Maintaining the Security and Availability of a Stream of Time-Dependent Secret Information in an Ad-Hoc Network.

by

John David S. Sprunger

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science in Computer Science

Auburn, Alabama
December 15, 2018

Keywords:
Network Security, Threshold Cryptography, Ad-Hoc Networking

Approved by

Dr. Alvin Lim, Professor of Computer Science and Software Engineering
Dr. David Bevly, Bill and Lana McNair Endowed Distinguished Professor
Dr. David Umphress, COLSA Corporation Cyber Security and Information Assurance Professor
Dr. Xiao Qin, Director of Computer Science and Software Engineering Graduate Programs

Abstract

In this thesis we present a system called Ad Hoc Security for maintaining the security and availability of a stream of time-dependent secret data in an ad-hoc network. Time-dependence refers to how each piece of data is only useful during a unique time window. The goal is to determine the effectiveness of the Ad Hoc Security system for distributing and securing secret information in a mobile ad-hoc network under a variety of connectivity scenarios, with different sets of behavior parameters. Ad Hoc Security makes use of threshold cryptography for both decryption of the data as well as authentication of the participating devices. It is implemented and tested in the network simulator called ns3. The results of these tests are compared to show how different connectivity scenarios and behavior parameters affect the overall performance and security. The tests demonstrate its ability to adaptively shift between higher security and higher reliability based on its surroundings. Ad Hoc Security is compared to a similar, theoretical system that is tuned for perfect reliability at the cost of security. Compared to the perfectly reliable system, Ad Hoc Security consistently has half the vulnerable time[1] and half as much decryption material saved[2] with a minimum (and often avoidable) decrease in reliability. Most of the unreliability was from rapid, random group separation that the system could not predict or adapt to fast enough. At the end we also present three potential ways of improving reliability.

[1] Local Decrypt Time, or LDT. Defined in Section 5.4.
[2] Key Time. Defined in Section 5.3.

Table of Contents

List of Tables

List of Illustrations

List of Abbreviations

GPS	Global Positioning System
OLSR	Optimized Link State Routing
CA	Certificate Authority
TC	Threshold Cryptography
MOCA	Mobile Certificate Authority
RFI	Redundancy Factor for Individual Shadow Functions
RFC	Redundancy Factor for Threshold Cryptosystem
LDT	Local Decrypt Time
KT	Key Time

Chapter 1

Introduction

1.1 Background

The United States military and many NATO allies use a precise, encrypted GPS signal for geolocation. This signal is only able to be decrypted by GPS receivers that have loaded a GPS decryption key corresponding to the current time. This GPS key typically comes from either a secure, physical device (such as a Simple Key Loader) or from a server with secure wireless transmission capabilities. These decryption keys are only valid for certain periods of time, after which a new key is required.

1.2 Motivation

Because these GPS keys need to be kept secret, there is a large amount of administrative overhead that is applied to every GPS receiver, especially ones loaded with a valid GPS key. The manufacturing facilities that produce the GPS receivers must be secured against vulnerabilities such as hardware backdoors. The receivers must be tracked and accounted for while being moved through the logistics system. The end users must not lose their GPS receivers while in theatre, or else they may be required to backtrack and retrieve one that is lost. Loading GPS keys only as needed, and requiring external authentication and assistance for decrypting those keys would help to relax these requirements. Furthermore, while the valid time periods for these keys may be very long (one or more days) there is an issue that arises: How are the receivers rekeyed when a changeover occurs and the server or key loader are unavailable? This leads to the more general problem that is the title of this thesis: How can we maintain the security and availability of a stream of time-dependent secret information while part of an ad-hoc wireless network? This is the situation we wish to analyze and provide a solution for.

1.3 Assumptions

First, we must state every assumption that we are making. The first assumption is that the devices will form groups and ad-hoc networks. To be exact, we are assuming that there will be some form of ad-hoc routing protocol running on all of the devices. We use OLSR, but any ad-hoc routing protocol that maintains a routing table should work. In the real situation stated above, there was no mention of devices forming ad-hoc networks. We make this assumption because lone devices have fewer and much less creative options for dealing with security. Here, we are trying to leverage the power of the group to provide a better solution than could be achieved without any cooperation. Next, we are assuming that the devices may become disconnected from any infrastructure for extended periods of time. This is reasonable for any wireless network, but even more so in environments where a military would be operating. Also, everything required to load the secret will come from the "key server." It is named as such because, as mentioned above, in the original situation the secret was a GPS key. Here, it provides both the secret, as well as any cryptographic keys necessary to decrypt it. Lastly, we are assuming that the key server is always connected to a trusted upstream source for the GPS keys, etc. The primary point of this last assumption is that the devices can always validate their trust in both the key server and the data that it provides.

1.4 Problem Statement

1. There are secrets provided by a wireless server that must be loaded into devices.
2. Each secret is only useful during a unique time period.
3. The secrets must be protected from unauthorized users.
4. The devices may be disconnected from the server for extended periods of time.
5. The devices will form ad-hoc networks that may merge, split, etc.

6. Device authentication may need to be established or revoked while the devices are disconnected from infrastructure-bound authentication servers.
7. The system as a whole should be as resistant as possible to small numbers of device compromises (i.e. having all memory contents read).
8. The system should adapt to the current situation, attempting to maximize both security and availability of the secrets.

1.5 Related Work

We have found three papers that propose systems that bear some semblance to our system that will be outlined in Chapter 2. First is a threshold signature protocol that can increase its threshold after the cryptosystem has been created [4]. To be more accurate, that threshold cryptosystem requires the participation of all peers who hold secret shares, but the number of distinct secret shares can be increased at will. The issue with this system is that the threshold of the cryptosystem must be equal to the number of distinct secret shares. In their system this threshold is also equal to the number of peer devices because each device is only permitted to hold a single secret share. Furthermore, they provide no means of reversing the process, allowing for better availability at the cost of security. In contrast, our system tries to dynamically adjust the tradeoff between security and availability.

Next is a means of creating a CA based around a local cluster which is then able to sign and trust another cluster's CA [5]. This system is effectively the Web of Trust model [7] with each discrete vertex on the graph being replaced by a local cluster of devices. The local clusters generate basic TC signature schemes to create their CA. This method is similar to how our system takes into account the locality of the devices, but they consciously leave out the means by

which the devices choose who gets to hold the secret shares of the CA. Our system is intimately concerned with how to make this choice, as will be outlined below.

Lastly is a secure, distributed file system for ad-hoc networks [6]. Unlike the previous two papers, this deals explicitly with the security of data in the face of device loss or capture. Their system, called Mobile Distributed File System or MDFS, takes the data to be protected and encrypts, fragments, and then distributes it among the local devices. Each fragment is a piece of the ciphertext combined with a corresponding secret share. While the system does present some interesting ideas, they completely ignore the aspect of reliability in potentially volatile networks. Our system would try to ensure that those fragments were available to devices that need them.

Chapter 2

The Solution: Ad Hoc Security

2.1 Design Principles

As mentioned above, the goal is to only allow authenticated users to access secrets, while denying secret access to all others while in an ad-hoc network. This would normally require the devices to have some knowledge of their current user's state. While there are technologies that do this directly, such as biometrics and security cards, they are out of the scope of this research. The goal here is to be able to automatically verify the authenticity of a device with cryptography while only requiring user interaction for fringe cases. Given the assumption of frequent disconnection from any infrastructure, the receiver groups must have some way of verifying authenticity among themselves. Lone receivers would have to rely on the technologies mentioned above (biometrics, cards) along with whatever other external information they could gather in order to verify their user. The assumption that devices will form groups allows for more creative solutions.

2.2 Ad Hoc Security Overview

The terminology used in this overview will be explained in detail in the sections below.

1. Use broadcast heartbeat messages to continuously push back the time when a device will wipe its own memory of all secrets (including decryption materials).
2. Have the devices continuously sign each other's identifying certificates using threshold cryptography with short time validity signatures. A certificate with no signature or an expired signature requires other users to manually tell their devices to trust the unsigned certificate. Each device holds a fixed number of signature functions (normally 1, unless under special circumstances).

3. Use threshold cryptography to decrypt the secrets. Each secret is encrypted with a different threshold cryptosystem. Each device can hold a variable number of decryption functions. The number held is based on the current Redundancy Factor of that cryptosystem.

2.3 Heartbeat and Trusted Peers

Ensuring that only the intended users are granted access to secrets starts by ensuring that adversaries are denied access to secrets. We use heartbeat messages as a kind of dead-man switch. These messages contain the current time as well as a cryptographic signature to prevent spoofing or replay attack. If a device goes for too long without receiving one or more heartbeats from trusted peers, it will empty its memory of all secrets. This is because a normal user will be within communication range of at least a few peers, and an abnormal user will likely try to take the device to another location. If the device remains in communication range of its peers, then it is the responsibility of the other users to proactively tell that device to wipe its memory of any secrets. Thus, adversaries are put into a no-win situation: whether they leave or stay, the secrets should be wiped. However, all of this relies on knowledge of which peers to trust.

2.4 Trusted Peers and a Mobile Certificate Authority

In an infrastructure network system, knowing which peers to trust usually involves an implicitly trusted certificate authority (CA) signing the identifying certificate of everybody involved. It is assumed that the reader is familiar with the basic function of a CA, as well as the properties of a cryptographic signature. This method of verification could be used as an absolute baseline of trust. However, as mentioned in the problem statement, there may be situations where the devices are disconnected from all infrastructure-bound resources and need to modify a peer's trust status. This is where a Mobile Certificate Authority (MOCA) system can be used.

A MOCA is a kind of flexible, distributed CA. This will be a brief description, see [2] for details. At its core, MOCA is threshold cryptography applied to cryptographic signatures. Threshold cryptography will be explained in detail in the Threshold Cryptography section below. In short, so long as a device can communicate with at least a threshold number of designated MOCA signing peers, it can have its identifying certificate signed by the MOCA. This signature works the same as any other CA signature, just the means of acquiring it is different. In [2] the authors only designate a subset of the devices as MOCA signers. However, we opted to make every device a MOCA signer, mostly for simplicity. The distributed nature of MOCA also enhances security. As will be explained in the Threshold Cryptography section, the MOCA system can only be fully compromised if a threshold number of participating devices are compromised. This fits with the last goal in the problem statement above: it should be resistant to small numbers of device compromises.

The primary change to the MOCA system used here is the duration of a signature's validity. To ensure that up-to-date authentication is maintained, the signatures must be valid for reasonably short amounts of time. The signatures are normally renewed automatically, but if it is unable to do so before the current signature expires then the users must manually authenticate the device. This continuous "check-in" kind of scheme is similar to the heartbeat system. It forces compromised devices to eventually become untrusted, whether automatically or manually. Furthermore, recall that the heartbeat system only recognizes heartbeats from trusted peers. This forms another layer of security: if a device is untrusted, then it cannot push back its wipe time even if it is within range of other devices.

2.5 Secret Encryption

A trusted device can receive secrets from either the central server or its peers. But, as mentioned in the problem statement, the secrets must be protected from unauthorized users. This means that the secrets should at least be encrypted. However, decrypting the secrets when they are needed poses a challenge. Traditional single key decryption schemes would make each device a single point of failure. A distributed decryption system is then needed. In a similar fashion as the authentication system, distributing the decryption power among multiple devices can protect the encrypted secret from single point compromises. Like with MOCA, we can make use of threshold cryptography for this task, with a few twists to enhance reliability.

2.6 Threshold Cryptography in Ad Hoc Networks

Threshold cryptography (TC) is a type of multi key cryptography that only requires the participation of a threshold number of peers out of the total number (t out of n) to complete a cryptographic function [1]. The term "participation" refers to the use of a secret function, sometimes called a secret share or shadow function, on the message to be signed or decrypted. We will use the term shadow function from here on. As seen with MOCA, TC provides a cryptographically verifiable and highly flexible way to enforce the number of peer contributions required, instead of relying on some kind of counter value saved in the software, or a multi-layered encryption.

The main difficulty of designing a system for ad-hoc networks is the connection volatility and unreliability. Devices may become connected or disconnected at random, and there is no guarantee that a specific peer will be available when needed. Traditional TC is synchronous [1], meaning that participating peers must be present throughout the entire process to be successful. While TC is better than other options, synchronous TC is far from ideal. In this system,

asynchronous TC is a much better option. Asynchronous TC allows for participation at any time by producing "decryption shares" [3] which are single messages that encapsulate a peer's entire contribution to the decryption. These decryption shares will be referred to as "partial decryptions" from here on, because the term is more true to its use. These partial decryptions are later verified and combined to produce the decryption or signature. This property allows for much needed flexibility on when the decryption can take place, because devices can simply store partial decryptions for later use.

However, if a device cannot contact at least a threshold number of peers, then the decryption or signature will always fail. One solution to this would be to reduce the threshold of the cryptosystem. This incurs obvious security detriments and does nothing to address the inflexibility of the threshold value. It would be better for the solution to be adaptable to the network's current state. This is where Redundancy Factor comes into play.

2.7 Threshold Cryptography Reliability Enhancements

Redundancy Factor is essentially a measure of confidence based on the status of nearby peers. It is the confidence that a threshold cryptosystem's functionality will be available by the time it is needed. There are two parts to Redundancy Factor: one for individual shadow functions and one for whole cryptosystems. Their formulations are given in Equations 1 and 2.

Equation 1: RF Individual, for individual shadow functions.

$$RFI(i, N, D, F, t, X, x, T) = \sum_{n \in N(i,D)} \frac{\min\left(1, \frac{T}{C(n, X)}\right) P(n, x)}{F(d(i, n), t)}$$

Equation 2: RF Cryptosystem, for entire threshold cryptosystems.

$$RFC(i, N, D, F, t, X, T) = \sum_{x \in X} \frac{\min(1, RFI(i, N, D, F, t, X, x, T))}{T}$$

Terms used in Equations 1 and 2:

i: our particular node in N

N: the set of all nodes in the network

N(i,D): set of nodes no further than D network hops from node i, includes node i at d = 0

x: individual shadow function in X

X: set of all shadow functions in the threshold cryptosystem

C(n,X): the number of shadow functions that node n has from cryptosystem X

D: maximum distance considered, also maximum distance in F

d(i,n): distance from i to n

T: threshold number of unique shadow functions needed for this cryptosystem

F(d,t): falloff factor matrix for distance (in terms of network hops) and time, see Table 1.

P(n,x): presence of shadow function x in node n, returns true (1) or false (0)

t: time until the secret is no longer needed[3]

Time Before Secret End	Falloff Factor for d = 0	Falloff Factor for d = 1	Falloff Factor for d = 2
t > 160 minutes	1.0	1.0	2.0
160 minutes > t > 140 minutes	1.0	2.0	4.0
140 minutes > t > 60 minutes	1.0	4.0	8.0
60 minutes > t > 0 minutes	1.0	1.0	2.0

Table 1. Example falloff factor matrix.

[3] The falloff factor's time parameter is in reference to the time when the secret is no longer needed, called the secret's "end time," as opposed to when the secret is first needed, called the secret's "start time." This is because nodes may still request and load the secret after its start time. The falloff factor matrix must be aware of the secret duration so that it can offset its times accordingly.

A falloff factor matrix is a way of defining what level of redundancy is acceptable, given the amount of time before needing the cryptosystem. To be more precise, it says how many copies of a shadow function are needed at certain distance to feel confident that this particular shadow function will be available when it is needed. While in this implementation the falloff factor only considers distance and time, its position should be seen as a more general scaling factor given the state of the device in question. In Table 1, the falloff factor increases as the secret's start time approaches, and then decreases once the start time has passed. The decrease at the end is not used in the simulations (because they will almost never encounter such idyllic scenarios) but is shown here as an example of a more general form that the matrix might take. This kind of falloff factor assumes that the vast majority of devices will load the secret on time, so maintaining the elevated redundancy afterwards would be unnecessary. The falloff factor matrix is what defines the devices' behavior towards security and availability, and should be the factor that is changed first when tuning the system.

In Equation 1, all that is happening is that our node is looking at its network neighbors and checking if they have a certain shadow function. If they do, then it considers how far away that neighbor is as well as how long until the function is needed. If a neighbor is further away and the deadline is very soon, then the falloff factor will probably be much higher, resulting in a lower confidence contribution. In Equation 2 we are asking whether we are confident in our ability to access at least a threshold number of functions, given our individual confidence in each of those functions. The minimum acceptable confidence level for both equations is 1. These two equations form the backbone of the solution, but they have some context-dependent issues that require explanation.

The minimum function in Equation 1 is to limit the contribution by nodes that hold a large number of shadow functions. Specifically, the nodes affected are those that have at least a threshold number of functions. With at least a threshold number of functions and a sufficiently small falloff factor, a single peer node can produce a RFC that is unreasonably high compared to a qualitative analysis. Multiplying all of its contributions by the threshold and then dividing by the number of functions held results in a maximum total contribution towards the RFC of 1 divided by its falloff factor. Thus, falloff factor dictates not only the minimum number of function copies required to be confident, but also the minimum number of peers as well. The minimum function in Equation 2 is to limit the contribution of a single function. Because the goal is to maintain the availability of a threshold number of different functions, it is necessary to put a cap on the contribution of each individual function. This does result in the loss of some information, but there would be little benefit gained from extending the confidence range beyond whatever is deemed acceptable.

