AN EFFICIENT PAIRWISE KEY ESTABLISHMENT SCHEME FOR AD-HOC MOBILE CLOUDS

A Thesis

by

SUBHAJIT MANDAL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2015

Major Subject: Computer Engineering

ABSTRACT

An Ad-hoc Mobile Cloud (AMC) is a new computing model that allows sharing computing power of multiple mobile devices. For a diverse group of individuals that employ such computing model, in an ad-hoc manner, secure peer-to-peer communication becomes very important. Using private or pairwise keys to secure such communication is preferable to public-keys because of computation and energy requirements. With the advent of sensor enabled mobile devices, a protocol (SekGens) that uses sensor data to generate pairwise keys on demand has been proposed. To work successfully SekGens requires devices to be closely located and becomes infeasible for devices situated multiple hops away. SekGens is also expensive in computation and slow in key generation. In this thesis, we investigate how to enable devices in an AMC to establish pairwise keys. We propose an efficient solution which tries to reduce the number of executions of SekGens in the AMC, and establishes pairwise keys between mobile phones multiple hops away by distributing parts of the key on multiple routing paths. Our results show a reduction of up to 75% in the number of SekGens required to establish keys in an AMC, when compared to a naive approach. Also the execution time to come up with the optimal pairs is within 10s of seconds for reasonably large networks.

DEDICATION

To my Father and Mother

who always believed in me and made a lot of sacrifice for my well-being

ACKNOWLEDGEMENTS

First and foremost I would like to extend my gratitude to Dr. Radu Stoleru, my adviser, for giving me this opportunity to pursue my master's thesis under his guidance. I am thankful to all the members of LENSS laboratory at Texas A&M University, for helping me on any and every doubt and problem that I faced during my research. In particular I am really grateful to Chen Yang for his help and advice while coming up with the solution to my research problem and during the writing of our paper. I would like to thank Dr. Alex Sprintson, one of my committee members, for helping me in identifying relevant prior state of the art. I would also like to thank Tamas Nepusz who helped me with doubts in using the igraph library while performing the experiments.

NOMENCLATURE

AMC	Ad-hoc Mobile Cloud
SMA	SekGens Minimization Algorithm
SekGens	Session Key Generated from Sensors

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Mobile devices, such as cellphones, tablets, etc. are ubiquitous in today's world. In addition to traditional communication tasks (e.g. phone calls, text messages), modern mobile devices enable more complex services such as mobile social networking, crowdsensing, etc [17, 32] due to powerful computing, sensing and networking capabilities. An *Ad-hoc Mobile Cloud (AMC)* is a recently proposed idea, where a collection of mobile devices form a distributed system in an ad hoc manner. Ad-hoc Mobile Cloud is defined here in a different way than traditional cloud computing. It is a collection of mobile devices that are owned by different individuals but collaborate among themselves sharing resources, which benefits all the participants [16]. Several interesting applications have been developed for AMC, including peer-to-peer file sharing, distributed computing [15, 21], collaborative data storage and processing [7, 8]. With an AMC, applications are easy to deploy, highly mobile, and do not rely on infrastructure-based networks.

Although there are multiple advantages for using Ad-hoc Mobile Cloud, it is paramount to establish secure links among peers in an AMC, especially for data sensitive applications. Key establishment for securing device to device communication in ad-hoc networks has been under active research [26]. None of existing solutions use the sensing capability of current mobile devices in generating security keys. In recent work [1], a protocol (SekGens) uses sensor data gathered from mobile phones to generate a strong cryptographic pairwise key for a pair of nodes within a certain range.

Despite its guarantee of generating a strong key for a pair of nodes, the use of SekGens in large networks remains challenging. First, it is impossible for users that are not within a certain range to generate keys with SekGens. Second, generating keys using SekGens takes non-negligible time and thus it is time consuming for a user to use SekGens to generate keys with multiple neighbors sequentially. Considering these two aspects, it is burdensome to use SekGens individually in a network of relatively larger size ($\sim$40-50) nodes.

We focus on the security issues concerned with such mobile collaborative networks. There is possibility of the presence of malicious devices in the network who might wish to listen to the communication exchange that takes place among two peers. With wireless medium the security threat becomes more pronounced. It is very easy for a malicious entity to listen and capture the information being exchanged. The content that the two peers share might be confidential classified government documents, military strategy plans or maps, even business proposals, that might have catastrophic consequences if it falls into wrong or unauthorized individual's hands. So securing such peer-to-peer communication is of utmost importance.

Key generation for securing device to device communication in ad-hoc network has been under active research. One of the novel approach for that called SekGens is presented in [1]. SekGens uses sensor data from the mobile phones to generate strong cryptographic pairwise keys. SekGens has its own drawbacks, in terms of failure to scale with large networks, user inconvenience in generating keys with large number of peers and finally its inability to function when peers are not within a certain distance. We address these problems through the combination of SekGens and intelligent routing of key fragments over Node-disjoint paths, enabling an ad-hoc MAC of mobile phones to generate secure pairwise keys among them. Since performing SekGens is expensive and inconvenient optimizing the number of SekGens for a given number of mobile phones is the problem we try to solve in this paper.

In this thesis, we propose to combine SekGens and *Node-disjoint Path Key Es-*

tablishment technique [25] to solve the pairwise key establishment problem in an Ad-hoc Mobile Cloud. The goal is to establish pairwise keys for all pairs of nodes while minimizing the number of SekGens executions. Our basic idea is to carefully choose pairs of nodes that are within a certain range to perform SekGens to establish secure links, while other pairs establish the keys by exchanging key fragments over multiple node disjoint paths, whose links are already secured. We analyze the formulated optimization problem and deduce the lower bound of the optimal value. In order to solve the problem efficiently, we develop a heuristic algorithm called SekGens Minimization Algorithm (SMA). We evaluate the proposed algorithm through simulations and show that it reduces the number of SekGens significantly (up to 75%) when compared to unoptimized naive solutions [27].

The contribution in this thesis is:

- An efficient algorithm for key establishment using node-disjoint paths combined with the key generation capability of SekGens for an AMC

2. BACKGROUND AND MOTIVATION

In this chapter, we introduce the two techniques for pairwise key establishment, namely *SekGens* and *Key Establishment Using Node-Disjoint Paths.* We then motivate our research by identifying the drawbacks when each of the techniques is used individually, and the advantages when the two are combined in an efficient manner.

2.1 Symmetric or Pairwise Key Establishment

Session Key Generated from Sensors (SekGens) algorithm [1] aims at establishing a secure long key (128 bits) between two mobile devices using their contextual information (i.e., data obtained from device's sensors). SekGens has three phases. In the Quantization Phase, the key is iteratively generated based on different sensor's data. In the Reconciliation Phase, the two mobile devices eliminate minor differences in the bits of their keys by using the Cascade reconciliation mechanism. In the PrivacyAmplication-and-Hashing Phase, the two devices omit all bits exposed during the reconciliation phase and apply hashing to the remaining secret bits to strengthen the key. SekGens was implemented and evaluated by modifying the Android code responsible for WiFi Protected Setup (WPS) protocol in Google Nexus 5 and Samsung Galaxy S2 smartphones. The evaluation results show that SekGens is efficient in generating keys with low mismatch ratio and with high Shannon entropy.

[25] proposes the idea of *Key Establishment Using Node-Disjoint Paths.* The idea is to use node-disjoint paths to secure the negotiation and establishment of a symmetric key between a pair of nodes. Depending on the number of available node-disjoint paths between a sender and a receiver, the sender divides the key into a fixed number of fragments. Each fragment is then sent along one of the node-disjoint paths. The receiver needs to collect all the fragments in order to recreate

the complete key. So an attacker has to compromise at least one node on each of the node-disjoint paths to capture all the fragments to discover the key.

2.2 Motivation

Even though SekGens proves its efficiency and robustness when it runs on same and different Smartphones that are within a certain distance [1], it has some limitations in case it is used to establish a key for many nodes in AMC networks (i.e., multi-hop networks). The time needed to establish a key (i.e., 6 seconds) is relatively long and impractical for a large number of nodes. Moreover, due to the requirement of closer distances (i.e., 0, 1.5 or 3 m), called **SekGens-distance**, between devices preferred by SekGens, it might be inconvenient or even impossible to generate keys for a large number of users. As a result, an efficient pairwise key establishment scheme is required to benefit from SekGens in an AMC.

The Node-disjoint path key establishment scheme has two major limitations. First, it assumes that the key is already generated and it only decides the path for distributing the key fragments. There is a need for generating the actual key with some key generation algorithm in the first place. Secondly, the key fragments can only be forwarded from one node to the next hop neighbor if there already exists a pairwise key between them. So keys need to be established between each neighbor node pair on the node-disjoint paths before the fragments can be securely forwarded from the source to the destination.

An efficient algorithm for key establishment using node-disjoint paths combined with the key generation capability of SekGens for an AMC is our contribution in this paper. Instead of blindly performing SekGens between all device pairs, we minimize the number of SekGens based on the network size, number of key fragments and node density. We present experimental results with respect to risk of key being compromised and required number of SekGens performed for a network using varying number of node-disjoint paths.

3. LITERATURE REVIEW

Security key is used to encrypt the data before sending over the network. Key establishment and management for wireless sensor networks is a widely studied area. Ad-hoc Mobile Cloud (AMC) is a better resource enriched version of the wireless sensor network deployment. So the solutions proposed for sensor networks can be applied with appropriate modifications to AMC systems.

Two types of keys are used in cryptographic systems. The asymmetric key cryptographic scheme, where each device has a private and a public key. The public key is distributed to the network for every device. The private key is not known by any one other than the owner. Any information encrypted by device d's public key can be deciphered by d's private key which is known to only d. The other is the symmetric key scheme, where each communicating pair have an unique private pairwise key between them. For any two devices d_1 and d_2 there is a key d_{12} for their exclusive use and unknown to any other device in the network.

Previous studies have shown that generation and use of asymmetric key requires a lot of computation power, time and research [18,31,34] has shown that it is not suitable for mobile devices which have a limited battery life. So symmetric, that is pair-wise key is the preferred solution for such mobile networks.

How to establish strong pairwise keys is another important research problem, with lot of work been done [2,23,30,35] on secret key generation. Most of the earlier works rely on physical layer properties and signal properties between the devices and fail to use any contextual information availably locally to the device. A new protocol, named SekGens, that uses sensor data gathered from the environment to generate cryptographically strong keys has been developed in one of our recent works [1].

We briefly introduced the protocol and described its shortcoming in the previous chapter. The problem that we are solving in this thesis is unique and specific to the use of the SekGens for a AMC. That is how it relates to key establishment problem studied in wireless sensor networks.

