StrongHold: A Secure Data Platform for Smart Homes

by

Jaques Clapauch

B.A.Sc., The University of British Columbia, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE
in
THE FACULTY OF GRADUATE STUDIES
(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

July 2013

Abstract

Security is an important concern in day to day life, far augmented when related to sensitive data, such as private information. As such, it is paramount to protect environments and devices wherein the greater amount of information is private, such as in a home, and its encompassing devices; especially so, as the level of privacy desired in a home is much greater than other private mediums. The issue is further exacerbated in a smart home scenario, where applications are ubiquitous and are constantly communicating with the outside world: this impending technological innovation results in previously private data suddenly becoming accessible through non-physical means, allowing potential breaches in privacy. The accessibility to previously unreachable means brings forth new threats that must be tackled to ensure proper confidentiality is kept. Furthermore, proper accessibility of the physical devices must be engaged, due to their newfound non-physically accessibly nature. We survey the security threats existent in smart home environments, along with possible solutions to mitigate the threats. We use methods ranging from human computer interaction, storage optimization, and behavioral learning to better ensure proper functionality. We integrate the solutions in a cohesive form to be applied to any smart home environment that wishes to best keep high confidentiality, availability and integrity. We test the system to ensure the secure data messaging platform provides sufficient throughput for high definition media storage in real time, as potentially necessary in a smart home

We supplement our study by creating a system capable of functioning on top of the smart server, which unobtrusively automates the daily task of recipe selection, via meal advocation through past meal selection; we do so unobtrusively in an attempt to prevent deviation from conventional home environments. We present a method which possesses higher granularity than the present meal recommendation technologies, and yet, requires less interaction than the web equivalents. Finally, we interface the algorithm to a real smart home server, evaluate the application on real users, and assess their reaction to the technology.

Table of Contents

List of Figures

Acknowledgements

I would like to extend my gratitude to those without whom this work would not have been completed. First and foremost, I would like to thank Dr. Sathish Gopalakrishnan for the idea, support, and instruction, without which this thesis would be no more than a mere concept. He has taken me in, and introduced me to the fascinating world of research, which has since become an integral part of my life. His passion for teaching and research, and his expertise in a vast number of areas have been a paragon of excellence which I hold as one of my greatest aspirations.

I would also like to extend my gratitude to Dr. Karthik Pattabiraman – who repeatedly aided me in my pursuit to extend my knowledge in engineering and beyond, and supported me every step of the way. Dr. Pattabiraman’s support has been instrumental for me: his knowledge, direction, and advice have been invaluable; and Dr. Kostantin Beznosov who inspired me to pursue the field of computer security, and sparked the inception of the security aspect of this thesis.

I would like to acknowledge the support of Ildar Muslukhov, Yazan Boshmaf, Anna Thomas, Shane Wang, Bader Alahmad and Theepan Moorthy, in terms of constructive discussions and ideas for furthering this work. Furthermore, I would like to thank my friends Augustine Wong, Bar Perry, Danial Tkach, Daniel Gilmour and Elan Polo for their tremendous support and help, and Elliot Barer for his continuous technical assistance. I would also like to thank Avi Jacobus, Amy Goodman, Salomon Saade, Sarah Barmazel, Tamara and Shiri Stern, Vadim, Yulia, and Yaniv Bronshtein, and Yoni Dayan, for being there. I am also indebted to my late mentor Reuben, and my mentor and friend Shamir for setting me in the path of science.

Finally, and most importantly, I would like to thank those whose unending encouragement and care have allowed me to be successful in my endeavors: my mother, my father, my aunt Marisa and uncle Roberto, my grandparents, my cousin Bruna, my aunt Ruth, my uncle Edward, and my girlfriend Maayan.

Dedication

To my role models: my parents, José Henrique Clapauch and Silvia Hoirisch Clapauch, my grandmother Regina Léa Clapauch, my grandfather Luiz Clapauch, and my late grandmother Frida Hoirisch.

Chapter 1: Introduction

1.1 Smart Home

The concept of smart environments and ubiquitous systems has been in existence for considerable time in literature [1], seldom making a substantial transition to real environments, beyond those transitions to positional goods [2, 3, 4, 5]. The lack of an all-encompassing infrastructure, allowing for collaboration between different manufacturers, as well as little emphasis on security efforts has hampered progress in this area. These topics nonetheless embody some of the most sensitive issues apposite to smart home environments: a smart home necessitates collaboration amongst devices, and unless the user only buys from one manufacturer, he will experience significant difficulty (if not outright impossibility) establishing inter-communication between devices of different manufacturers [6]; moreover, the sensitivity inherent in private home information brings about a number of security issues that must be tackled before the environment is allowed to be translated to conventional living quarters [6, 7, 9, 10, 11].

Home environments may ultimately be the most private of environments, and as such, much that occurs within it may be thought of as sensitive, and undesirable to be leaked to anyone outside. Further issues arise when determining which users are considered to be outsiders, and which ones should be granted access regardless of their residence or lack thereof [9, 12]: for instance, what should house guests, or even doctors and policemen be granted access to, and for how long? It is also of importance to understand what to protect: as much as one would not desire for their computer to be made public, lights or a toaster may not require considerable protection. Lastly, the means of access is also an important issue that must be addressed: how can technologically unskilled users securely access all these devices, without resorting to obtrusive techniques [9]?

Additional issues arise when attempting to protect the devices rather than the user. Ubiquitous devices may not all be trustworthy, and as such, malicious behavior must be prevented, particularly considering that embedded devices generally are lacking in resources, or rely on battery power [6, 13]. Furthermore, devices warrant their own communication methods, differing considerably than those required by users, further adding to the complexity of the system.

Though challenging at the time of inception, recent upsurges in technology (such as the proliferation of resources like smart phones and powerful computers) have appropriated sufficient resources to more easily turn the idea of smart environments into reality [8]. As those devices rise in popularity, a demand for more interconnected environments rises concurrently [7]; furthermore architectures must be devised to meet those demands, while maintaining usability and security equivalent, if not superior to non-smart environments. These architectures must allow for easy interactions by the computer illiterate, while still managing challenging tasks, such as priority regulation and escalation, device installation, secure communication, and interfacing. The abstraction of this distinct set of characteristics alone presents a significant challenge. Meanwhile the integration of the smart home would additionally require simple, comprehensive interfacing methods for the devices connected, as well as unobtrusive operation. As a further obstacle, this entire process should not require more than commodity equipment currently present in the home.

1.2 Smart Devices

Perhaps of the more interesting new applications of smart devices possessing sensors within the home is that of recommendations. Though recommendation systems have recently experienced incredible progress [14, 15, 16, 17, 18, 19, 20] they have by and large still required considerable amount of user interaction to extract viable information pertaining to recommendations. Furthermore, this information has been sporadic in nature, rather than continuous, likely missing valuable data points [16, 17, 18]. Smart devices located in the home have the potential to overcome that deficiency by automatically extracting the data as well as removing the burden of data input from the user.

Beyond improved data collection capabilities, smart devices have the capacity to increase immersion, allowing for direct communication with devices capable of acting on the recommendations, rather than requiring interactions with a less accessible website. Furthermore, the presence of recommendations may lead to better features stemming from the inter-communication characteristic inherent in ubiquitous systems; that is to say, the recommendation system intrinsic in one device may allow for better functionality in a coupled device: for instance, a refrigerator may recommend a recipe which the stove may cook for the user prior to arrival.

In comparison to other literature, such as video, music, or document recommendation, little work has been done in terms of recipe recommendations [21, 22, 23, 24, 25, 26, 27, 28]. Effort has been expended more-so to translate collaborative-filtering into the medium than to concentrate on the content-based nature of food recommendations; perhaps missing a very effective method of discerning and grouping food items for recommendation.

1.3 Contribution of the Thesis

In this thesis we address the two aforementioned topics. Firstly we suggest possible solutions to security, inter-communication, and usability issues born out of smart environments. Though a number of works have aimed at solving single issues related to all of these topics, an integrated solution was lacking; we present said solution here, adding to it further inter-communication and security features engendered by needs pertinent to producing a viable smart home, as yet unavailable in the literature. Features added include secure storage by devices and access by both devices and users, allowing for web-based interfaces for easy communication, as well as optimization features and a learning algorithm to allow for simple interfacing by users. Finally, we build the aforementioned system and test it against select benchmarks to prove viability.

Beyond the smart home server, we also tackle the creation of a smart device capable of recommending dishes intrinsically based on past dish preferences. We use a high level of granularity via a translation of the document advocation model to a meal advocation environment, and postulate more effective methods of meal advocation than those currently present. We further allow for an automated retrieval of food items, decreasing the burden on the user, while increasing granularity of input values. Ultimately, we test various methods of meal recommendation against each other to ascertain which technique is the most effective.

Lastly we test the smart device within the smart home server environment previously devised and determine the viability of the server in terms of storage, communication, and usability.

1.3.1 Primary Contributions

Through this thesis we create a usable platform, capable of encapsulating all needs pertaining to a practical smart home. The platform spans storage, security, and usability methods, facilitating full integration of devices, regardless of manufacturer; the methods advanced are, to our knowledge entirely novel for the field. We introduce a new, automated privilege escalation method, which removes from the home owner both the burden of set-up and escalation, greatly facilitating guest access control, as well as conventional access control by computer-illiterate users, such as those likely present in eHealth Homes. The method is further protected from attacks by timed escalations and de-escalations, and the burden of access is transferred to manufacturers. As a separate outcome we also introduce an Application Programming Interface (API) which abstracts the storage and communication modules in the home, and is accessible via any network-capable language.

We engineer a meal recommendation method embedded into a refrigerator which, unlike current equivalents, borrows from the document recommendation model and explores items more in depth, allowing more proficient retrieval. This method makes use of item weight as a metric for tag construction, leveraging much greater granularity than present methods, a yet unexplored option. The meal recommendation method moreover is made so as to follow the previously mentioned principle of ease of use, allowing no detraction from usability compared to normal fridges, while providing suitable recommendations to users; in contrast, present methods require considerable data entry effort to achieve lesser results. Finally we combine the two previously mentioned modules allowing for a more in depth exploration of smart home environments than is available in literature currently: coupling platform and application.

1.3.2 Smart Home Server

With the aim of producing a viable middleware to manage security and communication within the home, we survey relevant literature to establish extant challenges. We make use of solutions presented in surveyed works, and further determine areas necessitating improvement from literature and posit on how to do so. Following that, we suggest and implement methods to make the presented options usable by laities, as well as provide methods to improve operating speeds. We thus create a system capable of operating not only in test and laboratory environments like its predecessors, but also unlike the precursors, in real world environments. The method provided follows RESTful architecture [29], permitting easier interfacing for device manufacturers, and allowing freedom of programming language selection.

The server presented is devised to support a vast number of features (such as the choice of push or pull communication, different security levels, and storage methods), creating an adaptable experience. Ultimately, it is our belief that the elastic nature of the features and abilities presented by the server, should make it more desirable, attracting different manufacturers to a universal solution. We also advance that by automating most communication methods on the user-side, and removing burdens such as priority settings or escalations and guest set-ups, we enable the system to be used by a vaster group, allowing it to be finally appropriate for use in real home environments. Of foremost importance, we achieve these goals while maintaining or improving security techniques presented in the literature; especially more-so considering that since security features are automated, users cannot forgo them due to implementation difficulty or usability issues. Lastly, we demonstrate that the postulated system is practical for a real home environment, presenting suitable performance on outdated hardware to further establish the system's practicality.

1.3.3 Meal Recommendation

Determining the optimal method of recommendation for a home environment proves significantly trickier for a number or reasons: though the literature is vast for the general recommendation solution, few works translate it to the food model; moreover, the few translations presented only take into account either collaborative filtering or a very broad representation of content-based filtering, disregarding individual ingredient importance; lastly, no solution presented targets an intrinsic smart home environment where data retrieval is to be collected automatically, rather than manually, and continuously rather than sporadically. We undertake these challenges, and provide different methods to supply the given recommendations, making use of collaborative filtering, a more inclusive content-based filtering, and hybrid methods; some of which are directly translated from news advocation [17,18], and document advocation techniques [18, 30, 31, 32]. These methods are modified to satisfy the needs pertinent to the smart home environment, and more intrinsic data collection. We further interface the meal recommendation device (developed to be housed within a refrigerator) with the previously introduced smart home server to demonstrate functionality and ease of implementation. The algorithms are then tested by users, and the best recommendation method is determined via user rankings. Finally the inferences drawn out of these rankings are not only intended to advance the specific field of food recommendation, but also introduce a better base upon which future methods relative to meal advocation (as well as other similar advocation methods) may build upon to produce better, more reliable recommendations.

1.4 Organization

The remainder of the thesis is organized as follows: We introduce the StrongHold data messaging platform in Chapter 2; in this chapter we present our implementation, elucidate on the logic behind implementation choices, and showcase StrongHold capabilities through select benchmarks. In Chapter 3 we introduce the Meal Advocation for the Smart Home Algorithm (MASHA) and present results from user trials, concluding on its capabilities. Chapter 4 concludes the thesis, through a summary of our contributions and possible future work.

Chapter 2: StrongHold: A Secure Data Messaging Platform for the Smart Home

2.1 Introduction

Given recent advances in technology [8], the use of internet connected smart devices is on the rise, leading to a logical rethinking of the appliance paradigm. Along with technological advances, the idea of a connected home is rather an appealing concept to Americans according to a recent survey [7]. Furthermore, the notion of delegating challenging activities to smart appliances, from cooking to taking care of the elderly, seems rather an appealing development. Due to those factors, one would expect that a smart home would before long turn from a distant dream to a present commodity.

Owing to the large amount of data aggregation and transmission necessary for the proper operation of the smart home, developers must also concern themselves with both an optimal means of aggregation, as well as with the security risks involved: the vast number of real-time embedded applications incumbent in the proper operation of the Smart Environment, mostly necessitating ready information for seamless operation brings forth a challenge of easy aggregation and effortless communication with users often lacking knowledge of computer systems; furthermore, those users communicating with the devices would prefer their private data is not made public; nor would they like attackers to take control of their devices. With the advent of smart homes and appliances, one must then concern themselves with accommodating for these problems (and due to the mostly theoretical nature of smart homes, as they are yet to be widely deployed) and the devising of possible threat scenarios, coupled with solutions to these.

Smart homes, unlike office environments are not expected to feature skilled administrators, but rather average, often technologically unskilled, people. As such, the designers cannot expect the user to spend much time learning complex interfaces or learning how to perform or delegate security tasks [9]. Owing to this, controls and security settings should be simple and unobtrusive, as well as secure.

In order to address the previously stated concerns, we present StrongHold: A service aimed at secure storage, aggregation, and retrieval of data within the smart home. We present with StrongHold, the RESTful [29] API used to communicate with this server from both client and user perspectives, and elaborate on our choices. We also present the optimized algorithm which allows data aggregation and retrieval to remain rapid given a range of accessing devices. We expound on the means which allows the server to be usable by computer illiterate users through the advancing of a learning algorithm capable of escalation and de-escalation of rights dependent on server use. Lastly, we evaluate the functionality of StrongHold in real world circumstances. In doing so, we hope to conceive the first fully functional secure data messaging platform for both experimental and practical smart home environments.

2.2 Background and Motivation

The smart house template adopted for this work consists of several devices interconnected to one or more central stations. Each device may serve the purpose of a sensor, a hotspot or a servant [10]. A sensor would comprise a device that merely accumulates data, whereas a hotspot would be a device used for inter-communication between devices, and servants are the devices used to compute results.

Devices within the smart house need not be fixed to the house, but may be able to be removed from the smart home environment and used in a mobile manner (for example a mobile phone or a health monitoring device, such as a heart rate monitor) [10]. Devices in the smart home, much like devices in our own present homes should exhibit features of ownership, such that an unauthorized user should not be able to access a device he does not have the rights to use (such as one's own personal computer, or even the door lock mechanism) [9]. Continuing on the previous point, devices should be enabled to be used by more than one user if necessary. It is further necessary to allow external administrators and other outside users (such as policemen, repairmen, firemen, or simply network administrators) access to certain devices even when the users are not present [9].

It is also important to note that devices should be ultimately cost effective. As such, any device included in the house should not have any security feature severely impact its monetary cost [11]. Continuing in the cost paradigm, it is also assumed that not all devices can deal with high levels of computation (such as advanced encryptions) due to their embedded status, owing to primitive resources and battery-powered condition [6].

From current trends in the industry it can be foreseen that not all devices in the house will be from the same manufacturer, and thus some devices may possess similar protocols for communications (such as a standard), but we cannot assume that all devices will do so [6].

Under the same previously stated assumption that we cannot presume users are technologically skilled, we must also allow for simple interfaces with pre-defined customization abilities not only for security reasons, but for normal use as well [9].We should also assume that in such a template house, all electrical devices capable of exhibiting “smart” behavior will do so, and communications need not be solely within the house, but data may be transmitted outside (i.e.: not simply within devices). Lastly, though connections such as ZigBee and Wi-Fi are the norm in the smart home environment [33], other connections, such as wired connections or Bluetooth can be present [34].

In this work, we devise a server capable of encompassing all the concepts mentioned in this section, while able to protect from all threats mentioned in the Related Work section. As such we attempt to formulate a central device able to manage the home’s communications and data, as well as properly protect them from both external and internal attacks, while maintaining a simple to use interface for devices and users, ultimately allowing for standardized access by devices (regardless of manufacturer) as well as easy access and setup to technologically inept users. In order to allow for full protection from attacks, many concepts mentioned in the papers presented in the Related Work section are combined and new concepts absent in the literature are presented: the importance of this step lies in that although many papers exist in the smart home literature, little effort is employed in constructing a fully working smart home server; at most a work will present and implement a few ideas, and very few attempt a conjoined, fully implemented smart home server which tackles all concerns. Finally, it is our aim to provide fast access to stored data within the server, thereby eliminating any bottlenecks, even assuming high device or data load.

2.3 Threats

2.3.1 Ownership Threats

Of the most characteristic features inherent in the smart home is that of the enhancement of beneficial application attributes (for instance, having a stove cook by itself, or a wardrobe sort itself, or temperature to adjust itself) without any detrimental aspects being added. Of those, one of the most important aspects is to keep proper ownership; that is to say, the disallowance of non-permitted users from using things that do not belong to them. Some unknown attacker, as such, should not be able to control your stove from outside the house. This is indeed a problem in a smart house: an environment where communication with external networks is abundant and devices are made to provide external connections [6, 7, 9, 10, 11].

Another important point to consider is that of data ownership: Within the smart house (especially those specializing in eHealth) data is continuously recorded, be it by means of a camera (for security or eHealth purposes), or credit card information (employed when the user utilizes an online store feature of a device), among other things. Data recorded through those devices could technically be extracted by a skilled adversary if proper protection is not instilled. This category of data can be used to execute a number of malicious dealings, such as directly stealing user funds, mining user data, or even determining user presence [6, 11].

Data ownership threats also arise between house residents, rather than merely pertaining to outside attackers: in many cases, a resident of a smart-home will wish his actions to remain private from other residents in lieu of constant monitoring. The threat prominently arises in environments where film data is pervasive, and not merely processed in real time; for such environments, a specific resident would not want his house-mate covertly leaving a smart camera on in the room, and later querying and easily obtaining the other resident's actions. One's ownership of data should only extend as far as the data that is pertinent to him, and no more [35].

A further key threat is the possibility of malicious devices being introduced to the system. A malicious device could issue commands to hotspots demanding and propagating data the user may not wish to disclose. Furthermore, malicious devices may convert their non-malicious counterparts through several means, and thus monitor a non-consenting user [6, 13].

An ownership problem also arises upon third parties attempting to access private, though relevant, information: A doctor may, for instance require access to patient data that the smart house possesses, which on its own is a harmless procedure. A dilemma arises, however, when the doctor does access the information database; in such a case, he should only have access to pertinent information and no more [36, 37]. That is to say, we do not wish a caregiver to know more information than that which is necessary to care for the patient. The smart home environment is after all, very different from that of the hospital, as the hospital is not a private location; in the patient's home, the patient may engage his environment differently, and may not wish the caretaker to have access to all of the information pertaining to his actions.

2.3.2 Availability Threats

An important possible threat that should be addressed is that of denial of service: in many instances of smart houses, such as those related to eHealth, it is of the utmost importance that service is not halted, lest the patient being monitored be in danger. As such, any instance that disallows the service to the user should be meticulously protected against[1] [10, 11].

Correspondingly, since all existing items in the house are electronic, power or system failure poses serious safety threats to the user, as a user may be locked in or out, without access to any supplies (such as water, food or facilities) [38].

[1] It should be noted, however, that this is on a case by case basis: while it is important to protect against denial of service in an eHealth environment, it is not as important to protect one's toaster from being halted in producing toasted bread, for instance.

On the same note, resource limitation on devices, leads to vulnerabilities to threats such as “sleep deprivation attacks,” that is to say, continuous superfluous communication with the devices solely for the purpose of exhausting device battery [39, 40].

Furthermore, though less common, it is also possible to target the wireless networks through jamming attacks, where radio frequencies in use may be super-saturated. In environments such as a smart house, where communication is key for any device access, a well-targeted jamming attack can be catastrophic, completely crippling the entire infrastructure [40].

Similarly, the fact that many wireless protocols used by ubiquitous devices do not require re-authentication can be attacked via intermittent service failures, leading to man in the middle attacks [39]. A different, though relevant exploit was shown to target Vonage VoIP phones, where a short injection caused the service to shut down, and upon reset, the VoIP device was fooled into reconnecting to an attacker rather than the service provider [41].