Chapter 3

Implementation Details

3.1 Overview

The solution outlined above is the skeleton of the full implementation. Here we will flesh out the important implementation details. The actual implementation was written in C++ for the network simulator called ns3. We will skip over the minutiae of the actual statements in favor of a more pseudo-code type of approach. These details are important to the success of the application, as will be explained in their respective sections. There are six "major behavior loops" present in this implementation:

1. Heartbeat. Regularly sending heartbeat messages to prevent memory wiping.
2. Load Current Secret. Attempting to load the currently applicable secret.
3. Check Secret Buffer. Ensuring the secret buffer is filled to capacity.
4. Redundancy Check. Checking RFC and requesting more shadow functions if needed.
5. Prepare Next Secret. Gathering partial decryptions for decrypting the next secret.
6. Maintain Authentication. Ensure the device has a signature that is trusted by others.

3.2 Behavior Parameters

Every device in the ns3 simulation has a variety of behavior parameters. These parameters are modified in the test cases to compare performance differences. Below is a list of the parameters and their purpose.

1. server: Whether or not this device is acting like a key server. If it is then certain functions are disabled, such as heartbeat broadcasting and memory wiping. This should only be present for the simulation.

2. signing_cert: The unique identifying certificate of the device. This is the cert that is signed by the MOCA.
3. signing_key: The key associated with the signing cert.
4. heartbeat_cert: The cert for the threshold cryptosystem that dictates the number of unique heartbeats required to push back the wipe time.
5. heartbeat_keys: The TC shadow functions used for the heartbeat cryptosystem. Each device typically has only one function.
6. moca_cert: The cert that identifies the MOCA cryptosystem.
7. moca_keys: The shadow functions used for the MOCA. Each device typically has only one function. The number of MOCA functions is not checked or modified by any kind of redundancy factor calculation.
8. server_ips: The list of IP addresses that identify key servers.
9. falloffs: The list of falloff factor objects. Each falloff factor object lists the falloff values for different distances with respect to its single time offset value. In Table 1, every row would be a falloff factor object.
10. olsr_routing_protocol: A pointer to the routing protocol this device uses. This is necessary as several functions use the routing table for calculating distance.
11. broadcast_port: The port that the device listens on.
12. maxNumHops: The maximum number of network hops that broadcasts will travel over. This applies to both heartbeats and crypto statuses.
13. heartbeat_delay: The delay between sending heartbeat messages.
14. heartbeat_power: The amount of time (past the time of sending) that a heartbeat will push back the current wipe time.

15. auth_delay: The delay between sending MOCA signature requests, if it is time to start re-authenticating.
16. auth_start: The amount of time before the current MOCA signature expires that the device will begin sending MOCA signature requests.
17. auth_power: The amount of time that the requested MOCA signature will be valid for. Note: In this implementation MOCA signatures are only valid for auth_power duration time blocks that are the same for every device, instead of adding auth_power to the current time. This was done purely for simplicity of implementation.
18. status_delay: The delay between sending crypto status messages.
19. status_power: The amount of time that a crypto status message is considered valid.
20. secret_interval: The useful duration of each secret. Note: Every secret object also contains their start and end times, but this is just a simple way to get that information.
21. secret_buf: The length of the buffer of secrets. This buffer includes the currently loaded secret, so there are secret_buf - 1 extra secrets queued up.
22. load_secret_delay_short: A short delay between checking whether the currently applicable secret can be loaded (i.e. is able to be decrypted locally). Earlier in development there was a load_secret_delay_long, but it was found to be unnecessary when the next check can be scheduled for the current secret's end time.
23. check_buf_delay_short/long: The short and long delays between checking if the device's secret buffer is filled to capacity. The short delay is for when the buffer is not full, and the long delay is for when the buffer is currently full. We cannot simply schedule the next check around the current secret's end time as a memory wipe may occur at any point.

24. prep_delay_short/long: The short and long delays between checking on the status of the next secret to be loaded. The short delay is used when there is still work to do, and the long delay is for when there is nothing for the loop to do for a long time.
25. prep_start/finish_decrypt: The start and finish time offsets for secret loading. They are both in terms of time before the secret is first needed. The purpose of these values will be described in detail later.
26. check_delay: The delay between checking the redundancy factor of each secret's encryption cryptosystem.

3.3 Heartbeat Loop and Memory Wipe

The heartbeat system exists as a way of automatically wiping secret information if a device strays from the group for too long. Every device broadcasts heartbeats at regular intervals defined by heartbeat_delay. The broadcast distance is reduced to be no more than maxNumHops. Each heartbeat contains a timestamp that states the new memory wipe time requested by the device. If a device receives a heartbeat request from a peer that they trust, then they will send back that heartbeat message with a cryptographic signature and a flag indicating that it is a response. In this implementation, the heartbeats also make use of a threshold cryptosystem to allow for the requirement that multiple peers approve the device's heartbeats. However, this was added for flexibility and generality and is not a strict requirement.

The memory wipe event is a ns3 event that is deleted and recreated every time enough heartbeats responses are received. Its time is pushed back based on the timestamp in the last valid heartbeat received (or heartbeats, if more than one are necessary). The memory wipe event removes all buffered secrets and decryption shadow functions from memory. It does not remove authentication information such as certificates or MOCA shadow functions, nor does it remove

the currently loaded secret. It is assumed that a loaded secret is either stored in a secure enclave or already consumed by the user or the device. The presence of a secret in the "loaded secret" memory location is more of a way of keeping track of whether or not one is loaded for the current time slot.

3.4 Load Current Secret and Check Secret Buffer

The secret loading system is what ultimately loads the currently applicable secret. It is effectively a polling loop that checks the device's secret buffer for the currently applicable secret. If that secret is unencrypted or can be decrypted using locally available functions and partial decryptions, then it decrypts and loads. It then schedules the next run for the end time of the current secret. If the currently applicable secret is not loaded and cannot be loaded, then it waits a short amount of time and tries again.

The secret buffer checking system is what ensures that the device's secret buffer is filled to maximum capacity. It will keep the buffer filled with the secrets that are applicable for the current time block, the next time block, etc. If any secret that should be buffered is missing, it will attempt to retrieve it from the key server. If the key server is not in the routing table, then it will broadcast the request to its 1-hop neighbors. Because every device will be attempting to get the same secrets, it is unnecessary to extend the request beyond immediate neighbors. The secret buffer checker will check quickly again if it is missing any secret, and will check after a longer duration if the buffer is full.

3.5 Crypto Status and Redundancy Check

A crypto status message lists a device's address, decryption shadow functions, and MOCA shadow functions. Each device broadcasts this message at a regular interval to all neighbors within its max number of hops range, and it serves as the basis for redundancy check

calculations. A crypto status message will expire after a set amount of time, or will be replaced if the device receives a new status before the previous status expires.

A redundancy check is when a device looks through its crypto status messages, calculates the RFC, and then makes requests to nearby peers for additional shadow functions if necessary. A redundancy check loop is running for every encrypted secret in the buffer, and is started as soon as the secret is first received. The redundancy check loop continues as long as the secret's end time has not yet passed. The control flow of a redundancy check is as follows:

1. Get the falloff factor with the time that is closest to, but still less than, the current time, defaulting to the falloff factor with the earliest time.
2. If our device does not have any locally saved shadow functions for this secret that have not already been applied (in the form of partial decryptions), then add a request for one unapplied shadow function from our peers.
3. Calculate RFC for this cryptosystem.
4. If the RFC is between 0.5 and 1, then add requests for $1/16^{th}$ of a threshold number of shadow functions with the lowest RFI.
5. Else if the RFC is between 0.25 and 0.5, then add requests for $1/8^{th}$ of a threshold number of shadow functions with the lowest RFI.
6. Else if the RFC is lower than 0.25, then add requests for $1/4^{th}$ of a threshold number of shadow functions with the lowest RFI.
7. Send the shadow function requests.

One important detail is in how the lowest redundancy shadow functions are chosen. In an earlier version, it would simply request the functions with the lowest redundancy and the lowest identifier number. This lead to two issues. First, the lowest redundancy functions are inevitably

the ones with RFIs of 0. It is useless to attempt to request these functions, as a RFI of 0 indicates that it is completely unavailable. The solution was to only request the lowest non-zero RFI functions. Second, selecting the lowest redundancy and lowest identifier function caused every device in a group to mostly request the same functions. Then there would not be enough variety to reach the decryption threshold, despite every device locally storing more functions that necessary. The solution was to apply some controlled randomness to the selection process. To select the lowest RFI functions, first the functions are grouped by RFI value. Starting from the lowest RFI value group, the device would randomly pick functions to request, only moving to a higher RFI value group once the current one is empty. This lead to devices choosing a wider variety of functions, thus reaching the required threshold with less unnecessary duplication of functions.

Another issue encountered was in the treatment of the key server's crypto status messages. Since the key server has all the shadow functions for every cryptosystem, it could easily saturate the RFC of nearby devices. This could lead to those devices choosing to not request any functions before becoming disconnected, and then being unable to decrypt the applicable secret. This issue was corrected by applying a special case to the key server where it adds a negligible, but non-zero, value to each function's RFI instead of the normal 1/falloff value. This problem later lead to the realization that there should be per-device cutoffs for RFC contributions, culminating in the inclusion of the min function in Equation 1.

3.6 Prepare Next Secret

The "Prepare Next Secret" loop is what prepares the next secret for decryption. It accomplishes this through the limited gathering of partial decryptions. Recall that a partial

decryption is the culmination of a single shadow function's contribution towards a decryption or signature. The basic control flow is as follows:

1. If the currently applicable secret is not loaded, try to find the current secret.
2. Else find the secret for the next time slot.
3. If we found a secret from either of the above searches, check the current time.
    a. If it is time to finish decryption, then request partial decryptions until the secret can be decrypted locally or until it requires only one more shadow function contribution. The partial decryptions requested will never be for a function that is locally saved[4].
    b. Else if it is time to start decryption, then request partial decryptions until either the secret can be decrypted locally or half of a threshold number of partial decryptions have been acquired.
4. Else, wait for either a long or short time, depending on whether we are ahead or behind on decrypting the next secret.

The goal was to limit the risks posed by possession of decryption material (shadow functions and partial decryptions). Because each device must eventually save enough decryption material to decrypt the secret, the only way to limit the risk is by controlling the amount of time that the material is in memory. If every device has some probability of compromise for every point in time, then lowering the time that the device holds sensitive information will lower the overall probability of that information being compromised. Of course, this method does come with reliability detriments, as will be explored in Chapter 6, Results and Analysis.

[4] As stated in Crypto Status and Redundancy Check, the device will always try to keep at least one unapplied shadow function locally saved.

It should be noted that while Prepare Next Secret and Check Redundancy may seem to do the same thing, they actually have different goals. Prepare Next Secret works for the benefit of only the local device. Partial decryptions are not gathered with the intent of distributing to others. Check Redundancy works for the benefit of the group. It works to gather shadow functions that are exceptionally rare not only so that the local device can have access to them, but also so that its peers can access them as well.

3.7 Maintain Authentication

The "Maintain Authentication" loop is what ensures the device has a valid MOCA signature on its identifying certificate. The MOCA signature request method and timing is simpler than the Heartbeat or Redundancy methods. This is due to the fact that MOCA signatures were implemented to expire at the end of pre-defined time blocks. For example, the default MOCA signature duration (auth_power) is one hour, so the device would need a new signature that expires at hours 1, 2, 3, etc. It starts requesting new partial signatures from its peers at auth_start amount of time before the start of the next time block. The signature starts being valid from the time it is completed, regardless of the time that it expires. This is because only the expiration date is recorded in the signature. If it is unable to acquire a new MOCA signature before the next time block starts, then it waits until it is time to start requesting for the next time block. This leads to "chunks" of "unauthorized time" that are usually about auth_power – auth_start in size, or 30 minutes by default. This method was used for simplicity of implementation. The device uses the list of crypto status messages as a way of deciding which peer to send the next partial signature request to. Recall that crypto status messages also contain the sending device's list of MOCA functions, allowing the requesting device to ensure it is not requesting from the same device more than once.

Chapter 4

Testing Methodology

4.1 Overview

The testing methodology used was designed around confirming basic functionality and finding failure points. Comparing the performance between different sets of behavior parameters was a secondary goal. Since this is the first complete version of the system, it would not make sense to fine tune the behavior parameters when it is expected that the system behavior may change drastically. There are two parts to each test: the major and minor version. The major version (denoted by the first three numbers in the simulation filename) defines the macro-scale movements, timings, and sizes of device groups. There are 12 major versions, with many of them being slight variations of others. The minor version (denoted by the second three numbers in the simulation filename) defines the behavior parameter that is changed relative to the baseline. There are 11 minor versions, with the first being the baseline and every consecutive pair afterwards moving a single parameter in opposite directions. This means that every simulation file has a name like "sim.004.003.json" which means it is testing major version 004 with minor version 003. Here we will refer to tests by their major and minor versions. For brevity, a full scenario may be referred to as simply "version 004.003" for major version 004 with minor version 003. If a third trio of numbers is present, then this refers to a specific run of that particular major/minor version pair. For example, 005.000.001 would refer to major version 005 with minor version 000 and run number 001. As will be mentioned later, the tests are run multiple times to find inconsistencies due to randomness.

One final note on the testing methodology: the MOCA/authorization system is not tested heavily or put under much scrutiny. It was originally going to serve as a more static point of

comparison against the more dynamic decryption system. However, the authentication system is so static that measuring its effectiveness is trivial. It will be successful if, given a certain time window, the device in question is able to communicate with a threshold number of different trusted peers. If that time window is extended, or the density of the network increased, or the threshold of the cryptosystem reduced, then the system's likelihood of success will increase. The effect that density has on reliability is seen in the MOCA paper's success ratio graphs [2]. The only simulation to see authentication failures was the final simulation (major version 011) with its highly mobile, widely spread out group of devices. Even then, failing to get a MOCA signature on time is exceedingly rare. Furthermore, it would not be fair to compare a static system against a dynamic system. In major version 007, three devices split off from a group and become disconnected. With a single decryption function per device and a decryption threshold of eight, these three devices would be guaranteed to never be able to decrypt any of the secrets within their buffer, unless they request partial decryptions during their connected period. With these issues in mind, the authentication system's performance will be omitted from the analysis.

4.2 Device Groups and Movement

The way we defined movement was heavily influenced by the assumption that devices would form groups which would, for the most part, move together. The devices are generated by defining groups in the simulation file. Each group has a size, which says how many devices are in it. When the simulation file is parsed, this will tell the simulation program to generate that many devices and to apply the same movement timings to them. Each group also has a dispersion value, which says how randomly spread out they can be. This dispersion value is actually the side length of a square. This square is the area that the devices can move within. More accurately, it is the range of values that a random number generator can produce. The

random number generator is used to offset the device's position from the position of the bottom left corner of the box. The three arrays named "X," "Y," and "T" define the position of the bottom left corner of the box at a certain time. The position values generated for each device become waypoints that they move between. Note that a new waypoint is only calculated for every X,Y,T triple. If there is a single triple, then only a single waypoint will be calculated, and the devices will not move from that position. This is why, for stationary boxes, we repeat the same X,Y value for consecutive times. It will cause the devices to randomly move within their box, which is more in line with expected behavior. The key server is always positioned at point (0,0). The communication range of all devices is approximately 120 units. A distance of 150 units is often used to enforce disconnection.

4.3 Major Version List

The list of major versions is as follows:

000. Fully connected device network within range of the key server.

001. Partially connected device network within range of the key server.

002. Fully connected device network that starts within range of the server. It later disconnects from the server for a time less than its internally buffered secret time. It then reconnects to the server.

003. Same as 002, but the device network is partially disconnected.

004. Same as 002, but the device network is disconnected for longer than its internally buffered time.

005. Same as 004, but the device network is partially disconnected.

006. A group of devices just large enough to self-authenticate becomes disconnected from the rest of the device network and key server for a time less than its internally buffered secret time. Afterwards they reconnect to the main group and server.

007. Same as 006, but the moving group stays disconnected for longer than their internally buffered secret time.

008. One device group is connected to the server. Another device group is directly connected to the first group but not directly connected to the server, i.e. a straight line of two device groups.

009. Same as 008, but it is a straight line of 3 device groups.

010. One device group is disconnected from the server. Another device group "ferries" data between the server and the disconnected group every other hour.