We present a brief review of similar works done for pairwise key establishment for wireless ad-hoc networks. Most of the symmetric key cryptography protocols for establishing a pairwise key between two devices make use of an on-line key server. Mitchekk and Piper [28] uses a probabilistic key sharing which avoids the use of on-line key server. However, the storage requirement is decided by the number of nodes in the network and does not scale well with large networks.

The Random Key Pre-distribution proposed by Eschenauer and Gligor [14] uses pre loading nodes with a certain number of keys chosen at random from a larger pool of keys. After deployment two nodes within communication range exchange key-identifiers or challenges to discover common keys. Node pairs without common key establish a key through a secure path. Drawback of this way of key establishment is that the key gets exposed on each intermediate node to the destination, which has been called the per-hop key exposure problem in [25]. Another drawback is the requirement of pre-loading keys on each node without knowing what is the network topology and the context where the device is going to be used. So there is a need for generating keys on-demand.

Chan et al. [5] extended the idea of [14] proposing three new mechanisms for key establishment using probabilistic key pre-deployment and using multipath key reinforcement. Even this work uses pre-deployment of keys and does not use any contextual information surrounding the device.

Zhu et al. [36] proposed a scheme inspired by [5] and [14] that uses multiple logical paths to communicate the key shares. This thesis also proposes a similar idea. However there is no prior loading of keys on any node. Our solution is purely on-demand.

To address the per hop key exposure problem [25] proposed using multiple node-disjoint paths to establish pairwise keys between non-neighbor nodes. Based on these paths, the pair-wise key, K, is divided into multiple fragments, each of which is transmitted along one of the established secure paths. The solution works in combination with existing key pre-distribution scheme. The main idea is, given a network with n nodes, where a secure topology has been established using a shared-key discovery phase. Now a node *N1* needs to set up a pair-wise key with another node *N2*. This is achieved using the following steps:

- *N1* uses a Node-disjoint Path finding algorithm to get a set, *PS*, of node-disjoint secure paths to *N2*.
- Let $s = |PS|$, be the size of the Node-disjoint Path set *PS*. Node *N1* selects a key K and divides it into s fragments, $K_1, K_2 ... K_s$, such that $K = K_1 \cup K_2 \cup ... \cup K_s$, where $K_i \cup K_{i+1}$ means a concatenation of K_i and K_{1+1}. Also each fragment contains a sequence number so that after all the fragments are received, *N2* can assemble the fragments.
- *N1* sends K_i through the i_{th} secure path.
- After *N2* receives all the s fragments of the key, it can reproduce the key K and uses it for secure communication with *N1*.

The requirement of finding end-to-end secure path from the source to the destination is a strong requirement for [25]. To relax this condition [24] identifies nodes called *proxy* that share keys with both the source and the destination. Depending on how many fragments the source wants to break a key, it discovers those many proxies and sends each of them a fragment encrypted with the pairwise key it shares with the proxy. The proxy decrypts the key and sends the fragment to the destination with the key it shares with it. This avoids the strong requirement of having end-to-end secure paths. Gupta et al. in [19] propose a similar idea, just naming the *proxy* as *friend* nodes and requiring that the intermediate relay node share a key with the destination only. Thus making it easier to find friend nodes compared to proxy nodes. All these algorithms address the key establishment problem, leaving the key generation on the users. It assumes that *N1*, the source some how has the key present with it. These algorithms only solve the establishment of the already generated key between node pairs. Our solution takes help of these ideas, specifically from [25], but solves the key generation problem as well as efficient establishment of the key.

The protocol SekGens [1], solves the problem of on-demand key generation using sensor data. Users use their mobile phones to perform specific gestures. Simple gestures like, making letters of the English alphabet such as "O", "N", "V" etc. produces sensor data in the phones. These sensor data is collected from accelerometer, acoustic, temperature sensors present on the phone. This contextual data is used on the SekGens protocol to create a session key. The protocol quantizes the data gathered from sensors and generates the raw key at each of the device pairs. Based on the choice of best sensor data the protocol through an iterative algorithm generates a 320 bit key, KEY_Q. This local key is generated independently at the device pairs. So there is high probability of mismatch of the KEY_Q^u at node u and KEY_Q^v at node v. So by a process called "Reconciliation Phase" the protocol exchanges certain bits of the keys between the devices and fixes the differing bits in them, to generate the reconciled key, KEY_{Rec}. KEY_{Rec} on the two devices still might have small difference. The protocol runs an error estimation to determine the difference and depending on the error being within a pre-decided threshold, uses or discards the key. Finally if the KEY_{Rec} is usable, the protocol hashes the key to remove eavesdropper's knowledge of the KEY_{Rec}. Finally, both the devices apply MD5 hashing on the remaining secret bits to form the 128 bits KEY_{Final}.

The SekGens protocol for key generation is novel and is cryptographically strong. However, it has the biggest drawback in the aspect of time of key generation and significant user involvement for generating the key. An user needs to perform the specific gestures with the mobile phone for about 6 to 8 seconds for the protocol to establish the key. This process needs to be replicated for every other device that a user wants to establish a key with. It is easy to see that as the number of users increases this process becomes inconvenient for an user. Further, this will work only when two devices are within a certain distance. So for a bigger gathering of mobile phone users, the protocol fails completely.

We present a comparative study of the existing symmetric key establishment schemes in Table 3.1. We have identified the following relevant metrics for this thesis, to evaluate each scheme's characteristics and convenience in deployment in real systems. Firstly, is it *on demand*, meaning can the solution generate keys on the fly. Secondly, *scalability* which means how well the solution works as the number of devices is increased in the network. Thirdly, does it solve the *key exposure* problem while communicating the key over multiple hops. Finally, *context sensitivity* which means does the solution consider the device's surrounding information in anyway in the key establishment.

The problem we have solved in this thesis is, combine the capability of SekGens protocol to generate on-demand session key from sensor data and the node-disjoint path mechanism for communicating key fragments that allows an Ad-hoc Mobile Cloud of users to establish session keys among them to securely communicate pair-

Classification	Pairwise Key Establishment Solution	On-demand	Scalable	Solves Key Exposure
Context Insensitive	Probabilistic Key Pre-deployment [Mitchell et al., Eschenauer et al.]	No	No	No
	Pre-deployment and Multipath [Chan et al., Zhu et al.]	No	No	Yes
	Multipath and Friend/Proxy Assisted [Ling et al., Gupta et al., Li et al.]	No	Yes	Yes
Context Sensitive	SekGens [Altaweel et al.]	Yes	No	No
	Scalable SekGens (SMA) [Mandal et al.]	Yes	Yes	Yes

Table 3.1: Comparison of Pairwise Key Establishment Protocols for Ad-hoc Mobile Networks

wise. We have minimized the number of times an user needs to run the SekGens protocol and still be able to generate keys from sensor data.

3.1 Difference with Steiner Network Problem

The problem we solve in this thesis can be simplified to a know class of network problems called Steiner Network [33] problems under certain assumptions about the Ad-hoc Mobile Cloud. Given an AMC with the additional information that it is k-connected, that is there are k node disjoint paths between every pair of non-neighbor vertices, then the optimization problem is to find the minimal k-connected sub-graph, that is also k-connected. This problem is called the "Minimum k-vertex connected subgraph" problem, which is studied by other researchers [9, 22]. This k value is actually what we define as the Key Fragment Factor (KFF) during our problem formulation in Chapter 4. We are not making any such assumption or imposing such requirement on the AMC. We only assume that the AMC is atleast biconnected. That is between any pair of non-neighbor vertices, we are able to find two node-disjoint paths. Our solution will honor or try satisfying this KFF value, k for pairs that have those many number of node-disjoint paths between them. For pairs that do not have the required number of node-disjoint paths between them, will have their keys distributed over the best, that is whatever the number of node-disjoint paths that the AMC provides for them. This differential treatment makes this problem no longer just an optimization problem of finding the minimal k-vertex connected subgraph. Since in Steiner Network problem, the starting assumption is that the graph is k-vertex connected and then it finds the minimum k-vertex connected subgraph from there, our problem differs from it by not making that initial assumption about the AMC.

The different connectivity requirement between the vertices is a problem of finding the minimal subgraph from a given graph that satisfies the requirement. To the best of our knowledge, there has been very few works that solve the node-disjoint connectivity requirement problem for undirected simple graphs. One of the interesting work is [4]. It uses a metaheuristic greedy approach to come up with an approximate solution. It uses a iterative search to improve over the earlier solution. The goodness of the solution is determined by the number of iterations. It seems that this approach might be slow in converging to its solution and become impractical for our AMC, where we need fast convergence, since user convenience is one of the desired property. The solution proposed in this thesis is expected to be faster, since it calculates the set of node-disjoint paths between all pairs and then identifies the critical edges. So our solution might perform better than existing solutions like [4].

3.2 Application of Network Coding

The use of multiple node-disjoint paths for distributing the keys has some shortcomings. The information even though divided over multiple mutually exclusive paths makes it difficult for a curious node to get the complete picture, but does not guard against the leakage of information. Ideas can be burrowed from network coding techniques to prevent this information leak [3,13]. The adversary node functions as a wiretapper, who can read the packets that it forwards. With redundant information added to the packets, even an internal node that helps in the packet forwarding, acting maliciously can not gather too much information that can actually compromise the complete key. Such network coding based solutions [12] that guard against wiretappers would make our proposed key establishment protocol more robust.

4. PROBLEM FORMULATION AND ANALYSIS

In this chapter we describe the system model and formulate the problem mathematically.

4.1 System Description

4.1.1 System Model

Consider an Ad-hoc Mobile Cloud (AMC) having a collection of N mobile phones, $\{v_1, v_2, ..., v_N\}$. The AMC is modeled as a graph $G = (V, E)$, where $V(G)$ is the set of vertices, i.e. phones and $E(G)$ is the set of edges. An edge $e = (v_i, v_j) \in E(G)$, implies v_i and v_j are within SekGens-distance. G is the connectivity graph. The goal is to establish pairwise keys between all N devices.

There are two ways to establish key between any pair, SekGens and Node-disjoint path key generation. For any v_i and v_j, if there exists a $e = (v_i, v_j) \in E(G)$, then we can use SekGens. We can use Node-disjoint path key generation also between v_i and v_j if there are adequate number of such paths between them, which we explain in the next paragraph.

As mentioned in [25] to avoid the per hop key exposure problem, a key needs to be broken into k fragments and sent over k node-disjoint paths, where $k \geq 2$. The receiver needs to receive all k fragments in order to recreate the key. We introduce parameter k as **Key Fragment Factor (KFF)** in our model.

To make the communication of the key fragments secure over the multiple node-disjoint paths additional technologies can be augmented over our solution. The process of distributing the key fragments, is basically a multicast network communication. To secure the key fragments while in transit, over the node-disjoint paths, we can use existing solutions from information theory [6, 12] for securely communicating the fragments. Further to make it hard for an adversary to recover the bits in the fragments network coding techniques [29] can be leveraged, that would reduce the information leakage, in the presence of adversary devices.