2.3.3 Locality Threats

The nature of the smart home environment, especially in regards to its pervasive characteristic, opens the smart home to a variety of location information leaks: Due to constant monitoring within the house, the user’s presence in specific locations can be determined with ease, be it through access to sensitive information (such as videos, or logs), or simply electrical monitoring of specific locations in the house [35, 42, 43]. An attacker can further determine a user’s current interactions with specific devices from snooping the wireless connection, which though encrypted, can still leak source and destination information with an accuracy of up to 90% [44].

Additionally due to mobile monitoring, users may be followed and tracked through the clever placements of a number of devices meant to snoop a communicating mobile sensor [45]. Attacks such as these are already in existence, and have been proven to work: an example of such an attack is that of locating a runner using the Nike+iPod Sports Kit, which communicates with a user’s iPod to provide the user with their running information. The device does not, however, encrypt its unique ID, and therefore, an attacker using properly placed sensors, can easily determine a target's location [45]. The problem of leaked ID information is further exacerbated with the eventual prevalence of large smart environments, with much farther reaches than those of houses. Smart work-places, and even smart cities are the logical consequent of the smart environment evolution, and with their progression, the use of monitoring equipment, as well as the use of wirelessly controlled personal identification will become more prevalent. Therefore, whereas currently an attacker might only be able to accurately locate a person within a house, in the future, it may be possible to accurately locate a person within a city [35, 37]. This is problematic even in cases where an ID is not actively coupled with a name or other identifiers, as an attacker can de-anonymize the ID by simply searching for the most prevalent ID in a target's house [37]. [2]

2.3.4 Data Leaks

With the increase of wireless communication, data can be more easily snooped. Though one would think current efficient encryption mechanisms would prove sufficient to halt this threat, it is not always the case. In one specific situation, it was proven that with a prior database of films and 10 minute traces of wireless data, one could determine with 73% certainty which film was being watched streaming through the Slingbox hardware, even though the transmission was encrypted. Given a 40 minute trace, the certainty rose to 89%, with some specific movies performing as well as 100% certainty [45]. The same paper that discusses this attack also notes that this threat does not simply lie with television information leakage, but rather that "one can infer the origins of encrypted web traffic or infer application protocol behaviors from encrypted data." Therefore, due to the high number of devices communicating wirelessly in a smart home, one can conclude that a larger number of data should be leaked, even given encryption.

In a similar attack as the one presented above, Enev et al. determine that power supplies in modern television sets produce discernible signatures when it

[2] Due to the small reach of the StrongHold, this last array of threats are not undertaken; Since the StrongHold (with some exceptions) is meant to operate solely from within a house rather than environment, it should not be an issue which would affect it, and is therefore out of the scope of this work.

comes to electromagnetic interference, allowing for the determination of content being viewed [46]. Through the analysis of electromagnetic interference using a prior database of films and 15 minute traces, a cross correlation of same content of upwards of 98% was yielded, even in the presence of considerable noise. The authors of this work further hypothesize that comparable attacks may be conducted to determine similar information in regards to computers, DVD players, printers, game consoles and washing machines. This attack is quite likely in a smart house that saves and propagates electrical information due to smart-grids.

Another similar attack shows the possibility of revealing the language spoken in a VoIP conversation from encrypted wireless data [47], further implying that as long as devices communicate wirelessly, even with encryption present, some private data is not safe.

2.3.5 Other Attacks

Following is a list of attacks that do not exactly fall into exactly one of the previous extensive categories (though they may be a combination of several categories of threats), yet are still important and should be tackled.

1. Through online or physical means, an adversary may impersonate a user, and with the user's credentials he may gain access to unauthorized appliances [6].
2. On the same track as the previous point, an adversary might create false credentials to respond to a patient alert in an eHealth smart home. Not only should this grant him access to the user's abode, but the user will likely not be rescued, because the system processed the fact as having already occurred [10].
3. Continuing on the same point, a user who may control appliances from a distance may pose physical danger to the user.
4. Appliances might be compromised, and used as zombie machines [6].
5. Appliances might be compromised, and their trusted status may be used to issue an attack on the user's personal computer on the same network [6].

6. Compromised machines may misdirect their output, thus executing possible phishing attacks to the user [6].
7. Cameras and other recording devices may be compromised, and may be used for wiretapping purposes [6].
8. On the topic of confidentiality, and not as much an attack, the lack of credentials may lend a friend of a user (i.e.: one with physical access, but perhaps not direct permission from the user) the ability to see data he normally should not be able to[3] [9].
9. A trusted device may be manufactured with malicious code, thus compromising the entire house [6].

2.4 StrongHold

2.4.1 Approach

StrongHold was devised with the aim of providing a secure, easy to use, configurable, and ultimately ubiquitous main hotspot for the smart home. The general architecture of the application was inspired by the central password system presented by Naqvi et al. in their Infosphere paradigm [39]. However, unlike the Infosphere model, control of encryption is set by the device manufacturer to remove the burden of configuration from the users. This is specifically significant as it should be the manufacturer that better understands whether their device can handle specific forms of encryption, rather than a layman-user.

2.4.1.1 Authentication and Management

Initial set-up is carried out through a simple handshaking protocol initiated by the user, wherein manual control is necessitated [6, 11, 12, 13]. The set-up simply requires that the user manually type the device name, location (the latter designating where the device should be installed, e.g.: kitchen, living room, among others), and the encryption level designated by the device onto the server, at which point the server will yield a 128 bit key (or 16 characters in ASCII representation) and an IP

[3] Another interesting point to note (albeit not entirely related to security), some users further find it embarrassing to reveal to their friends or acquaintances that they are not allowed access in specific devices, so credentials should desirably be as unnoticeable as possible [9]

address to input onto the device. Once the user reproduces this information to the device, the exchange is finished and the devices may begin communicating; no further user interaction is required beyond this point; however, users are able to configure specific elements of the system if they wish to and happen to have physical access to the server. Such configurations include the ability to setup super-users, or manually escalate user privileges; this specific capability consequently allows emergency super-user account activations, wherein master passwords may be set prior to device installation, and sent to necessary law enforcement devices for access during emergency situations, or set temporarily for doctors performing periodic visits [9].

The presence of device location information within the server exists as an extension of the zones paradigm presented by Manish [34], wherein it is elaborated that a device should know its area and thus may modify policies based on the selected area (for instance, if a device is placed in an area deemed “living room”, maybe it could be more liberal in terms of guest access than its area was named “office”). This specific feature is left to the device manufacturer to implement; the process is simply a matter of scanning a dictionary of viable location names and modifying its policy accordingly.

2.4.1.2 Communication and Encryption

Communication between StrongHold and its client devices is encrypted using the Advanced Encryption Standard (more specifically, AES-128) [48], and the key utilized is resettable at any given moment through the Simple Password Encrypted Key Exchange (SPEKE) protocol [49]. AES-128 was chosen since it is simple enough for a diverse set of devices to use, ranging from 8-bit smart cards to high performance machines [50], both of which may be encountered within smart homes.

Tests employing an Arduino Uno coupled with a Wi-Fi Shield, and utilizing AESLib [51] (an open source AES encryption and decryption library), yielded promising results, including a set-up time of roughly 0.3ms, encryption time of 0.6ms per block, and decryption time of 0.7ms per block. This throughput allows for communication speeds ranging from 20-27KB/s, which should be more than sufficient for most communications. Obtaining an effective throughput in Arduino is fairly significant, as it is one of the most popular platforms for embedded systems as of recently [52].

Stemming from the variability of devices ever-present in the embedded systems world, the server apportions vast configurability, allowing different devices different communication methods and forms of protection dependent on their capabilities: devices may choose to forgo encrypted communication altogether (via setting its communication level at 0), allow it (via a choice of communication level of 1), or allow encryption coupled with a nonce (communication level 2) in order to prevent relay attacks. Furthermore, users may use SPEKE to reset the device password whenever required (from a reset at every message exchange, to never). SPEKE was chosen as the password reset mechanism, as it is achievable by older, weaker devices (it was devised in 1996, with devices from that epoch in mind), and has presented no known flaws since its conception [53]. The ability of the devices to continuously change their password (as per Argwal et al.) through SPEKE should prevent threats such as those which affected the Nike+iPod platform [45]. This same method can be said to prevent issues concerning the determination of film content streamed through an encrypted wireless medium, via a database, referred to in the same paper. Lastly, devices may opt for pull communication between it and the users, using the server as a middle-man. Through this method, the device may issue pull requests whenever it wishes, from a queue located on the server, thus protecting it from sleep-deprivation attacks.

StrongHold allows transient devices, as it functions solely via push communication from the devices. This methodology allows other devices to be transient while StrongHold remains fixed. Communication by users and devices is available within the house by simply joining the local area network it is placed on; External communication with users and devices placed outside the home (or too distant to participate in the Local Area Network) should be accessible, though for server placement behind a router, Network Address Translation (NAT) set-up in the router should be necessary. However, as accesses within the house should be normally protected by the router's own encryption methods (WEP, WPA, among others), it is believed that keeping devices and users within the house would ultimately be more secure.

2.4.1.3 Storage and Usability

A device may store two types of data within the StrongHold: device data, and user data. Device data designates data which is closely tied to device function; for instance, a fridge may store its contents as device data, whereas a camera might store photos. Conversely, user data designates both raw data accessible to the user as well as web-pages with which the user may interact to access the device and its contents; via HTML and JavaScript, a device may create a page which allows the user to directly communicate with the device without using the StrongHold as a proxy if necessitated. The pages may also load the raw data stored within the Server to better serve the users. Both device data and user data are stored and retrieved in the manner described in Appendix A.

Though customization from a user's standpoint is available, StrongHold was designed with technologically unskilled users in mind, and as such, a user should need no more than a web enabled device to communicate with the smart home devices. Access from the user simply requires that the user type the StrongHold IP address in a browser followed by his preferred username, delimited by a forward slash. This access directs the user with a splash screen which designates which devices the user has access to. Initial access of the server by a user also registers the user with the server, binding the user's IP address with his username. Ultimately this disallows rogue users from accessing devices while pretending to be another user, as manual access to the user-accessing device is necessary, following the "Big Stick principle" [37, 54]. The automatic setup is in agreement with Jonson and Stajano [12] wherein it is argued that a guest shouldn't have to be registered to facilitate both the burden on the guest and on the home owner [9]. Through proper accesses of the previously mentioned web-pages, the user's access privileges automatically escalate in the manner described in Section 4.2.3 of this chapter, pertaining to the learning algorithm [11, 55, 56]; conversely, improper access decreases the user's privileges.

2.4.1.4 Accessibility

For the sake of commodity, guests to the home are not required to have a password and their communications are not encrypted; only upon extended residence will a user be prompted to get his or her password from the server, at which point, they must interact with the home resident in order to acquire access to further features. The manual acquisition serves two purposes:

1. It disallows snooping from the network
2. It ensures the user is trusted enough by the home owner to be allowed to interact with the devices in a protected manner.

Once the guest communicates with the owner, he must install an application (if on a computer) or app (if on a mobile device) to handle communications and decrypt replies from the server. Prior to escalation, a guest only really needs a browser and a mobile phone or computing device.

Ultimately, user access to specific features of the device is left up to the device manufacturer. The device manufacturer as such is able to set up to 5 levels of access, ranging from Guest to Trusted Guest, to Resident, to Power Resident, to System Administrator. At each level the device manufacturer can instantiate different rules through different web pages. Furthermore, access can be granted solely to specific people through the customizable user controls, and for instance, a specific doctor may be granted access to only one specific device. Lastly, all user accesses are logged in the system, coupling a username and IP address with all issued commands. As such, any anomalous behavior may be easily investigated, and the presence of extraneous devices in locations where they should not be can thus be easily detected and prevented.

2.4.2 Implementation

StrongHold was implemented using Java, and has been tested and works in Windows, Mac, and Linux Operating Systems. The server was coded using the HTTPServer and the Java-JSON libraries. As such, no outside software is necessary to run the application, and the code is standalone. The process of coding from the ground up further alleviates the user's inconvenience, as they need not concern themselves with third party software, such as proprietary databases or server applications.

2.4.2.1 Application Programming Interface (API)

StrongHold hopes to set a standard so devices from various manufacturers may inter-communicate, following Pishva and Takeda's view [6]. It does not discriminate by manufacturer, and, assuming non-malicious manufacturers, should allow proper, easy to use storage and interfacing options.

The Application Programming Interface follows RESTful design principles, wherein each additional URL layer in the URI signifies a resource, with each additional layer further specifying resources. Each URI further follows the Create, Read, Update, Delete (CRUD) principle inherent in RESTful APIs. Additionally, parameters and attributes of each state are placed behind a question mark, signifying further options to the query [57]. Appendix A illustrates the functionality of the Application Programming Interface for StrongHold.

2.4.2.2 Storage Optimization

As the usage of the StrongHold increases it is important to devise a method to leverage data storage and retrieval under load. Sample use cases are those involving various media devices necessitating the storage and retrieval of megabyte or greater media files. Particularly considering the fact that data retrieved from the server may comprise of more than one data point per iteration (such as for video streaming), it is important that expedient retrieval of sequential items be possible. Digital cameras at the moment, especially those using RAW formatting have long surpassed the single megabyte range, and stored videos and audio may also be large in size. As such, retrieval of specific entries would take excessive time to navigate through all the entries in disk to locate the desired item[4]. At worst case, a search would incur a runtime of O(n), where n represents the number of entries. With a large number of entries consisting of megabytes each, this search would be rather time consuming. Owing to this issue, we have devised an optimization method

[4] The usage of disk storage here rather than memory is explainable by the same process as search latency: as the number of large files increase, their placement in memory would be problematic, and would affect performance for other devices and other programs running in the server machine.

which decreases the disk runtime to constant time (specifically a worst case of 24 items), and incurs O(log(n)) worst case run time onto memory.

Inspired by the work of Rozier et al. [58] regarding the placement of slices of disk in memory when necessary, we have allowed for a file system, with a file size limit of 24 entries. The entries are placed in order in each file, and each file is named for the first entry's ID, appended with the number of items within the file. If the addition of an item to a file would set its size to greater than 24, the file is split into two files of 12 items each, the first file's order preceding the second, and each file still possessing all files in order. Deletion of an item might lead to one of four consequences:

1. The item is the only item in the file, at which point the file is deleted.
2. The item is the first item in the file. The item is deleted, the file name is modified to reflect the new first item, and the number of items within the file name is also updated.
3. The item is at a non-initial position in the file. The item is deleted, and the number of items in the file name is updated.
4. The item does not exist, and an error statement is returned to the device.

If an entry is to be placed and it is the first and only item in a folder, a file is created to accommodate it. All file names are then stored in memory in a Red Black Tree, keeping the IDs in order, and allowing for search, insertion and deletion worst case runtimes of O(log(n)). Each Red Black Tree representing files within a folder may be found through a hash of the folder address. Through this implementation, the search for a specific ID translates to a constant time hash, a logarithmic time search through a Red Black Tree, and finally, a constant time search through a file on disk. This is a large improvement from the linear search in disk. Figure 1 further illustrates the concept. This placement in memory occurs at item instantiation or item search, since it is plausible the server might shut off at a point due to power failure or other purpose, and it is burdensome to re-load the entire file structure to memory at set-up.

The file limit of 24 entries was elaborated as it allows for the industry-standard frame per second requirement for film streaming [59, 60]. As this optimization was developed with sequential retrieval in mind, the large limit[5] of 24 was chosen to match one of the more speed-essential applications of storage: that is, streaming film, wherein each data point consists of a single frame. For non-sequential storage and retrieval, StrongHold offers the option of individual storage through the path feature, as exemplified in Appendix A.

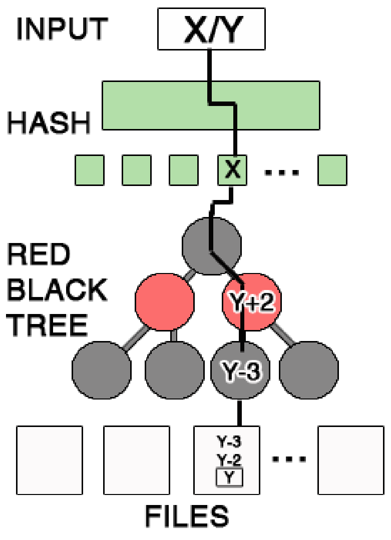


Figure 2.1 Illustrated Optimization Method

2.4.2.3 Learning Algorithm

In order to ensure user policies not need taxing interaction by laypeople, an adaptive policy was devised, allowing for minimal setup or user interaction. The modularization of groups, thereby initiating policy groups was inspired by Kim et al. [9], however, unlike their work, the policy groups may be either automatically set by the server or manually set by administrators, allowing for a level of granularity: if users are laypeople they can rely on the server's algorithm alongside the device policies; conversely, if they have technological prowess they may set it manually.

[5] We consider 24 a large limit, as any increase in this value will substantially increase file storage and retrieval (at each increase in limit the cost of storage increases by N at worst case). We must, therefore, be careful in selecting a suitable value, precisely following the Goldilocks Principle: that is to say, we must determine the best possible value, as going too low might be too cold, and too high might be too hot.

Further inspiration came from Hoque et al. [11] and Seigneur et al.'s [56] methods for privilege escalation. The method for escalation is set as follows: A user's initial GET request to the web resource will set up the user by linking the user's IP with the selected username (i.e.: the string the user appends to the URI), thereby inserting a soul into the device as per the resurrected duckling protocol. With the initial identification, the server will store a timestamp and set the user's experience points value to zero. Accesses so far are unencrypted and require only an ordinary web browser for communications. Every access by a user which does not yield an error will escalate the user's level by incrementing his experience points by 1 [55], assuming one circumstance: that the user's experience points not have been incremented within the past 30 minutes.

If the user's experience points have been incremented in the past 30 minutes, he is not deemed ready and as such, his experience is not evolved. Accesses which yield errors, however, such as trying to access devices which the user is not allowed to, devices or areas which do not exist, or web or data files which do not exist will result in experience points being decremented by 1.

Access to disallowed devices are deemed inappropriate, since a normal user should not reach that request through the splash page, and as such is likely attempting to knowingly access a device it should not. Likewise, access to devices or web files which do not exist may be an attempt to map the server, and as such are deemed improper. The only error statement inducing request which does not decrement a user's level is an access from a user's username with an improper IP address – This does not decrease a user's experience as it is unfair to users who had their names hijacked, and could be used as a denial of service method from malicious users to ordinary users. The time value for decrements is much stricter, consisting of five minutes intervals, unlike the lenient 30 minutes given to increments. The 5 minute value was chosen to discourage attackers from trying to do too much too fast. Likewise, the 30 minute granularity was chosen to discourage users from trying to access devices by escalating their privileges via an automated accessor [61]; the amount of time a user requires to escalate himself (often days or months) should allow sufficient time to notice something amiss in the logs.

Finally, when a user reaches the level of Resident, he is prompted to communicate with the administrator to retrieve his password from the server. This necessitates that a resident approve of the transition from guest to resident, thus disallowing rogue users from accessing sensitive information. At this point, the user must install some form of decrypting application to communicate with the server[6]. This follows the paradigm wherein a guest need not be burdened with set ups, meanwhile, a resident may incur light burdens.

2.5 Evaluation

In an endeavor to quantify the proficiency of select server features, and showcase suitable usability, several use-cases were created to which StrongHold has been subjected. This specific exposition ventures to demonstrate StrongHold abilities, and to prove its feasibility in a home environment. In running the experiments we strive to establish the following:

- StrongHold is capable of multi-platform communication.
- StrongHold performs satisfactorily when affected by computers and networks under heavy load.
- StrongHold can handle files in real time.
- StrongHold can handle files pertaining to real world scenarios.
- The security measures introduced onto the StrongHold system yield negligible delay, especially when compared to other inherent delays, and even considering larger files.

We also wish to elucidate the following issues:

- What features should be selected given differing file sizes and structures?
- How much of the impact in transmission performance is due to system specific functions and features (such as storage and encryption) as opposed to natural network delays?

[6] The creation of this application is outside the scope of this paper as it deviates from the personal computer platform (requiring coding in iOS, Android, or Windows mobile), however it should require no more than simple encryption and decryption algorithms and network communications.

- What is the expectant throughput in terms of storage and retrieval for differing file sizes?

To answer the posited questions, we decide to set a number of benchmarks pertaining to likely data to be transmitted within a smart environment. Rather than comparing equivalent storage technology speeds, we strive to simply determine whether we achieve throughput necessary for the feasibility of high definition streaming: though achieving faster results than any competing technology is not undesirable, it should not be a necessity for proper operation of the smart home, as such we rather simply determine whether we provide enough throughput for suitable usage. Though all data is streamed through a wireless G router, we do not focus on Wi-Fi streaming issues, as they are orthogonal to this work.