011. One large node group that is very spread out but can still be within range of the server.

4.4 Minor Version List

For the minor versions, only the baseline will list every parameter. Every subsequent version will only list the parameter that is changed. The list of minor versions is as follows:

000. Baseline.

- 6 secrets with 2 hour durations for a total of 12 simulated hours.
- Heartbeat threshold of 2 (not including self-signature).
- MOCA threshold of 3 (including 1 self-signature).
- Secret decryption threshold is 8 out of a total of 16 shadow functions.
- Falloff factor: See Table 2.
- maxNumHops = 2.

- heartbeat_delay = 1 minute.
- heartbeat_power = 120 minutes.
- auth_delay = 1 minute.
- auth_start = 30 minutes.
- auth_power = 60 minutes.
- status_delay = 1 minute.
- status_power = 10 minutes.
- secret_buf = 3, 1 loaded and 2 buffered.
- load_secret_delay_short = 5 minutes.
- check_buf_delay_short = 2 minutes.
- check_buf_delay_long = 10 minutes.
- prep_delay_short = 1 minute.
- prep_delay_long = 10 minutes.
- prep_start_decrypt = 90 minutes.
- prep_finish_decrypt = 30 minutes.

001. Lower secret encryption threshold:

- Threshold = 4, Total = 16.

002. Higher secret encryption threshold:

- Threshold = 12, Total = 16.

003. More aggressive falloff factor: See Table 3.

004. Less aggressive falloff factor: See Table 4.

005. Longer secret time, smaller secret buffer:

- 3 secrets with 4 hour durations.

- secret_buf = 2 (one loaded and one buffered).
- 120 minutes added to all falloff factor times.

006. Shorter secret time, larger secret buffer:

- 12 secrets with 1 hour durations.
- secret_buf = 5 (one loaded and four buffered).
- 60 minutes subtracted from all falloff factor times.

007. Earlier start and finish decrypt times:

- prep_start_decrypt = 180 minutes.
- prep_finish_decrypt = 60 minutes.

008. Later start and finish decrypt times.

- prep_start_decrypt = 45 minutes.
- prep_finish_decrypt = 15 minutes.

009. Larger number of hops for heartbeat, crypto status, and falloff.

- maxNumHops = 4.

010. Smaller number of hops for heartbeat, crypto status, and falloff.

- maxNumHops = 1.

Time Before Secret End	Falloff Factor for d = 0	Falloff Factor for d = 1	Falloff Factor for d = 2
t > 160 minutes	1.0	1.0	2.0
160 minutes > t > 140 minutes	1.0	2.0	4.0
140 minutes > t > 0 minutes	1.0	4.0	8.0

Table 2. Default falloff factor matrix.

Time Before Secret End	Falloff Factor for d = 0	Falloff Factor for d = 1	Falloff Factor for d = 2
t > 160 minutes	1.0	2.0	4.0
160 minutes > t > 140 minutes	1.0	4.0	8.0
140 minutes > t > 0 minutes	1.0	8.0	16.0

Table 3. More aggressive falloff factor matrix.

Time Before Secret End	Falloff Factor for d = 0	Falloff Factor for d = 1	Falloff Factor for d = 2
t > 160 minutes	1.0	1.0	1.0
160 minutes > t > 140 minutes	1.0	1.0	2.0
140 minutes > t > 0 minutes	1.0	2.0	4.0

Table 4. Less aggressive falloff factor matrix.

Chapter 5

Data Collection and Measured Metrics

5.1 Data Collection Overview

The data collection method is a log file that records the events that occur on each device in the simulation. Each event contains a timestamp relative to the simulation time, a device ID (which is just the memory location of the ns3::Application object), and an identifier string that states which kind of event is occurring, like loading a secret or calculating the RFI/RFC. Most events contain extra information after this, such as which secret is being loaded or the value of the calculated RFI/RFC. These event statements are later parsed by a separate program to produce four metrics: downtime, key time, local decrypt time, and unauthorized time. As unauthorized time was not tested, it will be omitted from most of the analysis. These metrics are calculated for each device, giving insight into how the movement and parameters affect the different groups. Each test is run multiple times, because of the randomness in the movement and behavior. Most of the tests are run four times, but some of the tests with more varied results are run eight times. The three metrics are then averaged across each device and test run before finally being compared to the other tests in the graphs that will be shown Chapter 6, Results and Analysis.

5.2 Downtime

Downtime is the sum of the amount of time that a device does not have a currently applicable secret loaded. For example, say there are two secrets with start times of 0s and 7200s. The device loads them at 600s and 7300s, respectively. The device's total downtime thus far is 700s. Note the loss of information that comes with the aggregation of those two values. Also, at the beginning of every simulation the devices are completely empty. This means that there will

always be some amount of downtime due to startup. However, this starting downtime is usually no more than 900s, or 15 minutes. Generally speaking, downtime is the most important metric. A system that is secure yet provides no benefit to its users is effectively worthless. Downtime is analogous to availability: being unable to decrypt a secret leads to downtime, which means the decryption system was unavailable.

5.3 Key Time

Key time is the integral with respect to time of the number of shadow functions a device has locally saved. Shadow functions are called keys in the implementation because they effectively do the same job as a normal cryptographic key. Because of key time's formulation, it has a somewhat strange unit of "key seconds" or key*s. This unit is actually quite useful. Recall the way that a threshold cryptosystem can be compromised. If a threshold number of its shadow functions are viewed by an adversary, then the system's security is lost. In this situation, every device that is lost could have its memory contents, including saved shadow functions, exposed to an enemy. It is similar to the situation of carrying a briefcase filled with money. More money in the briefcase means more financial loss if it is stolen. Carrying the briefcase for a longer time means there is a greater chance of encountering someone who will try to steal it. Thus, minimizing the number of shadow functions and/or the amount of time that those functions are in memory will help reduce the risk of full cryptosystem compromise. Key time is half of the equation for determining security. A decryption system can only be compromised by way of devices exposing the shadow functions in their memories.

5.4 Local Decrypt Time

Local decrypt time (LDT) is the integral with respect to time of the number of secrets a device can decrypt using only locally available materials (i.e. shadow functions and partial

decryptions). LDT is similar to key time in that its unit is “secret seconds” or secret*s. This is because a secret will likely be loaded (and thus implicitly locally decrypt-able) while accumulating partial decryptions and shadow functions for the next secret or two. Thus there is expected to be overlap between one or more secrets that are locally decrypt-able. The LDT presented here is a worst case scenario. It is assumed that the device will not remove partial decryptions from memory once a secret is loaded. In the best case scenario, the device would scrub its memory of all partial decryptions once the secret is loaded, since they will not use partial decryptions to assist peers in decryption. Calculating the LDT of the best case scenario would require making the distinction between being able to decrypt a secret using partial decryptions, and being able to decrypt a secret using only shadow functions. Currently, the logging system does not make this distinction. However, in Results and Analysis we will examine LDT in greater detail to show its relationship with the Redundancy Factor system. Local decrypt time is the other half of the security equation. Depending on how secrets are loaded and how partial decryptions are handled, having a larger LDT could indicate increased risk of secret compromise.

5.5 Unauthorized Time

Unauthorized time is the sum of time that a device does not have a certificate with a valid MOCA signature. This metric is not tested as heavily as the previous three, but it does allow for some knowledge of how volatile the ad-hoc network is, given that it uses static shadow function allocation. Major version 011 is the only test that had appreciable unauthorized time, which lead to some devices wiping their own memories.

Chapter 6

Results and Analysis

6.1 Overview

The results presented below are a selected subset of all the data gathered from the simulations. The results of many of the simulations are either too similar or provide no insight into the successes and failures of the Ad Hoc Security system. Furthermore, the averages presented in the graphs should be viewed as very coarse comparisons of performance. This is because the averages are over every device in a simulation run. Some of the device groups have wildly different connectivity experiences, and thus the average cannot accurately represent the overall performance. The real insight comes from examining the per-device performance and events. As such the graphs will be used as ways of comparing basic performance trends across minor versions, while the log files and per-device aggregates will be used to see real performance as well as failure points.

6.2 Illustration of Data

The performance data will be presented using three different bar graphs. The graphs are for downtime delta, key time delta, and local decrypt time delta. These three graphs illustrate the difference in performance that their minor version causes, relative to minor version 000. The value above minor version 000 in these graphs is its raw performance number. This value was included to provide a sense of scale to the compared values.

6.3 Perfectly Reliable System

First, we need a point of comparison. If this system were tuned for absolute reliability, then we can calculate the key time and local decrypt time and compare them to the experimental results. We will not need to consider the heartbeat or authentication systems, as we can assume

that the perfectly reliable system will simply lower the thresholds and/or increase the moca/heartbeat_power parameter to render both systems inconsequential.

To ensure a fair comparison, the perfectly reliable system will behave the same as the normal system. The only difference being that it has parameters and internal data tuned for perfect reliability. This means:

1. Every device will hold every secret needed for the duration of the simulation.
2. Every device will hold a threshold number shadow functions for every secret.
3. The devices will only delete secrets and shadow functions once the secret has expired.

The perfectly reliable system could delete the shadow functions after loading the secret. However, the normal system does not do this as it may need to assist a peer with decryption after the secret's start time. As such, for fairness, this behavior will be left unchanged.

Because every encrypted secret has a corresponding threshold number of shadow functions to decrypt it, every secret is able to be locally decrypted from the start time. Thus, calculating the local decrypt time means summing up the total amount of time that each secret is in memory. So, for N secrets with duration x, the local decrypt time is given by Equation 3. The key time is related to the local decrypt time in that a threshold number of shadow functions are saved for each secret. Thus, for N secrets with duration x and decryption threshold T, the key time is given by Equation 4. Calculating LDT_P and KT_P for the baseline secret time, longer secret time, shorter secret time, baseline threshold, higher threshold, and lower threshold results in the values listed below.

- LDT_P(6, 7200s) = 151,200 secret*s
- LDT_P(3, 14400s) = 86,400 secret*s
- LDT_P(12, 3600s) = 280,800 secret*s

- $KT_P(6, 7200s, 8) = 1{,}209{,}600$ key*s
- $KT_P(6, 7200s, 12) = 1{,}814{,}400$ key*s
- $KT_P(6, 7200s, 4) = 604{,}800$ key*s
- $KT_P(3, 14400s, 8) = 691{,}200$ key*s
- $KT_P(12, 3600s, 8) = 2{,}246{,}400$ key*s

Equation 3: Local Decrypt Time for the perfectly reliable system given N secrets with duration x

$$LDT_P(N, x) = \sum_{n=1}^{N} nx = \frac{xN(N+1)}{2}$$

Equation 4: Key Time for the perfectly reliable system given N secrets with duration x and decryption threshold T

$$KT_P(N, x, T) = T * LDT_P(N, x)$$

6.4 Major Version 004: Basic Disconnection and Reconnection

6.4.1 Overview

Major version 004 is a basic test where a single device group moves away from the key server and then comes back. This test has the group disconnected for longer than their internal secret buffers can handle. As such, there will be a guaranteed amount of downtime across every device. The movement of the group is given by Table 5. Recall that the key server is at (0,0), every device's communication range is approximately 120 units, and the positions defined in the table are for the bottom left corner of the dispersion box. The dispersion value here is 100, meaning every device will move within a 100 by 100 square, so there is very little chance of a

device becoming disconnected from the rest of the group. The results of the tests on major version 004 are in Figures 1, 2, and 3.

Group 1 Size: 16 Disp: 100	X	0	0	50	150	150	150	150	150	150	150	50	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12

Table 5. Major Version 004 movement definition.

6.4.2 Analysis of Results

To start, the 7,000s downtime of the baseline is mostly due to the devices running out of buffered secrets while disconnected from the key server. Normally, about 900s (15 minutes) of downtime is incurred at startup. The devices are completely empty and must gather secrets, shadow functions, and partial decryptions for the first secret. A three secret buffer with two hours each yields six hours of disconnected uptime. Because of the two hour increments, by the time of the disconnection the first secret's time has ended and the devices are buffering the second, third, and fourth secrets, leading to an end time at hour 8. The group does not reconnect to the key server until a bit after hour 9. To be exact, the log file says that the devices load the 8hr to 10hr secret at 34,500s (9hr 35min). Assuming reconnection at around 33,300s (9hr 15min) and 600s initial downtime, this means it took about 1200s (20 minutes) to gather and load the 8hr to 10hr secret.

Compared to the perfectly reliable system, the LDT and key time are drastically smaller. An LDT of just over 40,000 secret*s for the default situation is less than a third of the perfect system's 151,000 secret*s. The key time fares even better, being just under 200,000 key*s compared to the perfect system's 1.2M key*s for the standard case. The lower and higher threshold cases follow a similar trend.

An issue to note with the numbers stated above is that LDT and key time are directly related to downtime. The 8hr to 10hr secret could not be loaded until three quarters of the way through its duration, and its shadow functions were not in any device's memory either. However, this is still useful as a rough comparison, since only one secret out of six was effectively "left out." For comparison we can examine the values of major version 003. In version 003.000, the average key time is about 332,000 key*s and the average LDT is about 53,000 secret*s. While the key time is 30% higher, it still pales in comparison to the perfectly reliable system's value.

The most prominent feature at first glance is the drastic increase in downtime that minor version 006 shows. Minor version 006 is when the secrets have one hour durations instead of two, and the buffer is increased to five secrets from three. However, with some basic math the issue becomes clear: three secrets of two hours each means six hours of time, while five secrets of one hour each means only five hours of time. And the downtime delta perfectly reflects this, with an increase of approximately 3,000 seconds, or just under one hour. What is interesting, however, is that there is not a similar decrease with minor version 005, where secret times are four hours and it buffers only two secrets. This should result in two hours of less downtime, since it should buffer eight hours instead of just six. Here, the issue is in the timing: the devices are disconnected from the server by the time the first secret expires. This leaves the second secret, valid from 4 hours to 8 hours as the only one in the buffer. Minor version 000, with its trio of two hour secrets, grabs the 6 to 8 hour secret after the first secret expires at 2 hours. Before the 2 to 4 hours secret expires, the devices are already disconnected and cannot get the 8 to 10 hours secret. In other words, it worked out that the longer secret time did not yield any downtime benefit due solely to unfortunate timing of the events.

Another interesting development is in the increase and decrease of key time in minor versions 001 and 002. Versions 001 and 002 change the threshold to 4 and 12, respectively. This should intuitively lead to a key time that is either half or half-over that of the default, a threshold of 8. However, the changes are not as drastic as that, being closer to 30% under or 40% over, respectively. In a similar vein, minor version 005 does not drop by the expected one third due to losing one of the three secrets in the buffer, and minor version 006 however, does not increase by the expected two thirds, corresponding to the gaining of two buffered secrets over the original three. Instead, 005 drops by about 25% and 006 increases by about 30%. These kind of inconsistencies are difficult to pin to a specific cause, given the highly random nature of RFC/RFI and shadow function selection. For example, version 004.000 has devices with key times of almost 280,000 alongside others with key times of 143,000. To illustrate this point further, Figures 4, 5, and 6 show the number of shadow functions for devices with key times close to their respective test's average. The shape of the graph does not provide much to help in finding the reason for the inconsistencies. Finding the root cause would likely require analyzing each device in every run. However, it is very likely minor version 006's inconsistency is at least partially due to the extra hour of downtime where it has no shadow functions for any secret.

6.5 Major Version 005: Split Group Disconnection and Reconnection

6.5.1 Overview

Major Version 005 is similar to 004 except that the device group splits into two equally sized groups after disconnecting from the key server. Table 6 lists the movement of the two groups. Recall that the position definition is with respect to the bottom left corner of the box. The Y distance of 250 for Group 2 is necessary for total disconnection because Group 1 will extend up to a Y of 100. The results of the tests on major version 005 are in Figures 7, 8, and 9. For this

analysis we will be focusing on the baseline, minor version 000. The other minor versions do not provide much in the way of unexpected results.

Group 1 Size: 8 Disp: 100	X	0	0	50	150	150	150	150	150	150	150	50	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12
Group 2 Size: 8 Disp: 100	X	0	0	50	150	150	150	150	150	150	150	50	0	0
	Y	0	0	50	150	250	250	250	250	250	150	50	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12

Table 6. Major Version 005 movement definition.

6.5.2 Analysis of Results

As with major version 004, there is an expected 7,000s downtime on the baseline and an extra 3,600s downtime on minor version 006. However, the baseline for major version 005 extends up to almost 10,000s of downtime, indicating that there is some kind of issue. Examining the per-device values, it turns out that this is not a consistent problem. Figure 7 shows the per-device aggregates for trial runs 000 and 001. Trial runs 002 and 003 exhibit similar outputs, where one run will be fine and one will have much a larger downtime for the second group. Figures 8, 9, 10, and 11 illustrate how the RFC of two secrets' cryptosystems change in time for the same device across the two trial runs. Total separation of the groups should occur around hour 4, or 14,400s. In Figure 8 there are two momentary dips for the 4-6hr RFC at around 12,000s and 13,000s. Recall that the falloff factor changes based on time. In particular, it changes at 40 minutes and 20 minutes before loading. These momentary dips are because of this changeover and the resulting requests for new shadow functions. These same kind of momentary dips can be seen in Figure 9 for the 6-8hr RFC. The important thing to note in Figures 8 and 9 is that every time the RFC dips below 1, it is immediately corrected back to being more than 1. This means that Group 2 has at least a threshold number of shadow functions available.

