryptographically handle communication with perhaps multiple subsets of the group, and to support gradations of group membership. Membership in the session affords full privileges to send and receive (encrypt and decrypt) all messages sent to the session. Additionally, we require the application-driven existence of a single, swarm-wide session consisting of all group members.

Each session has a globally unique ID[3] and a current *epoch*. While a session itself is mutable, the session-epoch duplet is immutable and thus uniquely defines the session's attributes for the particular epoch (e.g. the session key).

Tactical swarm topologies can be described as either fully-connected or a proximity-based mesh with frequently updated edges [4].[4] In either case, over the course of a mission, a particular group member may need secured unicast, anycast, and/or multicast to a non-strict subset of the group.

The session-like interface we describe (§9.2) is manipulated via the following notional actions.

- *Create.* An agent creates a session. Distribution of keying material and the shared session key allows encryption of application data to all members of the session.

- *Send, receive.* An agent sends and receives application data multicasted to the session.

- *Update.* Agents can join and leave the session, following appropriate group access policies.

Finally, we make clear the distinction between a group and a session. We consider a *group* to be the concrete, logical collection of all agents in the swarm or UxS.

[3]Mission-unique IDs are acceptable, though global uniqueness is preferred for full flexibility

[4]E.g., the NPS swarm is fully connected, and the AFIT swarm can be described as a mesh with proximity-based mobility

Conversely, we consider a *session*[5] to be a logical subset of the main group.[6] The difference, as we describe, is in the protocol layers. That is, we describe a group at the physical layer, while a session is denoted at the application layer. For conceptual and operational simplicity, we enforce that an agent can be in only one group at a time, and every session must be a subset of its physical-layer group.

Required group-wide session

For conceptual and operational simplicity, we enforce that there must exist one session which includes all group members, denoted `all`. This must be initialized by the application, as the first session created, ideally during the secured bootstrapping period (though other solutions are possible, see §9.1.6).

The group-wide data (CRL, blacklist, whitelist) are stored as attributes of the `all` session, allowing updates to e.g. the blacklist to (eventually) be promulgated to all members of the group, serialized via the session's epoch.

For use cases like the NPS swarm, the `all` session is sufficient as the only utilized session (binary group access), while use cases requiring more fine-grained control over access to application data can make use of additional sessions on top of the `all` group-wide session.

Create a session

Regardless of utilized key management scheme, a session's full attribute set, replicated at every session member, includes the same metadata.

- *Session ID.* (*) The globally unique session ID.
- *Epoch.* (*) The session's current epoch, denoted as an integer incremented on each session update.
- *Manager.* The agent responsible for managing the swarm, and perhaps serializing key management decisions. The manager designation involves differing roles

[5]Conceptually similar to a *cryptonet* [24], though we opt for session-based terminology

[6]Notionally, a group versus a session could be considered a similar comparison to a broadcast versus a multicast

across key management contexts—anything from nominal managerial duties to full responsibility for secret distribution.

- *Membership view.* List of agents in the session. If the key management scheme allows, agents can be nested (e.g. a session listed as a member of another session).

- *Protocol.* The key management scheme employed. May contain additional, protocol-specific information from the application.

- *Options.* Miscellaneous options, such as whether to require session members to provide confirmation of session updates.

While the above items are coupled to a particular session, we include a set of session-relevant data shared across sessions in the following.

- *CRL.* (*) Mission-static certificate revocation list, accrued from previous missions of this and other entities under a particular certificate authority.

- *Blacklist.* (*) Localized blacklist, allowing rapid and perhaps overly risk-averse blacklisting.

- *Whitelist.* Application-specific whitelist, allowing the application to severely restrict the set of agents allowed into session.

The starred session ID, epoch, CRL, and blacklist are set by the ECU,[7] while all others are initialized by the application during the session *create* action. The blacklist and whitelist are coupled to the mission (i.e., at any point in time, they should be identical across sessions), but are promulgated as attributes of the `all` session.

Send and receive under a session

All *send* and *receive* operations must result in messages which include the relevant session ID and epoch. Together, these uniquely define an immutable set of session

[7]The initial blacklist must be loaded during the pre-mission phase or promulgated at a later point, while updates to it are processed via application-informed compromise detection

attributes, which the ECU can use to determine the appropriate session key for encryption and decryption of application messages. Thus, while the application sends its message m and the accompanying sessionID, the ECU's output message must include, additionally, the relevant session epoch.

Update a session

Session *update* operations, whether mutating the session's membership view or other attributes, requires the ECU to update (increment) the session's epoch as well. Additionally, since session updates can trigger substantial computation and message passing within the swarm, availability attacks must be considered.

Platform-level availability. Here we look primarily at the statefulness of the ECU's rekey messages—that is, whether an agent that missed all rekey messages for epoch n can seamlessly resume operation in epoch $n + 1$. For stateless protocols, such as the centralized direct distribution used in LOCKMA and the SCM, loss of all rekey messages for epoch n has minimal[8] or no effect on an agent's efficient participation in subsequent epochs.

For stateful schemes, however, we must account for a recovery mechanism as well as DoS prevention. Generic recovery involves first asking the manager for the rekey messages from the relevant epoch(s). If the manager no longer holds the rekey message, then the recovering agent can ask to be added back into the group.[9] This can result in the loss of application messages that may have been encrypted to the session under a particular epoch; we assume application-level mechanisms to be in place for providing reliability, ordering, and error-checking guarantees at the application level.

For DoS prevention under stateful schemes, we note (a) our threat model assumes the need for physical-layer DoS protection and zero allowed intrusion, meaning the majority of use cases will not have to deal with DoS protection at higher layers; and (b) regardless, for the sake of use cases like the FANET which allow "half-trusted"

[8]Some of the more intricate "stateless" schemes require a number of intermediary epochs to pass before seamless participation can be resumed, e.g. [112, 113]

[9]Adding an agent back to a group can look different across key management schemes—for centralized LKH, the manager can unicast the relevant keying material (subset of KEKs), while for decentralized key trees, performing successive *leave* then *join* operations is perhaps more efficient

nodes, stateful key management schemes should be supplemented with DoS-aware ECU functionality such as caching of rekey messages and prioritizing responses to well-behaving nodes.

System-level availability. Here we look at the effect of agent failure during the rekey process. For swarms with high loss rate, large session sizes, and/or imperfect availability, a rekey operation should succeed even if one or more session members fail to participate in the rekey operation.[10] Additionally, failure of the manager must be considered. In distributed schemes, a new manager can often be appointed without great effort, though manager loss in centralized schemes may require non-trivial computation to initialize the next manager. Such trade-offs are to be evaluated on a per-application basis,[11]

Sessions as privilege rings

Finally, we note a particular, novel benefit in using logical sessions rather than the binary "in-or-out" form of group membership. That is, since access to application data, such as application C2, is guarded by session access, successively subset sessions can be used analogously to privilege rings in architecture and operating systems contexts.

Generically, this allows swarms to have a privileged subset of agents given voting rights in swarm-wide decision making (a "small council"[12] of sorts), and even successively larger councils in an application-defined gradation of cryptographically-enforced voting rights, where individual members can be selectively promoted and demoted between privilege levels (see figure 9-2). In the AFIT swarm, for example, the cluster manager session could be granted exclusive input to making swarm-wide decisions.

[10]This is particularly relevant for many distributed schemes, which often rely on contributory mechanisms requiring input from all session members—such schemes would require fault-tolerant alternatives to be used in such high-churn or low-availability use cases

[11]One of the goals of this work is pointing toward an architecture where such application-driven evaluation and comparison of key management schemes is as easy as changing a parameter to the session *create* action

[12]`https://awoiaf.westeros.org/index.php/Small_council`

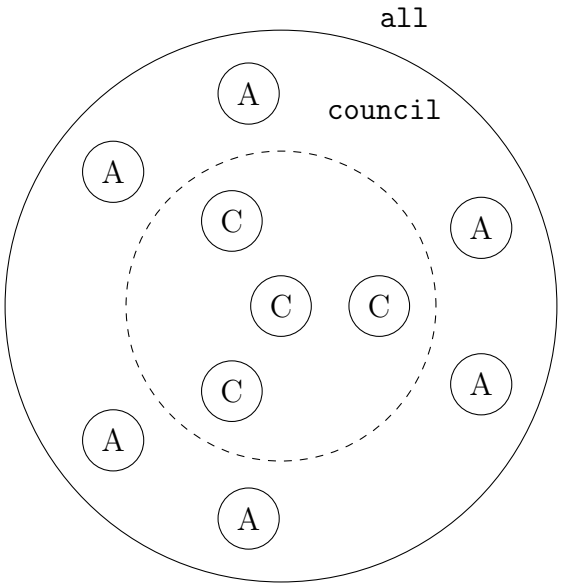

Figure 9-2: Conceptualization of successive subsets of sessions acting as privilege rings. In this example, the `all` session holds all members (As and Cs), while the `council` session holds only a subset of members (Cs), where `council` session members are granted additional decision privileges in the swarm.