2.5.1 Experimental Setup

Unless otherwise stated, all experiments have been performed using a Windows 7 laptop with a 2GHz Intel i7-2630QM processor and 8GB of memory to house StrongHold, and an OSX 10.6.8 iMac with a 2.8GHz Core 2 Duo processor and 2GB of memory to house the client[7]. Files transferred to the computer are grouped as follows:

1. **MP3** – The MP3 collection consists of 12 files, with sizes ranging between 2.53 MBs and 12.43 MB, with a mean of 8.89 MBs, and a standard deviation of 3.22 MBs. As can be seen form these values, the sizes lean towards files greater than 8 MBs. The total benchmark size is 106 MB.
2. **PDF** – The PDF collection consists of 35 files, ranging from 37.7 KBs to 40.15 MBs, with a mean of 3.05 MBs, and a standard deviation of 8.31 MBs. The PDF files were chosen specifically due to the inherent large deviation, allowing for an analysis of StrongHold behavior due to large variance in input files. Though the file sizes lean towards those of less than one MB, three file sizes greatly exceed the 10 MB range, including one file of 17.82MBs, one file

[7] The choice of such an outdated computer as a client (the iMac is as of now over 5 years old), was reasoned as a means to demonstrate the usability of the client computer as feasible even in outdated technology. It should be further noted that even current generation cellphones now surpass the performance of this computer [62].

of 25.54 MBs, and one file of 40.15 MBs. The total collection size is also 106 MB.

3. **Photos –** The photos collection consists of 110 JPEG files, ranging between 11.69KBs, and 6.39 MBs, with a mean of 3.65 MBs, and a standard deviation of 1.21 MBs. The files present in this collection are tailed more-so towards the larger files, with most of the collection ranging between 3 and 6 MBs, with less than 20 files possessing less than 3 MBs. Disregarding 10 of the pictures (the smallest ones), all other pictures were taken with a 12 Megapixel camera. The total collection size is 402 MB.
4. **Text** – Text files were files generated using a PERL script to create 150 random files consisting of web-safe (UTF-8 compliant) text. All files consist of 1MB exactly. Web-safe files are necessary as StrongHold yields an error when presented with non-ASCII data to prevent buffer overflows, and files otherwise must have non-ASCII characters replaced with their ASCII representation via filtering.
5. **Frames** – Frames were likewise generated by a PERL script to generate 150 random files equal to the necessary size for transmission of one 1/24th of a second video frame, considering a 180MB, 720p video, which runs for 23 minutes. All files consist of 150KBs exactly.
6. **Film –** Film consists of one 180 MB, 720p video which runs for 23 minutes.

Before files are sent, a number of transformations must first be performed, dependent on client configuration: unencrypted communication must be UTF-8 encoded before transmission in order to be web-safe; encrypted communication requires filtering[8], followed by either AES encryption or AES encryption and a nonce request and transmission; since all files are initially stored in the hard disk, we also account for time to load the file from hard disk to memory.

[8] Though AES encryption already makes the file UTF-8 compliant, the underlying file may still have unsafe characters used by the server (such as +, /, &, =, ? and \n), which must be transformed to UTF-8 format. Since this process is not as strenuous as a full UTF-8 conversion, it is deemed filtering, rather than encoding.

In order to simulate the performance of a real home network, the computer housing StrongHold continuously played a DVD while executing, and an HD film was streamed through Netflix though the network to a tertiary computer.

2.5.2 Sequential Transmission Results

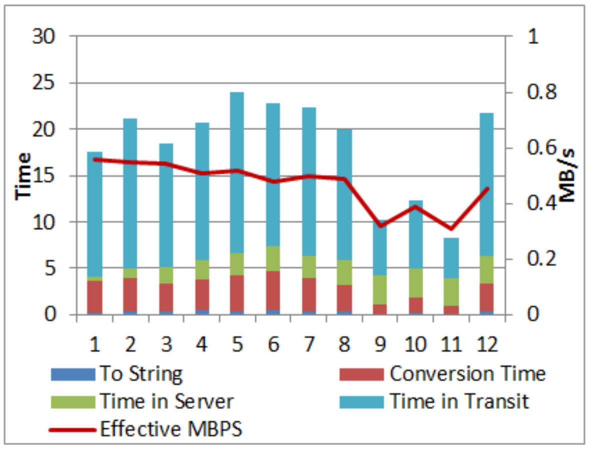
Sequential transmission consisted of first clearing all storage space in StrongHold, then transmitting all files to that space in mode 0 (unencrypted), mode 1 (encrypted) and mode 2 (encrypted with nonce), without any extraneous options. All leading times were then recorded, and portrayed through graphs. Time to nonce has been consistent throughout, and rather than including it in every iteration of the graphs, in order to save space it is declared to be on average 0.04 seconds per request.

2.5.2.1 MP3

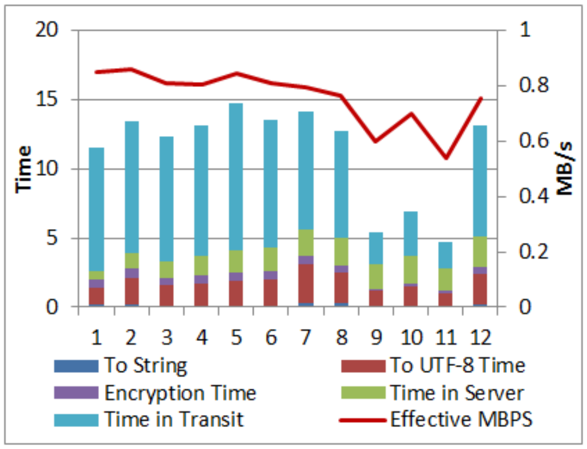
The MP3 collection allows for a representation of the transmission of consistently large files to the server. They furthermore allow for an exploration of the feasibility of the transmission of actual MP3 files to StrongHold. Since MP3 files contain web-unsafe characters, they needed to be converted when sent unencrypted, and filtered otherwise. UTF-8 encoding incurred a file size increase of 147% on average; filtering increased the file size by 4.4%, and AES encryption increased the file size by a further 33.4% on average, for a total increase of 39.25% file size increase when transmitting encrypted files. The significant bloating due to UTF-8 is due to the number of web-unsafe characters produced by MP3 encryption, since it contains a vast variety of characters outside the printable ASCII range. Ultimately, owing to the large increase due to UTF-8 encoding, encrypted transmissions outperformed the unencrypted counterpart, even though they necessitated both encryption and decryption on top of transmission and storage.

Figure 2.2.a demonstrates the performance of unencrypted MP3 transmission over the wireless network. “To String” represents the time to load the file from the hard disk into a string in memory; “Conversion Time” represents the time to encode the string into its UTF-8 representation; “Time in Server” represents the time the file took to store in the server; and finally “Time in Transit” is the total time taken to transmit the file from the client to the server through the network. As can be observed, each transmission transaction is mostly dominated by network latencies, with the server storage time only being significant for files 9, 10, and 11, which are smaller files. This is due to the fact that the storage algorithm re-orders files for more expedient retrieval: since these files were preceded by files as much as 5 times larger, they are subject to relatively longer delays as their storage time is overshadowed by the re-ordering of larger files; if all files are of the same size, or if the number of files exceed 24, there is a noticeable improvement to the algorithm after every 24th file, as seen in the later benchmarks. The mean throughput of the transmission is 0.467 MB/s, and conveys the transmission time accounting only for the initial file sizes (before UTF-8 encoding).

Conversely, figure 2.2.b shows the performance of encrypted MP3 transmission over the wireless network. This graph introduces two further metrics, non-existant in the first graph: “To UTF-8 Time” represents the time to filter, and “Encryption Time” represents the time to AES encode the string. The only significant difference between the transmission of encrypted and unencrypted MP3 files is the final file size of transmitted files; due to the size discrepancy, transmission time for files is greatly inflated for the unencrypted version, and server storage time is also magnified. Encryption time makes up an insignificant amount of the total time, and the only major effect it has on transmission time is due to the 33.4% file size growth. The mean throughput of the transmission is 0.76 MB/s, once again conveying the transmission time accounting only for the initial file sizes. Due to the reduced file size, throughput is greatly improved for this group.



(a) Unencrypted



(b) Encrypted

Figure 2.2 Sequential MP3 Transmission

2.5.2.2 PDF

The PDF collection allows for a demonstration of the effects of transmission of files constituting of highly variable sizes. Moreover, they explore the practicality of PDF file storage in StrongHold. Like MP3 files, PDF files are not web-safe, and require encoding when unencrypted, and filtering otherwise. UTF-8 encoding incurred a file size increase of 146% on average; filtering increased the file size by 4.7%, and AES encryption increased the file size by a further 33.2% on average, for a total increase of 39.65% file size increase when transmitting encrypted files. Like its MP3 counterpart, encrypted transmissions considerably outperformed the unencrypted in terms of throughput.

Figure 2.3 displays the unencrypted transmission of PDF files. While the larger file transmissions are dominated by transfer times, smaller files suffer more-so by the burden of the re-organization: since a vast amount of them was preceded by the large 40MB file, their server time is greatly augmented, leading to a significantly reduced throughput. The throughput is somewhat improved after the 24 file mark, but that is not enough to reduce the full weight of the previous lowered throughput. The mean throughput of this varied, unencrypted file size benchmark is 0.082MB/s.
Figure 2.4 shows the encrypted transmission of PDF files. In this instance, smaller files are less-so affected by the initial large ones (though they are still affected), simply because the sizes are greatly reduced. Nonetheless, storage still proves costly, yielding a mean throughput of 0.27 MB/s. It should be further noted that the 40MB file could not be transmitted in mode 2, as the timed nonce expired before transmission was complete, illustrating that this particular mode does not allow such large file transmissions. The 25 MB and 17 MB files however, completed successfully.

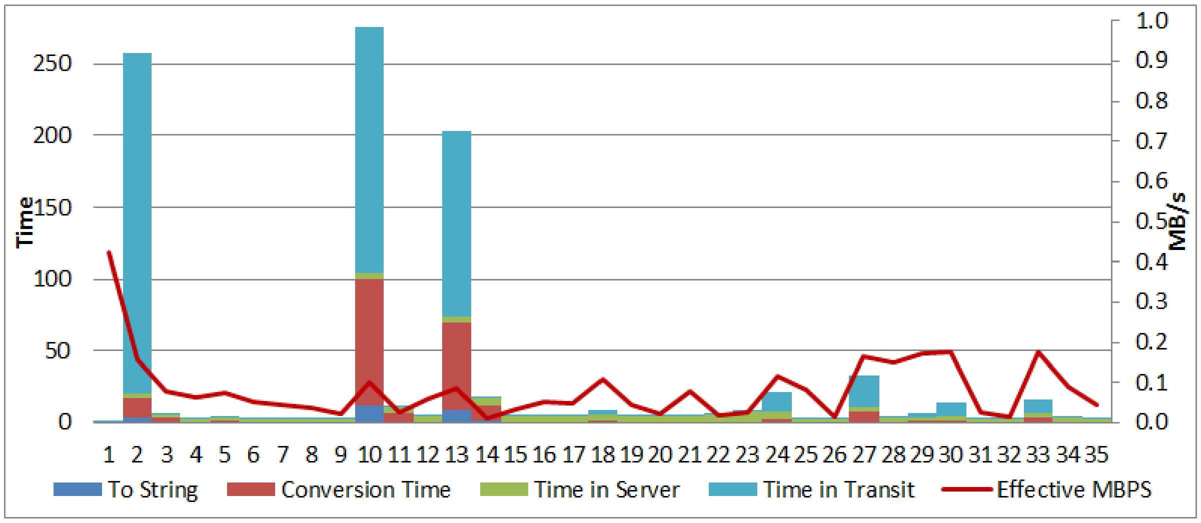


Figure 2.3 Unencrypted Sequential PDF Transmission

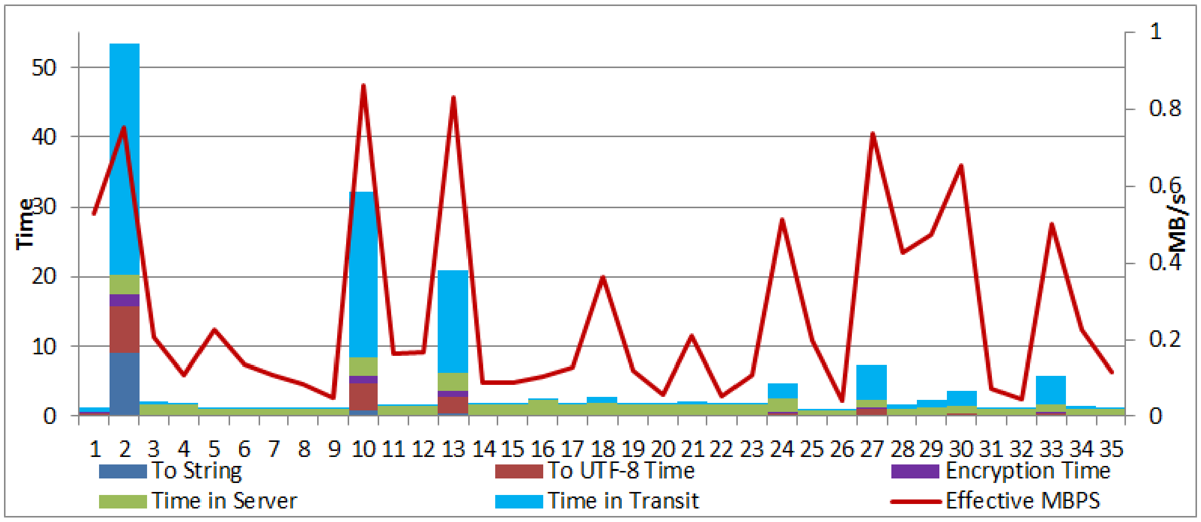


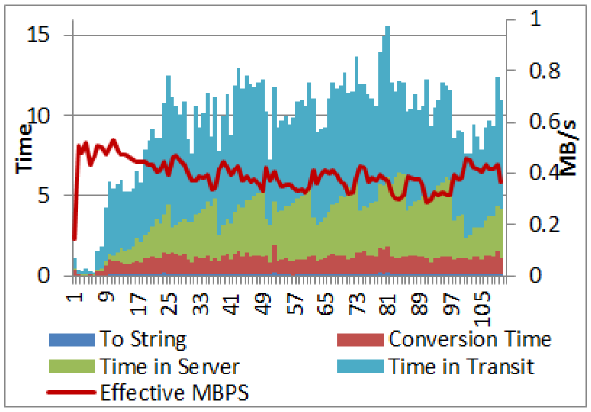
Figure 2.4 Encrypted Sequential PDF Transmission

2.5.2.3 Photos

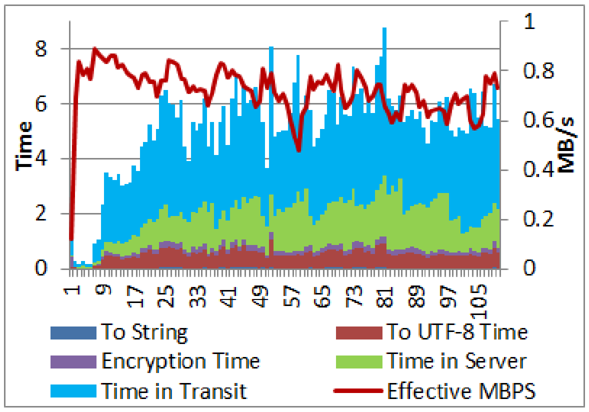
The photos benchmark illustrates storage of a large array of medium sized files with little deviation. Like PDF and MP3, Photos are not inherently web-safe and must be UTF-8 encoded. Encoding and Encryption size increases resemble those of PDF and MP3 and as such will not be reiterated. Also like PDF and MP3 encrypted transmissions outperform the unencrypted counterparts.

Figure 2.5 displays the unencrypted transmission of Photo files. Though transmission time is still a large factor, the medium file sizes comprising this benchmark leads to a decreased transmission time, with storage time consisting of a greater amount of total time (though still not consisting of the majority). The server time ramp up followed by a drop repeated throughout transmission is a characterization of the file split pertaining to the optimization algorithm. The average throughput of this transmission is 0.39 MB/s.

Encrypted transmissions fare much better, achieving a throughput of 0.72 MB/s, due to the smaller file sizes.



(a) Unencrypted



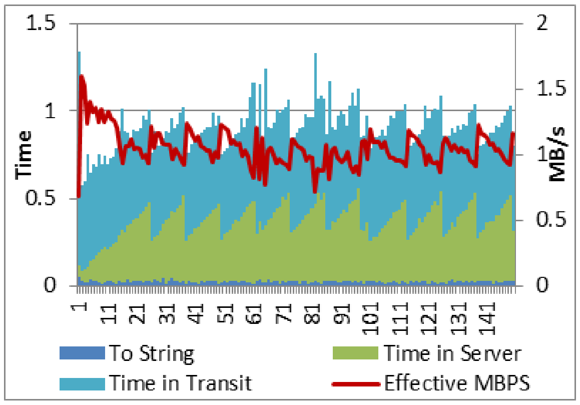
(b) Encrypted

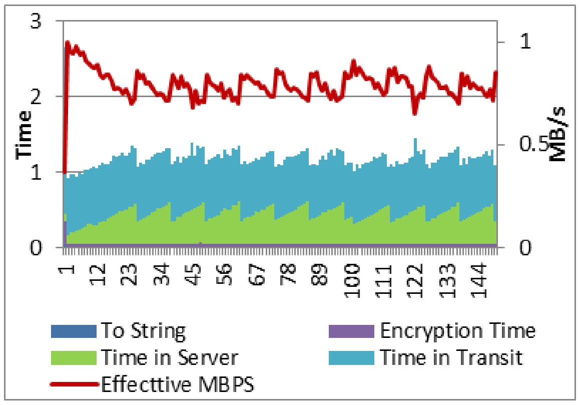
Figure 2.5 Sequential Photos Transmission

2.5.2.4 Text

Text allows for an analysis of the transmission of files of medium size which are of equal (or at least very similar) size, and are web-safe. As such, unencrypted transmission incurs no encoding penalties, be it in terms of time or size. Conversely, encrypted transmission increases file sizes by exactly 33.3%.

Unencrypted text's time in server shares the position of bottleneck more closely with the time in transit than the previous benchmarks. The average throughput yielded by these transmissions is 1.05MB/s, more closely resembling speeds necessary for effective real time storage. Conversely, encrypted transmissions underperform compared to the unencrypted equivalents due to their increased file size, yielding a mean throughput of 0.79MB/s.





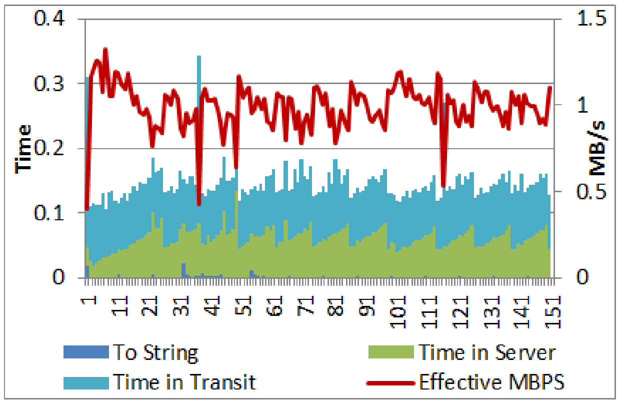
(a) Unencrypted **(b) Encrypted**

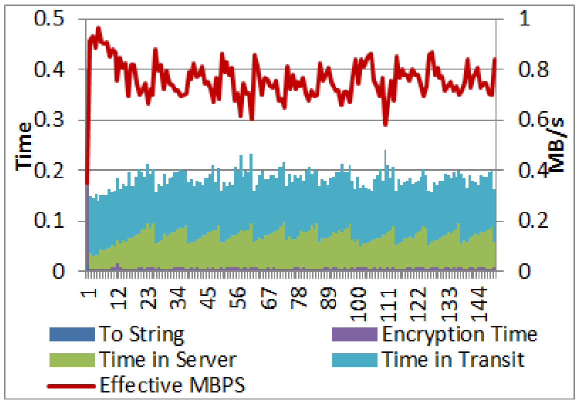
Figure 2.6 Sequential Text Transmission

2.5.2.5 Frames

Frames allows for an analysis of the transmission of web-safe video frames of small, equal size. Encrypted transmission increases file sizes by exactly 30% in this instance.

Unencrypted transmission yields a mean throughput of 0.97 MB/s, with server time sharing a much more significant proportion of the transmission bottleneck. Conversely, the encrypted transmission yields a mean throughput of 0.76MB/s.





(a) Unencrypted **(b) Encrypted**

Figure 2.7 Sequential Frames Transmission

2.5.2.6 Film

Unlike the previous benchmarks, transmitting a high definition film file without any segmentation proved challenging: UTF-8 encoding the 180 MB file ultimately required more memory than the unmodified JVM allowed[9]. Though StrongHold may

[9] The limit imposed is of 1GB of memory. After encoding, the file consisted of nearly 500 MB. For optimization purposes two copies of the file are required at the time to speed up server storage times.

be launched with an option set to allow more memory, this is a contradiction of the initial objective of a plug-and-play server for a smart home, as its execution requires significant technological prowess. As such, we simply attempted to transmit the film through the encrypted channel, allowing for less memory utilization. Furthermore, due to the sluggishness of the client in this transmission, we utilized another Widows 7 laptop with an i5 processor and 6GB of memory in its stead[10]. The total transmission time for this specific file was 526.152 seconds, of which 2.8 seconds were spent converting to string, 26 seconds were spent filtering, 36.8 seconds were spent encrypting, and 210 seconds were spent in the server. As such, this transmission allowed for a throughput of 0.34 MB/s. It should further be noted that since the 720p video ran for 23 minutes, and total transmission time was less than 9 minutes, the feasibility of transmission of real-time HD film is then confirmed for this platform.