Figures 10 and 11 reveal the issue plaguing Group 2 of run 001. Group 2 does not have enough shadow function variety for the 4-6hr and 6-8hr cryptosystems to raise their RFCs above 1. There are no momentary dips from falloff factor time changeover because this device has every shadow function available in Group 2. The log file reveals that for the 6-8hr cryptosystem, device 16 has shadow functions 0, 1, 6, 7, 11, 13, and 15. Given a consistent falloff factor of 1.0 for locally saved functions, having 7 out of a threshold of 8 leads to the resulting 0.875 RFC for this cryptosystem. The same is true of the 4-6hr cryptosystem. All but three of the other devices in Group 2 share this same fate.

In run 001, devices 11, 12, and 13 all indicate that they were able to decrypt a secret that their Group 2 peers could not. This type of inconsistency is due to the somewhat random nature of the Prepare Next Secret loop. Recall that Prepare Next Secret works to gather partial decryptions from peers in preparation for the next secret's loading. The default lead times for this process are 90 minutes for starting and 30 minutes for finishing. In the log file it says that device 13 was first able to decrypt the 4-6hr secret at 9,240s (2hr 34min), or 86 minutes before its start time. Device 12 could decrypt the 4-6hr secret at 12,000s (3hr 20min) and device 11 could decrypt it at 12,480s (3hr 28min). As for the question of why the other five devices could not decrypt the 4-6hr secret, unfortunately the log file cannot say for certain. The event for receiving a partial decryption does not list which shadow function was used, only that the device now has some new number of partial decryptions for a certain secret. However, given that the devices hold seven shadow functions it is reasonable to say that the other five devices in Group 2 were simply unlucky in their partial decryption requests. This is partly due to the fact that selecting peers' functions for decryption does not take into account the RFI of the available shadow functions. It only cares about which shadow functions are locally saved.

6.6 Major Version 007: Small Group Disconnection and Reconnection

6.6.1 Overview

Major Version 007 is one of the most punishing tests. A large group stays connected to the key server while a small group disconnects from everything else for longer than its buffered secret time. Table 7 lists the movement of the two groups and Figures 15, 16, and 17 show the average results of the tests. Given the relative sizes of the two groups, it's expected that the measurements, especially downtime, will be skewed lower. As such, we will not spend much time examining the averages. Similar to major versions 004 and 005, there will be an expected 7,000s downtime for Group 2. However, here the disconnection is timed such that minor version 006 will not induce any extra downtime. In short, between hours 3 and 4 the Group 2 devices have the 3-4hr secret loaded and the secrets for hours 4-8 in the buffer. At hour 4, total disconnection has occurred and thus the end of the buffered time stays at hour 8, the same as the baseline (2-4hr loaded with hours 4-8 in the buffer) and minor version 005 (0-4hr loaded with hours 4-8 in the buffer).

Group 1 Size: 13 Disp: 100	X	0	0	0	0	0	0	0	0	0	0	0	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12
Group 2 Size: 3 Disp: 70	X	0	0	0	125	250	250	250	250	250	250	125	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12

Table 7. Major Version 007 movement definition.

6.6.2 Analysis of Results

In minor version 000, there were eight total test runs and in three of them Group 2 was able to decrypt the 6-8hr secret. Figures 18, 19, 20, and 21 show graphs of the RFC for this secret's cryptosystem for runs 000 through 003. Note the wide range of final RFC values;

anywhere from 0.375 to 1.000. This makes sense given that we are effectively taking a random sample of three devices from a pool of 16 and checking how much shadow function variety they have. What is surprising is that there was any test run, not to mention three out of eight, in which they could decrypt the 6-8hr secret. The fact that they could consistently decrypt the 4-6hr secret is somewhat less impressive, given that they start gathering partial decryptions before disconnecting from the rest of the group.

Minor versions 001 and 002 had the expected downtime results. The lower threshold of 001 made it so that five out of the eight runs saw Group 2 decrypting the 6-8hr secret, while the higher threshold of 002 dropped that number to two out of eight. The real star of the show is minor version 003 with its more aggressive falloff factor. During its eight test runs, only once did Group 2 fail to decrypt the 6-8hr secret. Minor version 004, with its less aggressive falloff factor, was a spectacular failure. None of its test runs saw Group 2 decrypting the 6-8hr secret, and in two different runs one of the devices even failed to decrypt the 4-6hr secret.

Minor version 005 had interesting, though still unsurprising, results. Note that in this major version, the disconnection is timed such that every minor version will have the same amount of expected downtime. In six of the eight runs Group 2 performed perfectly. In the other two, two of the three Group 2 devices were able to decrypt the 4-8hr secret while the third device was left with a 22,000s downtime (14,400s from the secret and about 7,000s from the expected downtime). Like with major version 005, this inconsistency is due to the Prepare Next Secret system. Those devices simply got unlucky with the partial decryption selection. Minor versions 007 and 008 had very minimal effect on the downtime. The 4-6hr secret is the only important secret affected by this change, because even with the earlier start and finish decrypt times the 6-8hr secret will not start getting any partial decryptions before the disconnection occurs.

Minor versions 009 and 010 have unexpected results. Both have larger downtimes compared to the baseline because there were no test runs where Group 2 managed to decrypt the 6-8hr secret. Given that minor version 009 increases the number of hops that the devices can communicate over, it would seem reasonable that the downtime should be somewhat lower. However, the increased falloff distance is likely working against Group 2 in this scenario. The increased ad-hoc communication range will keep RFC higher than it would be otherwise, leading to fewer shadow functions requested before disconnection. Even if the falloff factor is large, it still adds a non-negligible amount to the RFC. Their longer reach makes no difference because they don't take advantage of it when it counts. Minor version 010 has the opposite issue, the shadow functions variety is limited by whatever devices happen to be within direct communication range right before disconnection.

6.7 Major Version 009: Linear Chain of Group Connections

6.7.1 Overview

Major version 009 involves no macro scale movement or timing. It is a straight line of three device groups with only one connected to the key server. Table 8 lists the positions of the groups. This test is meant to find how well Ad Hoc Security handles static dispersion of devices. Figures 22, 23, and 24 give the average downtime, key time, and LDT of this test.

Group 1 Size: 8 Disp: 100	X	0	0	0	0	0	0	0	0	0	0	0	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12
Group 2 Size: 8 Disp: 100	X	150	150	150	150	150	150	150	150	150	150	150	150	150
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12
Group 3 Size: 8 Disp: 100	X	300	300	300	300	300	300	300	300	300	300	300	300	300
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12

Table 8. Major Version 009 movement definition.

6.7.2 Analysis of Results

The baseline's downtime is relatively low. In a few of the test runs there could be upwards of 2,100s of downtime for some devices, but the log files show that it was always for the first secret. Every secret afterwards would be decrypted and loaded on time. Figure 25 shows the numbers for the first two runs of minor version 000 on the left, and minor version 009 on the right. The first eight devices are in Group 1, the next eight are in Group 2, and the last eight are in Group 3. The key time of the three groups seems to follow a distinct pattern. In minor version 000, Group 1 has the lowest key time at around 25,000-35,000 key*s, while Group 2 has a key time of around 35,000-50,000 key*s, and Group 3 has the highest key time at 55,000-70,000 key*s. The reason for this is quite simple: lack of variety due to "filtering." During times when there is less than a threshold number of different shadow functions available, the devices will locally save every shadow function available in the local group. This behavior is good for reliability, but obviously increases key time drastically. For example, Table 9 shows a comparison of shadow function availability for a device in each Group. As the device gets further from the key server, the number of available shadow functions becomes "filtered down" based on what devices nearer the key server have chosen to locally save. As a comparison, Table 10 shows the same availability comparison for minor version 009, where the maximum number of network hops for requests is four. Notice how version 009.009 does not show decreased availability with increased distance. The time of 4,740s is arbitrary, but by that time the dispersion of shadow functions for this cryptosystem had become mostly fixed. Returning to Figure 25, the right column shows the numbers for the first two runs of version 009.009. Note the more even distribution of key time, with the highest being just under 60,000 key*s, which is the average value of Group 3 devices in version 009.000.

The effects of filtering can be seen most prominently in minor version 004. In minor version 004 run 003 there are six devices, all in Group 3, that have downtimes of 8,400s or 8,700s. They all fail to decrypt and load the 6-8hr secret because of lack of shadow function variety. Figure 26 is a graph of the RFC of device 24 in run 009.004.003. Note how the CRF levels off to be 0.875, indicating that only seven out of the necessary eight shadow functions are available within two network hops.

Minor version 010, where the maximum number of network hops drops to one, also suffers somewhat from filtering. Run 009.010.006 has devices in both Groups 2 and 3 with downtimes between 6,300s and 9,300s. The reason the average downtime is not higher is because every other run has a max downtime of anywhere from 900s to 3,300s. Like above, all of these other downtimes are due to startup, and do not occur after the first secret is loaded. In run 009.010.006 however, several devices have trouble loading later secrets on time, though the issue does not culminate in an entire secret failing to be loaded like in 009.004.003. Instead, most of the secrets are simply not loaded on time for several devices in Groups 2 and 3.

		Shadow Function Availability by ID Number															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Device Number	1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	9	Y	Y	N	N	N	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N
	17	Y	N	N	N	N	Y	Y	Y	Y	Y	N	N	Y	N	Y	N

Table 9. Shadow function availability for the 4-6hr secret's cryptosystem for three devices, each in a separate group, in test run 009.000.000 at 4,740s.

		Shadow Function Availability by ID Number															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Device Number	1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	9	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y	Y	N	Y	Y	Y
	17	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y	Y	N	Y	Y	N

Table 10. Shadow function availability for the 4-6hr secret's cryptosystem for three devices, each in a separate group, in test run 009.0009.000 at 4,740s.

6.8 Major Version 011: Wide Area Random Movement

6.8.1 Overview

Major version 011 is a test of how well Ad Hoc Security can handle rapid changes in network connections with relatively low connectivity to the key server. The movement timing is given by Table 10. This test places 32 devices in a 500 by 500 box with the server in the lower left corner. Recall that every device, including the key server, has a communication range of approximately 120 units, meaning that less than one fifth of the total area is within communication range of the key server. Furthermore, the larger movement box will cause the devices to move faster, increasing connection volatility. Figures 27, 28, and 29 show the deltas for the downtime, key time, and LDT of major version 011, respectively.

Group 1 Size: 32 Disp: 500	X	0	0	0	0	0	0	0	0	0	0	0	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	Time (hours)	0	1	2	3	4	5	6	7	8	9	10	11	12

Table 11. Major Version 011 movement definition.

6.8.2 Analysis of Results

The most noticeable feature of the three performance graphs is the fact that almost none of the minor versions improve, or even match, the downtime of the baseline. This is likely due to the increased randomness of the test as compared to the other major versions. Figures 30 and 31

show the numbers for version 011.000 runs 000, 001, 005, and 007. Note that there is a wide variance of downtimes between runs and even between devices within the same runs. While certain minor versions such as 002 (less aggressive falloff factor) and 010 (max number of hops is 1) increase the overall downtime more than others, it is likely that the baseline simply had several lucky runs. For example, regardless of what the downtime average says it is unreasonable to believe that increasing the aggressiveness of the falloff factor (minor version 001) would directly cause an increase in average downtime.

While the average downtime may be relatively large, as stated before this value does not say exactly when the downtime occurs. For every test run in major version 011, the first two to four hours of the simulation is where all of the downtime is accumulated. After that startup period, every secret is loaded on time. This makes sense given the way that the devices handle shadow functions. The first falloff factor value applies as soon as the encrypted secret is buffered. The secrets that start later in the buffer are given more time to have their shadow functions distributed compared to secrets that start earlier. Similar to a microprocessor's pipeline, the next set of shadow functions are being "processed" (distributed) in the background.

Recall that one of the initial assumptions was that the devices would form ad-hoc networks. Ad-hoc networks are far more efficient when their topology changes slowly and for fewer links. The same is true here: the system becomes less efficient as connectivity changes faster and more randomly. This is mainly because RFC will be much lower due to much smaller networks, leading to more shadow functions stored locally. Thus, it is only fitting to compare this test against the perfectly reliable system, which is immune to any topology changes because each device is self-sufficient. Even under these conditions, the key time is never more than half of the perfectly reliable system's key time except for minor version 005. The LDT as well is under half

the perfectly reliable system's LDT for all but minor version 005. In minor version 005 the key time and LDT are both only about 60% that of the perfectly reliable system's key time and LDT respectively.

In Figures 30 and 31, the unauthorized time is higher than normal for some devices. This trend holds true across all of the other minor versions except 009, with an average of about 3200s for 009 and 2200s for the rest. The unauthorized time induced by the start of the simulation is 1860s, and any time below 3600s indicates that it just took more than one minute to get the first MOCA signature. In all of the minor versions other than 009, there are usually only two or three devices per run that have unauthorized times higher than 3600s. In minor version 009 however, upwards of half the devices have unauthorized times greater than 4000s. Figure 32 shows two runs of version 011.009. This behavior is counterintuitive at first glance because minor version 009 increases the distance that devices can request MOCA signatures. However, recall that this major version has much more connection volatility than the others. That, coupled with the longer reach, means that more MOCA signature requests go unfulfilled due to some disconnection along the way.

Chapter 7

Conclusions and Summary

7.1 Conclusions

In conclusion, the Ad Hoc Security system offers a way of adaptively managing the tradeoff between availability and security of time-dependent secret information while in an ad-hoc wireless network. In many of the simulations, the main source of non-transient down time is from group separation. In all of those scenarios a more aggressive falloff factor resulted in far better reliability. Even with more aggressive falloff factors, the security of Ad Hoc Security is far better than the perfectly reliable system. In most cases the Key Time and LDT numbers were less than half of what the perfectly reliable system would have been. Furthermore, there is a noticeable difference between the Key Time and LDT in scenarios depending on how volatile the situation is. Less volatile situations (like major version 004) had lower Key Time and LDT while more volatile situations (like major version 011) had higher Key Time and LDT. This shows that the system is performing the reliability/security tradeoff as expected. What follows is an examination of the failure points of the Ad Hoc Security system, and potential means of correcting or improving the performance.

7.2 Secret Buffer Improvement

In all of the tests, the LDT and key time were much lower than the perfectly reliable system's values. The formulations of the perfectly reliable system's key time and LDT show why: there is no limit on the number of secrets, and thus shadow functions, that a device can hold. The longer the simulation time, the faster those two numbers grow as well. By itself, the fixed size of the secret buffer puts a hard cap on the key time and LDT. Coupled with the self-limiting nature of the Redundancy Factor and Prepare Next Secret subsystems, the LDT and key

time values are greatly reduced. However, the fixed secret buffer size also puts a hard cap on the amount of time the devices can be disconnected from the key server. Simply making the secret buffer larger would obviously have detrimental effects on the security.

One possibility for extending the disconnection time could be to have an extended secret buffer which does not store every secret after the main buffer. Effectively, it would be like applying the Redundancy Factor system to a secret buffer as well as its shadow functions. As the secret gets closer to being moved into the main buffer, the requisite fraction of devices that store that secret should increase until that fraction reaches 1. One detriment of this proposal is that extending the number of secrets in the buffer would increase both the internal storage requirements as well as the size of the crypto status messages. Larger messages mean more network congestion and more power consumption.

7.3 Redundancy Factor Improvement

In major version 005, even with group sizes equal to the threshold of the cryptosystem there still exists the possibility being unable to decrypt a secret. This event was due to the random division of the larger group into two smaller groups. Without any foreknowledge of group divisions, the devices had no way to prepare for this. Furthermore, the only indication of separation was the increased number of two hop neighbors, which may trigger additional shadow function requests. Of course, the time between seeing more two hop neighbors and then being disconnected may be very short, leaving the device groups with very little time to prepare for the separation.

The issue here is of situational awareness. In its current form Ad Hoc Security only takes into account distance and time. There is no analysis of the topology or reliability of the network. Topology is important because of single or small number of failure points. To illustrate the

benefits of topology knowledge, imagine the scenario where every two hop neighbor is only accessible through the same one hop neighbor. It would be reasonable to increase the falloff factor of those two hop neighbors because of the consequences should that "choke point" link go down. Network reliability knowledge can be used in a similar way. A two hop neighbor at the end of a pair of 30% reliable connections should not have the same falloff factor as a two hop neighbor at the end of a pair of 90% reliable connections. However, this type of system increases the complexity and amount of information required to be constantly distributed. As with the extra secret buffer mentioned above, this increases storage requirements and power consumption.