4.1.2 Adversary Model

The adversary is a single or a group of malicious device(s), who wish to discover the key(s) that is being established between device pair(s) in the AMC. For an adversary to recreate the key, it has to compromise one node on each of the k paths. We focus on inside attacker only, as such devices have easy access to the key being exchanged. The attacker is only a passive entity. It does not jam the communication medium. Also the attacker does not modify the data it is entrusted to communicate or forward to other device, since data manipulation makes the attack an active one and we consider only passive attacker model in our system.

4.1.3 Assumptions

We make the following assumptions,

- We assume that the users use mobile devices that support at least one sensor. Sensor data is required to generate the keys whenever required.
- Two nodes can perform SekGens only if they are within the SekGens-distance.
- The communication of the key fragments over the node-disjoint paths takes less time than performing SekGens between two devices.
- There is no inherent vulnerability in the mobile devices themselves. That is the devices are not infected with malware a priori which could manipulate the key fragments.
- A node is able to generate cryptographic key from already stored sensor data

and break it into required number of fragments, when required for use in node-disjoint path key establishment.

- The topology of G does not change while our key establishment algorithm is running.

4.1.4 Problem Formulation

For N vertices in the connectivity graph G we need $\binom{N}{2}$ pairwise keys. It is trivial to notice that when G is a complete graph, each device can perform SekGens with every other node, but it is inefficient. It is possible to reduce the number of execution of SekGens by replacing some of these SekGens by node-disjoint paths to distribute the key fragments instead.

We formulate the problem as the following optimization problem:

Given a connectivity graph G and a KFF k, decide on the devices that should perform SekGens so as to minimize the number of executions of SekGens while establishing pairwise key for all $(u, v) \in (V(G) \times V(G))$.

4.2 Problem Analysis

In this subsection, we discuss the characteristics of the optimization problem. Before going into the details of our algorithm we present some properties that the connectivity graph G should possess for a feasible solution and argue on the hardness of the problem.

4.2.1 Feasibility

Feasibility is defined as the requirements that the system model needs to meet for generating pairwise keys for all nodes in a given connectivity graph G and for the specified KFF value k. The connectivity graph needs to have certain properties and the nodes must be able to perform some minimum number of SekGens individually, for the solution proposed later in the paper to work. We explain them and argue their necessity below.

We discuss the Lemma(s) and Theorem below for a connectivity graph where the KFF value $k = 2$, is the requirement. Afterwards we generalize it for $k > 2$ cases.

Lemma 1. *For a given connectivity graph G and any vertex pair v_i and v_j, both v_i and v_j need to perform at least two SekGens each to use node-disjoint paths to establish key.*

Proof. Let P_1 and P_2 be two node disjoint paths between vertices v_i and v_j. Let $P_1 = \{v_i, v_{11}, v_{12}, ..., v_j\}$ and $P_2 = \{v_i, v_{21}, v_{22}, ..., v_j\}$. Since we claim that P_1 and P_2 are node disjoint paths between v_i and v_j, therefore $(P_1 - \{v_i, v_j\}) \cap (P_2 - \{v_i, v_j\}) = \emptyset$. Hence none of the nodes that occurs on one path can occur on the other for the pair of nodes v_i and v_j. This implies that v_i must perform SekGens with v_{11} and v_{21} and they are distinct vertices. This is necessary because the key needs to be broken into at least two fragments and sent over two node-disjoint paths. Similarly v_j must perform two SekGens with the last node on the two paths. $\square$

Before presenting the next Lemma, we introduce a basic graph theory concept. In graph theory, a biconnected graph is a connected and nonseparable graph, meaning that if any vertex were to be removed, the graph will remain connected. Therefore a biconnected graph has no articulation or cut vertices.

Lemma 2. *For a given connectivity graph G, to find two node-disjoint paths between every pair of vertices v_i and v_j, G must be biconnected.*

Proof. Consider the graph in Figure 4.1(a).The vertex X is a cut vertex or articulation point for the graph, since the removal of vertex X would increase the number of connected components. Let G_1 and G_2 be the two subgraphs of the original graph G that are created on removing X. In the original graph G, for any two nodes v_i and v_j, such that $v_i \in G_1$ and $v_j \in G_2$, a path from v_i to v_j is $P = \{v_i, v_1, v_2, ...X, ..., v_{n-1}, v_n, v_j\}$. To find two such paths that are node-disjoint is not possible in G for such v_i and v_j, since the node X will be present on all such paths. By the assumption on our connectivity graph, such a topology will not be supported by our algorithm. $\square$

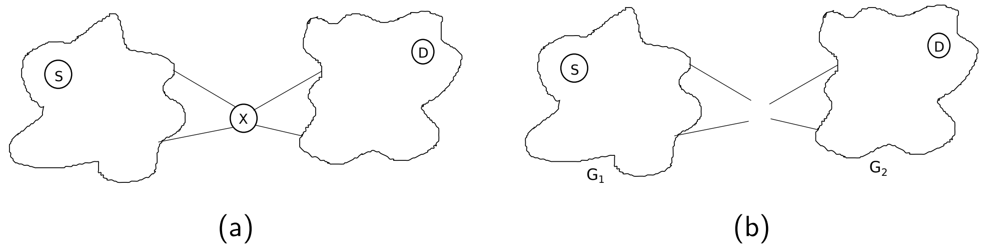


Figure 4.1: Our Solution assumes the connectivity graph does not have an articulation point for $k = 2$

The statements of Lemma 1 and 2 must be satisfied for all connectivity graphs where $k = 2$ is the KFF value.

4.2.2 Hardness

We show the hardness of the optimization problem and provide an intuition of how it can be reduced to a known **NP**-hard problem, for the case of $k = 2$. Hence for the general case k the problem is also **NP**-hard.

Theorem 3. *Let $G = (V, E)$ be a connectivity graph with N nodes and OPT be the optimal number of SekGens for generating key between all device pairs in G. Then,*

a. *$OPT \geq N$, for $k = 2$.*

b. *$OPT = N$, iff there exists a Hamiltonian Cycle in G.*

Proof. To show that the minimum number of SekGens for $k = 2$ cannot be less than N, lets assume that the minimum number of SekGens for G is $N - 1$ and then show a contradiction. The claim is that after performing $N - 1$ SekGens each node will have two unique node disjoint paths with every other node. If we start with N disconnected nodes and connect the pairs that perform SekGens, we get a graph, $G^{'} = (V^{'}, E^{'})$ that has $|V^{'}| = N$ nodes and $|E^{'}| = N - 1$ edges. By [20] , if $G^{'}$ is connected and $|V^{'}| = |E^{'}| + 1$, then every two nodes of $G^{'}$ are joined by only one unique, that is node-disjoint path. So there cannot be two node-disjoint paths between non neighbor nodes in $G^{'}$. Therefore for N node graph the number of SekGens has to be greater than or equal to N.

For the equality condition, we claim that for G the minimum number of SekGens is N. Let $G^{'}$ be the graph with N nodes and N edges where each edge denotes SekGens has been performed between the neighbors. We need to show that $G^{'}$ is a Hamiltonian Cycle of G. $G^{'}$ will have 2 node disjoint paths between every pair of node that did not perform SekGens, only then will those node be able to establish key using node-disjoint path. We remove one of the edges, $e = (u, v)$ from $G^{'}$. Since there was 2 node-disjoint paths before the edge was removed, there should still be 1 node-disjoint path between every pair in the new graph, which we call $G^{''}$. So $G^{''}$ is connected and $|V(G^{''})| = |E(G^{''})| + 1$. By [20], $G^{''}$ has to be acyclic and if the removed edge, $e = (u, v)$ is added back, where u and v were non-adjacent in $G^{''}$, since we had removed the edge between them, the resulting graph has exactly one cycle. This new graph is actually $G^{'}$ since e was removed from $G^{'}$ to obtain $G^{''}$ and was added back. So $G^{'}$ has N nodes and N edges. From definition, $G^{'}$ is then a

Hamiltonian Cycle for G.

Now assume that G is Hamiltonian and prove the equality condition. Let the graph G be Hamiltonian and assume G' is the corresponding Hamiltonian Cycle of G. So G' is a graph with N nodes and N edges. Every non neighbor node on G' has two node-disjoint paths between them. Hence performing SekGens between nodes that are connected by an edge on G' will be enough. Since these N nodes of G' are actually the same nodes in G, this will be the least number of SekGens for G, which is equal to the number of nodes N. □

Using the result from Theorem 3, for the case of $k = 2$, the problem of determining the minimum number of SekGens, is equivalent to finding the Hamiltonian Cycle for the connectivity graph. The *Hamiltonian Cycle* problem is a known **NP**-complete problem [10]. Hence for $k = 2$, if we can find the minimum number of SekGens for G, we are able to find the Hamiltonian Cycle, if it exists. Hence our problem is reducible to the Hamiltonian Cycle problem.

4.2.3 General k Value

The Lemma and Theorem stated earlier are for the case of $k = 2$. We generalize them for $k > 2$ case.

For $k > 2$ case, for the key to be broken into k fragments there needs to be k number of node-disjoint paths from the source to the destination. Using the reason of Lemma 1, to have k node-disjoint paths the source needs to perform at least k SekGens, with k neighbors.

For having k node-disjoint paths between a source and a destination node the connectivity graph should be *k-connected*. This requirement ensures that the algorithm used to find node-disjoint paths, will find k such paths. This is the generalized version of Lemma 2. In real life AMCs it is not always possible to have the connectivity graph to possess this property. For any k **KFF** value requested by the users, our solution is a best effort endeavor. This means that if it is possible to distribute the key over k-node disjoint paths, we would do so. Otherwise, whatever the maximum number of paths we can find between the vertex pair we distribute the key over them. This is called differential security level.

Using Theorem 3, for $k = 2$, we showed that the problem is reducible to a known **NP**-complete problem. So for $k > 2$ the problem becomes harder.

Therefore we propose a heuristic algorithm for solving this optimization problem in the next chapter.

5. THE MINIMIZATION ALGORITHM

5.1 Sekgens Minimization Algorithm (SMA)

In this section we propose our algorithm to establish keys between pairs of devices while trying to minimize the number of SekGens for the network.

5.1.1 Main Idea

The basic idea is to select neighbors v_i and v_j at each iteration and run SekGens between them, such that it helps other pairs to use node-disjoint paths to distribute their key, using the edge (v_i,v_j) that is already secured. This will allow those pairs to send key fragments on those secured paths and establish key between them. Here we are greedy in choosing a pair which is the best choice at that iteration, with the hope that it will minimize the total number of SekGens for the complete network. We continue doing this until all pairs have keys between them. Our algorithm determines the best pairs to perform SekGens for a given connectivity graph. We present this idea algorithmically in the next subsection.