2.5.2.7 Discussion

A number of insights may be grasped from the above experimentations. Perhaps most important of which, is that the server is capable of storing data in a sufficiently expedient speed, in most circumstances, disregarding those requiring storage of incredibly diverse information in terms of data size For such instances, the server possesses a simple feature which allows the storing of different files in different folders[11], which can transform such a widely varied benchmark as the PDF benchmark into two less varied benchmarks, resembling the MP3 and Frames benchmarks respectively. Since files of similar sizes benefit most from the storage algorithm, it is advisable that the aforementioned option of aggregating files by size on each path be utilized when there is vast divergence in file sizes.

The storage of significantly large files also adversely affects the storage mechanism, and as such it is advised that the storage of such files be avoided, or at the very least that the files be split before storage. Furthermore, it is important to

[10] We believe that if a machine is to transmit such a large file, it likely will possess considerable resources to begin with, since simply creating a file of this magnitude, and keeping it in memory requires substantial means.
[11] See in Appendix A, the path option, and Section 4.3 of this chapter for experimentation on this option.

note that if time is of the essence, and the data is not integrally web-safe, it is advantageous that the encryption option be used, lest excessive time may be spent.

Lastly, the transfer could be sufficiently sped-up if the data was already in memory and web-safe to begin with, since no time would be spent to encode or load the data.

2.5.3 Individual Transmission Results

In order to explore the separate file storage option mentioned in Section 5.2.7 of this chapter, we investigate the individual storage feature available through the path option. We store all files individually, rather than sequentially, as in Section 5.2 of this chapter. It should be noted however, that though storage will achieve a higher throughput, sequential file reading speed for small files deteriorates considerably, as presented in Section 5.4 of this chapter.

Figures 2.8-2.11 present a contrast of storage times for all benchmarks; transit time, encryption and encoding times are disregarded in this comparison as they should remain constant for each transmission. As conveyed in the figures, individual storage outperforms sequential storage: for the MP3 benchmark, individual unencrypted storage outperforms the sequential counterpart by a factor of 4.27 on average, and individual encrypted storage outperforms sequential by a factor of 2.57.

Since MP3 transmission time is dominated by time in transit, this modification does not increase throughput by more than 10 KB/s on average. PDF individual unencrypted transmission outperforms its sequential counterpart by a factor of 9.12 on average, and encrypted transmissions outperform by a factor of 6.35. Photos individual unencrypted file performance overtakes the sequential counterpart by a factor of 11.5 on average, and encrypted by a factor of 5. Text individual unencrypted transmission outperforms sequential by a factor of 9.3, and encrypted by a factor of 5.5. Frames individual unencrypted transmission achieve a throughput 3.6 times better, and encrypted achieves a throughput 3.22 times better.

It should be noted, however that disregarding small files such as those in the Frames benchmark, server storage time seldom comprises a large fraction of the transmission time, and as such, little throughput is generally gained by separating file storage, especially considering that at its worst each storage value will only be affected by 24 other files at most.

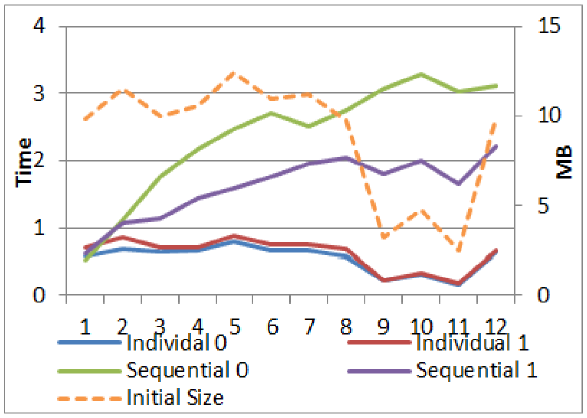


Figure 2.8 MP3 Sequential vs. Individual Storage

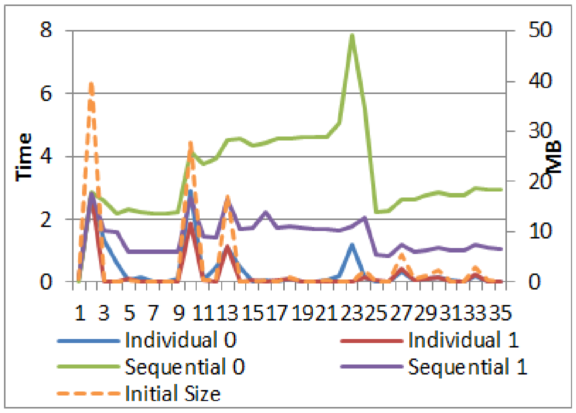


Figure 2.9 PDF Sequential vs. Individual Storage

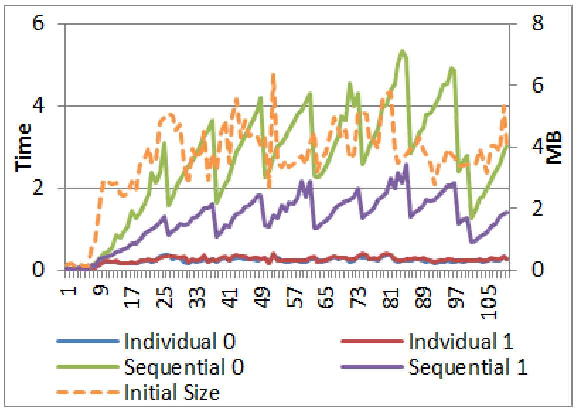


Figure 2.10 Photos Sequential vs. Individual Storage

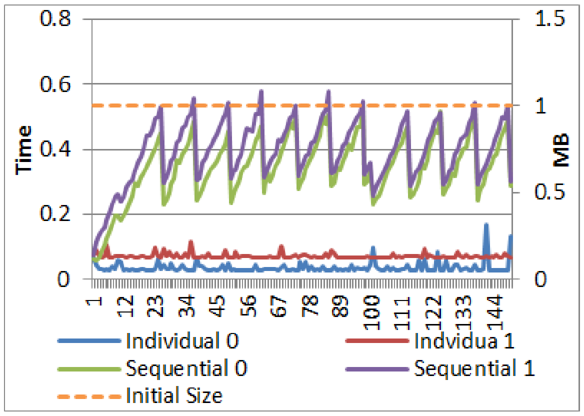


Figure 2.11 Text Sequential vs. Individual Storage

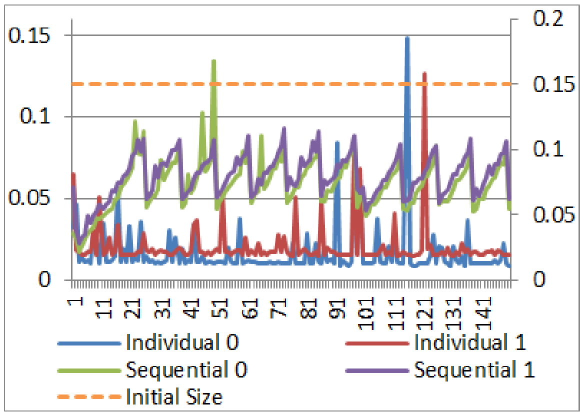


Figure 2.12 Frames Sequential vs. Individual Storage

2.5.4 Retrieval Results

Fast file retrieval may be an utmost necessity depending on the application. As such, we perform a simple benchmark to determine retrieval speeds for different file sizes, and different retrieval methods. Since sequential storage accumulated a maximum of 24 data points per file, we use 24 data points as the metric for retrieval. We divide retrievals into four cases: 24 data points retrieved sequentially, spread across two files (12 points in each file), 24 data points retrieved sequentially from one file, 24 data points retrieved separately from one file, and 24 data points retrieved individually from 24 paths. The files retrieved are divided into two categories: 24 files of exactly 1 MB each, and 24 files of exactly 150 KB each. Each of the files is retrieved through mode 0 (unencrypted) and mode 1 (encrypted). Mode 2 is unexplored, as retrieval does not require a nonce. The results for the retrievals are displayed in figures 2.13 and 2.14.

Files retrieved individually and sequentially throughout one file and two files perform almost equally for medium-sized files for the 1 MB benchmark. Conversely, the one file and two files retrievals are the clear winners for retrieval for 150 KB, indicating that retrieval in the form of frames (i.e.: small files retrieved sequentially) is best achieved through sequential storage, if retrieval time is more critical than storage time. Retrieval in this method achieves a 1.8 MB/s throughput for one file retrieval and 1.7 MB/s for two files, and 1.1 MB/s for individual files for the 150 KB benchmark. For 1MB, retrieval for one file is 1.75 MB/s, for two files is 1.7 MB/s, and for individual files is 1.77 MB/s.

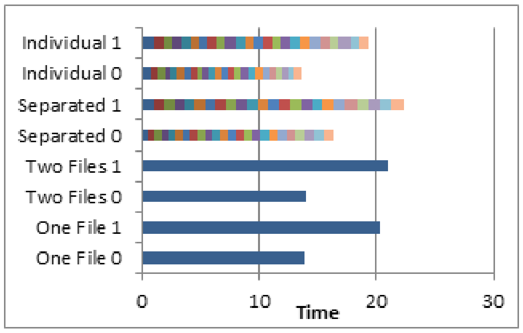


Figure 2.13 1 MB File Retrieval

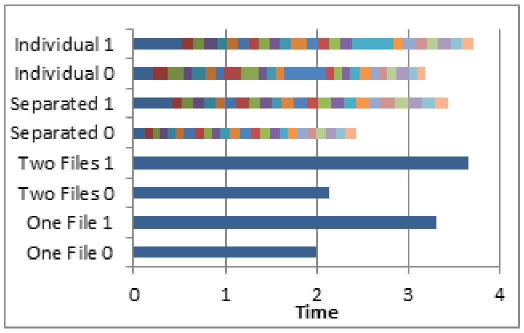


Figure 1.14 150 KB File Retrieval

2.5.5 Discussion

The allowance of varied workloads on StrongHold necessitates that a number of different features be available to accommodate all possible use cases. As no one feature can be elastic enough to appropriate all possible loads, the next best solution was to allow the choice of storage befall the client, rather than force a paradigm, which will encompass fewer cases. The plethora of elements available to StrongHold should as previously presented, allow for this accommodation. In cases where storage speed is paramount, the path option may be selected, and if large file sizes are considered, little will be lost in terms of retrieval. Encryption should be considered if files require considerable encoding to be web-safe, especially if storage speed is of importance, particularly considering encryption time is trivial. Sequential storage of smaller files is the clear winner if files must be retrieved more than one at a time and the files are sufficiently small. If an application produces sufficiently varied file sizes, the path option is the most advantageous choice, as it allows less perturbance on storage and retrieval. Large file sizes (those surpassing the 40 MB threshold) should be avoided, or at least split before transmission, as their transmission will ultimately be slower than if partitioned (as can be seen from the Text benchmark, where a total of 150 MB were transmitted in just under 150 seconds, as opposed to the Film benchmark, where 180 MB were transmitted in over 800 seconds). Finally, as long as these options are followed, the server should allow for sufficiently expedient retrieval for real time high definition storage and retrieval.

2.6 Related Work

2.6.1 Device Ownership

One of the first resolutions regarding the ownership threat dilemmas in ubiquitous computing was introduced by Stajano and Anderson in the form of “The Resurrected Duckling Security Policy Model” [40, 63]. Within this model, we are told to approach the problem analogously, identifying devices as a family of ducks: a master device is deemed a mother duck, whereas a slave device is a duckling. A duckling may be either imprinted or imprintable: an imprintable duckling may be granted a soul by the mother. A soul in this sense is a shared secret that binds the slave device to the master device. So long as the duckling is imprinted with this soul, he will obey the mother duckling and no other devices. A soul is extracted upon completion, with the death of the duckling, at which time, the duckling is imprintable again, and may be resurrected by another potential mother duckling. Any number of mother ducks may simultaneously instill souls into ducklings, meaning a slave device may be controlled by more than one master at a time.

The granting of the soul is recommended to be done physically via a non-wireless channel, or at the very least through “a channel whose confidentiality and integrity are axiomatically guaranteed”[37]. This is done in accordance to “the Big Stick Principle,” which states that “whoever has physical control of the device is allowed to take it over” [37, 54]. This is appropriate as it corresponds to the manner through which we access devices outside of a smart home environment, that is to say, through physical presence.

Initially, the concept allowed ducklings to interact among themselves, but disallowed devices issuing orders to other devices they do not control; however, in a later paper, Stajano expands on the concept, allowing mother ducks to instill policies to ducklings through the same secure channel. These policies would define what actions are allowed and by whom, yielding, if necessary, full control of the duckling to any other devices [63].

A centralized equivalent is presented by Naqvi and Riguidel, where ownership of data is dealt with by an “Infosphere”, and a “Security Domain” remains in charge of protection and control [39]. The authors further propose that encryptions be set by the user, allowing her to choose between performance and security for specific devices. Additionally, their implementation makes use of virtualization to disallow applications located within a servant device to interfere with one another. Another decentralized solution is provided by Lee et al. [13], where each device must be authenticated through a “Security Manager,” but their access control data is managed by a “Smart Portal Server.” Though lacking virtualization of applications within a servant, the remainder of this solution is quite similar to Naqvi and Riguidel’s.

After the Resurrected Duckling Model, a number of methods for distributed access control were defined for this medium, many expanding on the model, and others, taking tangential paths. Such means presented tend to have in common their necessity of a mobile method of authentication. Such methods for authentication range from biometrics to passwords, to devices (such as smartphones or pocket watches) to RFID tags [9, 10, 12, 38, 64]. Though the device which achieves authentication deviates, the elementary notion is always constant: the device is mobile and constantly placed alongside a user, be he a resident or guest, and achieves (through methods described in Section 6.3 of this chapter) bounding of user location in relation to the device to be connected to[12].

Conwell et al. provide an additional methodology that falls within the resurrected duckling model, wherein users use their smartphone devices to authenticate, configure and update access control lists [38]. The ubiquitous quality of smartphones and their ever-presence beside the user further allows the evolution of the model to easily permit reactive access control permissions; that is to say, the device may allow a user to respond to access grant requests from other users. Thus access list population may occur ad hoc, and rather than having to type out lengthy and complex lists, the population of the list is simply reduced to a modest prompt.

Also following up on the Resurrecting Duckling Model, Argyroudis and O’Mahony develop AETHER, whereupon they establish more detailed connection methods and interactions [64]. In AETHER, devices would come pre-installed with asymmetric key pairs, and have their own policy lists, wherein rights are directly associated with actions. It is lightly hinted that one may use auto-configuration to evolve policy sets over time, but this approach is not given much attention. The binding of devices is specified as a key exchange through a secure location-limited channel (such as touch or infrared link); following such an exchange, policies are passed through a secure channel, and refreshed given a short time period. The refreshing method is used in lieu of certificate revocation lists. If a device is to come out of range and for longer than the validity period, the binding expires, and the devices must reconnect via the same secure location-limited channel.

In regards to policies, AETHER allows both positive and negative policies, establishing what is specifically allowed and disallowed. The policies further allow conditions, which specify restrictions related to time, location, and other factors. The concept of conditions seems to be quite common among other models [13, 39].

[12] That is to say, it helps the device being connected to identify that a user connecting to it through an authentication device is physically present.

Moreover, AETHER allows a policy-maker to specify further policy-makers for devices, as well as their delegation depth; that is to say, to what extent the new policy making users may specify new policies.

Solutions such as these and other policy setting schemes similar to the Resurrected Duckling Model, though allowing decentralized management of devices and data, suffer from the requirement of onerous creation of policies, which is neither an easy feat for technologically untrained users, nor a much desired feat for those who are technologically able, due to the repetitiveness and laboriousness of the task.

Kim et al. suggest a mechanism where access control policies are pre-set into groups, capturing most, if not all home owners and visitors [9]. Through this method, whenever a visitor or another newly introduced resident is within the home, a resident or administrator may place him within one of the pre-set groups (Full Control, Restricted Control, Partial Control, and Minimal Control). Any access made outside of the policies groups would have to be manually granted by a resident or administrator. It is also suggested that on top of that all accesses be logged to be audited to account for discrepancies.

Hoque et al. tackle the problem in a tangential manner, rather than allowing for easy configuration, they attempt to create a self-configurable system, arguing that some elderly patients located within the e-health environments would find even simple policy creation unmanageable [11]. They further argue that common current policy configurations fail to account for intuitive intricacies of trust, merely focusing on an implicit, inaccurate view. In their approach, all devices possess a list mapping trust values for each service requested by a given neighboring device. Upon an interaction request, a device estimates and sets a trust level for the requesting device, based on the security level of the requested service, as well as a user's disposition value. Through proper use, trust values are increased, and more access is granted. Improper use leads to a decrease of trust, coupled with a decrease of access. This however, can be exploited with mimicry attacks such as those used to target on Intrusion Detection Systems, whereupon a malicious device can escalate its privileges by following the dictated set of rule, only to later attack undetected [61].

Likewise, Seigneur et al. endeavor to mimic the process of human trust in an attempt to automate trust policy configurations [56]. They argue that the ease we achieve through Plug and Play technology should be attempted for later management. Within this auto-configuration arrangement, devices may recommend trust levels to other appliances. The installation of appliances within the home would be propagated to other appliances, and dependent on the installer, a level of trust would be assigned to this appliance by others. For instance, were the house owner to install a new door lock, other devices would assign that device a higher trust than if it were installed by a lesser source. Furthermore, dependent on the risk associated with the corruption of the already installed devices (which can be preset by the manufacturer), they will choose to follow the recommendations given by this new device differently. In the previous example, given that an owner installs a door lock, a television set will likely easily follow the door’s recommendations, whereas more sensitive devices such as a safe or another door might not.

Temporal coexistence further augments trust between appliances, such that if two devices coexist within the home for extended periods of time, their trust levels towards each other rise. The authors further state that biometrics could further ease this classification of trust, allowing a process to place low trust users (such as guests) detected to interact with higher trust users (such as residents) at an escalated level of trust. Continuing on the previous example, a guest might enter the house through a key provided by a resident. As such, the house door would identify this user as trusted, as he possesses a house key, independent of the fact that he does not possess an owner’s equivalent biometric equivalence. This trust level is propagated throughout the house, and as such, the guest is allowed to use minor devices, such as the television. However, the guest has no access to other rooms or the kitchen, since they are considered to be more sensitive locations. The guest then might proceed to watch television for some time, thus building trust with the television set. Since the television set has, over time built trust with the kitchen, and since the user has escalated his trust level with the television over use, he is now granted access to the kitchen. Additionally, were the guest to engage in a phone conversation with a resident, a biometric on the phone might recognize the resident’s voice and further escalate the guests privilege due to the interactions with the user.

The approach of auto-configuration, however, fails to overcome a number of Section 3’s previously established vulnerabilities: One of the more severe threats this configuration fails to circumvent is that of malicious devices being integrated within the smart house. This specific class of threat is previously mentioned in Section 3.5 of this chapter, and concerns either the existence of devices manufactured with malicious code, or compromised devices. The compromising of a number of trusted devices may lead to further corruption of appliances, at which point an adversary might gain control of trust assignments within the house, even though he may lack control of the majority of appliances. Such methods may give an adversary access to the house, or even in some cases may lock a legitimate user out. Falsification is another method through which an adversary might take advantage of this configuration: through the recording of specific biometrics, an adversary might escalate his trust, such as the replaying of a legitimate resident’s voice [65, 66], or the use of masks or photos with the user’s likeness [66, 67].

A similar approach is taken by Azman et al. for automatic trust calculation [55]. Like Seigneur at al, trust escalates with interactions (be it with a house owner, or other devices), however, unlike Seigneur et al., this method also uses routing selections alongside temporal measurements to determine the trustworthiness of a user: In this approach, a user who detours from a normally traversed path, or presents temporal anomalies (that is to say, the user exceeds a temporal threshold in a location or activity), will be deemed suspicious and have his trust level decline.

Solutions such as IBM’s SPARCLE, conversely, attempt to rather facilitate the entry of access control policy data, by creating a more human-readable interfacing language [68]. In such cases, one can grant access to people and devices by writing the rules in simple English, such as “guests may access the television,” or “police may open the door.” Though such entries might be time consuming, it is more likely that laymen may be able to enter and understand the policies. Furthermore, unlike automated and pre-fabricated access control lists, the resulting actions will more accurately follow the exact desired outcome.

2.6.2 On the Prevention of Data leaks

Kim, Beresford and Stajano propose that to limit availability to sensitive data, only summaries of present data are stored in any pervasive storage device [36]. As such, real time measurements can only give access to data occurring at that time window and no prior data, to disallow more sensitive data to purvey unwanted inferences (For instance a caregiver who monitors a patient’s detailed heart rate might be able to imply details about a patient’s more intimate encounters). In such cases, a summary of all data will be just as useful for the caregiver, and as such more minute details could be foregone. It is also suggested that any further data necessary that cannot be acquired through the summary necessitate the patient’s consent.

It is additionally stated that any caretaker may only have access to data stored specifically in a temporary repository, out of which it cannot be transferred.