7.4 Small Group Disconnection Reliability Improvement

In major version 007, the test runs showed that it is sometimes possible to prevent downtime even when it involves a small groups splitting from a larger one. However, it also showed that it is definitely not consistent unless we focus less on security and more on reliability, like having a more aggressive falloff factor. However, this solution would apply to all of the devices, not just the ones that need the extra reliability. While Ad Hoc Security is relatively dynamic, the aspect of security vs. reliability through falloff factor is fairly static.

Ad Hoc Security leveraging the assumption of devices grouping up is, more generally, leveraging an assumption of some level of organization. In most cases, organization creates efficiency. If the network of devices were perfectly static, then the distribution of shadow functions could optimize security with no detriment to reliability. In much the same way, if there is knowledge or assumptions of organizational divisions, such as platoon, squad, fire team, etc., then the system could leverage that knowledge to enhance reliability. For example, a four-man fire team could have devices that coordinate so that no two devices store the same shadow

functions. With no overlap in function variety, they will be maximizing the probability of being able to decrypt a secret, instead of semi-randomly selecting functions and hoping for the best.

Beyond this, if there were a way of manually or automatically detecting the need to go into a kind of "super high reliability" mode, then the overall security and reliability would stay intact while only changing for the affected devices. While the Redundancy Factor system automatically does this to an extent, it is a purely reactive system. Having a proactive solution would prevent the issue of scrambling to acquire more shadow functions before disconnection occurs. Of course, this would likely drastically increase complexity or require user intervention, neither of which is desirable.

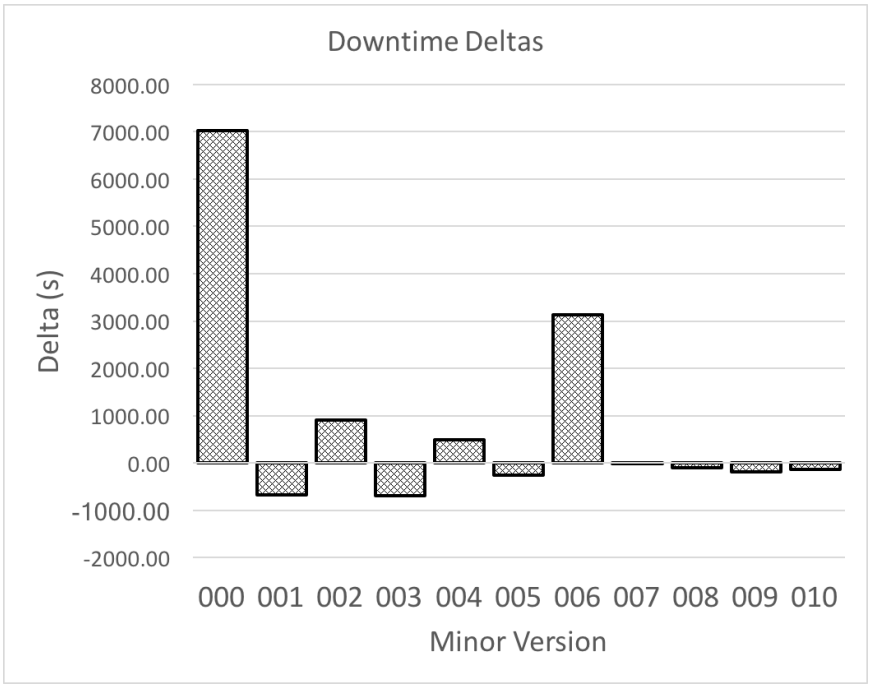

Figure 1. Major Version 004 Downtime Delta.

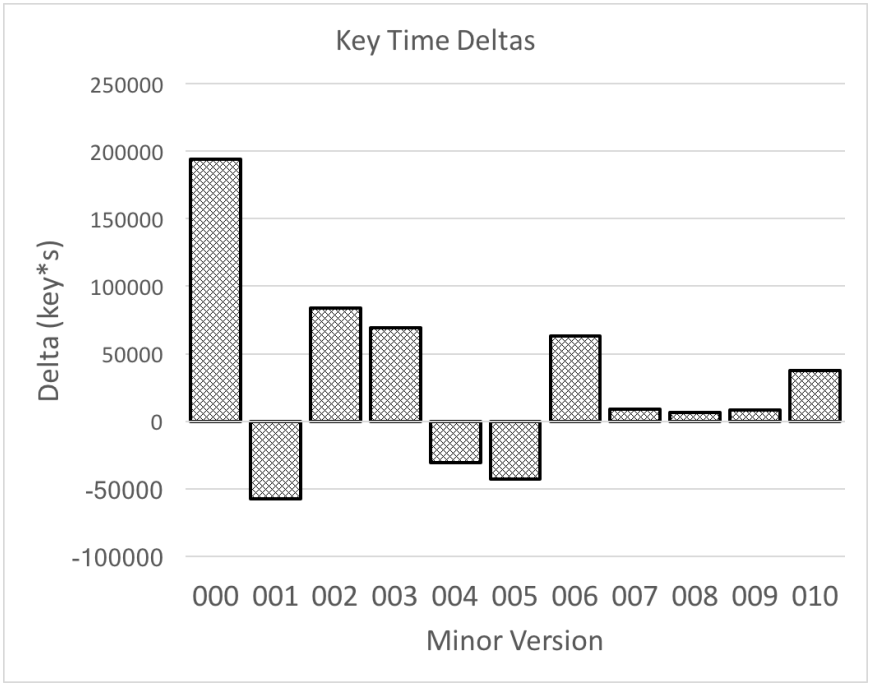

Figure 2. Major Version 004 Key Time Delta.

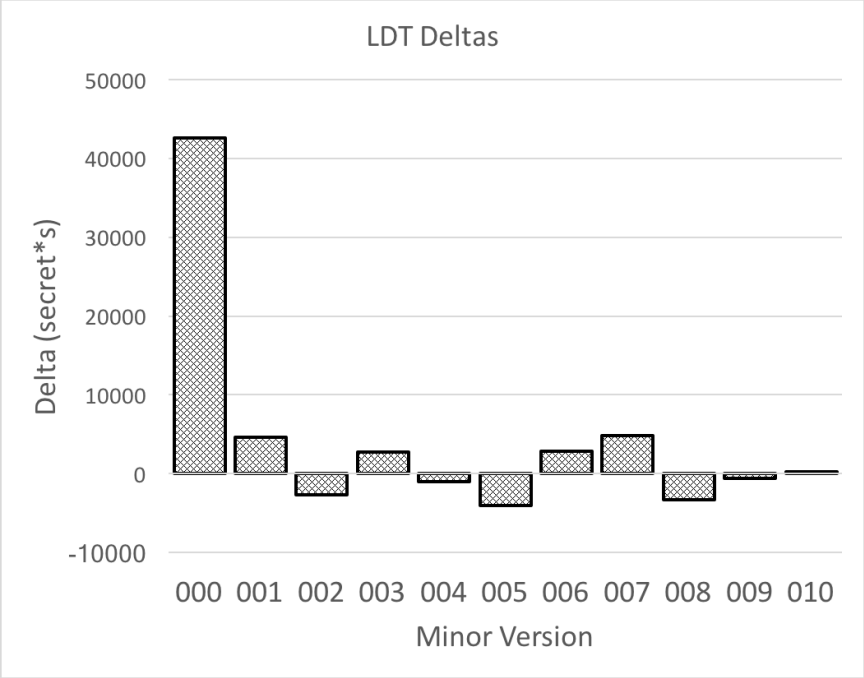

Figure 3. Major Version 004 Local Decrypt Time Delta.

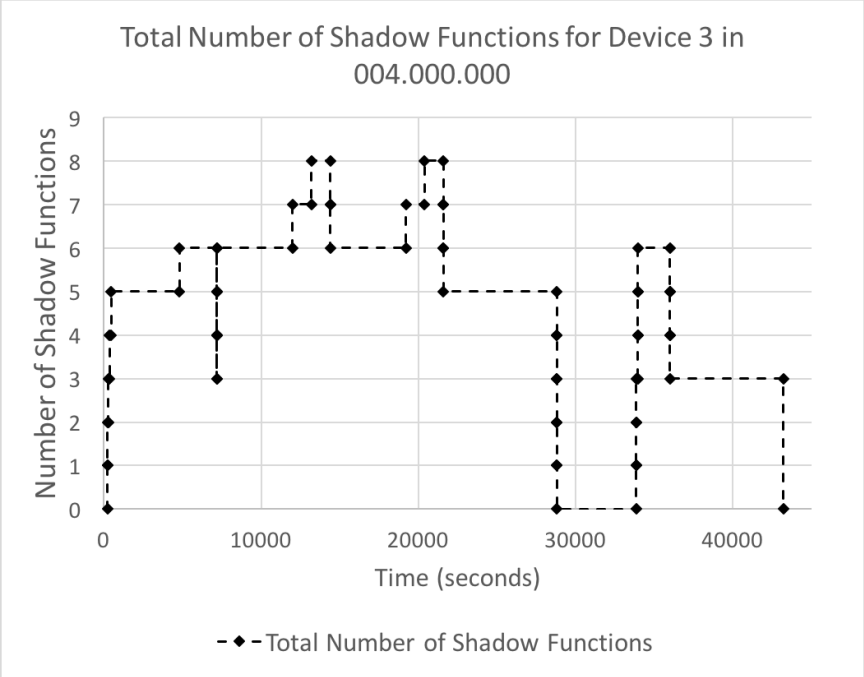

Figure 4. Total number of Shadow Functions for Device 3 in test run 004.000.000.

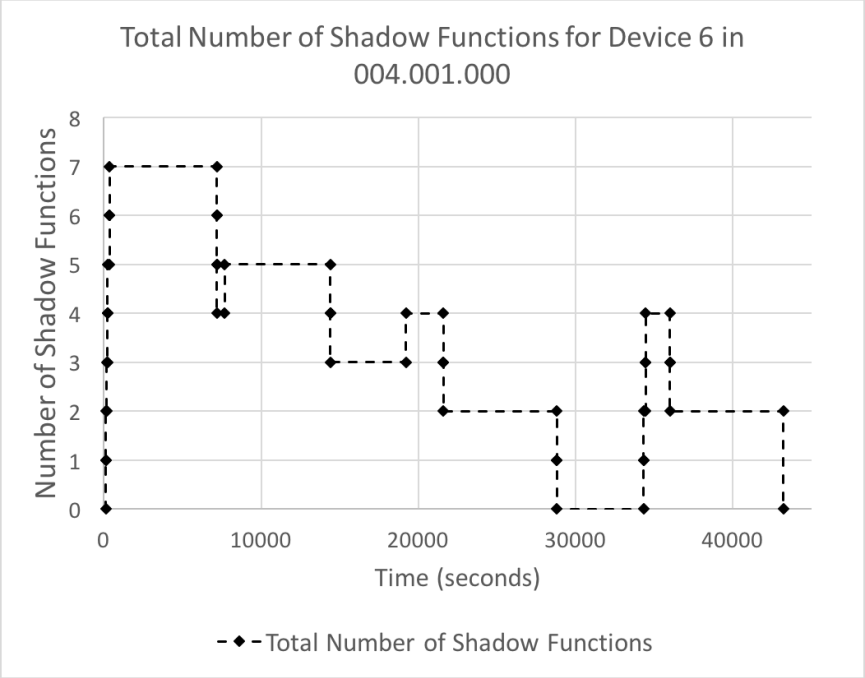

Figure 5. Total number of Shadow Functions for Device 6 in test run 004.001.000.

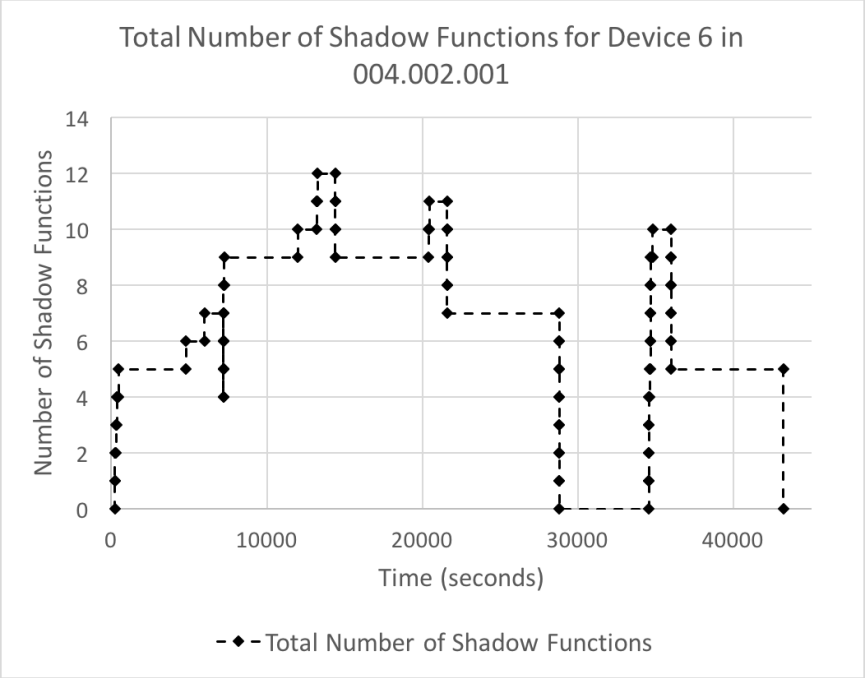

Figure 6. Total number of Shadow Functions for Device 6 in test run 004.002.001.

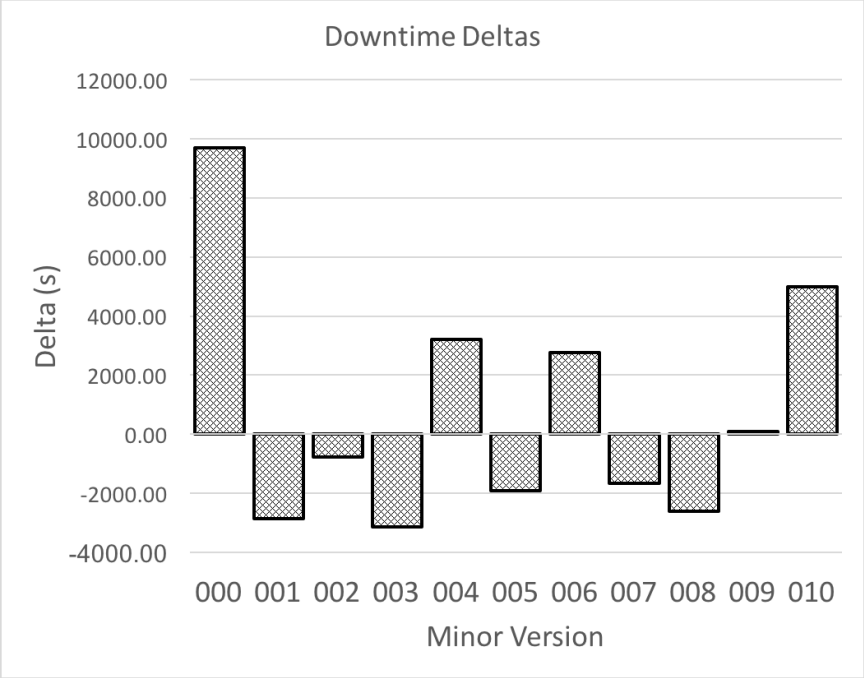

Figure 7. Major Version 005 Downtime Delta.

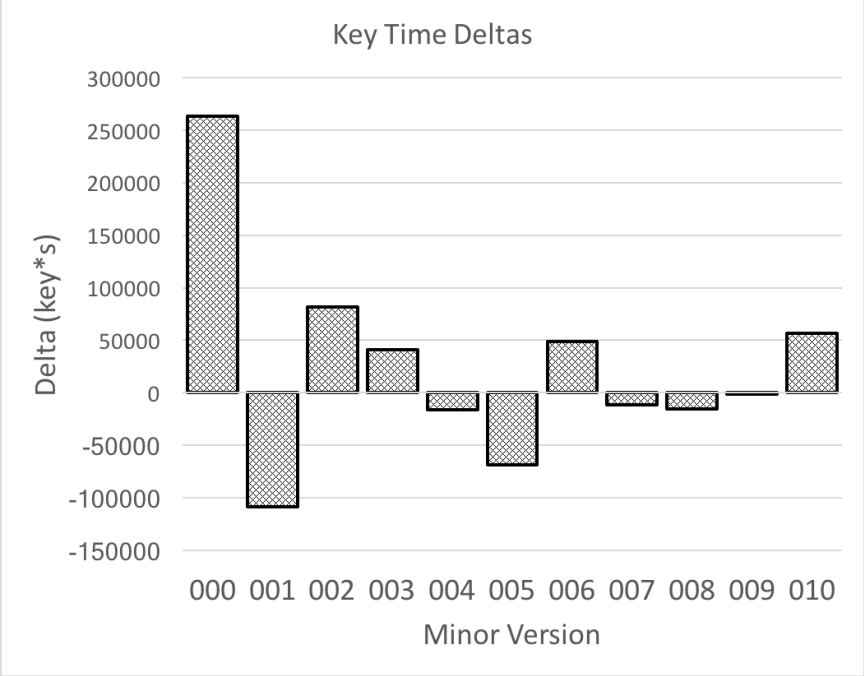

Figure 8. Major Version 005 Key Time Delta.