5.1.2 SekGens Minimization Algorithm (SMA)

The SekGens Minimization Algorithm (SMA) has two parts, which are Algorithm 1 called *Edge Importance-Value Calculator* and Algorithm 2 called *Schedule Generator*.

Algorithm 1 determines how many times an edge occurs on some node-disjoint path in G. This is recorded as the Importance-Value for that edge. It uses an existing Node-Disjoint path finding algorithm (NDPAlgorithm) [10] and calculates all the node-disjoint paths between each device pair (lines 1-2). The paths that are at least of length 2 are usable for sending key fragments. This is because two neighbors

Algorithm 1 SMA: *Edge Importance-Value Calculator*

Input: Network Topology G, Key Fragment Factor k

Output: Calculate the Importance of each edge, $e \in E(G)$

```
1: for all $(u, v) \in (V \times V)$ do
2:   $P_{uv}$ = NDPAlgorithm$(G, u, v)$
3:   Initialize counter $i = 0$
4:   for all $p \in P_{uv}$ do
5:     if PathLength$(p) \geq 2$ then
6:       $i \leftarrow i + 1$
7:     else
8:       Delete p from $P_{uv}$
9:   if $i \geq k$ then
10:    Add pair (u,v) to D {This pair has the required number of paths to satisfy
       KFF of $k$}
11:    Keep only $k$ shortest paths
12:  else
13:    if $i \geq 2$ and $< k$ then
14:      Add pair (u,v) to D, mark as reduced security
15:  else
16:    Delete (u,v) from node pair list
17: for all $e \in E$ do
18:   $F_e := 0$
19: for all $e \in E$ do
20:   for all $(u, v) \in (V \times V)$ do
21:     for all $p \in P_{uv}$ do
22:       if $e \in p$ then
23:         $F_e \leftarrow F_e + 1$
24: Sort $F$ in descending order of edge importance value
```

cannot communicate a key fragment securely between them unless they already have a key between them. So if v_i and v_j want to establish a key, the edge $e = (v_i, v_j)$ if exists cannot be used by them. The algorithm checks for such paths and deletes the ineligible paths (lines 3-8). It then checks if there are the required k number of node-disjoint paths between the node pair and stores such pair (line 9-10). This is used to decide whether this pair can use node-disjoint paths with the specified KFF of k to establish key. In case the required k paths are not available, the algorithm marks that pair as using reduced security in terms of less than k KFF value. It then computes the importance value of each edge in the graph by accumulating the number of times the edge occurs on a node-disjoint path between some source S and some destination D (lines 11-17). The algorithm then arranges the edges in a decreasing sequence of their relative importance and maintains it as queue (line 18), that is used by Algorithm 2.

Algorithm 2 decides the pairs that will use SekGens and the ones that are going to use node-disjoint paths for key establishment. This is done by greedily choosing an edge in each iteration with the hope that securing this pair will benefit the most. Initially all pairs are insecure, i. e. there does not exist a key between them. The algorithm adds each pair to a set (U), called the insecure set and initializes an empty set (S) that contains all the pairs that have established a key between them (line 1-3). It also initializes a timer (t) and a set (Q), called Schedule Maintainer, to keep track of the method that will be used to establish key between a particular pair (lines 4-5). In each iteration it dequeues the most important edge, e (line 7). It deletes the pair that makes up that edge from U and F (line 8). It adds that pair to the secure set S (line 9). The end vertices of e will perform SekGens, which is recorded in Q (line 11). Each element of Q is of the form $(v_i, v_j, 0/1)$, which denotes that node v_i and v_j should perform SekGens if the third entry in the tuple is 1, otherwise they use node-disjoint path to establish key. It then looks for all pairs that are still insecure (line 12) and have edge e on a path in their node-disjoint path set (line 13). If such a pair is found, then it checks if that pair has all its paths in its path set, have edges that have a key on them (line 16-27). If such a pair is found, then it adds that pair to S and also recording in Q that node-disjoint path is to be used for key establishment between them, by setting the third entry in the tuple as 0 (line 28-32). The algorithm terminates when F becomes empty and also all the pairs are secured, i. e. U becomes empty (line 33).

5.1.3 SekGens Minimization Algorithm Example

To demonstrate the execution of our algorithm, consider the connectivity graph of Figure 5.1(a). The connectivity graph, G has 5 nodes and 6 edges. In the

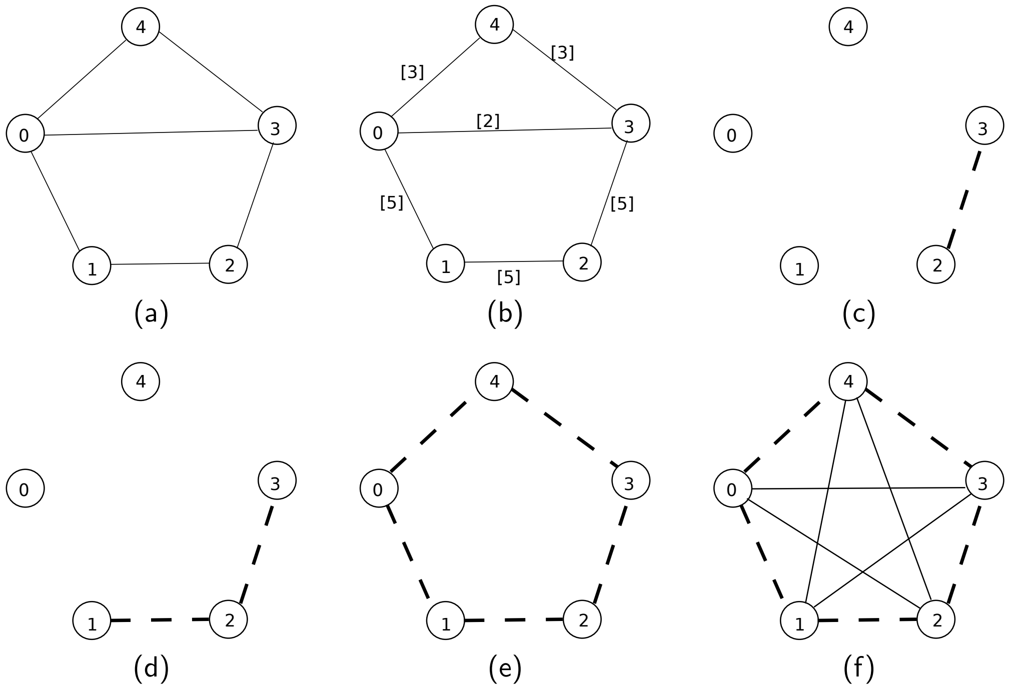


Figure 5.1: Example Graph (G) and intermediate steps to illustrate SekGens Minimization Algorithm

naive approach we perform SekGens on all possible pairs. It will still not be possible for pairs like $(1, 4)$ to establish a key, since they are not within *SekGens-distance*. We assume that the KFF parameter is $k = 2$, that is two node-disjoint paths are sufficient to send key fragments between any node pair. Using the *NDPAlgorithm* in *Edge Importance-Value Calculator*, we get the following node-disjoint paths between the node pairs as shown in Table 5.1. Only paths that are at least of length 2 can be used to send key fragments. One hop paths cannot be used to securely send a key, as

node-pair(v_i, v_j)	Path Set
(0, 3)	[(0, 4), (4, 3)], [(0, 1), (1, 2), (2, 3)]
(1, 3)	[(1, 0), (0, 3)], [(1, 2), (2, 3)]
(2, 4)	[(2, 3), (3, 4)], [(2, 1), (1, 0), (0, 4)]
(1, 4)	[(1, 0), (0, 4)], [(1, 2), (2, 3), (3, 4)]
(0, 2)	[(0, 3), (3, 2)], [(0, 1), (1, 2)]

Table 5.1: Node-disjoint Path Set between Node-pairs in G

explained earlier and hence are not considered. The algorithm assigns Importance-Value, shown in Table 5.2 and also Figure 5.1(b) to edges in G based on the number of time it occurs in some node-disjoint path using the data from Table 5.1.

edge(v_i, v_j)	Importance-Value
(2, 3)	5
(1, 2)	5
(0, 1)	5
(3, 4)	3
(0, 4)	3
(0, 3)	2

Table 5.2: Edges in G with their Importance-Values

The Importance-Value of the edges $(2,3)$, $(1,2)$ and $(0,1)$ are maximum and equal, so Schedule Generator algorithm picks one of them first. It marks SekGens to be performed between those two vertices, which is denoted by dotted line in Figure 5.1(c). Now $(2,3)$ are secured. It then checks for pairs that use this edge on any of its node-disjoint paths. It finds such pairs, like $(1,3)$, but it will see that none of the pairs have all the links on their path set secured, meaning a key fragment cannot be forwarded securely from the source to the destination node. It then takes the next important edge, which is $(1, 2)$. The node pair $(1, 2)$ perform SekGens, as shown in Figure 5.1(d). It does the same check as before, but fails to find any such pair that can use paths that are secured on each link from the source to the destination. The algorithm proceeds like this until it reaches the situation in Figure 5.1(e). Now it can find two node-disjoint paths between all the non-neighbor nodes and also between the neighbors $(0, 3)$ and uses node-disjoint paths to establish the rest of the keys. So in the Figure 5.1(f), the edges represent that the nodes have a pairwise key between each one of them. The dotted edges signify SekGens was used for key establishment and the solid lined edges signify use of node-disjoint paths. For the given G, we could establish all the keys between each pair with 5 SekGens.

5.2 Time Complexity Analysis

We provide an analysis of the execution time for the minimization algorithm. The total execution time for establishing keys between every pair in the network is calculated as,

$$\begin{aligned} Total\ Execution\ Time\ (TET) &= Schedule\ Generation\ Time\ (SGT) \\ &\quad + Real - time\ Execution\ Time\ (RET) \end{aligned}$$

where, SGT is the time it takes for the SekGens Minimization Algorithm to decide which pairs are to use SekGens and which ones will use Node-disjoint paths and RET is the time taken by the users in the network to execute this with their mobile phones in real time. We comment on the theoretical time and present experimental results for SGT only. The time for RET will depend on the user's level of engagement in the process and communication throughput in the location where this solution is deployed. Since, performing such real experiments were cumbersome and difficult in the limited scope and resources available, we are unable to show actual data on the RET portion of the total execution time.