9.1.4 Group access control

Group access control determines which agents can join a session. It consists of a policy and a specification for which agents can enact the policy. Our architecture makes use of a localized blacklists, application-specific whitelists, and long-term CRLs.

Policy encoding

For encoding the policy of which agents can join a session, one option is static capability certificates signed offline by a trusted third party (as in [114, sec. 2]). While conceptually simple, this static policy precludes dynamic group membership, and does not effectively match the session abstraction of non-binary group access. Instead, applications should be empowered to make session access decisions on a per-session basis, as informed by the application, for more fine-grained access control [3].

This desire for dynamic access control takes input from the application in making session access decisions. However, the application's decision should still be guarded by traditional access control mechanisms, for which we utilize the following.

- *Long-term CRL.* Loaded during the pre-mission phase, the group-wide CRL is not tied to a particular mission, is static over the duration of a particular mission, and is signed offline by a trusted third party. Optionally, functionality can be subsumed into carefully-initialized black/white lists.

- *Localized blacklist.* Aggressive, append-only list of agent identities disallowed from any form of communication or cryptographic distribution. Updates promulgated to all ECUs in the swarm to ensure compromised group members cannot gain re-entry to the group. Localization allows non-permanent designation of compromise, as well as shorter blacklist update messages.

- *Application-specific whitelist.* Minimal, well-chosen subset of possible agent identities allowed to participate in the mission. Chosen to strongly limit the number of agents who could participate in the mission, thus severely restricting the attack surface for potential swarm intrusion. The whitelist and blacklist (and long-term CRL) work together to provide the first phase of access control. The whitelist is optional, since its use restricts flexibility (and MANET-like ad hoc functionality) to the benefit of reduced attack vectors.

Policy enactment

For enacting the policy, the precise mechanism is dependent on the key management scheme in use. For centralized schemes, session access must touch the manager since membership view updates trigger a rekey by the manager; the manager could delegate certificate validation to trusted peers (§9.1.7), however. For distributed schemes, rekeys do not necessarily flow through the manager, so group access decisions can be made without originating from the manager.[13]

[13]E.g., after an application-layer vote, the swarm could agree that member Eve has been compromised, and that member Alice should initiate the distributed rekey operation

Example policy enactment

As an example, consider a scenario where member Alice would like to join the centralized session `council`. Under the proposed architecture, the following steps would occur.

1. Alice asks to be added to the `council` session by application-specific communication over the `all` session.

2. The ECU of `council`'s manager: consults the long-term CRL, rejecting if Alice is found; consults the local blacklist, rejecting if found; consults the application-specific whitelist, rejecting if not found.

3. The application of `council`'s manager decides whether to accept Alice, optionally consulting other members of the session via application-specific communication over the session.

4. The application of `council` forwards its confirmation of Alice's request to its ECU, which then initiates the session update to cryptographically *join* Alice to the session.

5. After the rekey, Alice's application can send and receive (encrypt and decrypt) messages under `council`.

For joining the `all` session (a proxy for joining the group), an application-specific mechanism must be employed (§9.1.6), but in this example we say the initial request to join the physical group is sent in the clear between radios, where a request to join `all` is then immediately generated (§9.1.6).

9.1.5 Compromise detection

Compromise recovery is achieved in two steps: compromise detection and handling the reported compromise.

Detection. As detailed in previous chapters, detecting the compromise of a group member must be accomplished by the application. Once detected, the compromise is

reported to a relevant ECU, which then decides how to proceed in a scheme-dependent manner. One potential mechanism is a periodic heartbeat message sent to the `all` session, with the additional benefit of forcing lagging agents to update to the current epoch (and thus current blacklist).

Blacklist update. Specifically, the `all` session's blacklist must be updated (and thus also its epoch). For centralized schemes, the compromise report can be forwarded to `all`'s session manager. For distributed schemes, we again delegate consensus on and serialization of blacklist updates to the application. Coupled with periodic heartbeats on the `all` session, this leads to eventual update of the group-wide blacklist.

Forced leave. At each ECU, an update to its blacklist requires removing the newly-blacklisted agent from every session. For both centralized and distributed schemes, we will have the manager handle initiating the session rekey.

Finally, one quirk in application-driven compromise detection: an application can accidentally DoS itself via misguided blacklist merges. Consider a group unintentionally partitioned in a 60-40 split, for a non-trivial amount of time. Now, the majority split does not assume any compromises, while the minority split consider all agents in the majority to be compromised, reports the compromises, and updates its blacklists. Then, the implicit partition heals, and the minority split convinces the majority split that all its members are compromised, resulting in immediate loss of the majority split.

The simplest solution to this quirk, which we adopt, is to strictly prohibit non-compromised agents from publishing information to agents they deem compromised, and utilize a generous grace period before deeming an agent as compromised. Thus, if a swarm is implicitly partitioned for an extended period of time, the partition cannot be healed.[14]

9.1.6 Lower-layer keying material

To protect against the assumed-adversarial environment faced by tactical swarms, TRANSEC is assumed necessary—that is, we assume physical-layer DoS is the norm,

[14]Other, consensus-based solutions are conceivable, but we leave them for future work

and require any key management service to accommodate the need for lower-layer keying material. Thus, unlike exclusively application-layer key management (the assumption in e.g. the IP multicast literature), there is no guarantee that physical-layer communication can take place at all without shared physical-layer secrets (shared TRANSEC keying material).

In remedy, we follow the approach taken by projects like Bluetooth mesh [53] and separate secrets into application-layer keys and a (single) lower-layer key. Once this single lower-layer key (the physical-layer key) is shared by all members of the group, application-layer key management can proceed as described above.

Lower-layer key is the current `all` key

Since we already utilize the `all` session as a proxy for (physical-layer) group membership, we now take the final step and describe the `all` session's shared key as the lower-layer key, and expose it to the radio via the ECU.

Ideally, the radio-defined metadata encryption and associated packet layouts of each layer would be defined as well, for full interoperability (cf. efforts like TinySec [75] and ZigBee [77]). However, as this is outside the scope of key management at the ECU, we simply expose a bitstream deterministically transformed from the physical-layer key and leave subsequent encryption choices to the radio.[15]

Updating `all` group membership

As with any other session, additive (*join*, *merge*) and subtractive (*leave*, *partition*) operations on group membership are supported by the ECU. Subtractive operations function as in other sessions: group membership is updated, and the subsequent rekey revokes access to the current physical-layer key from the removed members. Additive operations are similar, though with a bootstrapping quirk—the physical-layer key has to be sent without metadata encryption or physical-layer DoS protection, since there

[15]For a fully interoperable standard, of course, radio formats would need to be standardized; however, we focus on interoperability at the ECU in this work

is no shared physical-layer key.[16] This complexity supports the secured bootstrapping period as a desirable default affordance.

The mechanism for bootstrapping communication at the physical layer (that is, performing additive operations for the `all` session outside of the secured bootstrapping period) is necessarily left as a per-application decision. Some options include the following.

- Cleartext request over available (non-DoS'ed) physical medium; this is perhaps the expected case for many ISR and non-combat military applications
- Cleartext request over application-specific DoS-resistant LPI[17] mechanism
- Cleartext request over application-specific uncoordinated spread spectrum mechanism [94, 95, 96]

Note that, ideally, there is an atomic changeover of the physical-layer key. Once again, we accomplish this challenge of distributed systems by leaving coordination to the application. That is, while the ECU may update the `all` session key, the application must collectively decide when to actually install the new key at the radio.[18] Finally, for additive group updates to `all`, the application may for efficiency reasons wish to forego updating the physical-layer key. The application can accomplish this by setting scheme-specific options for `all` at the ECU.

Multiple lower-layer keys

Finally, while we use a single key for all sub-application layers (with focus on the physical layer), this could be extended to provide multiple, independent keys for layers between the physical and application layers. This would make sense in scenarios such as FANETS, where swarms are networked and potentially allow "half-trusted"

[16] The metadata could be encrypted via the receiver's public key, but then the message to message to acquire the receiver's ID is still sent without protections

[17] LPI: low probability of intercept

[18] This plays out as: the application receives a notification that the `all` session has been updated; the application confers with other members of `all` to decide when to commit to the key for the specific, new epoch; finally, the application informs the radio to begin using the new key

members—that is, members may be trusted unconditionally with the physical layer key, but not unconditionally with the network layer key. For simplicity, and because military operations have the capacity to forego reliance on marginally-trusted agents, we forego consideration in this architecture. In this case, an agent is trusted to be well-behaved in a binary manner (access to physical-layer key), while more fine-grained trust is reserved for application-layer data.

9.1.7 Delegation

In our architecture, we allow a restricted form of delegation that outsources work at a particular ECU to other, trusted ECUs in the group. This could involve delegating actions like certificate validation (e.g., to speed up the system-level bootstrapping process) or generating a rekey message under the centralized envelope-style scheme (i.e., a distributed version of the envelope-style scheme).

