STARS

Electronic Theses and Dissertations, 2004-2019

2014

Improved Internet Security Protocols Using Cryptographic One-Way Hash Chains

Amerah Alabrah

University of Central Florida

Showcase of Text, Archives, Research & Scholarship

IMPROVED INTERNET SECURITY PROTOCOLS USING CRYPTOGRAPHIC ONE-WAY HASH CHAINS

by

AMERAH ABDULRAHMAN ALABRAH
B.S. King Saud University, 2006
M.S. Colorado State University, 2008

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2014

Major Professor: Mostafa Bassiouni

ABSTRACT

In this dissertation, new approaches that utilize the one-way cryptographic hash functions in designing improved network security protocols are investigated. The proposed approaches are designed to be scalable and easy to implement in modern technology.

The first contribution explores session cookies with emphasis on the threat of session hijacking attacks resulting from session cookie theft or sniffing. In the proposed scheme, these cookies are replaced by easily computed authentication credentials using Lamport's well-known one-time passwords. The basic idea in this scheme revolves around utilizing sparse caching units, where authentication credentials pertaining to cookies are stored and fetched once needed, thereby, mitigating computational overhead generally associated with one-way hash constructions.

The second and third proposed schemes rely on dividing the one-way hash construction into a hierarchical two-tier construction. Each tier component is responsible for some aspect of authentication generated by using two different hash functions. By utilizing different cryptographic hash functions arranged in two tiers, the hierarchical two-tier protocol (our second contribution) gives significant performance improvement over previously proposed solutions for securing Internet cookies. Through indexing authentication credentials by their position within the hash chain in a multi-dimensional chain, the third contribution achieves improved performance.

In the fourth proposed scheme, an attempt is made to apply the one-way hash construction to achieve user and broadcast authentication in wireless sensor networks. Due to

known energy and memory constraints, the one-way hash scheme is modified to mitigate computational overhead so it can be easily applied in this particular setting.

The fifth scheme tries to reap the benefits of the sparse cache-supported scheme and the hierarchical scheme. The resulting hybrid approach achieves efficient performance at the lowest cost of caching possible.

In the sixth proposal, an authentication scheme tailored for the multi-server single sign-on (SSO) environment is presented. The scheme utilizes the one-way hash construction in a Merkle Hash Tree and a hash calendar to avoid impersonation and session hijacking attacks. The scheme also explores the optimal configuration of the one-way hash chain in this particular environment.

All the proposed protocols are validated by extensive experimental analyses. These analyses are obtained by running simulations depicting the many scenarios envisioned. Additionally, these simulations are supported by relevant analytical models derived by mathematical formulas taking into consideration the environment under investigation.

To My Father

ACKNOWLEDGMENT

This dissertation would not have been possible without the guidance of my committee, the help and support from friends and the love and encouragement of my family and loved ones.

First and foremost I am truly indebted to my advisor Dr. Mostafa Bassiouni who, from the very first day he met me, has shown me how a true scholar should really be. He never spared me his guidance, his support and his encouragement. He has been and will always be my true mentor. To him, I am grateful.

I am also thankful to my committee members, Dr. Sheau-Dong Lang, Dr. Cliff C. Zou and Dr. Yuanli Bai for their invaluable comments and suggestions throughout my dissertation writing process.

I also would like to thank my friends and colleagues in the Department of Computer Science and Engineering who provided me with a nurturing and fruitful research environment. I am also truly grateful to have known my best friend Sana Tariq during my years in the department. We have been together through the best and worst of times, and she has shown me how a true friend really should be.

I would like to thank my family for providing me with the support I needed. My mother has been keeping me up thorough this experience by her daily phone calls to cheer me up. My brothers and sister have provided me with their love, encouragement and sense of humor. I would also like to thank my soul mate and my friend, Husam Alawadh, my husband, who have never quit on believing on me and supported me take this route from the day we met.