In regards to wireless information leaks through encryption, Agarwal et al. propose that constant rate data production might prevent an attacker from determining the movie being streamed [45]. However, the authors also argue that solutions such as the one presented may significantly affect bandwidth consumption, and would not prevent an attacker from still determining when and for how long the user watches movies. In their paper regarding language leaks related to VoIP conversations, Ballard et al. present possible padding of packets to greater packet sizes [47]. Dependent on the amount of padding, the discernibility of language decreases from beyond 66% (with no padding) to 27% with 192 or 256 bits, and 6% with 512 bits. However, in the case of 192 or 256 bits padding this dimension of determinism is still quite above that of random guessing and is, as such, still undesirable. Furthermore, this solution still introduces a large overhead in terms of bandwidth, leading to nearly 42% overhead for padding to 512 bits. Libertore and Levine reach a similar conclusion in their paper of inferring the source of encrypted HTTP connections, wherein padding also dramatically decreases accuracy, but at a high cost to performance [44].

Canny and Duan attempt to impede an attacker from gaining access to sensor data recorded in their absence [35]. To do so, they propose a scheme where all data pertaining to a user’s presence (such as recordings and localization data) be encrypted with a randomly generated secret key, which in its turn is encrypted by the public keys of all users present during the recordings. Each different encryption is placed within a different tabular position, which is calculated via a hashing function that takes as input the user's public key. This function allows the location and placement of the encrypted secret key. All other empty locations are filled with random numbers. To allow access to privileged parties (such as policemen, repairmen, firemen, or network administrators), a master key is also saved, only decryptable by a matching private key in possession of the privileged parties.

An authentication device is a mobile device used to store a user's public key which interfaces with a smart location to grant access during physical presence; therefore, a user lacking his authentication device will not have access to data as a fail-safe default. This system also allows the exclusion of access control lists, and can thusly function without knowledge of the user's identity in situations where the user must interact with an untrusted smart environment. The authors further recommend that each smart location be equipped with a display which presents the current number of occupants within the room; if this number does not match the visible number of residents, it can be concluded that something is amiss and a rogue authentication device may be present.

2.6.3 Location Awareness

In order to circumvent outside users from using technology which they should not, Manish suggests that specific zones within the home should be created, and different features be enabled for each appliance within each zone. For instance, though you may check the temperature of a stove from a distance, you may not set it except when within the room. Locality can be established by using extra sensors, and for the more dangerous appliances, communications can only occur via shorter range signals (such as infrared, as opposed to Wi-Fi) [34]. It is, however not an infallible technique to use communication range to represent physical locality, as Capkun et al. [69] have shown; in their paper, they present a method of amplifying, and thus relaying signals from a keyless entry key onto an automobile, thus opening its doors and turning its engine on, when they key really is not physically present. It is feasible that such attacks could be adapted to the smart-home model, thus rendering locality establishment moot. As suggested in the paper, one must take immense care to ensure the distance bounding protocols are safe from relay, which according to the authors, can be achieved with a verifiable multilateration protocol.

The problem of locality does not only befall unto the device process of ascertaining the presence of a user, but also encompasses the charge of maintaining a user's location unknown in an environment where his presence is constantly checked. In their paper, Al-Muhtadi et al. present the novel idea of mist routing to circumvent locality threats [42]. In their model, sensors are able to detect the presence of users, but lack the ability to identify the users. Furthermore, this is combined with a novel routing protocol to further protect the user's location.

In their study of the Nike+iPod [45], Agarwal et al. argue that communicating mobile devices can be a threat to one's locality, as a leak of its persistent unique ID may allow an attacker to track a user's precise location through sniffing. As such, the authors maintain that strong encryption such as AES, using randomized IDs, recomputed at each idle moment, should be sufficient to circumvent the attack, however it may prove to be difficult due to the limited performance and battery life available to mobile devices.

2.6.4 Device Authentication

Authentication for devices is a necessity, as the configuration is meant to be elastic (that is to say, devices should be able to be added or removed at will), and in order for trust to exist, authentication methods are necessary. In one of the more interesting methods, items are connected via the physical interaction of checking the device for a physical code, and manually inputting it on a hotspot device, and vice versa. Furthermore, this method of authentication allows for different encryption for data transfer between all devices, as they each use their code as a key [6, 10, 11, 13].

Han et al. present a more detailed view of the method of authentication [13], wherein a user must register a device through a hotspot, which will issue a portal run by the device manufacturer a request to confirm the validity of a certificate issued by the device. Following such a check, a manual exchange of codes will be required by the user, to identify that the device being registered is indeed the one owned by the user. Following this step, the hotspot will manufacture a set of private and public keys and will exchange them with the device through an encrypted channel.

As per Pishva and Takeda, one of the greatest concerns in regards to device authentication and communication within a smart home is that of creation of standards [6]. Heterogeneity as it stands in regards to smart devices is quite prevalent [13], and if that quality carries on towards communications and authentications, it would present quite a problematic challenge in regards to security in an environment where collaborative communication is vital.

Lastly, as stated by Naqvi and Riguidel, common cryptography can be easily translated into this field to prevent eavesdropping, and protect authentications, and for all intents and purposes as it stands is enough for this problem, and should be utilized mostly unchanged [39].

2.6.5 Availability

In regards to circumventing sleep deprivation attacks, Stajano and Anderson propose data communication directed at devices with limited resources be directed through a reservation mechanism, which would prioritize actions and only enable them if their priority passes a threshold [40]. This could then be used for preventing communication from being flooded between devices, by only forwarding high priority messages, and sending a summary of other messages in a timely manner otherwise.

The authors also target jamming attacks by ascertaining that in their occurrence, devices may commence spread spectrum communications or frequency hopping to prevent them from achieving a denial of service. They however argue that in the commercial world, such attacks may be dealt with in a more physical manner, such as complaining to the authorities, and having the operator of the jamming station arrested.

Attacks such as those that target ubiquitous devices which do not require re-authentication and fall victim to man in the middle attacks, can be thwarted through periodical (though infrequent, due to limited resources) re-authentication [39].

[13] As can be seen quite predominantly in the smartphone market in terms of iPhone vs. Android vs. Blackberry vs. Windows Phone, etc.

Furthermore, devices can be kept from failing due to vulnerability exploits due to more extensive fuzz testing, and communications can be kept from being hijacked through encryption, such as SSL, TLS and SRTP [41].

2.6.6 Guest Access Control

A newfound issue apposite to the smart home is that of guest access control. Alongside the common problem of device utilization by habitual users, further dilemmas arise, such as that noted on Section 3.5 of this chapter: not only does a user wish to disallow a guest from a number of actions; he does not wish the guest to know he is deemed untrusted. Unless given some tool to properly circumvent such a social taboo, the user will tend to set looser policies of access controls to keep from disclosing his distrust [9].

Moreover, Johnson and Stajano argue that guests should not have to be given accounts [12], nor should they be dealt with as strangers would: Specifically, a guest should be able to access the television set, but he should not need to have to be registered as a permanent occupant to do so. They further maintain that in preserving past customs we should perhaps imitate the non-smart-environment instance wherein a guest is given possession of the house keys until they leave, by having certain rights granted upon entry and revoked upon exit.

Johnson and Stajano go on to provide a guest-specific[14] temporary access control scheme wherein a guest is authenticated manually rather than automatically, and without need of prior policy creations. Using this scheme, a guest would be required to press a physical button on the device to access it, which would in turn cause the device to produce a nonce, which the user would input on his authentication device to complete the pairing[15]. The authors continue, contending that the burden of guest access control policies be placed on the manufacturers. Such classes of actions to be provided by the manufacturers are subdivided, providing actions that any one physically present may perform, actions requiring physical presence and resident authorization once, actions requiring resident

[14] As opposed to the schemes provided on Section 6.1 of this chapter, which could be used for both users and guests.
[15] The observant reader will find some similarities shared between this method and that used in Bluetooth device pairings.

authentication at each access, and actions that may never be performed by non-residents. Lastly, if a guest is to reside for extended periods within the house, the guest will be given control of areas where he resides through a temporary account. Namely, if a guest is residing within a room, control of all devices within the room should be given to the user, however, the house owner will maintain administrative control over all devices, allowing him to remove the guest’s access control capabilities, but not vice–versa.

2.6.7 Central Smart Home Servers

Though we consider our work to be novel in terms of a secure central smart home server - containing all necessary aspects to properly protect against privacy and security threats in the literature - other work is present in terms of smart home servers, which do not concerns themselves with security aspects. The most complete example of these is perhaps HomeOS, as introduced in conjunction by Microsoft and IBM research:

Dixon et al. [84] create a central hotspot akin to StrongHold in terms of ease of use and elasticity of features. The method provided allows for significant abstraction, transforming the central server into an operating system with the possibility of abstracting network devices as applications atop it with the use of a language specific API; this solution is proven in their work to be significantly simple to use and code. However, unlike StrongHold, no focus is given to security and privacy issues or sensor-like data storage. Moreover, access rights are given manually rather than escalated automatically, allowing for a possibly more granular outcome than the learned algorithm (however, it is equal to the manual methods provided as system administrator tools of StrongHold); it is nonetheless far more challenging to allocate than simple plug and play, and falls prey to problems concerning guest access control (as noted in the third footnote of this chapter). It should additionally prove difficult to coordinate in eHealth homes or homes containing computer illiterate users. Lastly, it is tested using an Intel Xeon server allowing for functionality in costly environments, but perhaps providing more difficulty in conventional homes.

2.7 Future Work

The addition of further functionality and heavy stress testing are among future work for StrongHold, however, perhaps a more pertinent area to focus on the future is that of user and security experimentation: whereas this work has been more-so an exploratory attempt at solidifying, implementing, and proving the feasibility of assorted ideas, it focuses very loosely on ease of use evaluation and security experimentation. Ultimately it is a means to give form to the theory present in the few existing works surrounding this area. Given this server as a test-bed, it is assumed that a variety of tests may be implemented to ensure proper functionality, and even more importantly, implement user studies for the functionality and feasibility of a real smart environment.

Before release of the server onto the public, however, more effort is necessitated in terms of Human Computer Interactions (HCI) and design, because as much as the server presents various aspects of plug-and-play functionality, and is easy to use, that does not equate desirability. That is to say, that although the server is simple to use, that does not mean that users will want to use it, and as such, better graphical user interfaces still may be implemented. Human Computer Interactions, nonetheless are outside of the scope of this work, and as such are left for future work. User interactions must also be better studied to determine the best numerical values selected for each manufacturer's threshold values, as the threshold numbers designated so far have been more-so a approximate estimation rather than an exact collection.

The implementation of a user application tasked with encrypting and decrypting data was also not realized, as it diverges from the server side of the application, and would involve coding for several platforms. Due to time constraints, it was then left for future iterations.

Also in regards with user interactions, improper accesses are met with negative reinforcement in the form of lost experience, however, no attempt has been made towards direct punishment for destructive attacks. This is one aspect of the application which might be studied, wherein if an IP address is the source of multiple attack-like interactions, it can have its access blocked, or even issue a direct alarm to a resident or system administrator. Such issues were not tackled in this work, but can be engaged in the future.

Lastly, though we did not observe any resolution to safety apposite to power and system failures conundrum in the literature, we advise that in order to protect the user, the system should default to relinquish use of essential devices in such circumstances. Namely, devices constituting survival necessity, such as water, food, restroom facilities, doors allowing access to these areas and exits (such as those from a room, or from a house to the outside), should default to function for anyone. Conversely, entries other than those leading to facilities or kitchen area should remain closed, unless they lead to an area known to contain a user[16]. This area has not been elaborated upon, as it delves into usability territory, and as such falls outside of the scope of this paper.

2.8 Further Discussion and Conclusion

In our exploration of the world of smart home security, we have encountered a number of interesting solutions to tackle the problems this new medium carries. Although there are many challenges, few solutions exist due to the novelty of the field[17]. Very few areas of this vast topic have been fully explored, and though theoretical solutions exist, they are rarely implemented outside of a testing environment: though we will find select smart devices in the open, such as smartphones, smart TVs, VoIP enabled phones, among others; fully interconnected homes are rather rare outside of experimental settings.

We believe that we are reaching an era where pervasive computing is becoming a prevalent paradigm, and as such, this topic will likely soon undergo an upsurge of the likes of those experienced as of late by the fields of mobile phone technology and portable media players. In order to reach that level, we must first establish proper security and usability pertinent to a field of this sensitivity; after all, one should not build a car when they don’t know how to achieve stoppage. As can be seen by the literature, we seem to be reaching a point where the usability of this

[16] This instance is presented in case children, or incapacitated users reside in a specific area, and cannot exit on their own.

[17] Though the notion of electrified and automated homes have existed far prior, the idea of interconnected ubiquitous computers within the house was only first publically proposed by Mark Weiser in 1991 [1].

technology is accessible to non-experts, and the security is nearing a stage sufficient for the integration with living environments. In such a manner, it is best to incorporate dispersed concepts into a single model, to finally achieve a level of integration able to combat all possible vulnerabilities present in the current model of smart environments.

In light of the fact that no single solution managed to solve all problems presented prior, coupled with the necessitation for an all-encompassing solution presently, we immersed ourselves in a means of implementing a comprehensive solution, feasible for experimental and non-experimental settings. Furthermore, we try to bring with our server a standardized methodology wherein a device from any manufacturer may utilize our server alongside devices produced by a number of different manufacturers.

Our method uses current industry standards of RESTful APIs, and JSON, as well as AES encoded communications, and SPEKE methods for password resets. Concurrently we also introduce new concepts, such as a learning method to allow users to relinquish the burden of set-up and maintainability of policy, an optimized algorithm to maintain practical response times, and a standardization of communication between devices and users within a smart home. Parallel to those we implement a plethora of security methods which are invisible to the user, and yet allow for private and secure communications; methods which although exist in the literature separately, were never before conjoined. It is through these implementations that we hope to aim for a secure, easy to use, and intrinsic smart home.

As a last argument, it should be noted that it is paramount that these threats and their associated solutions presented as well as new threats be rethought and resynthesized occasionally, respectively. Only by frequently questioning our assumptions, can smart home security remain always one step ahead of attackers.

Chapter 3: MASHA: Meal Advocation for Smart Home Automation

3.1 Introduction

Since the turn of the century, recommendation systems have gained considerable popularity: systems such as the Amazon Recommendation Engine [14, 15], the Youtube Recommendation System [16], Google News [17, 18], and Netflix recommendations [19, 20] have become an ubiquitous commodity, and parallels can be often found across the web. Recent efforts have focused on porting the recommendation engines to different domains, such as song recommendations [70], and recipe recommendations [22, 23, 24, 25, 26, 27, 28].

In the field of recipe recommendations, the aim so far has been to port the mutually liked aspect of ordinary recommendation systems in order to suggest future meal choices, given a previously established database [21]. Further effort has been employed in determining healthy choices given a user's diet plan [22, 23, 24, 25]. However, little has been done to determine which recipe to recommend given a pool of recipes pertaining only to participating users. Furthermore, recipe recommendation algorithms so far, tend to require user input of exact like values through a Likert Scale, which is known to suffer from considerable bias [71]. Lastly, no effort has been employed in porting this class of recommendation systems to a home environment, where excessive levels of user interaction are considered intrusive due to their explicit nature, and as such, undesirable; it is, nonetheless, an environment where such a system should be most useful: where else should a person need recipe recommendations but in the kitchen where such recommendations may be employed?

Focusing on the above concerns, we create MASHA, a recipe recommendation system designed for the smart home. We create a system of finer granularity than those present in literature by foregoing the Likert Scale, and instead directly tracking the frequency with which recipes are prepared. We allow for day-to-day use by simplifying the system so user interaction is minimal and implicit, and we couple the system with a smart home server to showcase usability. Moreover, the system takes recipe ingredients into account by using recipe weight as an analogue to tag frequency, allowing for further means of determining item similarity for recommendations with greater granularity than tag methods currently present in literature. Lastly, we evaluate the system, ascertaining its usability, and the success level of its recommendations; by determining the success level of each recommendation method employed (recipe ingredient similarity, recipe like similarity, and a combination of the two), we may determine which specific system yields the best results, thus determining which method achieves best results when concerning recipe recommendations.

3.2 Background and Related Work

The field of recommendations currently comprises of several methodologies: content-based filtering [14, 21, 72, 73], collaborative filtering [14, 21, 72, 73], demographic models [74, 75], and cluster models [22, 72, 73]. Moreover, related work pertinent to the field of recommendations is incredibly vast, owing to the variety of systems to which recommendations may be applied. Systems related only to recipe composition, however, are nowhere near as extensive, and can be explored in their entirety with ease. As such, in terms of general recommendations, only key systems will be mentioned, whereas systems associated with meal advocation and composition will be thoroughly expounded upon.

3.2.1 Filtering Methods

Content-based filtering [76] comprises of determining similarities intrinsic in items contained in the recommendation system. This method warrants a comparison of previously selected items with other available, though previously unselected options. A simple example of this system taken in the context of document recommendations would look for important words in a user's previously read literature, and examine documents which contain those words, or similar words, and output to the user the documents which most resemble the previously read works. Though content-based filtering can be applied to various fields, such as film or music recommendations, it is most widely used in fields containing a dense library of descriptive, extractable content. That is to say, fields where the automatization of tag retrieval is difficult without user interaction (such as in movies, where it is difficult to extract the theme or crew of a movie without user intervention), tend to focus less on recommendations via this method.

Collaborative-filtering tends to focus on the relationship between users and items rather than the relationship between items alone: the aim of collaborative filtering is to eventually couple like-minded preferences of users. Extending the previously established document example, this means that rather than matching documents via similarity, the system would recommend a document to a user that another user with a similar reading history has enjoyed. This model is far more popular to systems where tagging is prohibitive, such as those musical or film related, and has gained much popularity recently due to its application and victory over other methods in the Netflix Prize competition [19, 20, 21]. The model, as such, generally consists of an N by M matrix, where N signifies the number of distinct users, and M the number of distinct items; each M, N position on the matrix relates to the numerical correlation between a specific user N, and an item M, with the result signifying the likelihood the user has enjoyed the item, with positive values relating a positive experience, and negative values relating a negative one [77]. In specific cases deemed One Class Collaborative Filtering [77], only positive examples are taken into account, and negative values are ignored. Collaborative filtering is further forked into two categories: item based, and user based [72]. Item based approaches tend to couple users and items, thus recommending items that tend to be enjoyed if a user has liked a previous array of items. Conversely, user based models couple like-minded users, recommending to one user items which a similar user has liked.

Ratings apposite to these two previous methods can proceed in two manners: implicitly and explicitly [16, 18, 72, 78]. Explicit ratings necessitate greater amounts of user interaction, often requiring a user to directly rate products in manners resembling the Likert Scale. On the other hand, implicit ratings are acquired without deviation from the user's normal interaction; that is to say, little to no interaction is necessary beyond the normal usability to acquire ratings.

For the methodologies of content-based and collaborative-filtering, similarities are often calculated through a matrix similarity formula, the most popular of which is the Pearson Correlation Coefficient (PCC) [14, 15, 19, 20, 72, 73]. PCC allows the system to measure the proximity between two selected vectors, returning a value between -1 and 1, relating how similar (or dissimilar) the two vectors are [79]. The formula for PCC is as follows:

$$\frac{\sum(x-\bar{x})(y-\bar{y})}{n * \sigma_x \sigma_y}$$

Furthermore, in order to ensure lesser known items are not discarded in lieu of widely used items, normalization techniques are often employed [14], such as the Term Frequency – Inverse Document Frequency (TF-IDF). This statistic quantifies a term's importance in a document, in relation to the corpus: documents which appear often in the corpus get weighed less heavily, whereas lower occurrences in the corpus yield higher values, since the inclusion of a rare term should be given higher importance. The formula for TF-IDF is as follows [30, 31, 32]:

$$\frac{n_{document}}{N_{document}} * log_2(\frac{N_{corpus}}{n_{corpus}})$$

where N represents the full count of values pertaining to the specific area (corpus or document), and n represents the number of occurrences of the item in the area. For instance, if we are examining the word "dog" in a document that possesses 1,000 words, 25 of which are dog; and in a corpus of 1,000,000 documents, where dog appears in 125,000 of those, the equation will fill out as follows:

$$\frac{25}{1{,}000} * log_2(\frac{1{,}000{,}000}{125{,}000})$$

As the name implies, cluster models attempt to cluster like-minded users [14]. This method of recommendation relies on very specific user similarity, creating groups of users of selected sizes and determining recommendations solely from these clusters. Due to its nature, optimal clustering is impractical for sets of large sizes, and greedy algorithms must be used in place of the optimal equivalent. Though clustering may achieve faster results than the previous methods (especially through the use of the greedy approach, coupled with offline computations), the recommendations produced from this system are fairly poor due to the inherent lower granularity [80], and as such no more detail on this method will be elaborated.

Lastly, demographic methods entail the grouping of users of like-characteristics [74]. This method relies on user profiles to couple users with similar demographic features (such as locations). Demographic methods, however, suffer from low granularity (since only a few factors tend to be available on profiles) and lack of adaptability (since no attention is directed at learning from user interests beyond basic profile characteristics) [75]. Due to those pitfalls, like cluster models, we will not elaborate further on this topic.

3.2.2 General Recommendation Systems

One of the earliest notable adopters of collaborative filtering systems was GroupLens, a news recommendation algorithm [79]. In their paper, Resnick et al., present Grouplens as a Usenet program, which allows users to read and rate news articles, providing suitable recommendations based on ratings. The rating system uses collaborative filtering to determine similarities between users (via the Pearson Correlation Coefficient), and through that data, the algorithm determines articles of interest to the user.