We discuss two forms of delegation to fully trusted peers: platform-level ECU functions and system-level ECU functions.[19]

Platform-level ECU functions. We do not provide a mechanism for delegation of platform-level ECU work, such as validating a single certificate or signing a single message. That is, if the agent is so SWaP-constrained that it cannot perform such platform-level ECU functions, we require the application to choose a different, less strenuous key management scheme. For heterogeneous swarms with a mix of high- and low-capacity agents, applications are restricted to placing the low-capacity agents in sessions with less strenuous ECU requirements. We note the one quirk of this strategy, which is the high- and low-capacity agents must all be members of the `all` session. In this case, the `all` session must have non-strenuous requirements.[20]

System-level ECU functions. We focus on the session-wide work performed by

[19]Restricting delegation to fully trusted peers precludes consideration of areas like secure multi-party computation, but the realistic restriction allows a simpler, potentially less complex model

[20]For example, in ISR scenarios, where DoS concerns are less prominent, we could imagine a use case where entrance to the `all` session is not protected at all, allowing a small swarm of Perdix drones to aggregate data from communication with various ground-based WSNs. In this case, at least one of the WSN motes would need to need to be higher-capacity, to support interfacing with the swarm under this architecture.

managers of centralized key management schemes. We allow two forms of system-level delegation: scheme-specific delegation (distributed envelope-style distribution, delegation of certificate validation, etc.) and soft manager updates. In the former, the individual key management scheme can provide objects and protocols for delegating work to other trusted ECUs; in the latter, a centralized manager can securely unicast relevant keying material to a session's next manager.

In normal, *hard* manager updates to a session, the new (centralized) manager must perform work largely equivalent to starting a new session. For example, in the centralized LKH example in chapter 4, creating a new session requires $\mathcal{O}(n)$ work, while updating the session takes $\mathcal{O}(\log n)$ work. Thus, we provide an interface for the application to communicate to its ECU that the ECU should securely transfer its centralized keying material to the particular session's next manager, resulting in a more efficient *soft* update to the session's manager (§9.2).

By supporting soft updates at the architectural level, we attempt to provide a mechanism for delegation via manager transition, in which the new manager delegates work to the previous manager. For example, consider a Perdix swarm in the pre-mission phase of operation, utilizing a key tree scheme for keying its `all` session, waiting to be launched from its aircraft. A high-capacity controller on-board the aircraft can act as the initial manager of the swarm's `all` session, validate certificates, and generate the required key tree. Then, in transitioning to the launch phase of operation, the controller's application directs its ECU to perform a soft update to `all` and transfer its `all`-related state to the next drone manager of the swarm.[21]

9.1.8 Passive agents

Allowing passive agents to join a session is possible when key management schemes know which agents in a session are passive. Support for semi-passive or fully-passive agents requires two considerations: when the agents should be added to a session, and how to perform a rekey without input from passive members.

[21] Of course, this could need to be implemented on a per-scheme basis, as the "state" objects need interoperable definitions

For understanding when to include passive agents in a session, we defer to the application. That is, for semi-passive agents, they can join the `all` session during a non-passive period, communicate their intentions to the group, and relevant group members can then add the semi-passive agent to sessions at desired points. For fully-passive agents, the application must necessarily know or be informed of the existence of passive nodes from external sources. This can involve, for example, communicating a list of relevant passive nodes to the non-passive agents in a swarm, along with their relevant public keys and identity certificates.

More interestingly, we need to ensure that rekey operations can succeed even when some members of the session are passive. For example, in initializing a centralized LKH, the manager performs a secure unicast to each member of the session.[22] When all session members are non-passive, we can use ephemeral, full Diffie-Hellman key agreement to provide a scheme with perfect forward secrecy (PFS) [17, chp. 12]. However, this requires active participation in the key agreement by both members. For sessions that allow fully-passive members, we must break this affordance of PFS—that is, PFS is lost for all session epochs which include a passive member. That said, as long as the application informs the ECU of which agents are passive (§9.2), the ECU can provide PFS in the general case while retaining functionality in the presence of passive session members.

9.1.9 Summary of the conceptual architecture

In summary, we look to provide a generic architecture providing a simple interface for secure communication, with pluggable key management options.

The cornerstone of this generic architecture is the session abstraction, together with the application-ECU-radio decoupling. Each platform (e.g. drone) has a unique identity, with accompanying certificates and private keys, loaded to its ECU. The application is then free to direct session-based multicasting via *creating* sessions, *updating* sessions by adding/removing agents from the session, and *sending/receiving*

[22] Actually, the more likely form is a single broadcast message, with the relevant KEK encrypted to the public keys of relevant session recipients

messages under the session. The ECU handles, in scheme-dependent ways, these session-based actions requested by the application; the radio is responsible for routing the encrypted material, as well as utilizing the ECU's exposed physical-layer key to provide DoS protections where directed by the application.

We require an special, group-wide session denoted `all`, which allows for easier updating of access control materials and sub-application layer keying material. We also promote, as the expected case, a secured bootstrapping period, allowing physical layer DoS protections by inducting all group members into the `all` session over a secured, DoS-protected channel.

For compromise detection, we rely on the application to inform its ECU when an agent has been compromised; the ECU then ensures an update to `all` is initiated, appending to its blacklist attribute. We support a restricted form of delegation via soft-updates to sessions' manager attribute, and allow passive agents if they are identified as such by the application.

Finally, the key management scheme chosen at the ECU can range from centralized envelope-style distribution to distributed key trees to many others (see [115] for an subset of options). Of the promising schemes we have identified, we promote key graphs as a first prototyping effort.

9.1.10 Extensions to the basic architecture

The basic architecture trades full flexibility for a simplified operational model. We enumerate a number of extensions to the architecture which could add back some flexibility at the cost of complexity.

Extension: asymmetric send/receive capabilities

In the basic architecture, access to a session grants full send and receive capabilities—access to the shared session key grants access to encryption and decryption of application messages, and authentication is provided using the session key, as $c' = \mathsf{Sign}(c, k_{\mathsf{session}})$

We can imagine extensions to the architecture which encode send/receive capabilities into the membership view, then use a mechanism for broadcast authentication to cryptographically enforce the distinction between senders and receivers (see [116, sec. 1]). Some options for enforcing this broadcast-based authentication include the following.

- Sign with sender's public key, as $c' = \mathsf{Sign}(c, \mathsf{sk}_{\mathsf{sender}})$. Receiving ECUs can then determine whether to discard the received message. Unfortunately, this introduces an order of magnitude increase in computation overhead per message sent.

- Follow the mechanism used in the TESLA protocol [116], utilizing loosely synchronized clocks and delayed key disclosure. This relies on only symmetric cryptographic operations.

Extension: multi-domain groups

In looking to provide multi-domain sessions,[23] to incorporate use cases like the AFIT swarm, we can provide domain-relevant information to the ECU, allowing the ECU to set up and manage the multiple domains within the session.

For example, we can store the domain-based information for an AFIT swarm in the membership view with a concrete object encoding something like the abstract object in figure 9-3. With this updated conception of a membership view, the swarm's ECUs can route messages across domain lines to the manager of the relevant domain, where translation between key domains can take place, as in the AFIT swarm.

We note this may not be the optimal strategy in all cases, or even in the AFIT swarm. The AFIT swarm setup can be constructed using only the default architecture, where cross-domain translation and cluster membership (session membership) is directed by the application.

[23] That is, a session with multiple symmetric keys, requiring translation across domains

```
1  domain (main) {
2      domain (clusterA) [manager] {
3          agent (A1) [manager]
4          agent (A2)
5          agent (A2)
6      }
7      domain (clusterB) {
8          agent (B1) [manager]
9          agent (B2)
10         agent (B2)
11     }
12     domain (clusterC) {
13         agent (C1) [manager]
14         agent (C2)
15         agent (C2)
16     }
17 }
```

Figure 9-3: Example conception of a membership view allowing the ECUs to provide sessions with multiple key domains.

Extension: networked swarms

Finally, networked swarms are supported by the basic architecture. However, use cases allowing half-trusted agents (i.e., trusted at perhaps the physical layer but not the network layer) require cryptographically-enforced secure routing policies. For example, consider a FANET where group membership updates cannot be agreed upon because a malicious node DoS's the network layer (but not the physical layer).

In this example, we need a mechanism for bootstrapping security at the network layer, similar to our bootstrapping of security at the physical layer, described above. See [117, 38] for further consideration.

Thankfully, this extension is not required when we make the (perhaps reasonable) assumption that all military agents can be trusted to cooperate effectively, at least until they are compromised.

9.2 Abstract interfaces

We now present conceptual abstract interfaces enabling the architecture described in the previous section. This contribution, along with the nature of the interfaces, is

adopted largely from the SCM work [3]. We present the interfaces with a minimal summary, ending with an example usage for clarity.

9.2.1 Application-ECU interface

The application-ECU interface is the most involved of the interfaces. It is this interface that provides the shift from a library of key management primitives (as in LOCKMA [2]) toward a decoupled architecture with session-like abstractions (as in the SCM [21, 3]). See figure 10-3 for a conceptualization of the abstract interface.