One final word of gratitude goes to someone who has always waited for this moment, a person whom I always dreamt of writing these words to, a person who saw his dream in me, but

left before I made it in reality; my father. I am sorry I couldn't make this earlier, but I know that

you hear me. Thank you is not enough!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Our world today is signified by its unprecedented reliance on the Internet in all facets of life. From online banking and shopping, e-governance, medical recording, military surveillance to manufacturing and production, we are witnessing a cyber revolution that never once had been anticipated. Admittedly, this realization does not come without its caveats. One of the major hurdles that is potentially slowing down this trend and threatening its ascending wave is represented in the security threats associated with the inherent nature of the Internet. As the Internet advances, so do the cyber threats, and we see numerous occasions, that take place on a daily basis, where billions of dollars are lost due to the weakness of Internet security, or lack thereof.

As an example of the counter-measures deployed to increase cybersecurity, one-way hash functions have been widely utilized in numerous applications to secure wired and wireless communication. One-way hash functions, which are based on cryptographic hash functions, are a practical example of how forth we have come in our battle against cybersecurity threats and the advancement in the field of cryptography. Modern cryptography has enabled much of our Internet today and made electronic commerce thrive by helping companies and individual alike to maintain their privacy and the integrity of their data, when their information is shared from virtually everywhere distributed environment.

Given the increasing amount of information spread over the Internet, the potential of information abuse by unlawful entities grows exponentially, thereby, emphasizing the need for powerful counter-measures to combat such abuse. Cryptography is by far the backbone of such

counter-measures, and its development will continue to evolve as long as threats prevail. Accordingly, cryptographic hash functions have evolved with this sole purpose in mind.

This dissertation investigates the wide utilization of one-way hash functions and presents schemes that are hoped to increase their efficiency. The contributions introduced herein utilize one-way hash functions in a variety of ways, but they all share a quest for achieving better performance with the least amount of resources consumed.

1.1   One-way Hashing

One-way hashing is a common practice within cryptographic approaches used today. Hashing is a widely utilized technique in cryptography in which typical algorithms are deployed to encode information quickly. Typically, a "hash value" is achieved when these algorithms are applied to strings of text creating a "digital fingerprint" of the message. The digital fingerprint is expected to be identifiable by the respective entities who have access to the hash function used to create the hash value. One of the main benefits of hashing is that it enables us to know if the transmitted message/data has been altered from its original form or not. Once received, the hashed value is checked by the receiving entity using the agreed upon hash function primitives, and if the message conforms to the expected results it is accepted. Otherwise, the message might have been altered from its original form, and, thus, should be questionable.

The use of hashing in cryptography is attractive for a variety of reasons. One important aspect of hashing revolves around the fact that the hashing algorithm generates a fixed length string to any piece of data to which the hashing algorithm is applied. Another feature that is worthy of noting here is that hashing is not analogous to encrypting. The two distinctive features that make hashing different from encryption have to do with one-wayness and the resulting message. Whereas encryption is a two-way process where data is transformed to a cipher-text

that is decoded at the destination site, hashing is a one-way operation that converts data into a compressed message digest that does not get decoded, but, rather the message digest is checked for integrity. Message integrity is the other distinctive feature making hashing different from encryption. One of the main goals of hashing is to check the integrity of the message between the communicating nodes. With hashing, we want to make sure the message is not altered from its original form during any stage of communication.

Examples of applications that utilize hashing as a backbone solution for their designs are vast but just to name a few, here is a list of applications:

1. Digital Signatures: were one of earliest applications of hash functions. Most digital algorithm schemes verify the authenticity of the message to be signed by utilizing hash functions. In 1978, Rabin in [1] introduced the idea using hash functions in the digital signature schemes. The idea is basically is to sign a hashed value of a piece of data M instead of signing this piece of data directly. The benefit from this technique is two-fold. First, instead of directly signing a large set of data with a public key incurring extraneous computational overhead, hashing the data to produce a smaller sized message is much more efficient. The second reason is as we mentioned above is that hashing ensures the integrity of the message.

2. Password hashing: Previously, storing passwords in plain text was a common procedure utilized by many service providers. Getting a hold of the passwords file stored in clear text can lead to compromising the privacy of all clients whose passwords are stored in the password file. Here, hashing techniques have been proven useful to overcome this problem. Instead of storing the password in plaintext in the service provider's database, a hashed value of the password is stored. Each time the client tries to access a service

3

using their password, the password is hashed, and the hash digest is checked against the stored hash value. Unless they do not match, the client is granted access.