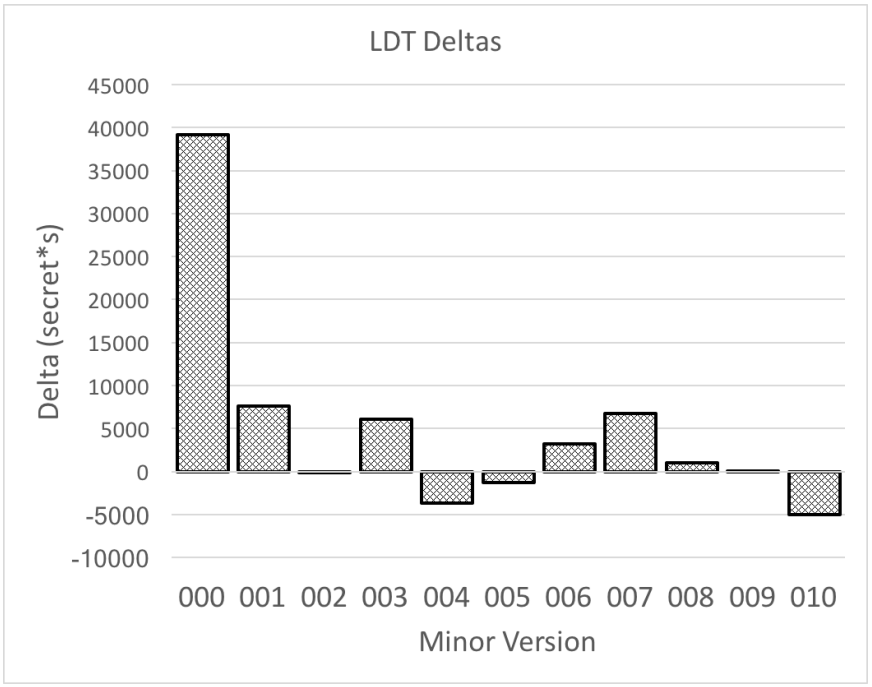

Figure 9. Major Version 005 Local Decrypt Time Delta.

1	6600	233820	41820	1860
2	6600	248280	43080	1860
3	6300	293400	43260	1860
4	6300	237660	42300	1860
5	7200	228240	41340	1860
6	6300	267840	42300	1860
7	6300	270660	44760	1860
8	6600	274980	45720	1860
9	6300	236160	42840	1860
10	6600	216480	42360	1860
11	6300	266520	45900	1860
12	6300	239040	42840	1860
13	6600	239760	41820	1860
14	6300	260520	46440	1860
15	6300	258660	46260	1860
16	6300	245940	42960	1860

1	7500	229560	41100	1860
2	7200	194940	41100	1860
3	6900	270120	42540	1860
4	6900	238200	42120	1860
5	7200	227400	41280	1860
6	6900	229800	42120	1860
7	6900	272880	42360	1860
8	6900	272220	42300	1860
9	21900	320100	23460	1860
10	21600	364320	23820	1860
11	14700	269760	32760	1860
12	14100	297240	33600	1860
13	14100	376440	36960	1860
14	21900	301257	22860	1860
15	21900	290280	22860	1860
16	21900	334500	23460	1860

Figure 10. Version 005.000 Trial Runs 000 (left) and 001 (right). From left to right the columns are: device number, downtime, key time, local decrypt time, and unauthorized time.

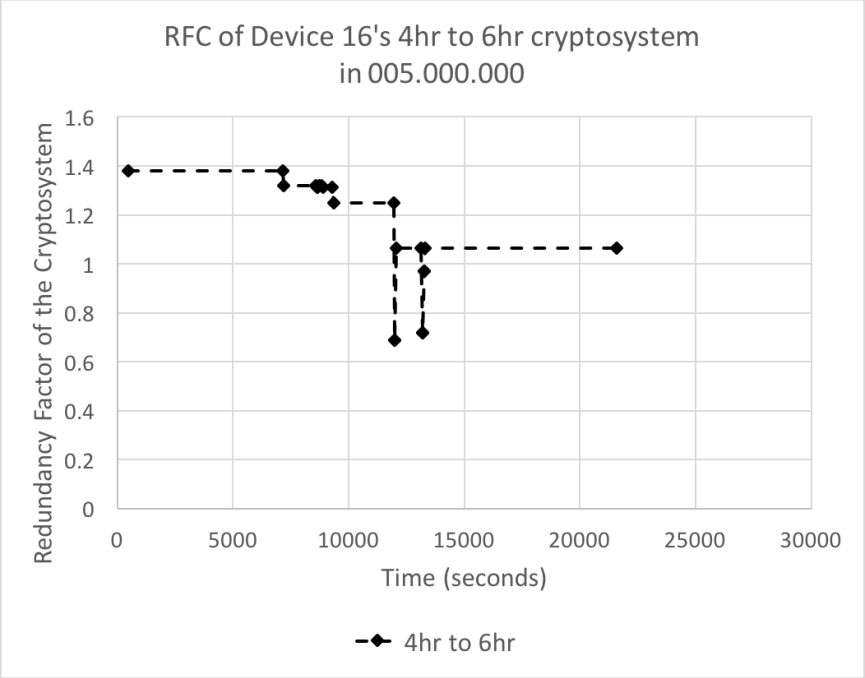

Figure 11. RFC for Device 16's 4hr to 6hr cryptosystem during run 005.000.000.

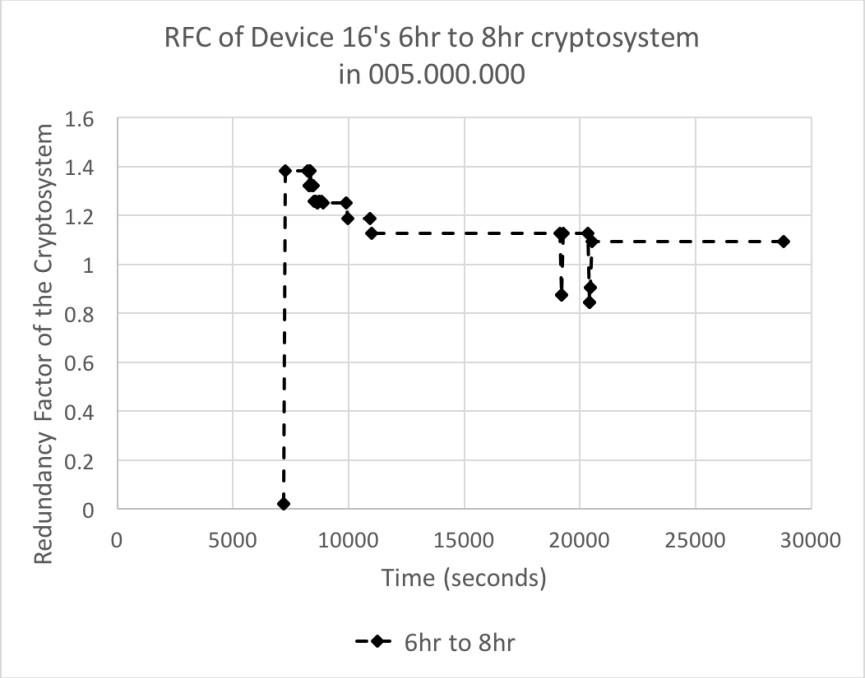

Figure 12. RFC for Device 16's 6hr to 8hr cryptosystem during run 005.000.000.

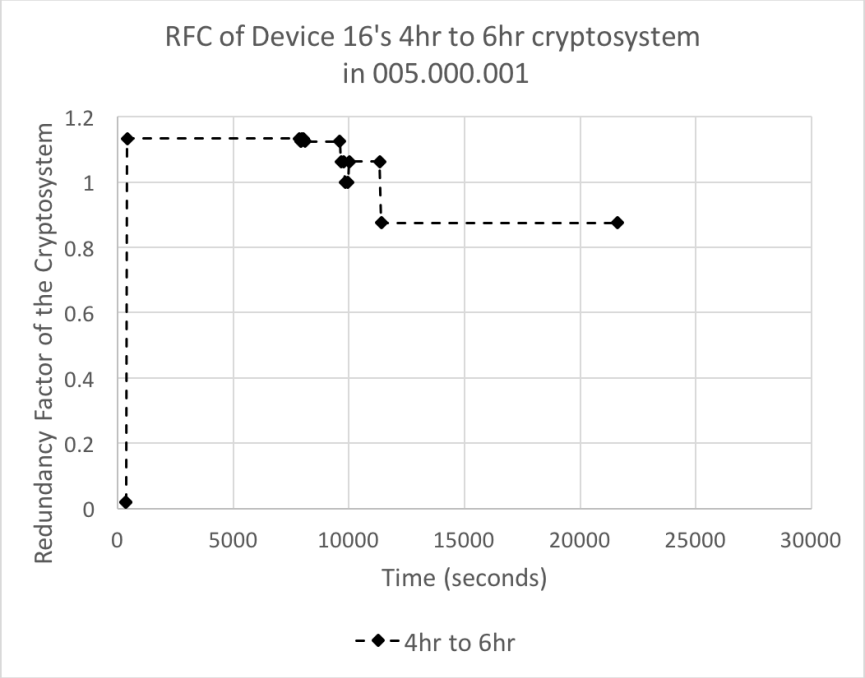

Figure 13. RFC for Device 16's 4hr to 6hr cryptosystem during run 005.000.000.

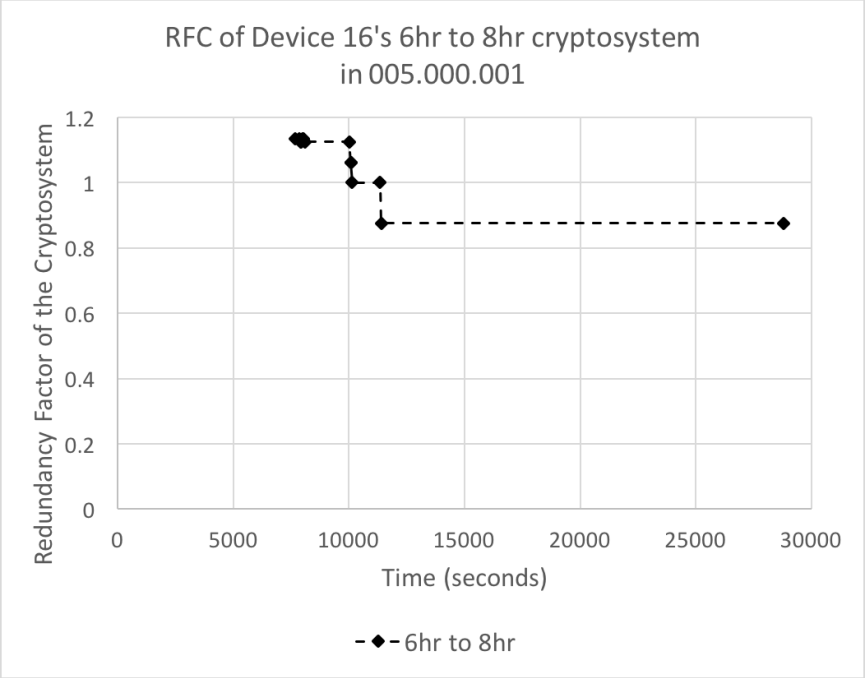

Figure 14. RFC for Device 16's 6hr to 8hr cryptosystem during run 005.000.001.

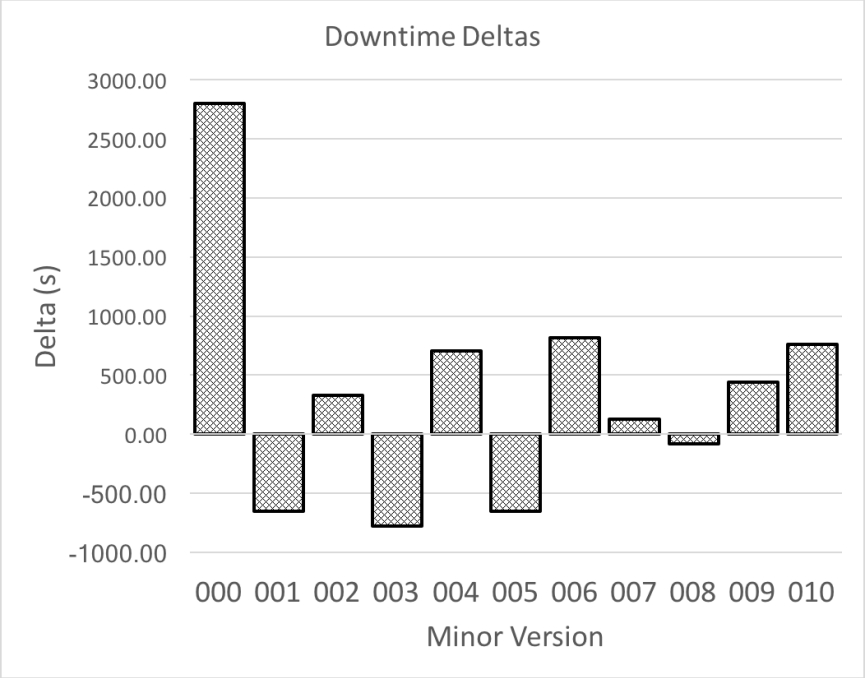

Figure 15. Major Version 007 Downtime Delta.

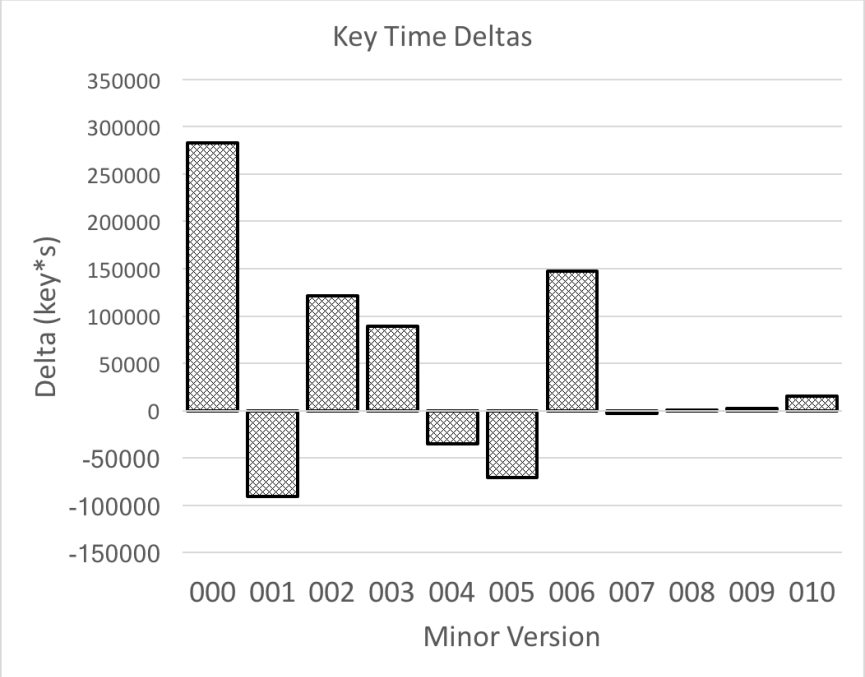

Figure 16. Major Version 007 Key Time Delta.

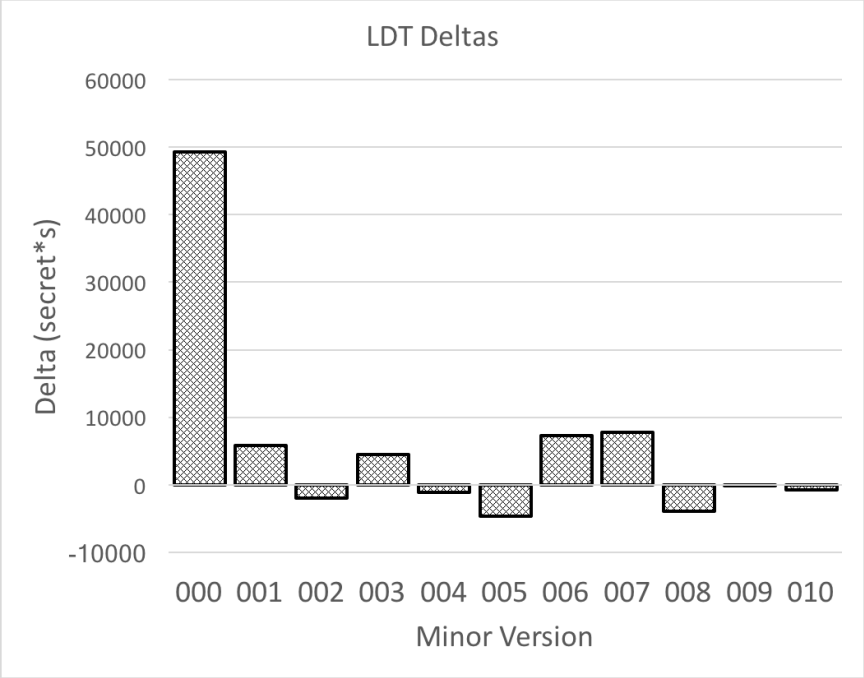

Figure 17. Major Version 007 Local Decrypt Time Delta.