The interface in figure 10-3 provides an agent with the capacity to know the identities of all active, in-range peers, authenticate the peers, and create a session with a desired subset of the peers. The agent is automatically made the session's manager, and can both update the session's membership view and its metadata, as well as send and receive data on the session.

In creating a session via `session_create`, the following information is provided.

- `peers`. The membership view, including all peers to be included in the group, with indication of any passive nodes.
- `hint`. Informal name of the session, used as hints to applications receiving `session_request` calls.
- `protocol`. Which key management scheme to utilize.
- `options`. Set session options, indicating whether this is the `all` session, providing the whitelist (if this is the `all` session), whether to require confirmation of session updates, whether to change the session key on additive session updates, etc.

Looking back to the session metadata (§9.1.3), we note the session ID and epoch are set by the ECU, but the membership view, protocol, options, and whitelist are initiated here in `session_create`. These attributes can be viewed later with `session_query` and updated with `session_update`.

Moving to specific functionalities, we note the following.

- Atomic changeover of the physical-layer secret is possible via the `session_update` notification from the ECU, allowing the application to coordinate when changeover at the radio should occur.

- Passive nodes are indicated in the membership view, passed to `session_create` and viewable/updatable as part of the `SessionInfo` object.

- The need for a response to session creation or update (e.g. rekey messages) from all non-passive nodes is indicated in the same `SessionInfo` object.

```
1  # Application to ECU
2
3  def peers_list() -> Peers
4  def peers_authenticate(peers: Peers) -> Bools
5  def session_create(peers: Peers, hint: str, protocol: KMProtocol,
       options: SessionInfo) -> SessionID
6  def session_destroy(session: SessionID)
7  def session_add(session:SessionID, peers: Peers)
8  def session_remove(session: SessionID, peer: Peers)
9  def session_query(session: SessionID) -> SessionInfo
10 def session_update(session: SessionID, update: SessionInfo, hard: bool)
11 def session_send(session: SessionID, data: bytes)
12
13 # ECU to application
14
15 def identification_request(peer: Peer) -> bool
16 def authentication_request(peer: Peer) -> bool
17 def session_request(session: SessionID, hint: str, peers: Peers) -> bool
18 def session_recv(session: SessionID, data: bytes)
19 def session_update(session: SessionID, update: SessionInfo)
```

Figure 9-4: Conceptual interface between the application and its ECU (Python-like syntax).

9.2.2 Radio-ECU interface

The radio-ECU interface is more succinct than its application-ECU counterpart. The basic functionality is `send` and `recv`, sending ciphertexts to a particular membership view and similarly receiving ciphertexts. See figure 9-5 for a conceptualization of the abstract interface.

```
# Radio to ECU

def recv(data: bytes)
def membership_view(session: SessionID, epoch: Epoch) -> MembershipView
def metadata_key(session: SessionID, epoch: Epoch) -> Key

# ECU to radio

def send(session: SessionID, epoch: Epoch, data: bytes)
```

Figure 9-5: Conceptual interface between the radio and its ECU (Python-like syntax).

We assume the ECU wraps the desired ciphertext in a PDU[24] indicating to receiving ECUs the nature of the message, indicating whether it is an application-layer message, ECU-driven session update, authentication request, etc.[25] This allows the `recv` call to return just the ECU's PDU, leaving the ECU to parse relevant session and epoch attributes. The session ID and epoch are included in `send`, however, to inform the radio which membership view should receive the message.

Two final functions complete the interface. First, `membership_view` is provided to allow the radio to query the ECU for the membership view for a particular epoch and session ID duplet, caching the response for future `send` calls. Finally, `metadata_key` allows the radio to query the ECU for the bitstream associated with the `all` session's key, to be used as the physical-layer key for metadata encryption and DoS protection. To protect keys of other sessions, the ECU will reject calls to `metadata_key` for session ID and epoch duplets for which the session is not marked as the `all` session.

9.2.3 Application-radio interface

Unfortunately, no abstraction is safe from the real world. In this case, we cannot fully decouple the radio from the application, motivated principally by the need for application-driven configuration of its radio. We leave specification of the application-radio interface to future work, noting that important aspects of configuration include directing the radio to install a new physical-layer key via a specific session ID/epoch

[24]PDU: protocol data unit

[25]For full interoperability at the ECU, the form of this PDU would be standardized within our architecture; we leave full standardization to future work

duplet.

9.2.4 Example usage

See figures 9-6, 9-7 for the full interaction. In this example, we describe three agents, Alice, Bob, and Carol, where Alice creates a session, later adds Carol to the session, makes Bob the manager, then sends data to the session. For concreteness, we could consider Alice, Bob, and Carol to be drones in a Perdix swarm.

```
1  ### Alice (initiator)
2
3  # Find all peers, then authenticate them
4  peers = peers_list()
5  if False in peers_authenticate(peers):
6      # handle failure to authenticate a peer
7
8  # Create the all session
9  options = {is_all: True, require_confirmation: True, change_on_add:
       False}
10 all_session = session_create(peers, 'all', 'centralized_lkh', options)
11
12 # Add passive Carol specifically as a new peer
13 carol = {peer_id: carol_id, is_passive: True}
14 session_add(all_session, [carol])
15
16 # Install Bob as the session manager
17 info = session_query(all_session)
18 info['manager'] = bob_id
19 session_update(all_session, info, False)
20
21 # Send to the session
22 session_send(all_session, b'this should be in an app-specific PDU')
```

Figure 9-6: Minimal example of session-driven interactions under the proposed interfaces; here, Alice initiates a session, then transfers her role as manager to Bob.

Find and authenticate peers. Line 3. Alice asks her ECU for a list of peer identities; her ECU forwards this request to her radio, which queries peers for their identities in radio-defined ways. Bob's radio receives this identification request, ensures his application wishes to acknowledge the request (`identification_request`), and responds appropriately. Finally, the peers list is returned to Alice's application, and Alice directs her ECU to authenticate the full peers list. Her ECU can choose to shoulder this work itself or look to progressively distribute the work to trusted (authenticated)

```
1  ### Bob
2
3  # Define the relevent RPC-style interface implementations
4
5  def identification_request(peer: Peer) -> bool:
6      return True
7
8  def authentication_request(peer: Peer) -> bool:
9      return True
10
11 def session_request(session: SessionID, hint: str, peers: Peers) -> bool
       :
12     return True
13
14 def session_recv(session: SessionID, data: bytes):
15     # forward for application-specific processing
16
17 def session_update(session: SessionID, update: SessionInfo):
18     # if update is to all session, decide whether to reconfigure radio
```

Figure 9-7: Minimal example of session-driven interactions under the proposed interfaces; here, Bob defines his reactions to relevant interface events.

peers. At some point, Bob's application receives the `authentication_request` call; he replies `True`, directing his ECU to send its identity certificate to Alice.

Create a session. Line 8. Alice creates the `all` session, which is required to be the first session created. Upon calling `session_create`, her ECU initializes a session utilizing centralized LKH as its key management scheme, in the following steps.

1. Initialize the LKH key tree for the requested agents.

2. For each member of the session: establish a symmetric key with the agent via ephemeral DH[26] key agreement,[27] then broadcast a single message K containing the relevant KEKs encrypted to their respective session members[28] (see figure 9-8).

3. Require each session member to confirm reception of its respective unicast subtree, e.g. via signing and returning a nonce from K with their personal signing key.

[26]DH: Diffie-Hellman
[27]For passive nodes, full, ephemeral DH agreement is replaced with e.g. one-sided, static DH
[28]When appropriate, K can be split into smaller, more manageable sub-messages

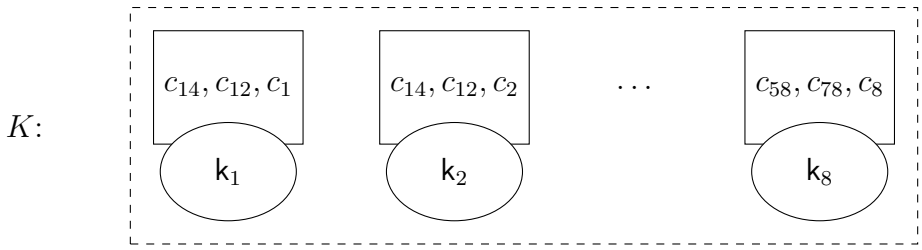

Figure 9-8: Notional LKH message K generated by Alice's ECU to initialize the `all` session for an 8-drone Perdix swarm (cf. figure 4-7). Each set of KEKs c_* is encrypted to session member i under the appropriate ephemeral key k_i. The actual message broadcasted is perhaps $K' = \mathsf{Sign}(K, \mathsf{sk}_{\mathsf{manager}})$. Note K here requires $n \log n$ space.

Before participating in the session initialization, Bob's ECU asks his application for permission via `session_request`. Finally, Alice's ECU has initialized a LKH-backed `all` session and received confirmation from the ECU of each member of the session. Alice's ECU returns the ECU-selected session ID to the application, stored in the `all_session` variable on line 10.