The Schedule Generation Time, involves running the SMA algorithm on a centralized machine. Referring to Algorithm 1, the time complexity of NDPAlgorithm is the major contributor to the overall execution time of the algorithm. The NDPAlgorithm involves finding the maximum-flow value from any source node to a destination node. The time complexity of the maximum-flow algorithm using the library used in our implementation is $O(|V|^3)$ [11], where V is the set of vertices of the connectivity graph, G. The NDPAlgorithm is called $O(|V|^2)$ times. So the overall time complexity of the loop in line 1 of Algorithm 1 is $O(|V|^5)$, which is within polynomial time. The inner loop takes less time compared to NDPAlgorithm and so does not affect the Big-O calculation. The other loop in line 13, has a time complexity of $O(|E| * |V|^2 * k)$, where E is the set of edges in G and k is the Key Fragment Factor. This loop is expected to take lesser time compared to the earlier loop. Hence, the overall time complexity of Algorithm 1 is decided by the execution of NDPAlgorithm and in our implementation it of the order of $O(|V|^5)$. For Algorithm 2 the loop in line 13-33 is the major contributor to the time complexity. The outer loop runs for $O(|V|^2)$ times. The inner loop in line 16-27, runs for $O(k)$ times with the innermost loop in line 19-23 depends on the number of edges in the longest path from the source to the destination node, which can at most be $O(|E|)$. Hence the total complexity of Algorithm 2 is $O(|E| * |V|^2 * k)$. So adding the time complexity of the two algorithms, the overall complexity of the SGT phase is $O(|V|^5)$.

Algorithm 2 SMA: *Schedule Generator*

Input: $V(G)$, Edge-Importance-List F

Output: Schedule $Q_t(u, v, 0/1)$

```
 1: for all $(u, v) \in (V \times V)$ do
 2:   Add pair (u,v) to U
 3: Initialize Secure Set $S := \emptyset$
 4: Initialize Timer $t := 0$
 5: Initialize Schedule Maintainer $Q := \emptyset$
 6: repeat
 7:   $e \leftarrow MAX(F)$
 8:   Delete $e$ from F and U
 9:   Add e$(= (x, y))$ to S
10:   Increment $t \leftarrow t + 1$
11:   Set $Q_t := (x, y, 1)$
12:   for all $(u, v) \in U$ do
13:     if $e \in some\ p \in P_{uv}$ then
14:       $no\_of\_paths$ := Size($P_{uv}$)
15:       Initialize counter $j := 0$
16:       for all $q \in P_{uv}$ do
17:         $path\_length$ := PathLength(q)
18:         Initialize counter $i := 0$
19:         for all $e \in q$ do
20:           if $e \in S$ then
21:             $i \leftarrow i + 1$
22:           else
23:             Break
24:         if $i == path\_length$ then
25:           $j \leftarrow j + 1$
26:         else
27:           Break
28:       if $j == no\_of\_paths$ then
29:         Delete (u,v) from U and F (if it occurs)
30:         Add (u,v) to S
31:         Increment $t \leftarrow t + 1$
32:         Set $Q_t := (u, v, 0)$
33: until $F \neq \emptyset$ & $U \neq \emptyset$
```

6. RESULTS AND EVALUATION

6.1 Performance Evaluation

In this section we present the performance evaluation of our *SekGens Minimization Algorithm*. We evaluate our algorithm based on the parameters that model the possible network and user requirements. In each case we plot the number of SekGens performed against one of the parameters.

- **Network Size (N)**: We vary the number of nodes in the graph to study this parameter. It reflects whether our algorithm reduces the number of SekGens that would otherwise be required for a group of mobile phones without using any optimization. The increasing number of nodes shows the scalability issue of SekGens that we identified as one of its major drawbacks.

- **Node Density (r)**: Node density depends on the number of neighbors that a node has. The value of r actually is the **SekGens-distance**. This parameter helps to understand how our algorithm performs as the density of the devices in the network changes. With more nodes within one hop, meaning denser network, our algorithm should be judicious in choosing the optimal number of neighbors to perform SekGens with. We study how many SekGens are performed as the number of neighbors for each device increases or decreases.

- **Key Fragment Factor (k)**: We defined the number of fragments the key is broken into as the KFF, as more the number of fragments the more difficult it becomes for an adversary to compromise the key. Generating more fragments implies we need to find more node-disjoint paths. As the number of required paths increases it takes more SekGens in securing all the links on such paths before we can use the node-disjoint solution of distributing the key to the destination. We expect a positive correlation between k and the number of SekGens.

The simulation environment we use for testing our algorithm is by generating random geometric graphs (RGG) using the python library "python-igraph" [11] and measuring the number of SekGens value in them. We compare our algorithm's performance with the *Naive*-algorithm and *Random*-algorithm.

The *Naive*-algorithm does not use any optimization or decision on choosing pairs to perform SekGens. It just performs SekGens between every pair that are neighbors. So the number of SekGens in all its applications on all graphs are equal to the number of edges in the graph.

The *Random*-algorithm uses a random number generator in each iteration to decide on an edge to perform SekGens between its end nodes. After each such SekGens it however checks to see if it can satisfy the required k end to end secured node-disjoint paths for any pair. If it can find such a pair it uses node-disjoint paths to establish key between those pairs.

6.2 Number of SekGens for each Parameter

Figure 6.1 shows the number of SekGens that are performed with the different network sizes. We construct the test networks by randomly placing N nodes in a two dimensional unit square and connecting two nodes if they are closer to each other than a given radius, which is actually the SekGens-distance in the graphs. We used the values for parameters $r = 0.8$ and $k = 3$ in all these simulation.

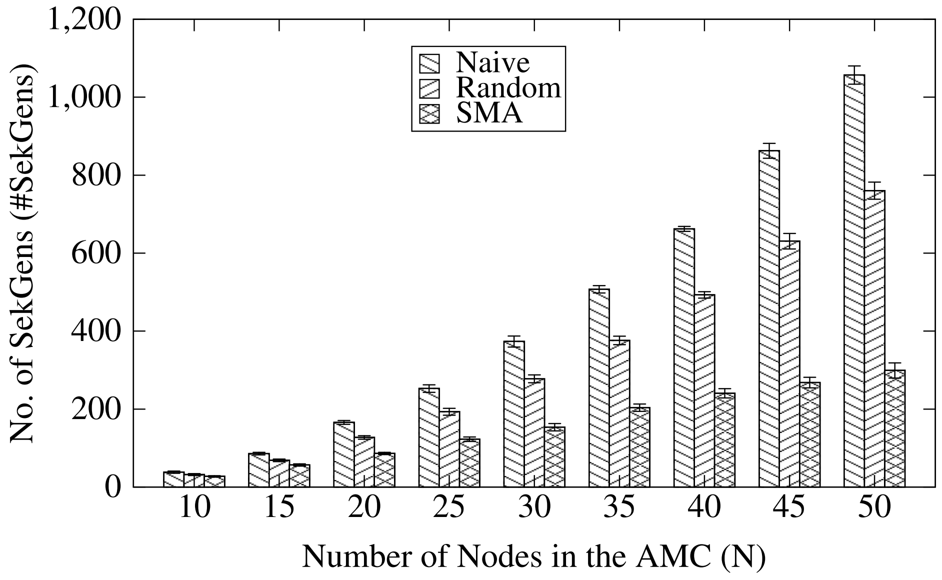


Figure 6.1: Number of SekGens in different sized networks

We get different but similar trends in the number of SekGens using other r and k values. The plots show the average number of SekGens that are performed for the network sizes, $N \in \{10, 15, 20, 25, 30, 35, 40, 45, 50\}$ with a confidence of 95%. Compared to the *Naive* and *Random* algorithm our SMA algorithm performs consistently better as the network size increases. For example in a 50 node network, the Naive algorithm performs SekGens on all the edges, resulting in more than 1000 SekGens and Random performs close to 800 SekGens. Whereas our SMA algorithm needs about 250 SekGens, which is about 75% less and about 65% less than the number of SekGens performed by Naive and Random algorithm respectively.

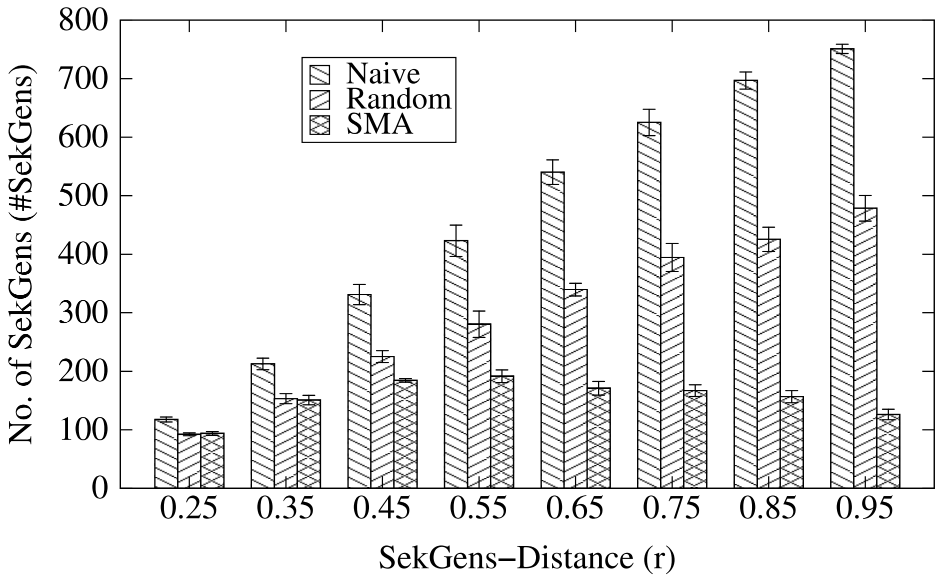


Figure 6.2: Number of SekGens on a network with 40 nodes with varying density

Figure 6.2 shows the results when the network size is kept constant, but the node density is varied. This is achieved by changing the SekGens-distance value. With larger SekGens-distance r, a node has more neighbors to possibly perform SekGens and hence more possibility of executing higher number of SekGens. We kept the network size fixed at $N = 40$ and KFF parameter at $k = 3$, which again with other values will give similar results. SMA performs significantly better compared to the other two. One interesting trend is observed in the results though. Intuitively, it is expected that as the number of neighbors of a node increases, SMA should have increasing value for number of SekGens. The results show that the number of SekGens increases till a certain density (here $r = 0.55$) and then falls slightly as the graph becomes denser. The reason behind this trend is, for sparse graph there is anyway less neighbors to do SekGens with, hence the lesser value. As the graph becomes dense more paths are available to distribute the keys using node-disjoint paths and SMA ends up being greedy and performs some redundant SekGens. As the node's degree increases further, even though more node-disjoint paths are found, there are shorter length paths available. Since SMA prefers shorter paths, the number of SekGens goes down slightly compared to less dense graphs but with longer length paths between pairs.