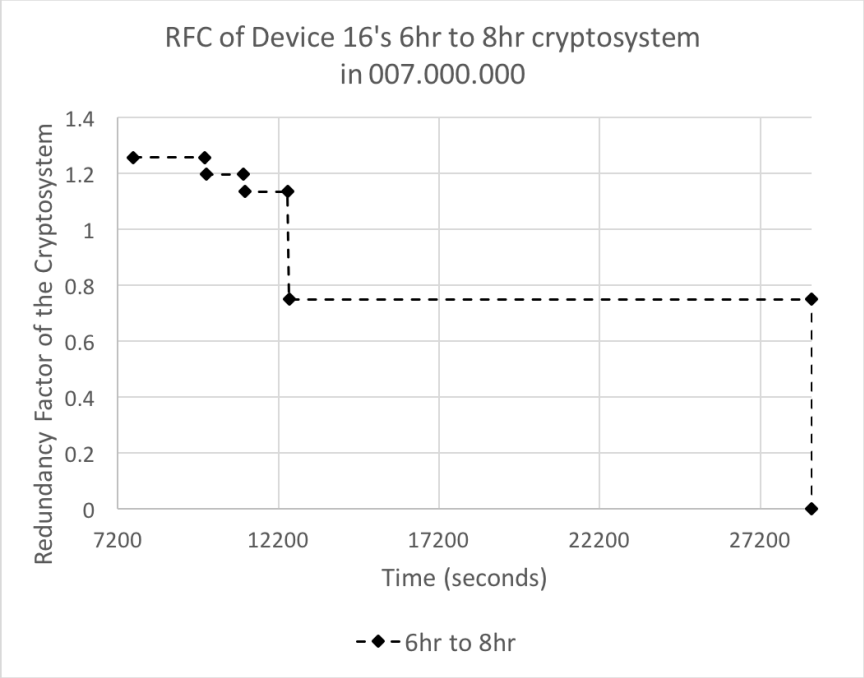

Figure 18. RFC for Device 16's 6hr to 8hr cryptosystem during run 007.000.000.

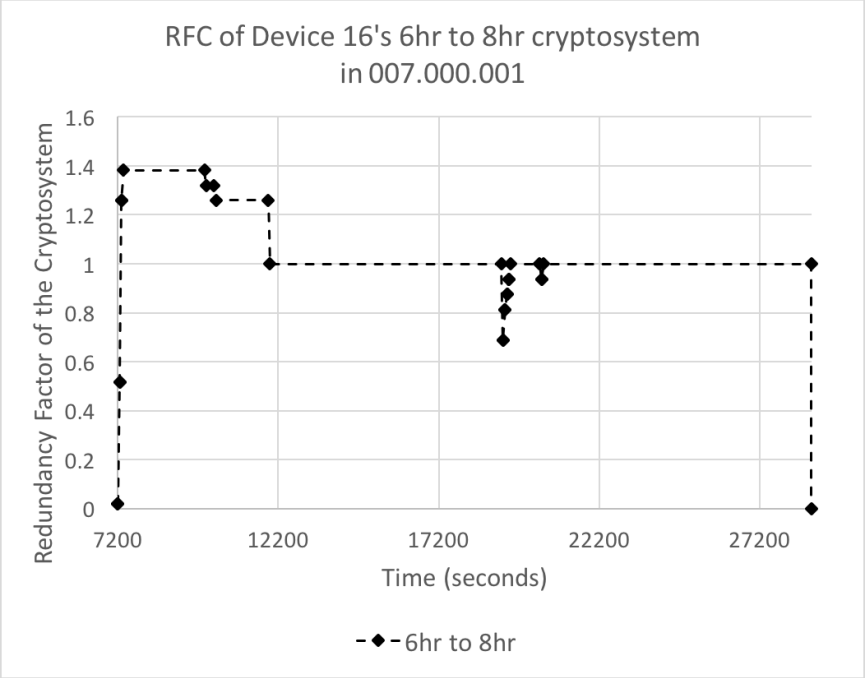

Figure 19. RFC for Device 16's 6hr to 8hr cryptosystem during run 007.000.001.

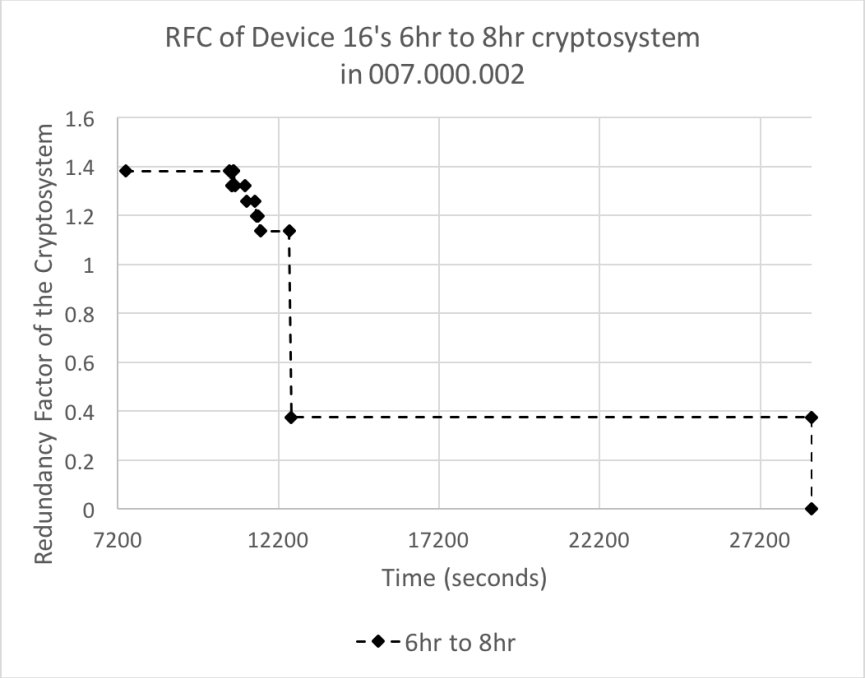

Figure 20. RFC for Device 16's 6hr to 8hr cryptosystem during run 007.000.002.

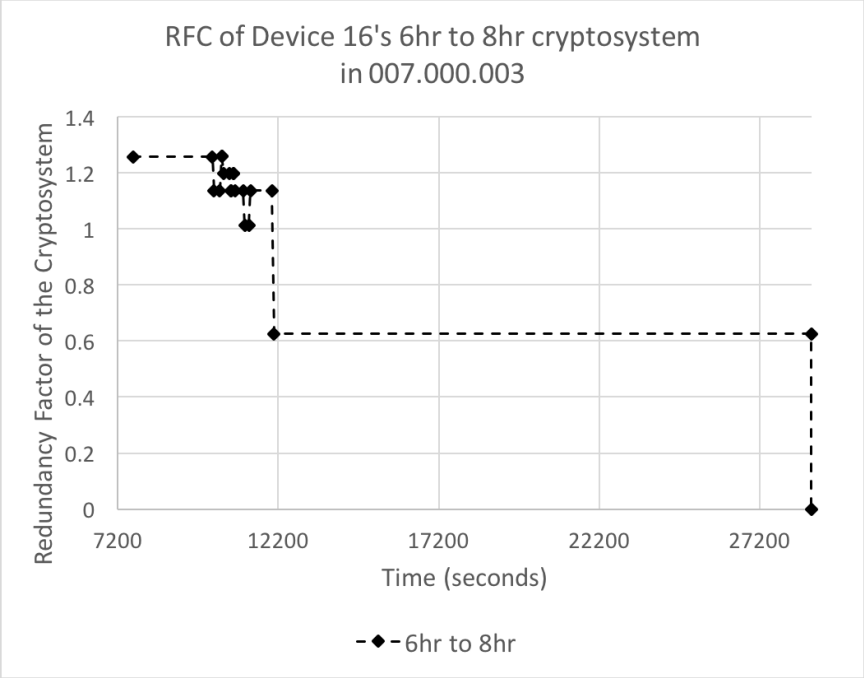

Figure 21. RFC for Device 16's 6hr to 8hr cryptosystem during run 007.000.003.

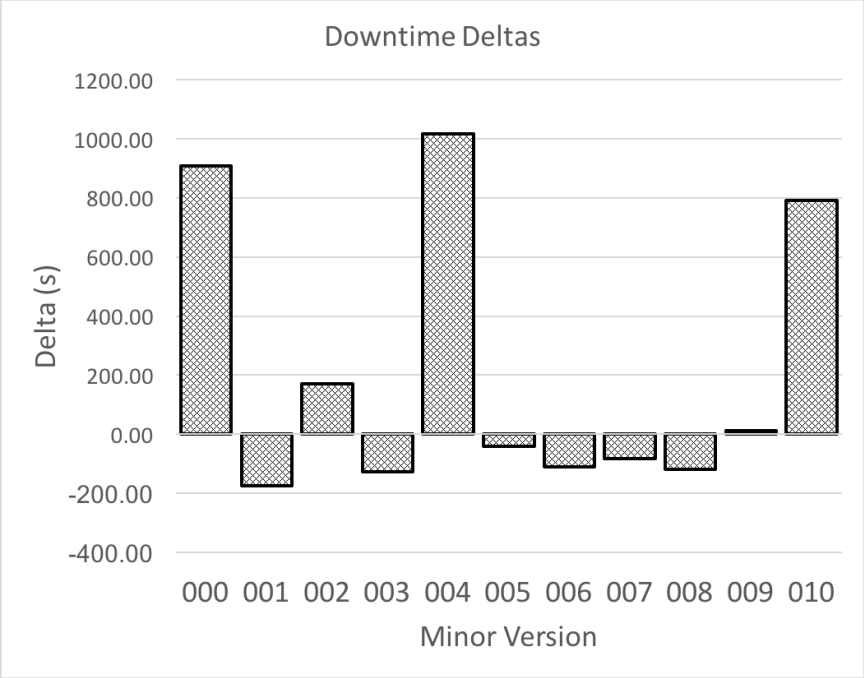

Figure 22. Major Version 009 Downtime Delta.

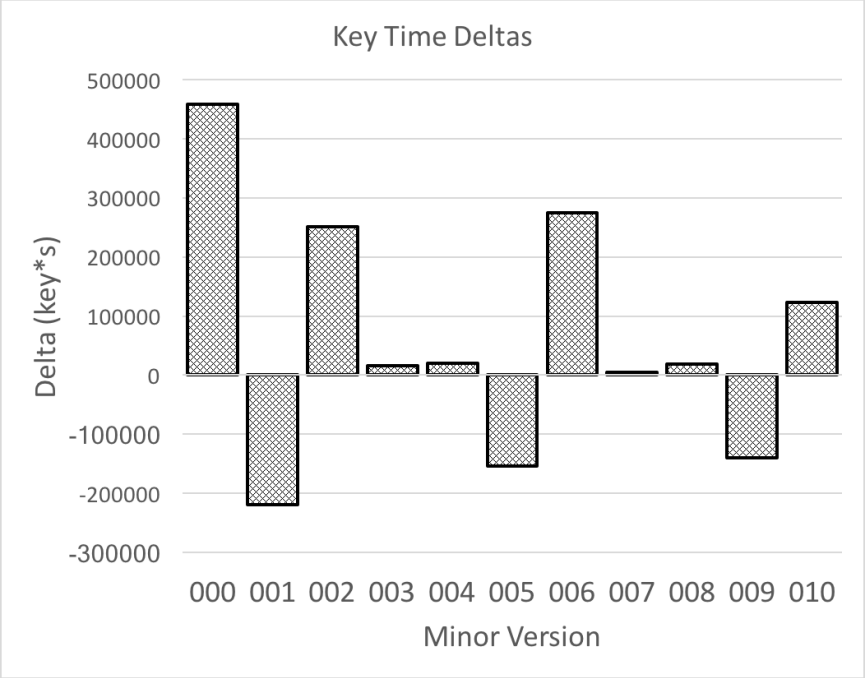

Figure 23. Major Version 009 Key Time Delta.

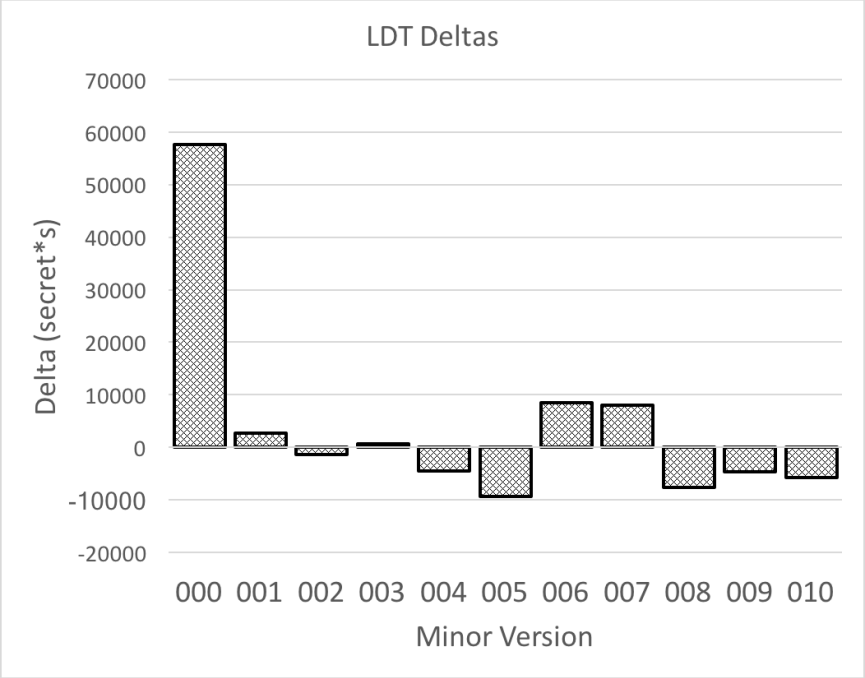

Figure 24. Major Version 009 LDT Delta.

1	600	363120	53820	1860
2	600	282840	53580	1860
3	600	396900	57780	1860
4	600	333300	50760	1860
5	600	376380	54540	1860
6	600	268980	53160	1860
7	600	343320	50760	1860
8	600	381300	51120	1860
9	600	367800	57360	1860
10	600	344520	56880	1860
11	600	353820	53580	1860
12	600	176040	49740	1860
13	600	395880	57840	1860
14	600	188820	49800	1920
15	900	343560	53940	1860
16	600	220380	53160	1860
17	2100	661320	66240	1860
18	1200	673560	63780	1860
19	1800	566400	63480	1860
20	600	633360	64920	1860
21	900	639300	67800	1860
22	2100	580020	59820	1860
23	2100	642900	63780	1860
24	2100	514200	61920	1860

1	600	275640	52920	1920
2	900	251040	49140	1920
3	600	230280	49920	1860
4	600	441660	60540	2160
5	900	307320	53160	1860
6	600	431880	60000	1860
7	1200	250620	49260	1860
8	600	387240	57720	2040
9	900	284820	49500	1860
10	900	302880	52980	1860
11	1200	161760	49140	1860
12	600	316500	53220	1860
13	900	173760	48840	1860
14	600	306480	49620	1860
15	600	342420	53340	1860
16	600	291360	50160	1860
17	900	314340	53040	2220
18	1500	384240	55380	1860
19	1800	256320	48540	1860
20	600	433380	57000	1980
21	1500	285420	49080	1860
22	1500	308760	50040	1860
23	2100	334560	55440	1860
24	1500	301500	51240	1920

1	600	345900	56700	1860
2	600	310080	53160	1860
3	900	253140	52740	1860
4	600	365700	53520	1860
5	600	232320	49920	1980
6	600	469020	64380	1860
7	600	379680	56700	1860
8	600	365040	56940	1860
9	600	413220	53340	1860
10	900	491940	60600	1860
11	1200	261240	52500	1860
12	900	358620	53340	1860
13	600	365580	57180	1860
14	1200	572760	60660	1860
15	1200	514078	56760	1860
16	600	567060	57300	1860
17	900	666720	60780	1860
18	900	639420	60540	1860
19	900	687300	63960	1860
20	900	664140	60720	1860
21	1200	624300	60360	1860
22	1500	683100	63540	1860
23	1200	587760	59820	1980
24	1500	657300	63540	1860

1	600	424500	63240	1860
2	600	410040	60960	1860
3	600	282060	52740	1860
4	600	379440	60180	1860
5	600	316140	53340	1860
6	900	226020	49380	1860
7	1200	217020	49560	1860
8	600	321720	53520	1860
9	900	293880	52800	1860
10	900	318360	52560	1920
11	600	429120	57240	1860
12	900	306840	52500	1860
13	600	496980	60780	1860
14	900	287520	52500	1860
15	600	515280	64680	1860
16	600	410460	57240	1860
17	900	598620	66540	1860
18	1200	414120	56040	1860
19	1500	503820	60060	1860
20	1500	392700	56520	1860
21	1200	424140	54180	1860
22	1200	459360	59640	1860
23	900	352380	52800	1860
24	1200	418680	56220	1860

Figure 25. Runs 000 (top) and 001 (bottom) for versions 009.000 (left) and 009.009 (right). From left to right the columns are: device number, downtime, key time, local decrypt time, and unauthorized time.