Add passive agent to the session. Line 12. Assume Carol is a passive agent, and Alice learned of her existence through external means (i.e., not via `peers_list`). Alice adds Carol to the `all` session via `session_add`. Alice's ECU follows similar steps as above (though the LKH is not truly rekeyed due to the `change_on_add` option), establishing a secured unicast channel with Carol (non-full DH) and communicating the relevant KEKs. The session's membership view and epoch are updated, resulting in Bob receiving a `session_update` message.

Install a new manager. Line 16. Alice updates the session's attributes, specifically its manager attribute. Calling `session_update` prompts Alice's ECU to install and promulgate this update, and look to perform a soft update by transferring its session-relevant state to Bob's ECU, such as the LKH KEK tree. Bob's ECU recognizes it is installed as the new manager, accepts the bootstrapped session state from Alice, and informs his application of the session update.

Send and receive on a session. Lines 21 (figure 9-6), 14 (figure 9-7). Alice directs her ECU to send a message to the `all` session. Her ECU encrypts the message with the session key for the current epoch, then forwards the ciphertext, session ID, and

epoch to her radio. Her radio makes a quick `membership_view` call to the ECU to pull the membership view associated with the received session ID/epoch duplet, caching the response. Finally, Alice's radio transmits the ciphertext. Once sent, Bob, Carol, and all other session members receive the ciphertext at their radios.[29] The ciphertext is forwarded to their ECUs, which parse the ECU-layer PDU for session ID and epoch information, decrypt the ciphertexts, and forward the plaintexts and session ID to their applications via `session_recv`.

[29]More specifically, a superset of any session's members receive the ciphertext at their radios, but effective key management precludes the ECUs of non-members from decrypting the ciphertext

Chapter 10

Prototyping the Architecture

We now describe our initial, in-progress prototype of the generic key management architecture proposed in chapter 9.

The objectives in building the prototype are twofold: (a) demonstrate the architecture is an effective extension to the architecture utilized by the SCM, generating QoS measurements and confirming the flexibility allotted to arbitrary applications; (b) generating performance benchmarks allowing effective evaluation and comparison between the pluggable key management schemes identified in chapter 8.

10.1 Software architecture

We now describe the software architecture of the prototype implementation. The aim for the implementation is to produce a prototyped implementation and simulation of cryptographically secured swarm communication.

- *Prototyped implementation.* The relevant application-facing interface, OTA[1] messages, and key management techniques are all specified and implemented, to a sufficient degree so as to support proper prototyping. Arbitrary code can be run as the application, and can be configured to run independent of the simulation framework.

[1] OTA: over the air

- *Simulation.* Support rapid prototyping of key management schemes, by allowing arbitrary code to be run as the ECU; configurable group sizes, node constraints, and network topologies and QoS should allow generation of useful data for further consideration.

- *Cryptographically secured.* All OTA messages are well-specified and properly encrypted and authenticated.

- *Swarm communication.* The application-facing interface should provide the full, secured, session-based communication abstraction across the swarm, not just pluggable key management.

10.1.1 Supporting libraries

The first decision is choosing a simulation framework. After reviewing options from ns-3 (used in e.g. [111])[2] to SimPy,[3] we settled on Mininet,[4] a network emulation framework supporting real code and arbitrary network topologies. While Mininet is not a high-fidelity simulator like ns-3, we prize its enabling of rapid prototyping, and assume Mininet's imperfect network QoS emulation will be asymptotically overshadowed by our simulated timings of per-platform cryptographic operations.

Next, we opt to develop in Python, and desire three supporting functionalities: an ASN.1 parser, cryptography primitives with certificate management, and an RPC library.

- *ASN.1 parser.* Pyasn1[5] is a generic implementation of ASN.1 types, supports BER, CER, and DER codecs, and can translate between ASN.1 structures and native Python types. Auxiliary functionality is provided by pyasn1-modules[6] for describing established ASN.1 structures in the pyasn1 data model and asn1ate[7] for generating pyasn1 code from formal ASN.1 definitions.

[2] `https://www.nsnam.org`
[3] `https://simpy.readthedocs.io`
[4] `http://mininet.org`
[5] `https://github.com/etingof/pyasn1`
[6] `https://github.com/etingof/pyasn1-modules`
[7] `https://github.com/kimgr/asn1ate`

- *Cryptographic functionality.* The Python Cryptographic Authority project[8] provides their cryptography[9] library implementing low-level cryptographic primitives and high-level functionality like X.509 certificate handling.

- *RPC framework.* Zerorpc[10] is a lightweight, reliable RPC framework built on ZeroMQ and MessagePack, with supported Python bindings[11].

10.1.2 Prototype functionality

At the system level, our implementation uses Mininet helper functions to define a network topology, then launch individual processes acting as the platforms (e.g. drones) in a swarm. The topology, along with throughput and latency at each edge in the topology, is configurable. In the future, we wish to explore incorporating Mininet-WiFi[12] as an extension to vanilla Mininet to better emulate the QoS constraints associated with wireless networks.

At the platform level, each platform is split into three processes, functioning as the application, ECU, and radio; see figure 10-1. Arbitrary code can be run in each process, though with the intent that prototyping and evaluation efforts iterate upon a single process at a time. The application-ECU interface and ECU-radio interface are implemented as RPC definitions at each process, listening on statically-defined ports within the Mininet-provided network namespace. In our basic, broadcast-based setup, the radio acts as a shim which just broadcasts messages from its ECU to the LAN, forwarding received messages to its ECU. The application is similarly straightforward, sending leader-driven heartbeat requests to all members of the swarm. The ECU, finally, is where we devote the majority of our effort.

[8] `https://github.com/pyca`
[9] `https://cryptography.io`
[10] `https://www.zerorpc.io`
[11] `https://github.com/0rpc/zerorpc-python`
[12] `https://github.com/intrig-unicamp/mininet-wifi`

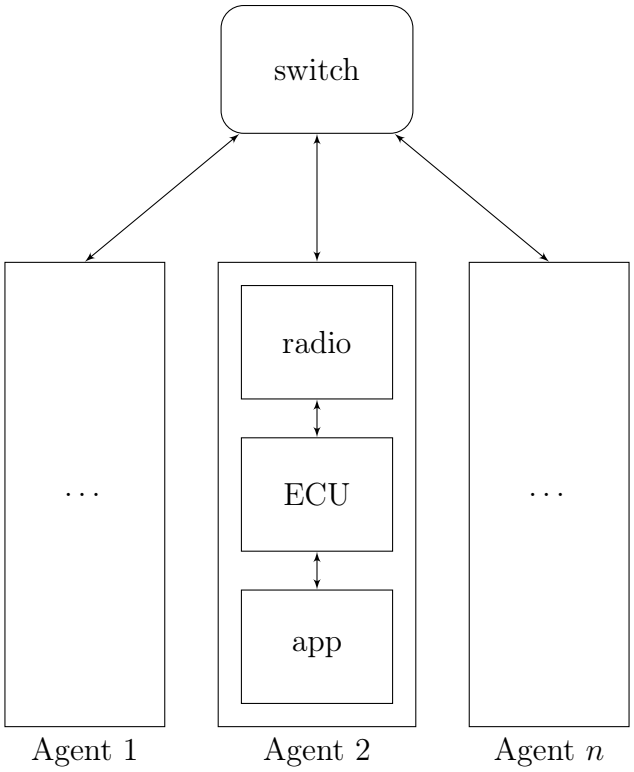

Figure 10-1: Default topology in our prototyped implementation. Each agent is composed of three processes (app, ECU, radio) communicating over RPC. Radios communicate by sending a LAN broadcast echoed to all other agents by the switch. Mininet provides helper functions to generate and namespace the network topology, as well as simulating link QoS variables like latency, throughput, and packet loss.

10.2 Abstract objects

At the ECU, we have two principal concerns: standardizing the generic architecture (including the format for secured application-layer messages), and implementing specific key management schemes. Specifying the abstract objects associated with the proposed generic architecture makes concrete a number of the interoperability-based decisions left underspecified in the architecture as presented. We present a set of simplified, prototypical abstract objects related to the application-ECU-radio interface in figure 10-3; see figure 10-2 for definitions of ECU messages for specific key management schemes (envelope-style, tree-based envelope forwarding).

```
RecipientInfo ::= SEQUENCE {
    to AgentId,
    encryptedKEK OCTET STRING
}

ForwardingInfo ::= SEQUENCE {
    to AgentId,
    forwardTo Agents
}

-- Option for keyingMaterial in session-related objects
Envelope ::= SEQUENCE {
    session SessionInfo,
    encryptedGroupKey OCTET STRING,
    recipientInfo SET OF RecipientInfo
}

-- Option for keyingMaterial in session-related objects
EnvelopeForward ::= SEQUENCE {
    session SessionInfo,
    encryptedGroupKey OCTET STRING,
    recipientInfo SET OF RecipientInfo,
    forwardingInfo SET OF ForwardingInfo
}
```

Figure 10-2: Initial ASN.1 object definitions supporting envelope-style and tree-based forwarding schemes, simplified from [47, sec. 6]. In both cases, the manager generates an envelope-like object; in the forwarding-based scheme, however, the manager generates the envelope for a small subset of the swarm, prescribing how they should then recursively forward new envelopes to the rest of the group.