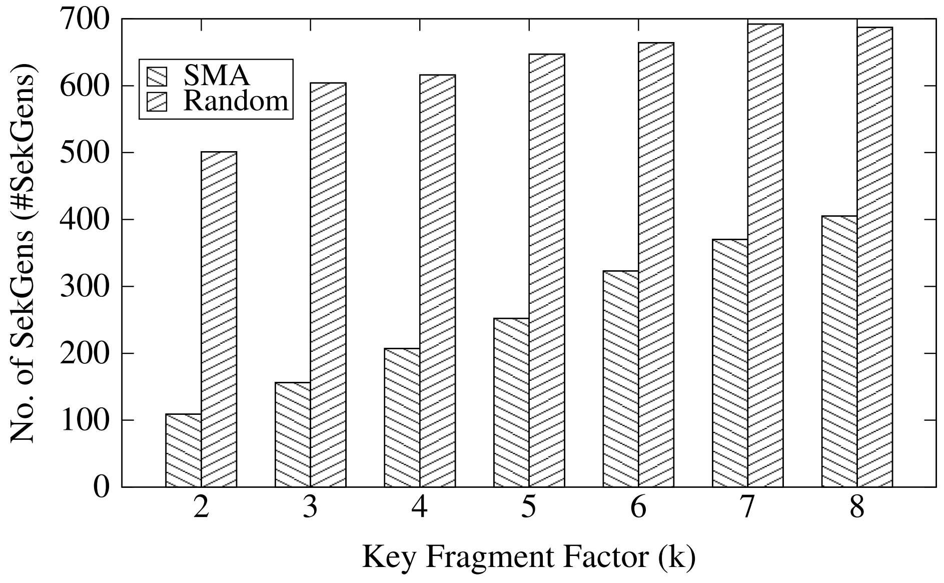


Figure 6.3: Number of SekGens on a network with 40 devices with varying key fragments

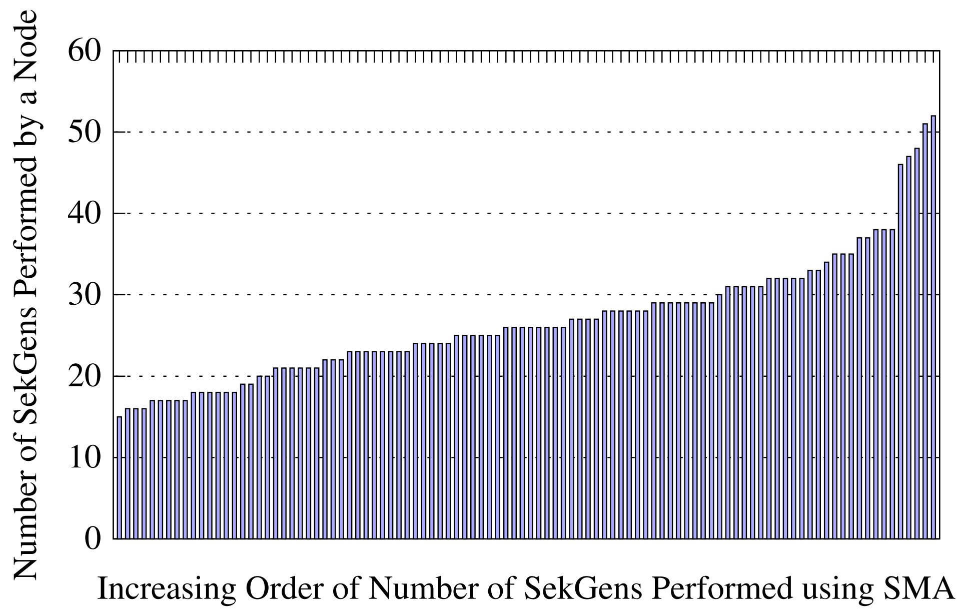
Figure 6.3 captures the performance of the *Random* and *SMA* algorithms as the parameter for number of node-disjoint paths, i. e. KFF k between pairs is changed. We fix the network size to $N = 40$ and SekGens-Distance $r = 0.7$. We did not show the values for the Naive Algorithm, since it will be equal to the number of edges for all the k values. SMA's performance is clearly better than the Random Algorithm and keeps the total SekGens value at almost 50% of what the Random approach gives, when k is varied from 2 to 8.

6.3 Load Distribution

We define load as the number of SekGens performed by an individual node in a network. In lemma 1 we proved that for having k node-disjoint paths, that is for a key strength of k, each device needs to perform at least k SekGens. Since a node might fall on the path of a node-disjoint path between a pair, the actual number of SekGens performed by a node would be more than the minimum requirement. A good algorithm should try and distribute the number of SekGens evenly across all the nodes in the network. Figure 6.4 shows the load on each node in a 100 node network for SMA and Random algorithm respectively. Our SMA achieves a better distribution with the average number of SekGens per node being around 26, while the corresponding value on the same graph for Random edge selection is 60. Using the max-min fairness index, we can measure how fair is the load distribution in the network. Using SMA we get a fairness value of 0.32. While for Random algorithm the fairness value is around 0.34. Even though the fairness measure is comparable for the two algorithms, the absolute value of the number of SekGens performed by a node is more in the Random approach than our SMA approach.

6.4 Differential Security

We discussed the relaxation of the vertex connectivity requirement on the AMC in Chapter 3. We mentioned that when the users ask for a **KFF** of k, to distribute the key over those many node-disjoint paths, the connectivity graph needs to be k-vertex connected. For very high k value, the AMC might not be that richly connected. For such scenarios, our solution falls back to the best effort security that it can provide. Our algorithm optimizes on the number of executions of SekGens, while distributing the key fragments over the maximum possible node-disjoint paths for every vertex pair as the connectivity graph can support. The number of fragments that we break



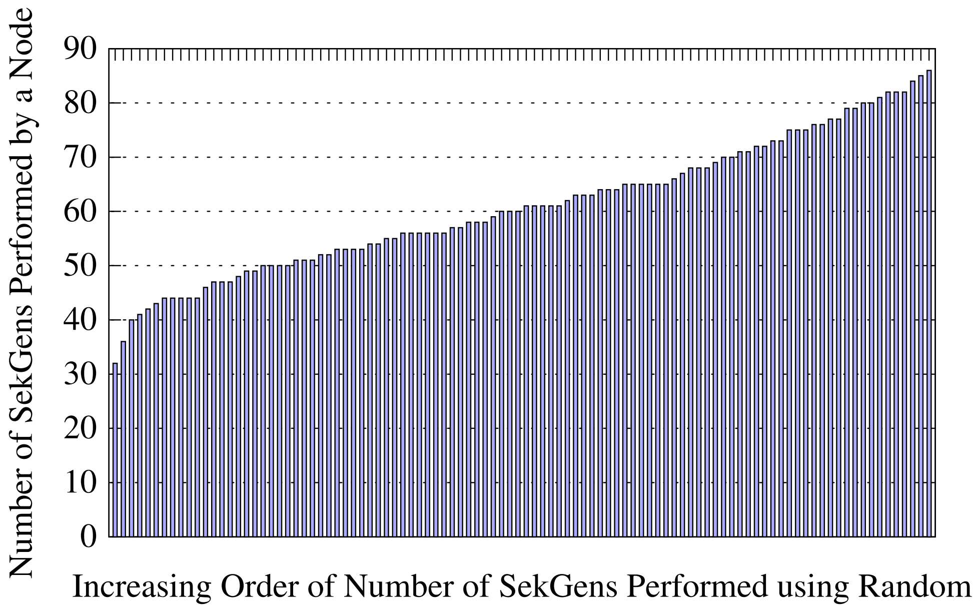


Figure 6.4: SekGens performed per Node in a 100 node network

the key into determines the security level of the key establishment. More fragments implies hard it is for an adversary to capture all the parts of the key and hence higher the security level. With different vertex connectivity in between different vertex pairs, the security level is thus different for the pairs in the AMC. We present in Figure 6.5 a case where the **KFF** value is 20. Since here the key is to be distributed into such a large number of fragments, the number of SekGens is considerably high. Still compared to the Random and Naive approaches, our solution performs consistently better. To study what is the security level we supported for the different pairs, in

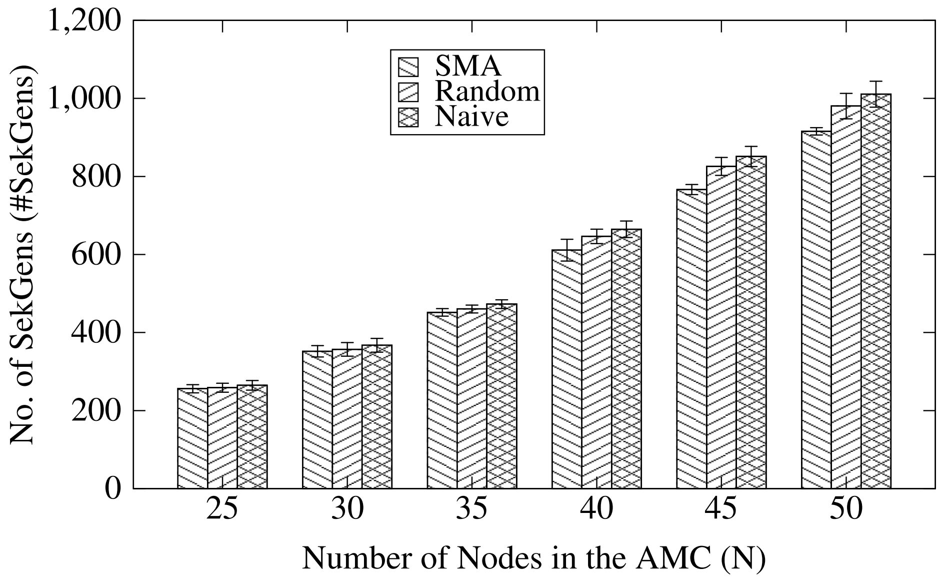


Figure 6.5: Number of SekGens for k = 20, a high KFF value

Figure 6.6 we show the distribution of the vertex connectivity value between the vertex pairs in a 40 node AMC, that we used in Figure 6.5 to see the number of execution of SekGens.