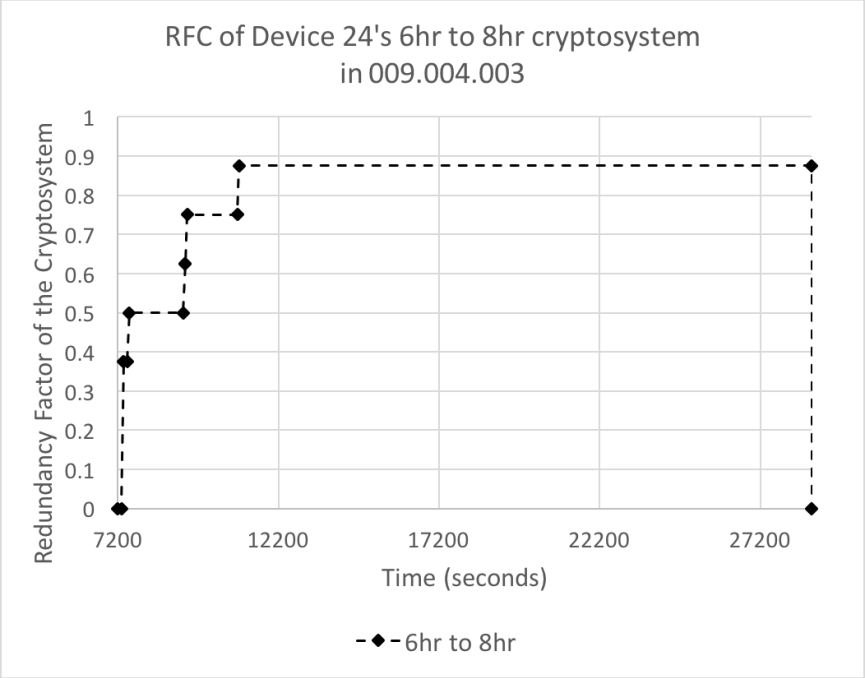

Figure 26. RFC for Device 24's 6hr to 8hr cryptosystem during run 009.004.003.

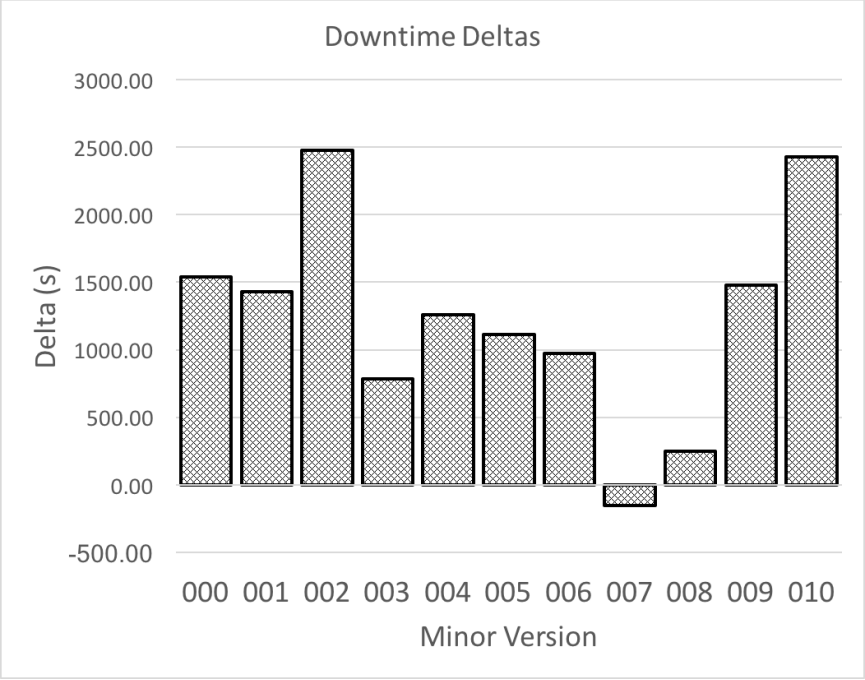

Figure 27. Major Version 011 Downtime Delta.

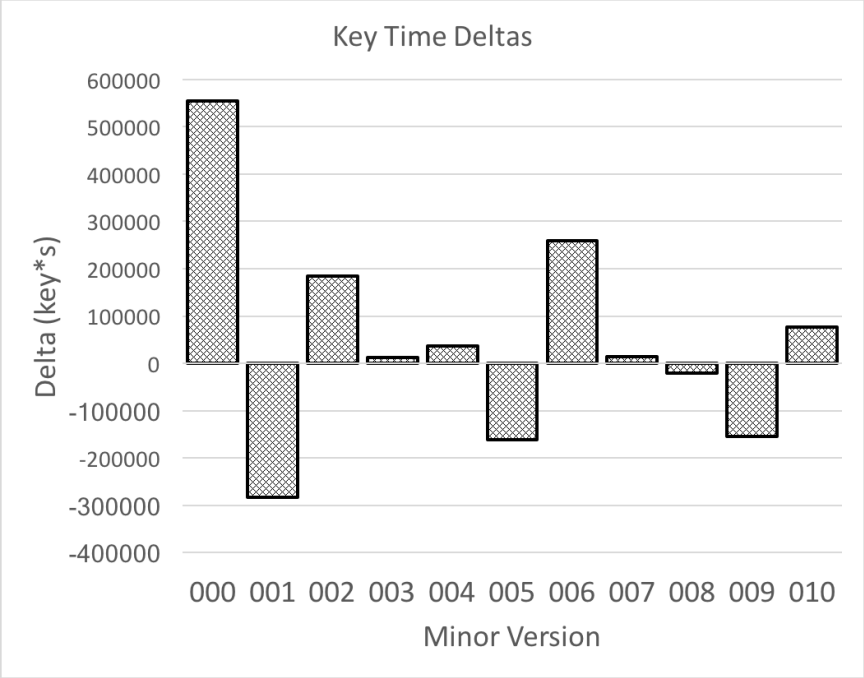

Figure 28. Major Version 011 Key Time Delta.

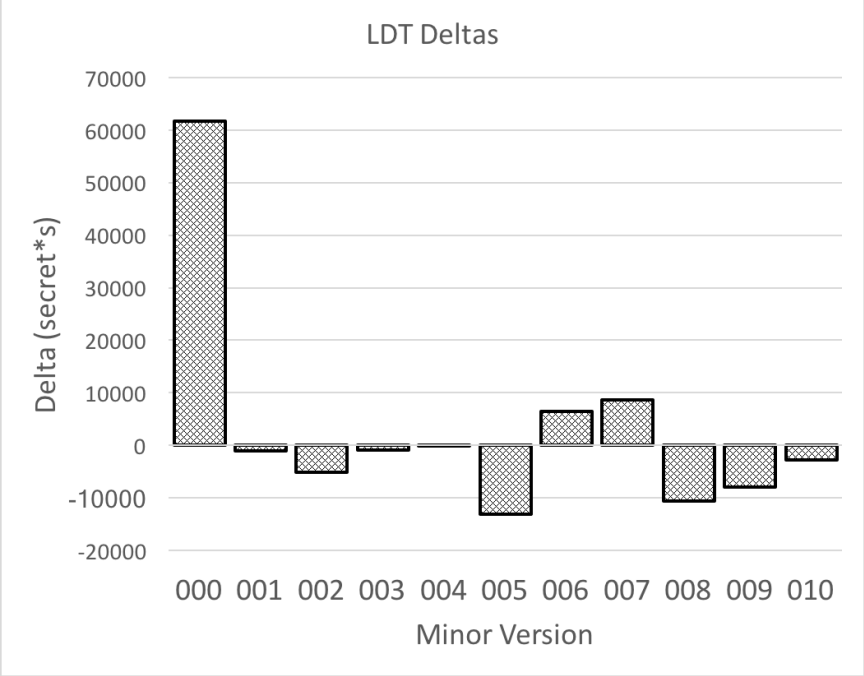

Figure 29. Major Version 011 LDT Delta.

1	1200	643740	63360	2160
2	900	725460	67200	1860
3	600	704400	68280	4020
4	1200	643440	62340	2160
5	1500	696540	65340	2280
6	1200	609780	62100	1860
7	600	623040	64860	1860
8	900	620640	60720	2100
9	900	680880	66960	1860
10	900	646440	67080	1980
11	600	659820	67920	1860
12	1200	673860	66360	2160
13	900	668820	65460	1860
14	900	639660	63840	1860
15	900	505020	58500	2460
16	1200	621240	64380	1920
17	900	695760	64080	1920
18	600	606240	64620	1920
19	900	732600	65640	1860
20	600	652380	67380	1980
21	1200	639960	65460	1920
22	1200	598380	61320	4560
23	900	576120	60600	1920
24	900	689160	67140	2340
25	1200	548040	61440	2160
26	900	660960	61980	1920
27	600	658380	63720	1860
28	900	593280	61260	1920
29	900	675660	64380	1860
30	900	594840	66300	2160
31	900	688200	66360	1860
32	1200	660780	64680	2340

1	1200	666300	66060	1920
2	3000	531720	57420	2820
3	600	645480	66720	1860
4	1200	659940	66540	1860
5	1200	519480	62220	1920
6	1200	626699	62940	2520
7	1200	495180	63240	1980
8	600	460380	56280	2340
9	1200	533940	62580	2100
10	600	469200	60240	1860
11	1200	650400	66060	1980
12	1200	587100	62940	1860
13	1200	490980	61980	1860
14	1200	466380	54900	7440
15	900	362460	51900	2400
16	1200	605220	61860	2040
17	600	485872	64380	2100
18	1500	583620	60000	1980
19	1200	572820	60480	2100
20	1200	573120	62280	2760
21	1200	436920	59160	2160
22	1200	540180	62040	2160
23	1500	591480	62580	4800
24	1200	541439	63060	2640
25	1200	643740	61440	2640
26	900	405660	52200	2160
27	1200	593700	66120	1860
28	1200	509700	59460	1860
29	1200	613980	63000	1860
30	1200	592920	63360	2280
31	1200	539340	66420	1860
32	1200	613260	62820	2340

Figure 30. Version 011.000 Trial Runs 000 (left) and 001 (right). From left to right the columns are: device number, downtime, key time, local decrypt time, and unauthorized time

1	2700	606840	63240	2760
2	600	635520	63660	2400
3	2400	450899	54120	1980
4	2700	692880	64260	1980
5	2700	548220	59160	2040
6	2700	548340	63000	1860
7	2400	563040	61080	1860
8	2700	606480	64260	1860
9	2700	498840	60540	1860
10	2700	550920	60540	2220
11	2700	498960	57720	2100
12	2700	575220	64620	3180
13	2700	526800	60060	2340
14	2400	604380	60240	1980
15	2700	569880	63360	2640
16	2700	500880	57900	1860
17	2700	617280	62700	3900
18	2700	596460	57360	2340
19	2400	652860	63120	4500
20	2700	566100	61200	1860
21	2700	608580	62640	2040
22	2400	499800	58320	1860
23	2700	523320	64140	1860
24	2700	517380	57720	2640
25	2700	598860	62280	1920
26	2700	694560	64020	1920
27	2400	599220	61740	3840
28	2700	599340	64380	2100
29	2700	681540	64800	1860
30	2700	552960	62760	4380
31	2700	555660	57540	1860
32	2400	610020	62100	1980

1	2700	432420	54240	2040
2	900	617340	64080	2040
3	1800	471780	55380	2160
4	900	549240	63300	2040
5	3900	504600	56460	2520
6	3300	483060	60840	2100
7	1800	500940	59220	1860
8	3900	445620	60840	1860
9	900	487020	62940	1860
10	2100	338880	54120	2100
11	600	448980	59340	1920
12	900	538560	66600	2040
13	900	546480	64260	1860
14	3600	375240	56640	1860
15	600	646740	67560	2040
16	600	464340	63120	1860
17	900	531240	66300	1920
18	600	553860	66420	1860
19	3600	417720	54480	2700
20	600	434880	57600	2100
21	600	593100	63480	4260
22	3600	411000	56100	2400
23	900	550260	63780	2040
24	1800	386880	55980	2100
25	1800	565680	63180	1980
26	900	465540	59160	1980
27	3900	347400	53340	1920
28	3000	440520	57720	3000
29	1800	393300	96600	2040
30	1800	374040	55020	1920
31	2700	501180	57900	2340
32	900	598740	64140	2040

Figure 31. Version 011.000 Trial Runs 005 (left) and 007 (right). From left to right the columns are: device number, downtime, key time, local decrypt time, and unauthorized time.

1	5700	445919	53880	2460
2	5100	435960	50640	2580
3	5700	451080	53580	3240
4	5700	467219	52560	2760
5	5100	372539	47940	1860
6	5100	353820	49860	1860
7	5700	634319	56040	4140
8	5400	325380	47340	1860
9	6000	348660	50220	4500
10	7200	430379	49680	1860
11	4800	405180	52500	4560
12	4800	417660	51720	2340
13	5100	446879	55020	5340
14	5700	540897	56340	3780
15	5100	557940	57720	5220
16	5100	472440	52620	2700
17	3600	548998	55680	5281
18	6300	481077	51720	5040
19	5100	536880	52080	6960
20	5100	434999	51780	1920
21	4800	526140	54060	5101
22	9900	453419	48180	5221
23	5700	542339	57840	1980
24	6300	459478	52920	1860
25	5700	522476	54480	4981
26	7500	315296	43800	1860
27	5100	357839	47880	1860
28	5100	425219	51180	1920
29	5400	293519	46980	7921
30	5700	451799	52500	1860
31	5100	465837	53700	2100
32	6300	329700	46080	5340

1	8400	273957	46020	1980
2	7800	317577	45720	1920
3	8400	302337	47340	1860
4	7800	339296	50160	4501
5	9000	308276	82320	6780
6	7800	339538	46020	1860
7	9600	374458	48000	1920
8	8700	354357	47220	2280
9	8100	334976	50340	4260
10	8100	403380	52980	4501
11	7800	287696	47400	2160
12	7500	418259	50460	3180
13	7500	388678	50100	4921
14	8100	404038	52800	1860
15	7800	241319	46680	4740
16	7500	364796	44640	3060
17	8100	360836	47400	2820
18	7800	350332	48060	2400
19	6900	363055	51900	1920
20	7800	211916	42600	4320
21	6900	290579	47880	3300
22	9300	336892	48000	1860
23	7500	269639	48180	2940
24	9000	319138	43140	2460
25	7800	338997	51060	1860
26	7800	384298	47580	3060
27	7800	317280	47220	2340
28	7500	383458	53400	4140
29	8100	286618	46920	2040
30	7800	442675	52800	2880
31	8100	385380	50640	2580
32	7800	425876	51900	3060

Figure 32. Version 011.009 Trial Runs 000 (left) and 001 (right). From left to right the columns are: device number, downtime, key time, local decrypt time, and unauthorized time.

References

[1] De Santis, Alfredo & Desmedt, Yvo & Frankel, Yair & Yung, Moti. (1994). How to share a function securely. Conference Proceedings of the Annual ACM Symposium on Theory of Computing. 522-533. 10.1145/195058.195405.

[2] Yi, Seung and Robin Kravets. “MOCA : MObile Certificate Authority for Wireless Ad Hoc Networks.” (2003).

[3] Libert B., Yung M. (2011) Adaptively Secure Non-interactive Threshold Cryptosystems. In: Aceto L., Henzinger M., Sgall J. (eds) Automata, Languages and Programming. ICALP 2011. Lecture Notes in Computer Science, vol 6756. Springer, Berlin, Heidelberg.

[4] Di Pietro, Roberto, Luigi Vincenzo Mancini, and Giorgio Zanin. "Efficient and adaptive threshold signatures for ad hoc networks." *Electronic notes in theoretical computer science* 171.1 (2007): 93-105.

[5] Xu, Gang, and Liviu Iftode. "Locality driven key management architecture for mobile ad-hoc networks." *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on*. IEEE, 2004.

[6] Huchton, Scott, Geoffrey Xie, and Robert Beverly. "Building and evaluating a k-resilient mobile distributed file system resistant to device compromise." *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*. IEEE, 2011.

[7] Caronni, Germano. "Walking the web of trust." *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.(WET ICE 2000). Proceedings. IEEE 9th International Workshops on*. IEEE, 2000.