10.3 Performance projections

In anticipation of a concrete performance evaluation, we project the performance improvements for one of the identified key management schemes. Specifically, the improvements in using centralized LKH over centralized direct distribution as the rekey mechanism.

For comparison, we target the Cortex-M0+, marketed as the "most energy-efficient ARM processor available."[13] Public speed benchmarks[14] identify the M0+ as performing on the order of 1 RSA operation per second,[15] which we use as a rough proxy for

[13] `https://developer.arm.com/ip-products/processors/cortex-m/cortex-m0-plus`

[14] `https://bearssl.org/speed.html`

[15] Benchmark uses 2048-bit key, public exponent 65537, and evaluates public-key and private-key operations

```
1  Tkm-top-level
2      { iso(1) member-body(2) us(840) mit(113554) lincolnlab(2)
3      tkm(8999) modules(2) top(0) }
4  DEFINITIONS
5      AUTOMATIC TAGS ::=
6  BEGIN
7
8  ll-OID OBJECT IDENTIFIER ::=
9      { iso(1) member-body(2) us(840) mit(113554) lincolnlab(2) }
10 tkm-OID OBJECT IDENTIFIER ::= { ll-OID tkm(8999) }  -- unregistered
11
12 AgentId ::= OCTET STRING
13 Agents ::= SEQUENCE OF AgentId
14 SessionId ::= OCTET STRING
15 Key ::= OCTET STRING
16
17 SessionKeyId ::= SEQUENCE {
18     id SessionId,
19     epoch INTEGER
20 }
21
22 SessionInit ::= SEQUENCE {
23     manager AgentId,
24     hint UTF8String,
25     protocol INTEGER,
26     parameters ANY DEFINED BY protocol OPTIONAL,
27
28     id SessionId,
29     membershipView Agents,
30     keyingMaterial ANY DEFINED BY protocol OPTIONAL
31 }
32
33 SessionUpdate ::= SEQUENCE {
34     id SessionId,
35     membershipView Agents,
36     keyingMaterial ANY OPTIONAL -- DEFINED BY protocol
37 }
38
39 SessionInfo ::= CHOICE {
40     init SessionInit,
41     update SessionUpdate
42 }
43
44 SessionInfoAck ::= SEQUENCE {
45     me AgentId,
46     session SessionId
47 }
48
49 END
```

Figure 10-3: Some initial, simplified ASN.1 objects defined for our prototype implementation.

the considered asymmetric operations.

As a single example, we examine an energy-conscious, 100-drone Perdix swarm equipped with ECU cryptoprocessors resembling the M0+. Under this scenario, an envelope-style rekey, requiring around 100 asymmetric operations at the manager, could induce a group update latency of 100 seconds; a centralized LKH rekey, requiring around 7 asymmetric operations at the manager, could induce a group latency of less than 10 seconds—an order of magnitude improvement.

Looking back to the proposed taxonomy from chapter 7, we notice that when the platform is constrained, group update latencies naturally rise as well, prompting the application to explore schemes supporting lower group update latencies such as centralized LKH.

Part IV

Conclusions

Chapter 11

Conclusion and Future Work

We began by framing the problem of providing scalable secure communication for tactical swarms, and identifying that no single key management scheme would be sufficient as a lone solution. This transitioned to a prescription of solutions which ultimately pointed to the idea that scalability, in the context of tactical swarms, is achievable. To combat the span of scalability challenges, we describe the decoupling of application and cryptographic concerns to allow a pluggable, application-driven choice of key management schemes appropriate to the particular application. Scalable key management for tactical swarms can be accomplished, as we begin demonstrating via our prototype implementation; futher, presenting application code with a session-driven interface simplifies secure communication in large systems.

Our contributions, presented in the final chapters, include a novel taxonomy of swarm needs, allowing informed mappings between specific swarms and applicable key managment schemes; a recommendation and characterization of key management schemes which collectively service the span of tactical swarm use cases; a notional presentation of a generic architecture for secure communication in swarms of unmanned systems, adopting a session-like interface to decouple application and cryptographic concerns, while providing implicit access to scalable key management; and an initial, in-progress implementation of the prescribe architecture, with the objective of demonstrating the efficacy of our architecture, as well as benchmarking the comparative effectiveness of the recommended key management schemes.

Future work for the contributions presented in this thesis include completion of the in-process implementation to enable concrete benchmarks of the recommended key management schemes; iterating on the proprosed architecture in simulation and via integration into existing SCM-based libraries; and eventual evaluation as part of a live-fly evaluation, similar to the online evaluation in [7].

Future work for general secure communication in tactical swarms should look to provide solutions to the problem of properly witnessing to fully autonomous swarms; investigate the possibility of recovering from premature compromise reporting, with the objective of providing support for safely healing non-trivial implicit partitions;[1] and, more generally, the set of live-fly swarm prototypes with secured communications and dynamic group membership should grow over the coming years as the area moves from prototypes toward production-ready tactical swarms.

[1]We note that these problems (witnessing to autonomous agents and healing implicit partitions) may reduce to the same problem, described perhaps as cryptographically-informed proof of liveness or proof of non-compromise

Appendix A

Common Abbreviations

- AFIT. Air Force Institute of Technology.
- BER. Basic encoding rules, an ASN.1 encoding format.
- C2. Command and control.
- CER. Canonical encoding rules, an ASN.1 encoding format.
- CIA. Confidentiality, integrity, and availability.
- COMSEC. Communications security.
- CONOPS. Concept of operations.
- CRL. Certificate revocation list.
- DER. Distinguished encoding rules, an ASN.1 encoding format.
- DH. Diffie-Hellman.
- DPG. Dynamic peer group, a form of multicast peer association described in [48].
- DoD. Department of Defense.
- ECU. End cryptographic unit.

- ISR. Intelligence, surveillance, reconnaissance.
- KEK. Key encryption key.
- LAN. Local area network.
- LKH. Logical key hierarchy, a form of key graph introduced in [50, 51] and surveyed in [45].
- LOCKMA. Lincoln Open Cryptographic Key Management Architecture, a key management architecture targetting unmanned systems summarized in [2].
- LPI. Low probability of intercept.
- NPS. Naval Postgraduate School.
- OCU. Operator control unit.
- OTA. Over the air.
- PDU. Protocol data unit.
- PFS. Perfect forward secrecy; a protocol is said to have PFS if compromise of long-term keys does not compromise past session keys [17, def. 12.16].
- QoS. Quality of service.
- RPC. Remote procedure call.
- SCM. The Security/Cyber Module, an ECU described in [3].
- SWaP. Size, weight, and power.
- TRANSEC. Transmission security.
- UAS. Unmanned aerial system, referencing a system of UAVs and auxiliary entities.
- UAV. Unmanned aerial vehicle.

- UxS. Unmanned system of unspecified type (e.g. aerial, undersea, etc.).
- WSN. Wireless sensor network.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] Fiona Higgins, Allan Tomlinson, and Keith M. Martin. "Survey on security challenges for swarm robotics". In: *Proceedings of the 5th International Conference on Autonomic and Autonomous Systems, ICAS 2009*. IEEE, 2009, pp. 307–312.

[2] Roger Khazan and Dan Utin. *Lincoln Open Cryptographic Key Management Architecture*. Tech. rep. 2012.

[3] Hoa G Nguyen et al. "A compact end cryptographic unit for tactical unmanned systems". In: *SPIE Unmanned Systems Technology XXI*. 2019.

[4] Manuele Brambilla et al. "Swarm robotics: a review from the swarm engineering perspective". In: *Swarm Intelligence* 7.1 (Mar. 2013), pp. 1–41.

[5] Gerardo Beni. "From swarm intelligence to swarm robotics". In: *Proceedings of the 2004 international conference on Swarm Robotics*. Springer-Verlag, 2005, pp. 1–9.

[6] Mathieu Le Goc et al. "Zooids: building blocks for swarm user interfaces". In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16*. 2016, pp. 97–109.

[7] Richard B. Thompson and Preetha Thulasiraman. "Confidential and authenticated communications in a large fixed-wing UAV swarm". In: *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. IEEE, Oct. 2016, pp. 375–382.

[8] Chuck Hagel. *Defense Innovation Days opening keynote*. 2014.

[9] Robert Martinage. "Toward a new offset strategy". In: *CSBA* (2014).

[10] Department of Defense. *Unmanned systems integrated roadmap*. Tech. rep. 2017.

[11] Peter Hess. *Perdix drones are the "riskiest, most exciting things" coming out of the Pentagon*. 2017.