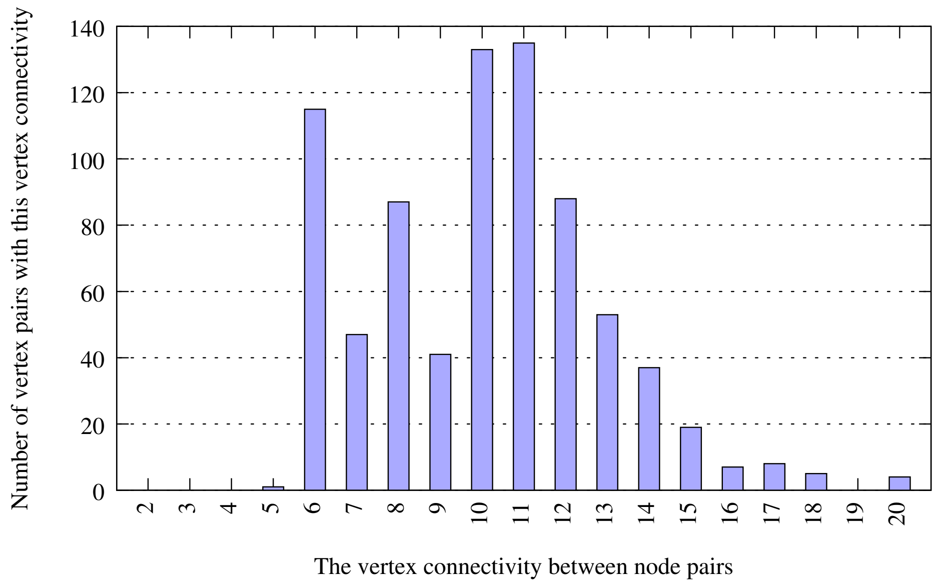


Figure 6.6: Example showing the idea of differential security, because of different number of node-disjoint paths between vertex pairs

If we had tried to optimize on the number of SekGens by limiting the vertex connectivity value to the least for the AMC, that is 5, then even though we had better connectivity for 95% of the pairs, we would not have leveraged it. Thus going by Steiner Network based solution, which searches for the overall vertex connectivity of the graph, the security level would be gravely lower for almost all the pairs. With our differential security level approach we are able to support more robust (from security perspective) key establishment.

6.5 Execution Time

We refer to the time of execution discussion that we presented in the Chapter 5 and present the system time needed for deciding the pairs that would use SekGens and Node-disjoint path respectively for key establishment. We tested our algorithm on a portable laptop which has an Intel T7500 (dual core @2.20 GHz) processor and runs Ubuntu 14.04 (32-bit). We ran the following experiments. First, keeping the SekGens distance (r) fixed at 0.7 and the KFF (k) at 3, we varied the number of nodes (N). We generated different Random Geometric Graphs (RGGs) and took the average time of Schedule Generation Time (SGT). We plotted the average SGT value with 95% confidence interval in Figure 6.7.

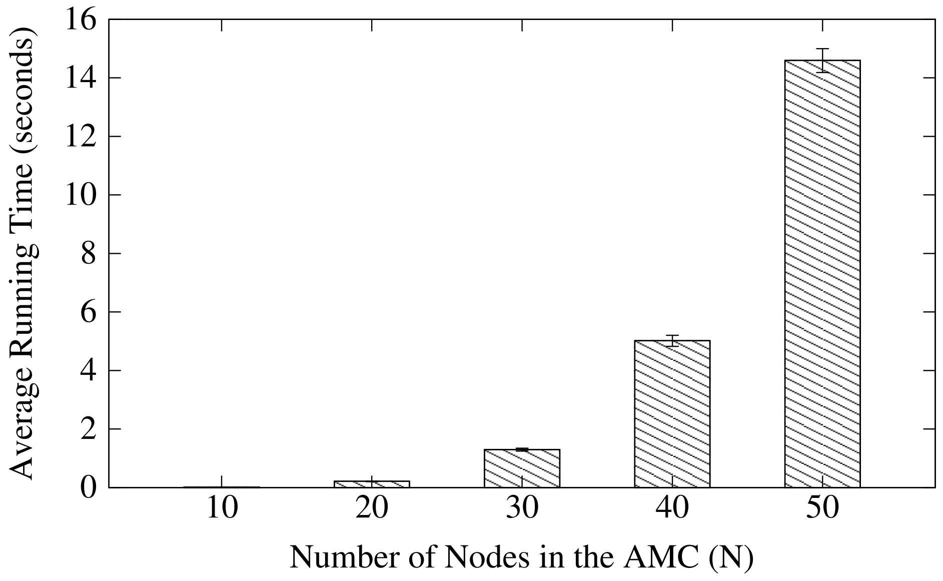


Figure 6.7: Schedule Generation Time for different sizes of the network

The time to decide on the pairs for SekGens execution for networks of sizes till 30 is less than even 2 seconds. For a 40 node network it is around 5 seconds. While for a 50 node network it is bit higher taking around 14 seconds. Still considering the significant number of less SekGens that is performed, this initial time expenditure is worth while. With other values of “N”, “r” and “k” we would get similar values of execution times.

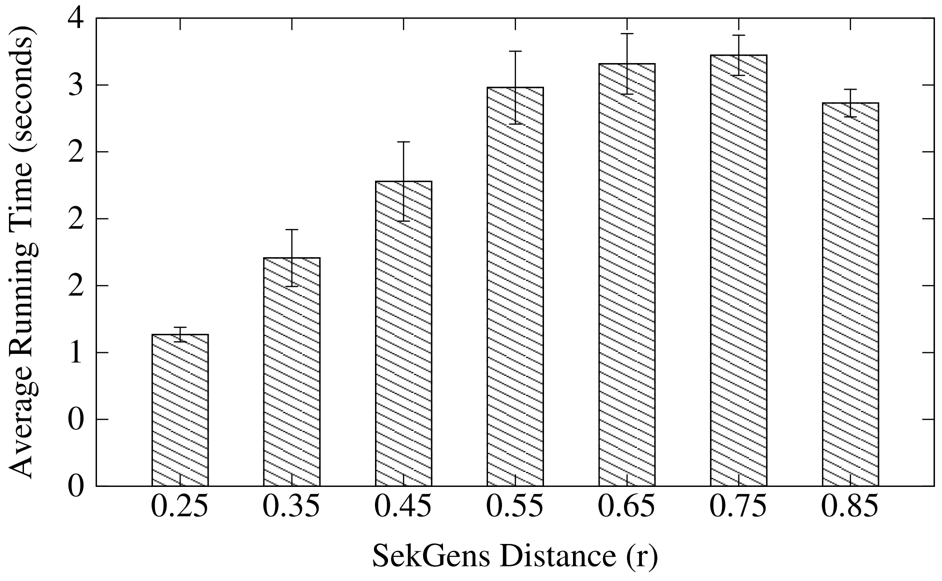


Figure 6.8: Schedule Generation Time for same sized network, with varying SekGens-distance

For the next experiment we kept the network size, "N" fixed at 40 nodes and KFF (k) at 3, but changed the SekGens-distance, "r", Figure 6.8. It is expected that as we increase the SekGens-distance, there will be more and more neighbors for a particular node. With more neighbors, that is denser graph, the greedy SMA algorithm would perform more number of SekGens, hence longer time to decide on those SekGens. Following the similar trend we saw in Figure 6.2, where the number of SekGens increased upto certain density and then fell slightly, we see exactly similar plot for the execution time as well. The same argument we presented explaining the results in Figure 6.2 applies here.

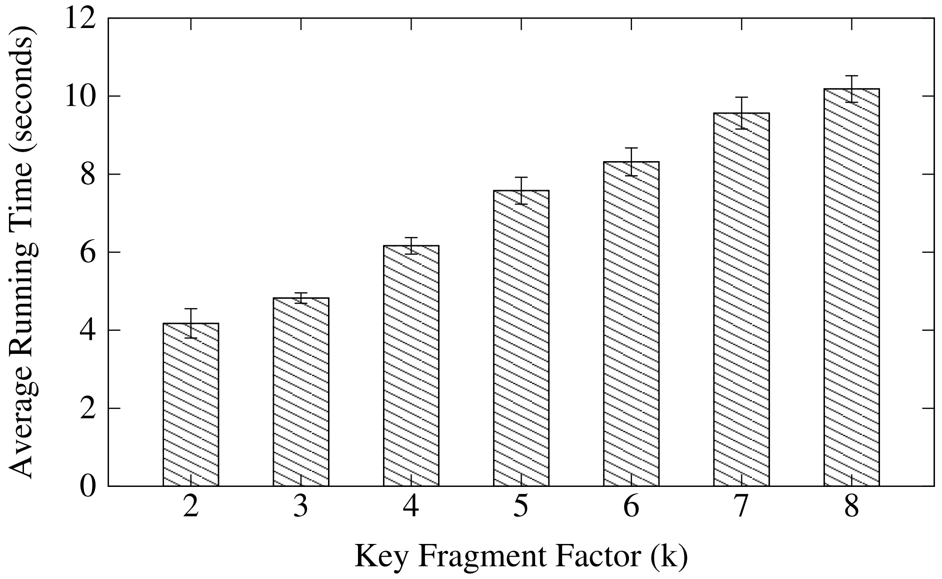


Figure 6.9: Schedule Generation Time for same sized network, with varying Key Fragment Factor

For final experiment, we changed the Key Fragment Factor (k), but kept the network size fixed at $N = 40$ and SekGens-distance $r = 0.75$. With more number of fragments the key being required to break into, the SMA algorithm requires more time to decide on the best node pairs that would perform SekGens. Hence, as expected, as we increased the value for "k" the SGT value also increased. However, it is encouraging to observe in Figure 6.9 that even with considerable large network of 40 nodes, the time required to decide on the SekGens pair is just 4 second for $KFF = 2$ and not more than 10 seconds for $KFF = 8$.

So we can claim that SMA can decide on the optimal pairs for SekGens within a reasonably short time, within tens of seconds, for real network deployments meeting the required Key Fragment Factor, for both dense and sparse node placements.