Systems like GroupLens, however, were not geared to handle the large user bases later systems possess. As such, as time progressed, new solutions were engineered to allow for better performance and recommendation quality. Among the more recent collaborative filtering systems, one of the more successful and largely public implementations is the Amazon Recommendation Engine [14, 15]: In their paper, Linden et al. introduce item-to-item collaborative filtering, a collaborative method focusing on item similarity rather than customer similarity. The authors argue that traditional collaborative filtering methods incur sizeable overhead for large sample sizes, necessitating a different solution to scale to Amazon’s significant user-base. Items as such are coupled with items which are often purchased together, and recommended to users who have bought those other items.

Alongside e-commerce and news recommendation, frameworks like the Youtube Video Recommendation System emerged, aimed at media recommendation [16]. Media recommendation algorithms like Youtube present considerable challenge due to the intricacy of obtaining relevant item information solely through automation: whereas text may be tagged, video themes are not as easily obtained without human interaction. Further challenges arise due to short and noisy channels associated with the medium; that is to say, the clicking of a short video, unlike the buying of an item at Amazon, fails to be a very clear declaration of intent, and as such, fails to be a very clear indicator of like or dislike. Youtube takes a unique approach to tackling these challenges: they factor co-visitation counts, which determine how often two videos have been co-watched within sessions. These values are normalized before being used to retrieve a recommendation, to ensure less popular videos are not discarded in lieu of more popular ones. Furthermore, videos before a certain time threshold are disregarded, since they are deemed irrelevant for being too far in the past. Finally, in selecting actual recommendations, view counts and ratings of candidate videos are taken into account, alongside with actual similarities between the user’s previously viewed videos and the candidates. Lastly, to increase the diversity of recommended videos, videos which share too many similarities are discarded.

Google News features as another prominent recommendation framework which has garnered significant popularity. Google News recommendations manage a similar system as GroupLens, however, unlike GroupLens, they rely solely on implicit user clicks, rather than explicit ratings. In their initial iteration [17], Google News, relied on both clustering algorithms and co-visitation scores, recommending stories which had a high correlation of visitation with a user’s past history (also expiring values which precede a certain date threshold), present in a familiar cluster. In a newer iteration of the system [18], this method is further coupled with content-based recommendation, relying on a multiplication of both methods to produce the recommended article.

Lastly, the mention of Netflix should not be foregone of any list declaring recent developments in collaborative filtering, especially considering the attention it has garnered following the Netflix Prize competition [81]. The BellKor Pragmatic Chaos Solution [19, 20] allowed Netflix a new, more accurate means of recommending films to its large user base. The Netflix recommendation system, much like Youtube or GroupLens operates in the basis of ratings, and much like the other systems in this section, disregards ratings preceding a certain threshold of time in the past. In their collaborative filtering model they further include baseline predictors to better the rating output, by overcoming rating bias.

3.2.3 Recipe Recommendation Systems

Recommendation Systems in the recipe medium take many forms. One of the more popular applications of recipe recommendations comes in the fashion of health advocation: frameworks which allow for recommendations of diets that fit a specific user's health paradigm. Phanich et al. create a system capable of recommending recipes tailored to fit a diabetic's diet [22]. Their system uses clustering alongside a neural network, to respectively group like nutrient groups, and normalize their weights. This method allows for the system to allocate both a varied and nutritional regime that fits the diabetic user's current nutritional needs. The system, nevertheless, necessitates an excessive level of user interaction, as nutritionists need to populate the database of nutrients by hand, leading to a fairly large overhead.

Freyne et al. cite obesity as motivation for the development of a recipe recommendation system [23, 24], maintaining that an engaging diet recommendation system should be more advantageous to recommend suitable substitutes to unhealthy meals. The paper proceeds to compare various recommendation systems suitable for recipe recommendations. The evaluated algorithms include combinations of clustering algorithms (through ratings in the Likert scale) and ingredient similarity measurements, as well as machine learned algorithms. Advantage was ultimately awarded to the machine learned algorithms, though little granularity is given to the ingredient weights, since they are being ranked in binary values, and little experimentation is done in terms of collaborative filtering beyond clustering models.

Wagner et al. present a system aimed at improving a user's cooking experience as well as incentivizing the preparation of healthy foods [25]. Though their paper does little beyond recommending the existing system be coupled with a recommender system, they devise noteworthy methods of extracting data relevant to recipe recommendation based on skill level: by installing sensors on kitchen utensils, the system may gauge the skill level of users, such as speed of preparation, allowing for a better perception and may notice the development of user skills.

Sobecki et al. attempt to group users by their demographic attributes [74]. Through data extracted from user profiles, specific users are grouped, and recipes which are well liked by many members of that specific group are propagated to other members of the same group. Furthermore, rather than the system determining similarities on its own, this system was fully programmed with over 170 hand written rules to extract similarity data. The system is ultimately not evaluated, giving little information as to the efficacy of its recommendations.

Forbes and Zhu create a system for food recommendation relying solely on standard collaborative filtering [21]. The system employs both recipe-like correlations as well as content-based filtering, in terms of ingredients. The system, however, treats ingredient co-occurrence as a binary value, giving no weight to ingredients which constitute a greater portion of the recipe. Furthermore, the recipe-like correlation relies on the Likert scale, which as previously mentioned, suffers from considerable bias.

Kuo et al. devise a menu recommendation algorithm, in which recipe recommendations are translated into a graphing problem to create rather than one relevant recipe, a number of recipes constituting a menu [26]. Similar values are coupled via content-based filtering, and placed in a graph; a minimum spanning tree of the graph is then generated to determine viable recipe combinations, considering a user’s previously consumed recipes.

Li et al. propose a distributed peer-to-peer system to recommend recipes based on flavor [27]. In their systems, users have a reputation score based on their trustworthiness, and exchange recipes only if their profiles are sufficiently similar. Recipes are chosen based on their similarity, factoring in specific flavors (such as salty, sour, sweet, bitter, and umami) as similarity features.

Finally Teng et al. use data from allrecipes.com to obtain recipe recommendations [28]. In their paper, they obtain user ratings from the allrecipes database, and couple recipes through co-similarity scores of recipes, as well as co-occurrence scores of ingredients. Ingredient granularity, as for previous works, is binary.

Although the aforementioned studies make important contributions towards the recommendation literature, they are limited in their detail for both item similarity, and collaborative filtering metrics: item similarity when computed is calculated through binary valued ingredients, rather than taking into account more prominent values in recipes; this detail comes in useful in instances where recipes like hamburger and veggie burgers are taken into consideration: both recipes feature very similar ingredients, though the most prominent ingredient (that is, the patty) is not relevant enough in the binary model, perhaps leading to some discontent among vegetarian users. Also, in the aforementioned models, collaborative filtering requires manual entry of items through the Likert scale, a requirement which will not only lead to bias, but will suffer from the fact that a number of users will prefer not to interface with the applications in this manner, leading to a limited dataset.

3.3 Methodology

3.3.1 Refrigerator Model

The smart refrigerator is essentially a refrigerator archetype, embodying all of the usual features of a refrigerator, coupled with the capability to recommend dishes based on its use. In order to allow for recommendations, the refrigerator further possesses storage features, which keep track of food items contained within, as well as previously consumed recipes. For the purpose of collecting these values, we have created a simple web-page which interfaces with both Smart Server and the smart refrigerator to consume this data. The smart refrigerator client then simply comprises of a refrigerator coupled with a simple embedded computer, capable of Wi-Fi connectivity, and simple processing capabilities. It should nonetheless be noted that the capability of implicit data acquisition is a very feasible possibility, and it has only not been pursued due to insufficient knowledge in the area of electronics. However, the system could contain a camera to extract information from shopping receipts [82], and as such determine the food being input into the system.

Furthermore a scale could be introduced to the system to track food item weights: in this manner, assuming two items with differing weights, the system can conclude when one item was removed in lieu of another by calculating the amount of weight removed from the refrigerator. This method also allows for a user to remove an item, subtract a portion of that item, and return it to the fridge; this allows for a fridge to track recipes, as the removal of portions of items may represent part of a recipe. A full recipe can be represented as all removed ingredients from the time the fridge door is opened until it is closed.

3.3.2 Device Communication

The smart refrigerator client communicates with two external servers: Smart Server and the MASHA server. Both forms of communication follow RESTful paradigms [29]. Communications with the Smart Server comprise the storage of webpages used to interface with the client, the storage of past recipes for easy retrieval, and the storage of commands issued by users, to be pulled by the client. Communications with the MASHA server comprise the following:

1. Creation of a user account for the specific refrigerator.
2. Storage of past recipes for the purpose of computation of recommendations.
3. Storage of recipe ingredients.
4. Recommendation of recipes.

Unlike Smart Server, there is no direct retrieval of stored data, and the storage of data on that server serves merely as metrics to calculate viable recommendations.

In order to communicate with the Smart Server, an encryption and decryption application has been devised to allow smartphone users to access the protected websites as if they were in a real smart home. All code for the MASHA server, client, and the encryption and decryption application have been written in the Java programming language.

A typical recommendation use case, then, considering the mentioned devices would be as follows:

1. A user accesses the client webpage via the Smart Server hotspot, using his smart phone. Dependent on the user’s privileges, he may interact with the refrigerator differently:

a. A trusted guest will only have access to refrigerator contents, with solely the ability to add or remove items from the fridge. These communications are unencrypted.

b. Resident level users and above may add or create recipes to the user's recipe set, as well as get recommendations for future recipes. These communications are encrypted.

2. The user removes items from the fridge, updating the fridge client's ingredients as he does so, via the webpage; this is done via a direct, unencrypted communication[18] with the refrigerator through webpage requests.
3. The user uses the removed ingredients to make a recipe, utilizing the webpage to add the new recipe to the Smart Server through encrypted communication. This specific step may only be employed by residents.
4. The fridge client retrieves commands from the Smart Server periodically and encounters a new recipe.
5. The fridge client proceeds to update the MASHA server database with the new ingredient.
6. The fridge client asks the MASHA server for a recommendation for the user given the new update.
7. The fridge client updates the webpages within Smart Server to contain the new recommendations, to be consumed to resident level users.

[18] Recipe input commands make use of encrypted communications since perturbation from malicious sources may cause destructive consequences to the MASHA database. Conversely, the worst effect a perturbation to ingredient input and removal from the fridge may have is a temporary alteration to fridge contents. Furthermore, the disclosure of past recipes consumed may be undesirable for users, whereas the disclosure of simple ingredient interchange may present no discomfort.

The model for communication among devices and servers mentioned above is illustrated in Figure 3.1.

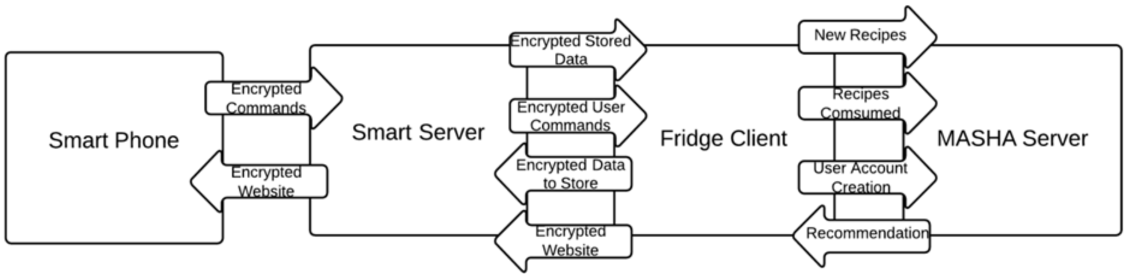


Figure 2.1 Device Interconnection in the Smart Home

3.3.3 MASHA Server

As mentioned in Section 3.2 of this chapter, the MASHA server features four main functions, all of which collectively serve one main purpose: recipe recommendation to the client. In order to issue the recommendations, different steps are employed, yielding ultimately 5 dishes to the client for recommended consumption. The entire process can be divided into two aspects: collaborative filtering and content-based filtering.

3.3.3.1 Collaborative Filtering Module

The collaborative filtering module utilizes data apposite to past meal consumptions. This module collects data in regards to recipe co-selection, acquired from standard fridge operation. Unlike conventional co-occurrence mechanisms, MASHA assembles the co-occurrence score through the sum of the minimum number of times each user has prepared both recipes, rather than simply the numbers of users which have had this co-occurrence arise. For instance, if one user has prepared recipe A once and recipe B fifteen times, the non-normalized co-occurrence score is one for that user. Formally, the non-normalized co-occurrence formula is as follows:

$$C_{AB} = \sum_{u=0}^{U} \min\{O_A^u, O_B^u\}$$

Where C_{AB} signifies the co-occurrence of items *A* and *B*, *O* signifies the occurrence of an item (*A* or *B*) for a specific user (*u*), and *U* signifies the total collection of users.

The method of calculation associated with this algorithm allows for a finer assessment of a user's contentment or discontent with a specific item, without necessitating a user's direct interaction or the use of any rating scale. A low score often signifies that a user who enjoys one of the items might not enjoy the other, and vice-versa. In order to produce a better interpretation given the body of data, however, this formula must be modified: this simple score alone gives no reference to the system as a whole. That is to say, we cannot determine if the value is significant by itself. Intrinsically, the normalized co-occurrence score was produced:

$$C'_{AB} = \frac{C_{AB}}{\sqrt{O_A * O_B}}$$

where C'_{AB} signifies the normalized co-occurrence of items *A* and *B,* and O signifies total occurrence between all users, rather than a single user's occurrence of an item. This final equation was derived from a modification of the co-occurrence equations present in both the Youtube (represented as a relatedness score) [16] and Amazon (represented as a commonality index) [15] recommendation systems. Each of these values is finally stored in an N by N matrix, containing normalized pairings for all co-occurrences.

Finally, the similarity score is calculated via Pearson Correlation Coefficient between an item's co-occurrence vector and a user's equivalent. The user's equivalent to a co-occurrence vector simply keeps track of the number of times each recipe has been eaten. Since Pearson yields results between -1 and 1, and the hybrid ranking aspect of the recommendation engine requires only positive scores, one is added to each result. If a specific recipe has been paired with other recipes in a similar manner to a user's pairings, it is deemed that the user should enjoy this specific recipe. For instance, if recipe A has been paired most often with recipes C and D, and recipe B has been paired most often with recipes E and F, and user U_1 has mostly eaten recipes C and D, and user U_2 has most often eaten recipes E and F, we assume U_1 should enjoy A, and U_2 should enjoy B.

Figure 3.2 exemplifies the non-normalized User-Recipe matrix and the Recipe-Recipe matrix, respectively.

$$\begin{bmatrix} \mathbf{2} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{3} & \mathbf{2} & \mathbf{3} & \mathbf{0} \\ \mathbf{4} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{0} & \mathbf{3} & \mathbf{5} & \mathbf{1} \\ \mathbf{3} & \mathbf{0} & \mathbf{3} & \mathbf{0} \\ \mathbf{5} & \mathbf{3} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix}$$

(a) User-Recipe (b) Recipe-Recipe

Figure 3.2 Non-Normalized Collaborative Filtering Matrix Example. (a) An example representation of each user's past recipe preparation dataset (i.e.: User 1 has prepared recipe 1, twice, recipe 2, once, recipe 3, once, and never has prepared recipe 4). (b) An example representation of the non-normalized co-occurrence matrix between recipes (i.e.: recipe 1 has co-occurred thrice with recipe 2, five times with recipe 3, and once with recipe 1).

3.3.3.2 Content-Based Filtering Module

For the content-based filtering module we attempt to model the food recommendation system in an analogous manner to document systems, correlating an ingredient to a tag, and its physical weight to the virtual weight of the tag. That is to say, if the recipe for French Fries contains 5 grams of salt, 30 grams of vegetable oil, and 1 kilogram of potatoes, the tags will be salt, vegetable oil, and potatoes, with respective weights of 5, 30, and 1000. These values are normalized using Term Frequency-Inverse Document Frequency, and input into a Recipe-Item matrix.

Finally, to determine the similarity scores, the Pearson Correlation Coefficient between each recipe in the database and all of a user's tried recipes is calculated. Like the content based filtering module, each value is added to 1, to ensure positive results are returned. This score is further multiplied by the number of times each of the user's recipe has been prepared by the user, in order to properly weigh each item: the reasoning behind this final weighing is to grant greater significance to similarities to items which the user enjoys more. This value is finally divided by the total number of recipes prepared by the user. The formula for this final normalization is as follows:

$$S_{AU} = \frac{\sum_{i=0}^{I} PCC(A,i) * O_i^U}{\sum_{i=0}^{I} O_i^U}$$

where S_{AU} signifies the similarity score for item *A* for user *U*, *PCC(A, i)* signifies the Pearson Correlation Coefficient between item *A* and current item *i*, *I* represents the complete list of items, and O_i^U signifies the occurrence of item *i* for user *U*.

$$\begin{bmatrix} 500 & 30 & 0 & 5 & 0 & 0 \\ 0 & 0 & 800 & 8 & 0 & 75 \\ 0 & 20 & 0 & 0 & 200 & 50 \\ 0 & 20 & 1000 & 0 & 0 & 80 \end{bmatrix}$$

Figure 3.3 Non-Normalized Content-Based Filtering Matrix Example. In this specific example, recipe one contains 500 grams of item 1, 30 grams of item 2, and 5 grams of item 4.

3.3.3.3 Recommendations

Final recommendation is determined by several factors, based on the two aforementioned filtering methods. However, prior to final ranking, further filtering is performed: any recipes which have been selected previously or recipes which contain items that are not present in the fridge are discarded. Finally, ranking is divided into four sub-categories: hybrid ranking, collaborative filtering ranking, content-based filtering ranking, and serendipitous ranking [72, 73].

Hybrid ranking results comprise of the top two scored results of the multiplication of each item's collaborative filtering score with its equivalent content-based filtering score. The reasoning for the previously mentioned addition of 1 to each Pearson Coefficient calculation for the collaborative filtering and content-based filtering modules lies in dealing with values where both results are negative, yielding positive results incorrectly. This hybrid method of computation has been inspired by the Google News recommendation engine [18], where a similar multiplication takes place to yield a more through recommendation result, allowing the recommendation to factor for more items.

Collaborative filtering ranking consists of the top result of all collaborative filtering scores, excluding the results selected from the hybrid ranking. This selection allows for retrieval of items which are well liked by users with similar past choices, albeit not necessarily similar in ingredients. Content-based filtering ranking produces the top result of all content-based filtering scores, omitting items selected by the two previous rankings. This ranking allows for recommendation of items which are similar in terms of ingredients, but are not necessarily well liked by users with similar past gastronomic experiences.

Lastly, serendipitous ranking retrieves an item at the midpoint of the list of hybrid recommendation scores. Serendipity allows a user to experience items outside of his comfort zone, lest he only experience very similar items. This allows for exploration of items which the user might not normally experience due to the fact the recommendation system factors only similarity for selections. Ultimately, this permits expansion of a user's range of gastronomic experiences.

3.4 Evaluation

In this section, we evaluate the efficacy of the recommendation system, alongside the ease of use of the fridge client, coupled with the Smart Server. For the purpose of evaluation, we query 20 different users with a choice of their ten favorite dishes, as well as the frequency with which these dishes would normally be prepared. The users are then given instructions as to how to install the system, as well as to populate the MASHA database through the fridge client. The users then assess the level of difficulty for the installation and use of the system. Finally, the users appraise the efficacy of the system through a quantitative ranking of results, as well as qualitative evaluation of their satisfaction with each dish.

The dishes presented to the users consist of the dish name and the dish ingredients and weights, and the five modules presented to each user consist of: two hybrid recommendations, one collaborative filtering recommendation, one content-based filtering recommendation, and one serendipitous recommendation; these results are not labeled to the users, and are meant to represent the exact choices given to the user when the system is fully functional: that is to say, the final functional MASHA should provide the user with all selections rather than simply the best one; they should however be presented to the user in order from best to worst, to allow better usability. This representation is finally meant to gage the proficiency of the system in terms of recommendation, as well as determine which method of recommendation prevails in this field.

3.4.1 Installation Results

Users were tasked with following a simple README file, consisting of six short lines. The necessary steps consisted of simply double clicking, switching windows, typing a short command consisting of 3 words and a number ("*setd refrigerator kitchen 1*"), and manually copying the resulting passkey from one window to another. The authors attempted to make the process as similar as possible to future manuals which would be included in products such as a smart refrigerator. The ease of use was finally evaluated through the Likert scale, where a value of 1 signified very difficult, and a value of 5 signified very easy; we understand the irony of utilizing the Likert scale after denouncing it for being ineffective, however it should be noted that it is sufficiently effective for simpler topics, as is the case in this specific instance of experimentation. Due to distance impediments, only 10 out of the 20 users were queried as to the difficulty of installation, as the tester’s physical presence was necessary to evaluate the installation completion.

Ultimately, users were content with the ease of use of the applications, some of them expressing surprise at the ease of installation of a server. Difficulties were met by some users who were unfamiliar with command-line environments; however, all users were ultimately able to install StrongHold and the fridge client without difficulties, all in less than two minutes. The Likert scale ranking averaged at 4.2, with 4 users rating the installation at very easy (5), 4 users rating it as easy (4), and 2 users rating it as moderate (3). It was generally recommended that a graphical user interface be implemented rather than command line to further simplify matters.

3.4.2 Usability Results

After the installation, each of the ten users was tasked with populating one recipe in the MASHA server[19]. To do so, the users had to open a web-page with the server ip, concatenated with their user-name, and navigate through it to get to the recipes page, and then populate a simple form. All users found this process straightforward, awarding the ease of the process a 5 in the Likert scale. The users further generally expressed that the method illustrated in this chapter’s Section 3.1, of automating recipe construction would be incredibly more convenient, though the method in place is not undesirable on its own.