[12] Timothy H. Chung et al. "Live-fly, large-scale field experimentation for large numbers of fixed-wing UAVs". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2016-June. IEEE, May 2016, pp. 1255–1262.

[13] Strategic Capabilities Office. *Perdix fact sheet*.

[14] John Enright and Peter R Wurman. "Optimization and coordinated autonomy in mobile fulfillment systems". In: *Automated Action Planning for Autonomous Mobile Robots: Papers from the 2011 AAAI Workshop* (2011).

[15] Raffaello D'Andrea. "Can drones deliver?" In: *IEEE Transactions on Automation Science and Engineering* 11.3 (July 2014), pp. 647–648.

[16] Michael Rubenstein et al. "Kilobot: a low cost robot with scalable operations designed for collective behaviors". In: *Robotics and Autonomous Systems* 62.7 (July 2014), pp. 966–975.

[17] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. Vol. 19964964. Discrete Mathematics and Its Applications. CRC Press, Oct. 1996.

[18] Yi Sheng Shiu et al. "Physical layer security in wireless networks: a tutorial". In: *IEEE Wireless Communications* 18.2 (Apr. 2011), pp. 66–74.

[19] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Tech. rep. Aug. 2018.

[20] Keith Moore and Chris Newman. *Cleartext considered obsolete: use of Transport Layer Security (TLS) for email submission and access*. Tech. rep. Jan. 2018, pp. 1–26.

[21] John Yen et al. "Cybersecurity for unmanned systems". In: May 2017 (2017), 101950R.

[22] Robert C. Nickerson, Upkar Varshney, and Jan Muntermann. "A method for taxonomy development and its application in information systems". In: *European Journal of Information Systems* (2013).

[23] Yacine Challal and Hamida Seba. "Group key management protocols: a novel taxonomy". In: *International Journal of Information Technology* 2.1 (2005), pp. 105–118.

[24] Committee on National Security Systems Instruction (CNSSI) No. 4009. *National Information Assurance (IA) Glossary*. 2016.

[25] Guowei Cai, Jorge Dias, and Lakmal Seneviratne. "A survey of small-scale unmanned aerial vehicles: recent advances and future development trends". In: *Unmanned Systems* 02.02 (2014), pp. 175–199.

[26] Andreas Kolling et al. "Human interaction with robot swarms: a survey". In: *IEEE Transactions on Human-Machine Systems* 46.1 (Feb. 2016), pp. 9–26.

[27] Riham Altawy and Amr M Youssef. "Security, privacy, and safety aspects of civilian drones". In: *ACM Transactions on Cyber-Physical Systems* 1.2 (2016), pp. 1–25.

[28] Axel Bürkle, Florian Segor, and Matthias Kollmann. "Towards autonomous micro UAV swarms". In: *Journal of Intelligent & Robotic Systems* 61.1-4 (Jan. 2011), pp. 339–353.

[29] Ivan Maza et al. *Classification of multi-UAV architectures*. Tech. rep. 2015.

[30] Jon Harper. *Pentagon's Strategic Capabilities Chief wants to see networks of "expendable" platforms*. 2017.

[31] Reg Austin. *Unmanned Aircraft Systems*. 2010.

[32] Alan F. T. Winfield, Christopher J. Harper, and Julien Nembrini. "Towards dependable swarms and a new discipline of swarm engineering". In: *Swarm Robotics Workshop*. Springer, Berlin, Heidelberg, 2005, pp. 126–142.

[33] Iñaki Navarro and Fernando Matía. "An introduction to swarm robotics". In: *ISRN Robotics* 2013 (Sept. 2012), pp. 1–10.

[34] Adrian N. Phillips et al. "A secure group communication architecture for autonomous unmanned aerial vehicles". In: *Security and Communication Networks* 2.1 (Jan. 2008), pp. 55–69.

[35] Brad Lendon. *U.S. Navy could "swarm" foes with robot boats*. 2014.

[36] Hui Liu et al. "Survey of wireless indoor positioning techniques and systems". In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 37.6 (Nov. 2007), pp. 1067–1080.

[37] Hung Chi Chu and Rong Hong Jan. "A GPS-less, outdoor, self-positioning method for wireless sensor networks". In: *Ad Hoc Networks* 5.5 (July 2007), pp. 547–557.

[38] Ozgur Koray Sahingoz. "Networking models in flying ad-hoc networks (FANETs): concepts and challenges". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 74.1-2 (2014), pp. 513–527.

[39] Mohammad Mozaffari et al. "A tutorial on UAVs for wireless networks: applications, challenges, and open problems". In: (2018).

[40] H. Zakeri, Fereidoon Moghadas Nejad, and Ahmad Fahimifar. "Image based techniques for crack detection, classification and quantification in asphalt pavement: a review". In: *Archives of Computational Methods in Engineering* 24.4 (Nov. 2017), pp. 935–977.

[41] R.A. Brooks et al. "Lunar base construction robots". In: *IEEE International Workshop on Intelligent Robots and Systems*. IEEE, 1990, pp. 389–392.

[42] Yurong Hu and Victor O. K. Li. "Satellite-based Internet: a tutorial". In: *IEEE Communications Magazine* 39.3 (Mar. 2001), pp. 154–162.

[43] Alexander G. Madey and Gregory R Madey. "Design and evaluation of UAV swarm command and control strategies". In: *Proceedings of the Agent-Directed Simulation Symposium* (2013), p. 7.

[44] Elaine Barker. *Recommendation for key management part 1: general*. Tech. rep. 2016.

[45] Sandro Rafaeli and David Hutchison. "A survey of key management for secure group communication". In: *ACM Computing Surveys* 35.3 (Sept. 2003), pp. 309–329.

[46] Ondrej Hyncica et al. “Performance evaluation of symmetric cryptography in embedded systems”. In: *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS'2011*. Vol. 1. IEEE, Sept. 2011, pp. 277–282.

[47] R. Housley. *Cryptographic Message Syntax (CMS)*. Tech. rep. Sept. 2009.

[48] Yongdae Kim, Adrian Perrig, and Gene Tsudik. “Tree-based group key agreement”. In: *ACM Transactions on Information and System Security* 7.1 (Feb. 2004), pp. 60–96.

[49] H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Architecture*. Tech. rep. July 1997.

[50] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. “Secure group communications using key graphs”. In: *IEEE/ACM Transactions on Networking* 8.1 (2000), pp. 16–30.

[51] D. Wallner, E. Harder, and R. Agee. *Key management for multicast: issues and architectures*. Tech. rep. June 1999.

[52] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2*. Tech. rep. Jan. 2012.

[53] Martin Wolley. *Bluetooth mesh networking*. 2017.

[54] Bluetooth Mesh Working Group. *Mesh profile: Bluetooth specification*. Tech. rep. 2017.

[55] Ross Anderson and Markus Kuhn. “Tamper resistance — a cautionary note”. In: *The Second USENIX Workshop on Electronic Commerce Proceedings*. 1996, pp. 1–11.

[56] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. “Tamper resistance mechanisms for secure embedded systems”. In: *17th International Conference on VLSI Design. Proceedings.* 2004, pp. 605–611.

[57] Ran Canetti et al. “Multicast security: a taxonomy and some efficient constructions”. In: *Proceedings - IEEE INFOCOM*. Vol. 2. IEEE, 1999, pp. 708–716.

[58] Stephen E. Deering and David R. Cheriton. “Multicast routing in datagram internetworks and extended LANs”. In: *ACM Transactions on Computer Systems* 8.2 (May 2002), pp. 85–110.

[59] B. Quinn and K. Almeroth. *IP multicast applications: challenges and solutions*. Tech. rep. Sept. 2001.

[60] Sylvia Ratnasamy et al. “Revisiting IP multicast”. In: *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 06. 2006, pp. 15–26.

[61] Suman Banerjee and Bobby Bhattacharjee. “Scalable secure group communication over IP multicast”. In: *IEEE Journal on Selected Areas in Communications* 20.8 (Oct. 2002), pp. 1511–1527.

[62] Stefano Basagni et al. *Mobile ad hoc networking*. IEEE Press, 2004, p. 461.

[63] Imrich Chlamtac, Marco Conti, and Jennifer J.N. Liu. "Mobile ad hoc networking: imperatives and challenges". In: *Ad Hoc Networks* 1.1 (2003), pp. 13–64.

[64] Bing Wu, Jie Wu, and Mihaela Cardei. "A survey of key management in mobile ad hoc networks". In: *Handbook of Research on Wireless Security*. 2008, pp. 479–499.

[65] Hao Yang et al. *Security in mobile ad hoc networks: challenges and solutions*. 2004.

[66] Marco Conti and Silvia Giordano. "Mobile ad hoc networking: Milestones, challenges, and new research directions". In: *IEEE Communications Magazine* 52.1 (Jan. 2014), pp. 85–96.

[67] Kerry A McKay et al. *Report on lightweight cryptography*. Tech. rep. 2017.

[68] Nils Gura et al. "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs". In: 2004, pp. 119–132.