6.6 Security Risk vs Number of SekGens

[25] proposes an analytical model for quantifying the risk, denoted as r involved in using a set of s node-disjoint paths to establish key between two nodes. For a network of n devices, with x of them being adversary or malicious devices that collude among themselves, the probability of the key being discovered for a single pair is given by,

$$\begin{aligned} r &= prob[\{P_1, P_2, ...P_s\}, x, n] \\ &= \frac{\sum_{j_1=1}^{l_1} \sum_{j_2=1}^{l_2} \cdots \sum_{j_s}^{l_s} \binom{n-\sum_{m=1}^{s} l_m - s + \sum_{i=1}^{s} j_i}{x-s}}{\binom{n}{x}} \end{aligned} \tag{6.1}$$

where P_i is one of the s node-disjoint paths and l_i is the intermediate hop count of path P_i. We simulate graphs with $n = 20$ nodes, where $x = 16$ nodes are malicious

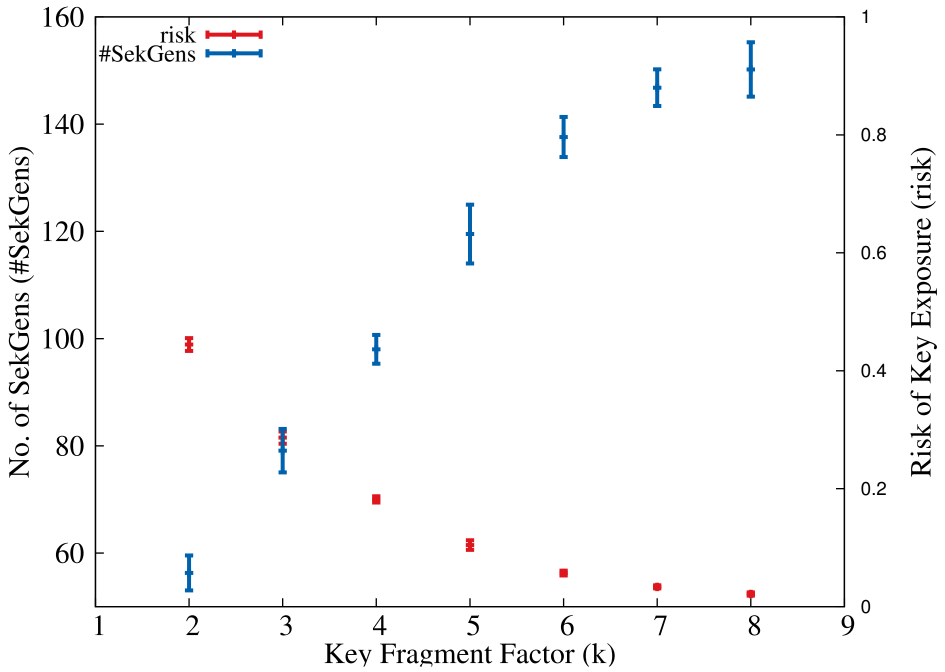


Figure 6.10: Comparison of Security Risk and corresponding No. of SekGens

and vary the KFF parameter k on these graphs and study the risk, calculated using Equation (6.1). We measure the corresponding number of SekGens for each k value. Intuitively, as the number of node-disjoint paths increases the risk of key getting revealed to an adversary should reduce, since more number of nodes need to be compromised to gather all the key fragments. Likewise as the number of node-disjoint paths increases, as we saw in Figure 6.3 the number of SekGens will also increase. We calculated the average security risk for a pair of nodes for each k value and the corresponding number of SekGens performed in the network and present it in Figure 6.10. It is clear from the figure that as the number of fragments or k value increases, the risk or probability of key being compromised decreases, while the required number of SekGens increases. With more number of SekGens, the time establish pairwise keys between all the devices will also increase. Thus depending on the risk and time expenditure thresholds, a practical deployment of our solution will need to have some trade-off that the users need to decide. To keep the risk of key compromise low, we have to divide the key into more fragments and the users in the AMC need to perform more SekGens, hence spend more time in establishing pairwise keys among all the devices.

7. CONCLUSION AND FUTURE WORK

Generation of on demand pairwise key in an ad-hoc network composed of mobile phones is advantageous in real world scenarios like disaster area, battle field, under developed communities, etc where infrastructure network is not available or unreliable. Using sensor data to generate such key is an innovative idea. However the existing solution does not scale over more than a pair of mobile devices.

In this thesis we proposed a solution that can generate on demand pairwise keys through efficient use of SekGens for an Ad-hoc Mobile Cloud. For a dense network where each user might have to perform SekGens with every other device in the worst case, our solution reduces it to less than 50% of the theoretical maximum value [27]. For sparse network, where SekGens completely fails for devices that are not neighbors, our solution is able to enable key establishment for such devices.

Instead of the centralized approach that is taken in this thesis, a distributed solution is more practical. It would be a new problem to explore a distributed solution instead of this centralized approach as future work. Use of other heuristic methods, like Simulated Annealing to solve the optimization problem are future extensions to this work. Deciding how to schedule the SekGens in the network, so that parallel such executions can be performed, is a related problem.

The problem addressed in this thesis is NP-Hard, as we have already explained while analysing the problem. This thesis proposes an approach to solve this problem and achieve an approximate solution. There are existing work that solve similar graph problem(s), particularly Steiner Network class of problems. Those problems can be mapped to the SekGens Minimization Problem if modified in appropriate way. An interesting future work would be to compare the performance of the Sek-

Gens Minimization Algorithm, proposed in this thesis with similar network design solutions [4].

REFERENCES

[1] Ala Altaweel, Subhajit Mandal, and Radu Stoleru. On secure shared key establishment for mobile cevices using contextual information. *Technical report, Texas A&M University*, 2015.

[2] Babak Azimi-Sadjadi, Aggelos Kiayias, Alejandra Mercado, and Bulent Yener. Robust key generation from signal envelopes in wireless networks. In *Proceedings of the 14th ACM conference on computer and communications security*, pages 401–410. ACM, 2007.

[3] Ning Cai and Raymond W Yeung. Secure network coding on a wiretap network. In *IEEE Transactions on information theory*, pages 424–435. IEEE, 2011.

[4] Héctor Cancela, Franco Robledo, and Gerardo Rubino. Network design with node connectivity constraints. In *Proceedings of the 2003 IFIP/ACM Latin America conference on towards a Latin American agenda for network research*, pages 13–20. ACM, 2003.

[5] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *Proceedings of symposium on security and privacy*, pages 197–213. IEEE, 2003.

[6] Pak Hou Che, Minghua Chen, Tracey Ho, Sidharth Jaggi, and Michael Langberg. Routing for security in networks with adversarial nodes. In *International symposium on network coding*, pages 1–6. IEEE, 2013.

[7] Chien-An Chen, Myounggyu Won, Radu Stoleru, and Geoffrey G Xie. Energy-efficient fault-tolerant data storage & processing in dynamic networks. In *Pro-*

ceedings of the fourteenth ACM international symposium on mobile ad hoc networking and computing, pages 281–286. ACM, 2013.

[8] Chien-An Chen, Myounggyu Won, Radu Stoleru, and Geoffrey G Xie. Energy-efficient fault-tolerant data storage and processing in mobile cloud. In *IEEE Transactions on cloud computing*, pages 28–41. IEEE, 2015.

[9] Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. In *SIAM journal on computing*, volume 30, pages 528–560. SIAM, 2000.

[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 3, pages 1091–1095. MIT press Cambridge, 2001.

[11] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. In *InterJournal on complex systems*, volume 1695, pages 1–9. New England Complex Systems Institute (NECSI), 2006.

[12] Salim El Rouayheb, Emina Soljanin, and Alex Sprintson. Secure network coding for wiretap networks of type ii. In *IEEE Transactions on information theory*, volume 58, pages 1361–1371. IEEE, 2012.

[13] Salim Y El Rouayheb and Emina Soljanin. On wiretap networks ii. In *IEEE International symposium on information theory*, pages 551–555. IEEE, 2007.

[14] Laurent Eschenauer and Virgil D Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on computer and communications security*, pages 41–47. ACM, 2002.

[15] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: a survey. In *Future generation computer systems*, volume 29, pages 84–106.

Elsevier, 2013.

[16] Niroshinie Fernando, Seng Wai Loke, and Wenny Rahayu. Dynamic mobile cloud computing: ad hoc and opportunistic job sharing. In *Fourth IEEE International conference on utility and cloud computing (UCC)*, pages 281–286. IEEE, 2011.

[17] Niroshinie Fernando, Seng Wai Loke, and Wenny Rahayu. Mobile crowd computing with work stealing. In *15th International conference on network-based information systems (NBiS)*, pages 660–665. IEEE, 2012.

[18] Gunnar Gaubatz, Jens-Peter Kaps, and Berk Sunar. Public key cryptography in sensor networks revisited. In *Security in ad-hoc and sensor networks*, pages 2–18. Springer, 2005.

[19] Arpan Gupta, Pavan Nuggehalli, and Joy Kuri. An efficient scheme for establishing pairwise keys for wireless sensor networks. In *2nd International conference on communication systems software and middleware*, pages 1–9. IEEE, 2007.

[20] Frank Harary. *Graph Theory*. AddisonWesley, 1969.

[21] Gonzalo Huerta-Canepa and Dongman Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond*, page 6. ACM, 2010.

[22] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. In *Combinatorica*, volume 21, pages 39–60. Springer, 2001.

[23] Suman Jana, Sriram Nandha Premnath, Mike Clark, Sneha K Kasera, Neal Patwari, and Srikanth V Krishnamurthy. On the effectiveness of secret key extraction from wireless signal strength in real environments. In *Proceedings of*

the 15th annual international conference on mobile computing and networking, pages 321–332. ACM, 2009.

[24] Guanfeng Li, Hui Ling, and Taieb Znati. Path key establishment using multiple secured paths in wireless sensor networks. In *Proceedings of the 2005 ACM conference on emerging network experiment and technology*, pages 43–49. ACM, 2005.

[25] Hui Ling and T Znati. End-to-end pairwise key establishment using multi-path in wireless sensor network. In *IEEE Global telecommunications conference (GLOBECOM'05)*, volume 3. IEEE, 2005.

[26] Wenjing Lou and Yuguang Fang. A survey of wireless security in mobile ad hoc networks: challenges and available solutions. In *Ad hoc wireless networking*, pages 319–364. Springer, 2004.

[27] Subhajit Mandal, Chen Yang, Ala Altaweel, and Radu Stoleru. An efficient pair-wise key establishment scheme for ad-hoc mobile clouds. In *11th International conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2015.

[28] Chris J Mitchell and Fred C Piper. Key storage in secure networks. In *Discrete applied mathematics*, volume 21, pages 215–228. Elsevier, 1988.

[29] L.H. Ozarow and A.D. Wyner. Wire-tap channel ii. In *Technical journal*, volume 63, pages 2135–2157. AT&T Bell Laboratories, Dec 1984.

[30] Neal Patwari, Jessica Croft, Suman Jana, and Sneha Kumar Kasera. High-rate uncorrelated bit extraction for shared secret key generation from channel measurements. In *IEEE Transactions on mobile computing*, volume 9, pages 17–30. IEEE, 2010.

[31] Krzysztof Piotrowski, Peter Langendoerfer, and Steffen Peter. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the fourth ACM workshop on security of ad-hoc and sensor networks*, pages 169–176. ACM, 2006.

[32] Wei-Tsung Su and Kiat Siong Ng. Mobile cloud with smart offloading system. In *IEEE/CIC International conference on communications in China (ICCC)*, pages 680–685. IEEE, 2013.

[33] Vijay V Vazirani. *Approximation Algorithms*. Springer science & business media, 2013.

[34] Arvinderpal S Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Third IEEE international conference on pervasive computing and communications (PerCom)*, pages 324–328. IEEE, 2005.

[35] Qian Wang, Hai Su, Kui Ren, and Kwangjo Kim. Fast and scalable secret key generation exploiting channel phase randomness in wireless networks. In *Proceedings of IEEE INFOCOM*, pages 1422–1430. IEEE, 2011.

[36] Sencun Zhu, Shouhuai Xu, Sanjeev Setia, and Sushil Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. In *11th IEEE International conference on network protocols*, pages 326–335. IEEE, 2003.