3.4.3 Recipe Recommendation Results

In order to populate the recipe database, users were asked to present their favourite dishes, coupled with the frequency with which these dishes were consumed. After all recipes were populated, users were given the recommendations as exemplified in

[19] The reasoning for the population of a sole recipe rather than all ten is that only one recipe is necessary to gauge the ease of use. Anything beyond is solely repetition and can be accomplished by the authors without taking time from the users in this specific instance. Especially so as in normal use cases, a user should not have to populate more than one recipe at a time (i.e.: during meal preparation).

Section 3.3.3 of this chapter. In order to evaluate user response to recipes, we qualitatively compare the first four results with the serendipitous result, treating it analogously to a random recommendation[20]. Logically, if the first four results surpass the random equivalent, they at least suggest some tendency of satisfaction. Moreover, through the ranking we can determine the best method to establish recommendations. Finally, users were asked to qualitatively evaluate each of the results, and whether they manage to represent their tastes.

The non-random results were generally met with positive reviews, with the first hybrid recommendation frequently taking first place in the rankings, and the second hybrid recommendation recurrently taking second. The random recommendation never attained first rank, consistently placing in the latter half of the rankings (though it did rank second, once), maintaining the claim that the recommendations yielded by MASHA express some tendency of like. The quantitative results are displayed in figure 3.4.

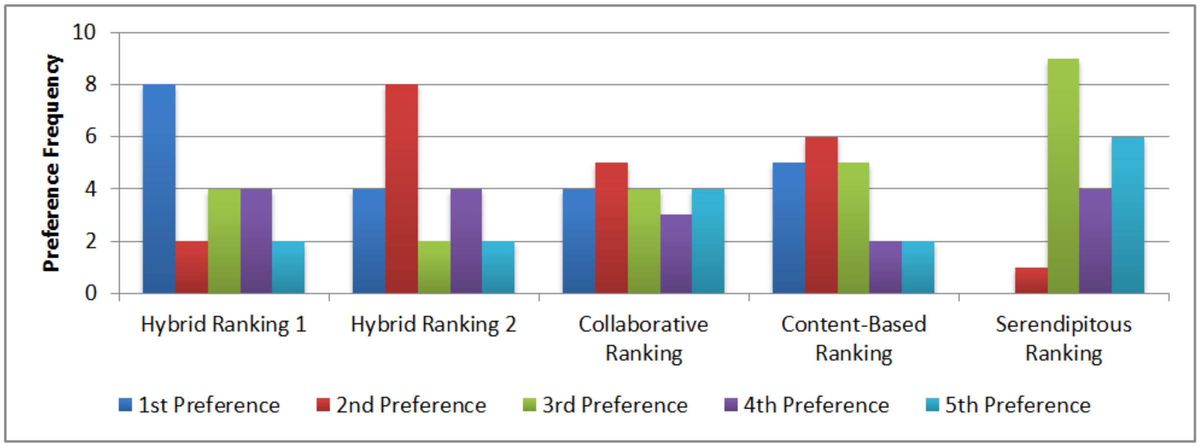


Figure 3.4 Ranking Results. The above histogram represents the number of users who have preferred each of the recommendation methods in each distinct ranking. For instance, 8 users have placed the first Hybrid Ranking result as their number one preference, 2 as their second preference, 4 as their third preference, 4 as their fourth preference, and two as their last preference. In this specific ranking, the lower the preference, the less effective the recommendation.

Of the two filtering methods, content-based filtering more frequently was regarded as producing good recommendation by users both quantitatively and

[20] We understand that the serendipitous result is by no means truly random, and typically should suggest some tendency of like due to its placement in the middle of a ranked preference list. Its nature however - especially in this small sample size - should represent a value which does not frequently agree with content or collaborative correlation, representing a somewhat random result.

qualitatively. The reasoning behind this result may be the granularity of both calculations, with collaborative filtering relying on data from 20 users, whereas content-based filtering relied on 96 distinct recipes with 156 distinct ingredients. Users were nonetheless content with the efficacy of both methods, most times. Content-based filtering and hybrid filtering further managed to weed out meat from recommendations to vegetarian users, as well as managing to recommend dishes falling within other users' dietary restrictions, since dishes containing familiar ingredients were given more weight than those with dissimilar contents. Conversely, collaborative filtering results were unable to filter for these users, though they generally were able to introduce users to desired dishes that did not befall their previously selected norm.

Most users were content with their overall recommendations, at times simply disliking the serendipitous result. In other cases, the serendipitous result proved useful, as users recognized they had forgotten about a specific favourite dish which the serendipitous result presented; often, this result deviated significantly from the user's selected dish choices, allowing for further variety. In a small number of cases, users were unhappy with their non-random recommendation choices, with collaborative filtering being the most common, with four dissatisfying results (three of which were due to dietary reasons). The second hybrid recommendation came in second, with three dissatisfying results (one of which consisted a dish a user had not tried, and as such did not think would be appreciated), and the first hybrid result, in third, with one dissatisfying result.

Assuming a numerical sum, wherein a first rank awards a recommendation method 5 points, a second rank awards 4 points, and so on, the recommendations rank as follows: The first Hybrid Recommendation and Content-Based Recommendation tie with a score of 70, the second Hybrid Recommendation follows closely with a score of 68, Collaborative Recommendation comes next with a score of 62, and Serendipitous Recommendation comes last with a score of 45. This ranking method however, is simply provided to allow better perception of the overall quality of ranking across all methods, and unlike the quantitative equivalent should be taken with a grain of salt: the hybrid methods place sufficiently well in the first and second ranking respectively, with the content-based method contending solely because it is effective across all rankings (i.e.: it is almost equally likely to be chosen within the first three preferences), though it is not guaranteed to rank highest at all times like the hybrid methods. This ranking process however, allows seeing just how far the difference lies between recommended values and the random equivalent, with the serendipitous method lying at less than 65% efficacy of the best ranking methods.

3.4.4 Discussion

Ultimately, the recommendation system proved successful, with many users remarking the need of such a device in their home. All results proved significant, with the hybrid results ranking highest, followed by content-based filtering, and collaborative filtering. Collaborative filtering would probably prove most useful, given a much larger user-base which we were unable to provide for these preliminary tests. Lastly, not all recommendations are easily accomplished, since an appropriation for immeasurable variables such as involuntary memory brought by dishes, such as those famously referenced by Proust [83], must be adopted.

3.5 Future Work

As mentioned prior, a number of items can be further explored in this study: a future iteration with a greater user-base would be a welcome addition, allowing for a better understanding of the different recommendations when the number of dishes and ingredients grows sufficiently large. The user interface of the client and server can be greatly improved upon to allow easier interaction with the user, and the refrigerator client can be greatly improved through the scale module, allowing for easier use. Most importantly, MASHA has only been tested in a laboratory environment, and would gain from a home environment study, wherein recipes are entered as dishes are consumed, rather than through a questionnaire. Moreover, it would be important to explore the possible limitation of the use of MASHA for an entire family rather than for one user, where dishes would be prepared by one user for an entire family, and rather than only for her; in such a situation, dishes recommended would pertain to an average of all users, perhaps appeasing no one.

3.6 Conclusion

The determination of preference from a pre-existing dataset is a challenging endeavor, pertaining to a field where one solution might not be directly translatable to other challenges. We design and implement a method capable of automatically determining culinary preferences based on past eating habits, translated and combined from current leading recommendation engines from different fields, and test it on a group of 20 people. From the results we determine both the usability and efficacy of the method, and show the algorithm allows for satisfactory recommendations, while requiring minimal effort from the user. We further prove the facility of interfacing with StrongHold, to further establish the simplicity of installation and implementation of a smart home.

Chapter 4: Conclusion

4.1 Introduction

This thesis was introduced with two topics in mind, and in a manner each was crafted to support the other. After all, it is paramount to have a foundation before a building is erected; or in this specific case, it is important that a communication and storage method be devised before a system is ready for home use. In finding a lack of systems that fit our various specific criteria in the literature, we chose to devise our own. We hoped that such a system could ultimately be utilized in standard home environments, and further expanded to fit future needs. With that aim in mind, we resolved a number of challenges:

- The creation of a viable smart home server, capable of protecting a user's privacy, confidentiality, all the while maintaining availability.
- The amalgamation of numerous isolated techniques existing in the literature onto one system.
- The allowance of utilization no more challenging than that pertaining to a conventional home though a learning algorithm devised for automatic privilege modification based on the manner of server access.
- A devising of a system with sufficient elasticity necessary to permit generalization for the use by a variety of devices from differing manufacturers.
- The development of a method to allow fast storage and retrieval to appropriate for the needs of a smart environment.
- The fabrication of a smart home server presently deployable outside of a testing environment.
- The creation of an unobtrusive meal recommendation system for the smart home.
- The advancing of the field of recommendations; specifically an increase in granularity in the sub-field of recipe recommendations.
- The positing a new more granular method of tag extraction through direct gram to weight ratio of ingredients rather than previously advanced method of binary measurements.

- The determination of the most suitable filtering methods to employ in the sub-field of recipe recommendations.

Moreover, we provide with the meal recommendation application both a realistic utilization to the storage infrastructure and a benchmark pertaining to that infrastructure. Intrinsically, with the devising of the application, we prove the feasibility of the storage infrastructure. The coupling further provides a container for the meal recommendation system, allowing a fully symbiotic relationship for both components.

We furthermore answered questions as to the usability of the system, and the speed and functions available by testing the system on realistic benchmarks, and implementing the system for a representative function: that of a smart refrigerator. Our conclusive goal was to determine what steps would be necessary to produce a smart home environment usable by the public, which disallowed any detrimental actions non-existent in unconnected environments. In doing so we chose and expounded on possible methods, and presented their practicalities and capabilities.

4.2 Contributions

For the StrongHold portion of the work, we suggest a modification of the resurrected duckling protocol, with an emphasis on ease of use brought by automated policies, pre-set by device manufacturers. The logical assumption being that the least amount of burden left to the home users the better, as a shorter learning curve both expedites a transition and ensures fewer detractors. Web-page access rather than separate application access for each device is chosen to lessen saturation, and its inherent confusion; ease of use is also a powerful motive, since web technologies are more familiar than standalone applications. Installation is made seamless as to require minimal user interaction to allow for easier transition from traditional homes. All communication within the home is secured by AES, and key resets are obtainable via the so far unbreakable SPEKE; the elasticity of these features allow manufacturers the ability to establish the use or disuse of AES, as well as the option to use SPEKE sporadically or continuously. Security is further maintained via the big stick principle, wherein a home resident must be physically present to grant access to a new user, and augmented via an automatic escalation process based on the principle of behavioral learning; that is to say, proper use is rewarded, and improper use punished. Finally, an optimization algorithm is devised to appropriate sufficient performance for real time media storage and retrieval. A means of general access through the use of RESTful APIs is chosen to allow effortless interfacing through any programming language capable of network communications.

To ensure the viability of StrongHold in a real home environment, rather than a laboratory equivalent, a number of tests are devised, including user tests to determine ease of use. It is ultimately determined that StrongHold is sufficiently robust to handle high definition streaming in real time, given that a number of guidelines are followed dependent on file size and type stored. Moreover, it is concluded that the server is sufficiently easy to install and operate, requiring an unobtrusively small number of interactions. It is further determined that the encryption mechanisms put in place have little to no detrimental effects on data throughput, especially contrasted to transit time in the wireless network; this effect is further substantiated due to tests being performed in older computers. Through these results and implementations, we introduce – to our knowledge – the first fully integrated and generalizable laboratory- and home-ready service aimed at secure storage, aggregation, and retrieval of data within the smart home.

Owing to the small amount of research extant on the subject of meal recommendations, we choose to expand it, as it is a significant issue possibly allowing for better health-minded dietary recommendations in the future: in doing so, our focus is to expand the field beyond sole collaborative-filtering, and include a high level of content-based filtering. We translate filtering and normalization methods from the document paradigm, and exhibit the effectiveness of the inclusion of a high-level content-based filtering algorithm on meal recommendation by assessing user preferences. We also adapt the mechanism to function in a smart environment in which interactions deviating from the current norm may be burdensome. Finally we prove the efficacy of the filtering system by demonstrating its effectiveness against a random equivalent, and rank filtering methods presenting that a hybrid approach, combining both collaborative- and content-based filtering, is the best option for effective recommendations.

4.3 Future Work

Though the results presented and solutions postulated are significant, the fields researched are novel, and as such tests that are truly representative of the environment in which they should be utilized can by no means be all-encompassing. As the fields evolve and more data is present, better means of validation and gaging of viability may be deemed necessary. Nonetheless, as it stands, user studies and test with a greater number of participants, and in real environments, rather than laboratory environments should greatly improve the work.

More robust security experimentation to ensure proper functionality of StrongHold within such a sensitive realm as one's home would greatly aid in the study. As further threats arise, additional effort should be spent to continually secure StrongHold, since the field of security by no means stands still. Human Computer Interactions (HCI) is another area in need of improvement, as this work utilized only command shell interactions; a better means of user interface should greatly contribute to the work, since ease of use is paramount in the field. A more extensive learning algorithm for user escalation should also be considered for future versions, including a means of devices to share trust levels between each other. Lastly, emergency situations should be further explored to appropriate better resolutions in situations which necessitate interactions which significantly differ from the norm.

MASHA, currently, is merely a software tool, and lacks the hardware necessary to be truly unobtrusively incorporated into a smart home. As is, it still requires excessive interactions for a smart home; a refrigerator which could keep track of food items inserted and retrieved, and their weights, as presented in chapter 3, would suffice, though significant hardware effort is necessary to bring it to fruition. Finally an integration of the meal recommendation service with a dietary-plan recommendation service should prove a valuable addition to the system, hopefully aiding in combatting health issues from dietary sources.

Bibliography

[1] M. Weiser. “The computer for the 21st century.” *SIGMOBILE Mob. Comput. Commun. Rev. 3*, pp. 3-11, Jul. 1999.

[2] M. Honan. “No One Uses Smart TV Internet Because It Sucks.” Wired. http://www.wired.com/gadgetlab/2012/12/internet-tv-sucks/, Dec. 27, 2012.

[3] A. Nusca. “Are consumers really asking for smart TVs?” ZDNet. http://www.zdnet.com/are-consumers-really-asking-for-smart-tvs-7000009479/, Jan. 7, 2013.

[4] PC Authority, “Whatever Happened to: the Internet-Connected Fridge.” http://www.pcauthority.com.au/News/152906,whatever-happened-to-the-internet-connected-fridge.aspx, Aug. 14, 2009.

[5] Sydney Morning Herald, “Failure to Launch.” http://www.smh.com.au/digital-life/digital-life-news/failure-to-launch-20100120-mk8g.html, Jan. 25, 2010.

[6] D. Pishva and K. Takeda. “A Product Based Security Model for Smart Home Appliances.” Carnahan Conferences Security Technology, Proceedings 2006 40th Annual IEEE International, pp.234-242, Oct. 2006.

[7] M. Dawson. “Smart Kitchens Could Cook Up a Strong Future.” Realty Times. http://realtytimes.com/rtpages/20050222_smartkitchens.htm, Feb. 2005.

[8] M. Shiels. “Cisco Predicts Internet Device Boom.” BBC. http://www.bbc.co.uk/news/technology-13613536, Jun. 2011.

[9] T. H. Kim, L. Bauer, J. Newsome, A. Perrig and J. Walker. “Challenges in access right assignment for secure home networks.” Usenix HotSec '10, 2010.

[10] L. Compagna, P. El-Khoury, F. Massacci, and A. Saidane. “A dynamic security framework for ambient intelligent systems: a smart-home based eHealth application.” In Transactions on computational science X, Marina L. Gavrilova, C. J. Kenneth Tan, and Edward David Moreno (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 1-24, 2010.

[11] M. Hoque, M. Rahman, S. Ahamed and L. Liu. “Trust based security auto-configuration for smart assisted living environments.” In Proceedings of the 2nd ACM workshop on Assurable and usable security configuration (SafeConfig '09). ACM, New York, NY, USA, pp. 7-12, 2009.

[12] M. Johnson and F. Stajano. “Usability of Security Management: Defining the Permissions of Guests.” In Proceedings of Security Protocols Workshop, 2006.

[13] D. Lee, G. Kim, J. Han, Y. Jeong, D. Park. “Smart Environment Authentication: Multi-domain Authentication, Authorization, Security Policy for Pervasive Network.” In Ubiquitous Multimedia Computing. UMC '08. International Symposium on, pp.99-104, 2008.

[14] G. Linden, B. Smith and J. York. "Amazon.com recommendations: item-to-item collaborative filtering." Internet Computing, IEEE , vol.7, no.1, pp.76-80, Jan./Feb. 2003.

[15] G. Linden, J. Jacobi and E. Benson. “Collaborative Recommendations Using Item to Item Similarity Mappings.” United States of America. US 6,266,649 B1, Jul. 24, 2001.

[16] J. Davidson. B. Liebald, J. Lie, P. Nandy and T. Vleet. “The YouTube Video Recommendation System.” ACM conference on Recommender systems (RecSys '10), pp. 293-296, Sept. 2010.

[17] A. Das, M. Datar, A. Garg and S. Rajaram. “Google News Personalization: Scalable Online Collaborative Filtering.” International conference on World Wide Web (WWW '07), pp. 271-280, May 2007.

[18] J. Liu, P. Dolan and E. Pedersen. “Personalized News Recommendation Based on Click Behavior.” International conference on Intelligent user interfaces (IUI '10), pp. 31-40, Feb. 2010.

[19] Y. Koren. “The BellKor solution to the Netflix Grand Prize.” 2009.

[20] A. Toscher and M. Jahrer. “The BigChaos Solution to the Netflix Grand Prize.” Sept. 2009.

[21] P. Forbes and M. Zhu, M. “Content-Boosted Matrix Factorization for Recommender Systems: Experiments with Recipe Recommendation.” ACM Conference on Recommender Systems (RecSys '11), pp. 261-264. Oct. 2011.

[22] M. Phanich, P. Pholkul and S. Phimoltares. "Food Recommendation System Using Clustering Analysis for Diabetic Patients." Information Science and Applications (ICISA), 2010 International Conference on, pp.1,8, 21-23 April 2010

[23] J. Freyne and S. Berkovsky. “Intelligent Food Planning: Personalized Recipe Recommendation.” Conference on Intelligent user interfaces (IUI '10). ACM, New York, NY, USA, pp. 321-324, Feb. 2010.

[24] J. Freyne, S. Berkovsky and G. Smith. “Recipe Recommendation: Accuracy and Reasoning.” Conference on User Modeling, Adaption, and Personalization (UMAP'11). Springer-Verlag, Berlin, Heidelberg, pp. 99-110, 2011.

[25] J. Wagner, G. Geleijnse and A. Halteren. “Guidance and Support for Healthy Food Preparation in an Augmented Kitchen.” Workshop on Context-awareness in Retrieval and Recommendation (CaRR '11). ACM, New York, NY, USA, pp. 47-50, Feb. 2011.

[26] F. Kuo, C. Li, M. Shan and S. Lee. “Intelligent Menu Planning: Recommending Set of Recipes by Ingredients.” Workshop on Multimedia for Cooking and Eating Activities (CEA '12). ACM, New York, NY, USA, pp. 1-6, Nov. 2012.

[27] Q. Li, W. Chen and L. Yu. “Community-Based Recipe Recommendation and Adaptation in Peer-To-Peer Networks.” Conference on Ubiquitous Information Management and Communication (ICUIMC '10). ACM, New York, NY, USA, Article 18, 6 pages, Jan. 2010.

[28] C. Teng, Y. Lin and L. Adamic.” Recipe Recommendation Using Ingredient Networks.” Web Science Conference (WebSci '12). ACM, New York, NY, USA, pp. 298-307, June 2012.

[29] R. Fielding. “Architectural Styles and The Design of Network-Based Software Architectures.” Ph.D. dissertation, Dept. Inf. And Comp. Sci., Univ. California, Irvine, CA, 2000.

[30] K. S. Jones. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," Journal of Documentation, 28 (1). 1972.

[31] G. Salton, E. Fox and W. Wu. "Extended Boolean Information Retrieval." Communications of the ACM, 26 (11), 1983.

[32] H. Wu, R. Luk, K. Wong and K. Kwok. "Interpreting TF-IDF Term Weights As Making Relevance Decisions." ACM Transactions on Information Systems, 26 (3), 2008.

[33] Y. Yan, Y. Qian, H. Sharif. “A Secure Data Aggregation and Dispatch Scheme for Home Area Networks in Smart Grid.” Global Telecommunications Conference (GLOBECOM 2011), IEEE. pp.1-6, 5-9, Dec. 2011

[34] T. Manish. “A Location Based Security Implementation in Smart Home.” In Proc. 10th IEEE International Conference on High Performance Computing and Communications (HPCC '08), IEEE Computer Society, Washington, DC, USA, pp. 1007-1011, Sept. 2008.

[35] Y. Duan, and J. Canny. “Protecting user data in ubiquitous computing environments: Towards trustworthy environments.” In Workshop on Privacy Enhancing Technology, pp. 167-185, 2004.