[69] Andrew Targhetta. *The design space of ultra-low energy asymmetric cryptography*. Tech. rep. 2015.

[70] Carl Hartung, James Balasalle, and Richard Han. *Node compromise in sensor networks: the need for secure systems*. Tech. rep. January. 2005.

[71] Anthony D. Wood and John A. Stankovic. "Denial of service in sensor networks". In: *Computer* 35.10 (Oct. 2002), pp. 54–62.

[72] John Paul Walters et al. "Wireless sensor network security: a survey". In: *Security in distributed, grid, and pervasive computing* (2006).

[73] Seyit A Camtepe and Bulent Yener. *Key distribution mechanisms for wireless sensor networks: a survey*. Tech. rep. i. 2005.

[74] Adrian Perrig et al. "SPINS: security protocols for sensor networks". In: *Wireless Networks* 8.5 (2002), pp. 521–534.

[75] Chris Karlof, Naveen Sastry, and David Wagner. "TinySec: a link layer security architecture for wireless sensor networks". In: *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*. New York, New York, USA: ACM Press, 2004, p. 162.

[76] Ronald Watro et al. "TinyPK: securing sensor networks with public key technology". In: *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks - SASN '04*. New York, New York, USA: ACM Press, 2004, p. 59.

[77] Paolo Baronti et al. "Wireless sensor networks: a survey on the state of the art and the 802.15.4 and ZigBee standards". In: *Computer Communications* 30.7 (May 2007), pp. 1655–1695.

[78] Roberto Sadao Yokoyama, Bruno Yuji Lino Kimura, and Edson dos Santos Moreira. "An architecture for secure positioning in a UAV swarm using RSSI-based distance estimation". In: *ACM SIGAPP Applied Computing Review* 14.2 (June 2014), pp. 36–44.

[79] Victor P. Hubenko et al. *Improving satellite multicast security scalability by reducing rekeying requirements*. 2007.

[80] Cory J Antosh. "The evaluation of rekeying protocols within the Hubenko architecture as applied to wireless sensor networks". PhD thesis. 2009.

[81] Craig W Reynolds. *Flocks, herds, and schools: a distributed behavioral model*. 1987.

[82] Srivaths Ravi et al. "Security in embedded systems: design challenges". In: *ACM Transactions on Embedded Computing Systems (TECS)* 3.3 (2004), pp. 461–491.

[83] Michael Vai et al. "Secure architecture for embedded systems". In: *2015 IEEE High Performance Extreme Computing Conference, HPEC 2015*. IEEE, Sept. 2015, pp. 1–5.

[84] Yongbin Zhou and Dengguo Feng. *Side-channel attacks: ten years after its publication and the impacts on cryptographic module security testing*. 2005.

[85] Junfeng Fan et al. "State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures". In: *Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, June 2010, pp. 76–87.

[86] Tushar Deepak Chandra and Sam Toueg. "Unreliable failure detectors for reliable distributed systems". In: *Journal of the ACM* 43.2 (Mar. 2002), pp. 225–267.

[87] Xiaozhou Steve Li et al. "Batch rekeying for secure group communications". In: *10th International World Wide Web Conference*. 2001, pp. 525–534.

[88] Yi Ruei Chen and Wen Guey Tzeng. "Group key management with efficient rekey mechanism: a semi-stateful approach for out-of-synchronized members". In: *Computer Communications* 98 (Jan. 2017), pp. 31–42.

[89] Reza Olfati-Saber, J. Alex Fax, and Richard M. Murray. "Consensus and cooperation in networked multi-agent systems". In: *Proceedings of the IEEE* 95.1 (Jan. 2007), pp. 215–233.

[90] Naohiro Hayashibara, Adel Cherif, and Takuya Katayama. "Failure detectors for large-scale distributed systems". In: *Proceedings of the IEEE Symposium on Reliable Distributed Systems* (2002), pp. 404–409.

[91] Hubert Zimmermann. "OSI reference model—the ISO model of architecture for open systems interconnection". In: *IEEE Transactions on Communications* 28.4 (Apr. 1980), pp. 425–432.

[92] Konstantinos Pelechrinis, Marios Iliofotou, and Srikanth V. Krishnamurthy. *Denial of service attacks in wireless networks: The case of jammers*. 2011.

[93] Raymond L. Pickholtz, Donald L. Schilling, and Laurence B. Milstein. "Theory of spread-spectrum communications—a tutorial". In: *IEEE Transactions on Communications* 30.5 (May 1982), pp. 855–884.

[94] Christina Pöpper, Mario Strasser, and Srdjan Čapkun. "Anti-jamming broadcast communication using uncoordinated spread spectrum techniques". In: *IEEE Journal on Selected Areas in Communications* 28.5 (June 2010), pp. 703–715.

[95] Tao Jin, Guevara Noubir, and Bishal Thapa. "Zero pre-shared secret key establishment in the presence of jammers". In: *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing - MobiHoc '09*. New York, New York, USA: ACM Press, 2009, p. 219.

[96] Yao Liu et al. "Randomized differential DSSS: jamming-resistant wireless broadcast communication". In: *Proceedings - IEEE INFOCOM*. IEEE, Mar. 2010, pp. 1–9.

[97] David R. Raymond and Scott F. Midkiff. "Denial-of-service in wireless sensor networks: attacks and defenses". In: *IEEE Pervasive Computing* 7.1 (Jan. 2008), pp. 74–81.

[98] Vikram Gupta, S. Krishnamurthy, and M. Faloutsos. "Denial of service attacks at the MAC layer in wireless ad hoc networks". In: *MILCOM 2002. Proceedings* 2 (2002), pp. 1118–1123.

[99] Klas Fokine. "Key management in ad hoc networks". PhD thesis. 2002.

[100] Nils Ole Tippenhauer et al. "On the requirements for successful GPS spoofing attacks". In: *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*. New York, New York, USA: ACM Press, 2011, p. 75.

[101] Hung Jen Liao et al. *Intrusion detection system: a comprehensive review*. 2013.

[102] Yih Chun Hu and Adrian Perrig. *A survey of secure wireless ad hoc routing*. May 2004.

[103] Ahmad Y. Javaid et al. "Cyber security threat analysis and modeling of an unmanned aerial vehicle system". In: *2012 IEEE International Conference on Technologies for Homeland Security, HST 2012*. IEEE, Nov. 2012, pp. 585–590.

[104] David W Carman, Peter S Kruus, and Brian J Matt. *Constraints and approaches for distributed sensor network security*. Tech. rep. 2000.

[105] Ohad Rodeh, Kenneth P Birman, and Danny Dolev. *Optimized group rekey for group communication systems*. Tech. rep. 1999.

[106] Alan T. Sherman and David A. McGrew. "Key establishment in large dynamic groups using one-way function trees". In: *IEEE Transactions on Software Engineering* 29.5 (May 2003), pp. 444–458.

[107] Wei Chi Ku and Shuai Min Chen. "An improved key management scheme for large dynamic groups using one-way function trees". In: *Proceedings of the International Conference on Parallel Processing Workshops*. IEEE Comput. Soc, 2003, pp. 391–396.

[108] Xuxin Xu et al. "Preventing collusion attacks on the one-way function tree (OFT) scheme". In: *International Conference on Applied Cryptography and Network Security*. Springer, Berlin, Heidelberg, 2007, pp. 177–193.

[109] Lakshminath R Dondeti, Sarit Mukherjee, and Ashok Samal. *Distributed group key management scheme for secure many-to-many communication*. Tech. rep. 2001.

[110] Ling Cheung et al. *Collusion-resistant group key management using attribute-based encryption*. Tech. rep. 2007.

[111] Sanjeev Setia et al. "Kronos: a scalable group re-keying approach for secure multicast". In: *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (2000), pp. 215–228.

[112] J. Staddon et al. "Self-healing key distribution with revocation". In: *Proceedings - IEEE Symposium on Security and Privacy*. Vol. 2002-Janua. IEEE Comput. Soc, 2002, pp. 241–257.

[113] Donggang Liu, Peng Ning, and Kun Sun. "Efficient self-healing group key distribution with revocation capability". In: *Proceedings of the 10th ACM conference on Computer and communication security - CCS '03*. New York, New York, USA: ACM Press, 2004, p. 231.

[114] L.R. Dondeti, S. Mukherjee, and A. Samal. "A dual encryption protocol for scalable secure multicasting". In: *Proceedings IEEE International Symposium on Computers and Communications (Cat. No.PR00250)*. IEEE Comput. Soc, 2003, pp. 2–8.

[115] Sandro Rafaeli, Laurent Mathy, and David Hutchison. *An efficient one-way function tree implementation for group key management*. Tech. rep. 2001.

[116] Adrian Perrig et al. "The TESLA broadcast authentication protocol". In: *RSA Cryptobytes* 5 (2005).

[117] Jack L. Burbank et al. "Key challenges of military tactical networking and the elusive promise of MANET technology". In: *IEEE Communications Magazine* 44.11 (Nov. 2006), pp. 39–45.