[36] J. Kim, A. Beresford and F. Stajano. “Towards a security policy for ubiquitous healthcare systems.” In Proc. of the 1st international conference on Ubiquitous convergence technology (ICUCT'06), Frank Stajano, Hyoung Joong Kim, Jong-Suk Chae, and Seong-Dong Kim (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 263-272, 2006.

[37] F. Stajano. “Security Issues in Ubiquitous Computing.” H. Nakashima, H. Aghajan, J. C. Augusto (Eds.). Handbook of Ambient Intelligence and Smart Environments, Springer, 2010.

[38] J. Cornwell, I. Fette, G. Hsieh, M. Prabaker, J. Rao, K. Tang, K. Vaniea, L. Bauer, L. Cranor, J. Hong, B. McLaren, M. Reiter and N. Sadeh. “User-Controllable Security and Privacy for Pervasive Computing.” In Proc. of the Eighth IEEE Workshop on Mobile Computing Systems and Applications (HOTMOBILE '07). IEEE Computer Society, Washington, DC, USA, pp. 14-19, 2007.

[39] S. Naqvi and M. Riguidel. “Security and trust assurances for smart environments. In Mobile Adhoc and Sensor Systems Conference.” IEEE International Conference on, pp.8, pp.234, 7-7, Nov. 2005.

[40] F. Stajano and R. Anderson. “The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks.” In Proc. of the 7th International Workshop on Security Protocols, Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe (Eds.). Springer-Verlag, London, UK, 172-194, 1999.

[41] R. Zhang, X. Wang, R. Farley, X. Yang and X. Jiang. “On the feasibility of launching the man-in-the-middle attacks on VoIP from remote attackers.” In Proc. of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS ’09). ACM, New York, NY, USA, pp.61-69, 2009.

[42] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas and S. Yi. “Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments.” Distributed Computing Systems, 2002. Proc. 22nd International Conference on, pp.74,83, 2002.

[43] M. Spreitzer and M. Theimer. “Providing location information in a ubiquitous computing environment.” In Proceedings of the fourteenth ACM symposium on Operating systems principles. ACM, New York, NY, USA, pp.270-283, 1993.

[44] M. Liberatore and B. Levine. “Inferring the source of encrypted HTTP connections.” In Proc. of the 13th ACM conference on Computer and communications security. ACM, New York, NY, USA, pp.255-263, 2006.

[45] T. Saponas, J. Lester, C. Hartung, S. Agarwal and T. Kohno. “Devices that tell on you: privacy trends in consumer ubiquitous computing.” In Proc. of 16th USENIX Security Symposium on USENIX Security Symposium, 2007.

[46] M. Enev, S. Gupta, T. Kohno and S. Patel. “Televisions, video privacy, and Powerline electromagnetic interference.” In Proc. of the 18th ACM conference on Computer and communications security. ACM, New York, NY, USA, pp. 537-550, 2011.

[47] C. Wright, L. Ballard, F. Monrose and G. Masson. “Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?” In Proc. of 16th USENIX Security Symposium on USENIX Security Symposium, Niels Provos (Ed.). USENIX Association, Berkeley, CA, USA, Article 4, pp.1-12, 2007.

[48] *Specifications for the Advanced Encryption Standard*, Federal Information Processing Standards Publication 197, October 2, 2012.

[49] D. Jablon. “Strong Password-Only Authenticated Key Exchange.” Computer Communication Review 26 (5): pp.5–26

[50] J. Daemen and V. Rijmen. “The block cipher Rijndael, Smart Card research and Applications.” LNCS 1820, Springer-Verlag, pp. 288-296.

[51] D. Landman. “AES Library.” https://github.com/DavyLandman/AESLib, 2013.

[52] J. Lahart. “Taking an Open Source Approach to Hardware.” The Wall Street Journal. http://online.wsj.com/article/SB10001424052748703499404574559960271468066.html, Nov, 2009.

[53] P. MacKenzie. “On the Security of the SPEKE Password-Authenticated Key Exchange Protocol.” Cryptology ePrint Archive, Report 2001/57, pp. 1-19, 2001.

[54] F. Stajano. “Security for Ubiquitous Computing.” John Wiley & Sons, Ltd, 2002.

[55] S. Nasution, P. Hartel, N. Suryana, N. Azman and S. Sahib. “Trust Level and Routing Selection for Mobile Agents in a Smart Home.” In Proc. of the 2010 Second International Conference on Computer Modeling and Simulation, Vol. 3. IEEE Computer Society, Washington, DC, USA, pp. 445-450, 2010.

[56] J. Marc Seigneur, C.D. Jensen, S. Farrell, E. Gray and Y. Chen. “Towards Security Auto-Configuration for Smart Appliances.” In Proc. of the Smart Objects Conference, 2003.

[57] APIGEE. “Teach a Dog to REST.” http://blog.apigee.com/detail/restful_api_design, 2011.

[58] E. Rozier, W. Sanders, P. Zhou, N. Mandagere, S. Uttamchandani and M. Yakushev. “Modeling the Fault Tolerance Consequences of Deduplication.” In Proc. of the 2011 IEEE 30th International Symposium on Reliable Distributed Systems. IEEE Computer Society, Washington, DC, USA, pp.75-84, 2011.

[59] P. Read and M. Meyer. “Restoration of motion picture film. Conservation and Museology.” Butterworth-Heinemann. pp. 24–26, 2000.

[60] K. Brownlow. “Silent Films: What Was the Right Speed?” Sight & Sound 49 (3): pp. 164–167, 1980.

[61] D. Wagner and P. Soto. “Mimicry attacks on host-based intrusion detection systems.” In Proc. of the 9th ACM Conference on Computer and Communications Security. V. Atluri, Ed. ACM, New York, NY, pp. 255-264, Nov. 2002.

[62] GSM Arena. “Samsung Galaxy S4 Technical Specifications.” http://www.gsmarena.com/samsung_i9505_galaxy_s4-5371.php, 2013.

[63] F. Stajano. “The Resurrecting Duckling - What Next?” In Revised Papers from the 8th International Workshop on Security Protocols, Bruce Christianson, Bruno Crispo, and Michael Roe (Eds.). Springer-Verlag, London, UK, pp. 204-214, 2000.

[64] P. Argyroudis and D. O’Mahony. “Securing communications in the smart home.” in Proc. of International Conference on Embedded and Ubiquitous Computing, pp. 891-902, 2004.

[65] J. Lindberg and M. Blomberg. “Vulnerability in speaker verification - a study of technical impostor techniques.” In Proc. of the European Conference on Speech Communication and Technology, vol. 3, pp. 1211-1214, 1999.

[66] M. Faundez-Zanuy. “On the vulnerability of biometric security systems.” In Aerospace and Electronic Systems Magazine, IEEE, vol.19, no.6, pp. 3-8, 2004.

[67] Z. Akhtar, G. Fumera, G. Marcialis and F. Roli. “Robustness analysis of likelihood ratio score fusion rule for multimodal biometric systems under spoof attacks.” In Security Technology, 2011 IEEE International Carnahan Conference on, pp.1-8, 18-21, 2011.

[68] C. Brodie, C. Karat and J. Karat. “An Empirical Study of Natural Language Parsing of Privacy Policy Rules Using the SPARCLE Policy Workbench.” In Proc. of the Usable Privacy and Security, 2006.

[69] A. Francillon, B. Danev and S. Capkun. “Relay attacks on passive keyless entry and start systems in modern cars.” In Proc. of NDSS, 2010.

[70] Pandora Internet Radio, www.pandora.com.

[71] W. Hill, L. Stead, M. Rosenstein and G. Furnas. “Recommending and Evaluating Choices in a Virtual Community of Use,” ACM CHI’95 Conference on Human Factors in Computing Systems, pp. 194–201, 1995.

[72] L. Cao and M. Guo. "Consistent Music Recommendation in Heterogeneous Pervasive Environment," Parallel and Distributed Processing with Applications, pp.495, 501, 10-12, Dec. 2008.

[73] J. Herlocker, J. Konstan, L. Terveen and J. Riedl. “Evaluating Collaborative Filtering Recommender Systems,” ACM Trans. Inf. Syst, pp. 5-53, Jan.2004.

[74] J. Sobecki, E. Babiak, M. Słanina. “Application of Hybrid Recommendation in Web-Based Cooking Assistant,” International conference on Knowledge-Based Intelligent Information and Engineering Systems, vol. 3, B. Gabrys, R. Howlett, L. Jain (Eds.), Berlin, Heidelberg: Springer-Verlag, pp. 797-804, 2006.

[75] M. Montaner, B. Lopez and J. De La Rosa. “A Taxonomy of Recommender Agents on the Internet,” Artificial Intelligence Review, pp. 285-330, 2003.

[76] K. Iwahama, Y. Hijikata and S. Nishida. "Content-based filtering system for music data," Applications and the Internet Workshops, pp.480, 487, 26-30, Jan. 2004.

[77] R. Pan, Y. Zhou, B. Cao, N.N. Liu, R. Lukose, M. Scholz and Q. Yang. "One-Class Collaborative Filtering," ICDM '08. Eighth IEEE International Conference on, pp.502, 511, 15-19, Dec. 2008.

[78] Y. Koren and R. Bell. “Advances in Collaborative Filtering,” in Recommender Systems Handbook, 1st ed., vol. 1. F. Ricci, L. Rokach, B. Shapira, P. Kantor, Eds. United States: Springer, pp.145-186, 2011.

[79] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl. “Grouplens: An Open Architecture for Collaborative Filtering Of Netnews,” ACM conference on Computer Supported Cooperative Work, pp., 175-186, 1994.

[80] J. Schafer, J. Konstan and J. Riedl, J. “E-Commerce Recommendation Applications,” Data Min. Knowl. Discov., pp. 5, 1-2, 115-153, Jan. 2001.

[81] The Netflix Prize, http://www.netflixprize.com/.

[82] I. Liu, I. Chen and M. Chen. “Le Festin: Shop Sign Recognition Assisted Food Recommendation System,” Wearable Computers, 2010 International Symposium on, pp., 1, 8, 10-13, Oct.2010.

[83] M. Proust, “Remembrance of Things Past”, (Transl.: C. K. Scott Moncrieff, Terence Kilmartin, and Andreas Mayor (Vol. 7). New York: Random House, 1981).

[84] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, “An Operating System for the Home,” in *NSDI*, USENIX, Apr. 2012.

Appendices

Appendix A Application Programming Interface

This appendix section serves the purpose of expounding on the construct of the Application Programming Interface of StrongHold, and will be subdivided into the top level resources, with each subsection further divided into each of the CRUD states.

A.1 The *home* Resource

Accesses to the sub-URI labeled "*home*" are accessible by devices to store, retrieve, create, or delete device data. Device data does not require very rigorous formatting, and follows two simple rules:

1. Each device data entry should be preceded by an entry ID. This is not a strict necessity; however, failure to follow this rule will significantly slow down storage and retrieval times for reasons illustrated in Section 4.2.2 of chapter 2. The ID may consist of any number of alphanumeric characters.
2. Values within an entry are delimited by a plus sign. All entry values may consist of any number of alphanumeric characters, with no limit to length.

A.1.1 Create

Prior to utilizing the data storage mechanism, folders may be set up to organize data from the device. A folder is created by POSTing to the home resource. The secondary resource must contain the location the device is placed within, followed by the device name. All this data is unencrypted. All queries following the question mark however, are encrypted if the device's encryption level is higher than 0. The reasoning behind this implementation is so that the communications not fall prey to the same threats WEP fell prey to [51]. Since repeated values in a formatting can aid an attacker in guessing the encryption of an exchange, we try to keep repeated or guessable values to a minimum. Due to the fact that a device's name and location can be guessed from the originating IP, these values were left unencrypted. Furthermore, it should be of no real use to the attacker to know which device is communicating; the data within should be the only value of importance. The data encryption also serves the secondary purpose of identification: only a device which knows its key should be able to encrypt the option name. That is to say, only a device in possession of its own key can properly encrypt "*?path=x*" in a POST query. This is verifiable by the server attempting to decrypt the message.

There exists only one possible query to this method of the resource, and that query is "*path=*". Following "*path=*" the device is expected to enter the new folder name to be created. It is recommended that this name be randomized, as it is to be repeated, assumingly repetitively in home GET queries. As such, if randomization does not occur, an attacker can more easily guess the encryption key.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/home/kitchen/toaster/?path=new_path
```

A.1.2 Update

In order to send device data to the server, a device must send an update request in the form of a PUT. PUT requests are formatted similarly to POST requests, with the device location following the resource name, and the device name following the location. Encryption for PUT also functions in the same manner as POST. The only difference in formatting lies in the options; the possible queries to PUT are as follows, and are delimited by an ampersand sign:

- **path –** path represents the folder the data will be placed within. Attempting to place to non-existent folders results in an error sent back to the device. If a path is not specified, the data is placed in the top folder for the device.
- **data –** data represents the data to be stored in the server. As previously mentioned, it must be delimited by the plus sign, and a preceding ID is recommended.
- **time –** if a device is set at level 2 (encryption + nonce), the time option must be set. This option disallows relay attacks, as current time (with a variability of 30 seconds) must be encrypted somewhere within the message to the server. Time value is in Unix Time. Section A.4 describes how to acquire current system time.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/home/kitchen/toaster/?path=path1&data=123+456+1
01112131415161711181920&time=1359446309
```

A.1.3 Read

To retrieve data stored within the server a device must issue a read request in the form of a GET. GET requests to home follow the same pattern as create and update requests, with the options being the only divergent variables. The acceptable queries for GET are:

- **path –** the path variable, as in previous instances specifies the folder from which the data will be retrieved.
- **id –** the id variable specifies the specific ID (or IDs) to be retrieved. In order to diminish load to less powerful devices, ID will only return the first ID match. If the device wishes to return more than one instance of an ID, the ID must be placed twice within this option, delimited by a plus sign. If the device wishes to return a span of IDs, the start ID must precede the end ID, and should be delimited by space. An id value of "*" returns all matches.
- **match –** the match variable allows one more degree of configuration to the user, allowing for a match anywhere within the line. If the entry matches the match variable, that entry will be returned.
- **limiting –** limiting specifies the maximum number of entries which will be returned by this query. If unspecified in the query, limiting is set as 1. A limiting value of 0 returns all matching values.
- **offset –** offset specifies the displacement from which the retrieval should begin. For instance, if offset is set to 3, offset will return from the third entry which matches the query. If unspecified, offset is set to 0.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/home/kitchen/toaster/?path=path1&id=123&match=4
56&limiting=1&offset=0
```

Queries to get are returned as JSON objects for easy retrieval of information.

A.1.4 Delete

If a device wishes to delete a specific entry, it may send a DELETE request to the server. The formatting once more does not change in comparison to create, update or read except in terms of options. Acceptable queries are as follows:

- **path** – the path variable, as in previous instances specifies the folder from which the data will be deleted from.
- **id** – id follows the same logic in here as in read, and returns id matches.
- **match** – match follows the same logic here as in read, and returns entries which match the match variable.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/home/kitchen/toaster/?path=path1&id=123&match=4
56
```

A.2 The *web* resource

The "*web*" resource exists as a means of managing and retrieving web pages and user data for users. It is through this resource that devices and users may communicate. Devices may assume the identity of a "user-device" if they wish to communicate with other devices. That is to say, they must assume their own username, and may access GET requests if they wish to use the server to inter communicate. However, through these accesses, they are susceptible to the same restrictions users follow, including access levels.

A.2.1Create

POST queries in the web resource create the usage policy of a device. Through sending a POST request, a device may specify what the threshold is for each user level for the specific device. User levels range from Guest, to Trusted Guest, to Resident, to Power Resident, to System Administrator. The POST hierarchy is similar to all home hierarchies, with the device location and device name directly following "*web*". The queries available to web are as follows:

- **threshold –**threshold dictates the minimum experience threshold to access this level of the resource. Threshold values should be delimited by + signs, and if left blank are set to default to 0+75+1500+7500+15000. This value sets a user as a guest at 0 hours of non-stop access, a trusted guest at 1.5 days, a resident at 30 days, a power resident at 150 days, and a system administrator at 300 days. As normal users do not access devices 24/7, the time to escalation is expected to be much greater than these values in reality.

- **level –** The level couples threshold with the user level. For instance, level=0+1&treshold=0+75 makes guest access be at 0 points and trusted guest at 75 points. If left blank, the coupling defaults to in order coupling, meaning that each value in threshold will pair with its placement. For instance, if threshold is set to 0+50+60, level 0 is set to 0, level 1 is set to 50 and level 2 is set to 60.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/web/kitchen/toaster/?level=0+2+4&threshold=0+10
0+5000
```

A.2.2 Update

Update requests allow devices to set web pages and user data on the server. Those are achieved via PUT requests. Though the URI is formatted just like the create method, queries are formatted differently. Also, unlike other resources, delimiting of options requires three ampersands (i.e.:&&&), and equal signs are likewise tripled (i.e.: ===) to not conflict with ampersands and equalities extant in JavaScript code. Following are the available options for update requests:

- **level –** level specifies which level the user must be to see access this data. Level ranges from 0 to 4.
- **filename –** filename specifies the name of the file to be accessed.
- **datatype –** datatype may be set to data or web. Web files are html files visible to the user directly by accessing the device. Data files house other necessary resources, such as images or data which may populate the web page.
- **data –** data represents the data which is to be input onto the server. If the datatype is web, data is expected to contain HTML, JavaScript, and CSS

code. Any instance of “$user$” is replaced with the username when retrieved by a user.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/web/kitchen/toaster/?level===0&&&filename===hom
e.html&&&datatype===web&&&data===<html><title>hello</title><bo
dy><p>hello world</p></body></html>
```

A.2.3 Read

Read requests correspond to retrieval of user specific data. URI formatting differs here from other instances, where the hierarchy is as follows: /web/username/location/device, where username corresponds to the user’s username, location corresponds to device location, and device corresponds to device name. A user may simply access this resource by typing web followed by his username, whereupon an HTML splash page will direct him to available locations and devices at his current level. Devices not within the user’s current level will not be accessible by him and will return an error. Requests are not encrypted, but data returned form the server is. Options are as follows:

- **filename –** filename refers to the file to be accessed. A blank filename will retrieve home.html
- **datatype –** datatype refers to the datatype to be retrieved. This value can be data or web.

An example of a viable URI for a toaster residing in the kitchen accessed by newuser would be:

```
server_ip:8080/web/newuser/kitchen/toaster/?filename=home.html
&datatype=web
```

A.3 The *password* Resource

The “*password*” resource exists to reset the user’s password via SPEKE. Only a PUT request option exists for this resource. To reset a password, the familiar URI consisting of the resource name followed by device location, followed by device name is used. The only available option is speke_b, and that must be encrypted. The encryption of this variable simply exists to ensure the device is who it says it is, as only the device should be able to encrypt the words speke_b, followed by the SPEKE B variable, correctly. The SPEKE A variable is returned by the server. The new key is calculates as follows:

$$K = (g^{a} \bmod p)^{b} \bmod p$$

Where K is the new key, a is SPEKE A, B is SPEKE B, p is a large prime integer, and g is calculated as follows:

$$G = \text{hash}(\text{previous key})^{2} \bmod p$$

Where hash represents the addition of each of the characters within the password multiplied by 256^{i}, where i is the character location within the string.

An example of a viable URI for a toaster residing in the kitchen would be:
`server_ip:8080/password/kitchen/toaster/?speke_b=13579`

A.4 The *time* Resource

A "*time*" resource request retrieves the current system time at the server. The time resource only consists of a GET request. If the device or user wishes for this request to be returned encrypted by the device or user key, they merely require appending the location followed by the device name or the user name, respectively, in the URI, delimited by a forward slash.

An example of a viable URI for a toaster residing in the kitchen would be:
`server_ip:8080/time/kitchen/toaster`

A.5 The *storage* Resource

The "*storage*" resource allows users of resident level or above to send commands to devices, using StrongHold as temporary storage for the commands until pulled by the device. This method disallows sleep deprivation attacks on devices. The reasoning behind allowing only password enabled users (resident level and above) to use this resource is so rogue users may not flood StrongHold with requests, and thus flood the disk, denying service to normal users. Storage, from a user perspective, should only logically be accessed via a device-written webpage on the web resource, as it is too complex for a user to access manually.

A.5.1 Read

In order to pull requests from users, a device may issue a GET request from StrongHold. Commands are returned in JSON format, and deleted from the server upon retrieval. Both request and options are encrypted by the device's key. The URI formatting mirrors all URI formats from the home resource, and the options are as follows:

- **limiting** – limiting selects the number of requests to retrieve. If limiting is not set, all commands are retrieved.
- **level** – level specifies to retrieve commands only from the selected user level. Commands are returned from highest level to lowest level in order. If level is not specified, all levels are returned.

An example of a viable URI for a toaster residing in the kitchen would be:

```
server_ip:8080/storage/ kitchen/toaster/?limiting=1&level=3
```

A.5.2 Update

A user may push requests onto the server for a device by issuing a PUT request to the server. The URI is formatted like the read request for the web resource, however, options in this instance are encrypted, and differ as follows:

- **data –** data represents the commands to be stored.
- **time –** the time variable here is similar to the time variable for the PUT request in the home resource. It serves as a nonce to avoid relay attacks, and time must be within 30 seconds of server time. Furthermore, time disallows duplicate requests, so that a duplicate request with the same data sent from the same time is rejected.

An example of a viable URI for a toaster residing in the kitchen accessed by user newuser would be:

```
server_ip:8080/storage/newuser/kitchen/toaster/?data=toast+bre
ad+medium&time=1359446309
```