3. Identity based encryption: hashing has also been beneficial for identity based encryption. Like digital signatures, users' IDs can be large and incur high computational overhead to be encrypted, and also require to be handled privately. The use of hashing technique can help provide privacy as well as achieve efficient encryption.

4. Hash trees: in this approach, the one-way hash functions are used in a binary tree to minimize the cost of authentication. The one-way hash functions are applied to data so that their sources can be verified via the use of the root hash and authentication path information.

5. Hash calendar: a special type of hash tree in which one-way hash functions are applied to conceal time since a special point in the past.

6. Message authentication code: this approach commonly known as MAC is used to protect a message's authenticity and integrity. The hashing technique is central to this approach. A message authentication code is a keyed hash function, where input to the hash function is a message as well as a key. Both inputs are subjected to the hash function and the receiver can verify, given that he has the key, both the authenticity of the messages and its integrity.

Given these great properties of hash functions, their application is endless and their versatility makes them a top choice for many developers for encryption and authentication purposes. Areas where hashing is utilized are vast, but just to name a few we take for instance

(e.g. wireless sensor networks [2], [3], [4], [5], smart card based authentication schemes [6], [7] and banking authentication schemes [8], [9]). This dissertation presents an investigation of a few applications of the one-way hashing techniques and proposes several solutions to try to optimize their configurations to achieve better performance. By introducing new and innovative solutions to overcome some of the inherent problems in typical hashing procedures, the main objective is to make hashing much more suitable for applications with limited recourses.

1.2   Problem Statement

Many web sites encrypt the user's password and perform robust authentication using SSL based encryption during the initial login only, but do not apply the same level of costly HTTPS protection in further transactions. Web servers rely on session cookies saved locally at the client side to perform authentication after the initial login. Among the information stored in these cookies is a shared hashed secret, which is used as a proof that the user has been successfully authenticated at the initial login. As these cookies are transmitted over a wireless link using the unsecure HTTP protocol, the communication between the user and the web server is vulnerable to session cookie hijacking. An attacker could take over a user's account by sniffing out the transmitted HTTP cookies. By hijacking session cookies, it becomes possible for the attacker to impersonate the victim and interact with the web site without proper authorization. Admittedly, despite the advancement of security measures to combat such attacks, today's Internet is much more prone to security breaches especially with the wide utilization of handheld and mobile devices.

A similar threat is seen in the context of wireless sensor networks where tiny sensor networks cooperate to gain information of interest to the network administration. Two of the major shortcomings of today's WSN are the scarcity of energy and memory resources. These

shortcomings make it difficult to devise security measures that suit them while at the same time provide robust and effective protection. To protect communication in WSNs, solutions based on public-key have been proposed. However, the aforementioned shortcomings make such solutions impractical. Alternatively, solutions based on the keyed-hash chains have been proposed.

One way to secure the connection in both scenarios is utilize hashing techniques. In the first scenario, the session cookies can be, as suggested in [10], substituted by easily computed one-way credentials to achieve better handling of cookies and mitigating the possibility of them being sniffed out resulting in a potential hijacking of a user's session. In the WSN context, the authentication of wireless sensor networks and users is a two way process. To secure this two way process between a user(s) and WSN sensor nodes, keyed-hash chains can be used to generate verifiable authentication tokens in both sides.

The major difficulty with the one-way hash technique revolves around the fact that the efficient use of the chain requires an accurate estimation of the number of transactions expected during the lifetime of the session. If the number of transactions is overestimated, the authentication in the early stages will suffer from an unjustified large computational overhead. If the number of transactions is underestimated, there will be the undesirable synchronization overhead of establishing a new secret and a new number for the remaining transactions. The shortcoming is detrimental for any context, but is more pronounced and risky if the devices handling communication are of limited resources.

Like WSN's, the context of Radio Frequency Identification (RFID) has benefitted from the one-way hash functions advantages. Namely, several researchers have proposed employing one-way hash functions in authentication schemes. However, due to scalability issues, using one-way hash functions in their standard format can be challenging. Therefore, it can be helpful to

consider Merkle Hash Trees to reduce the computational overhead associated with straight forward implementation of one-way hash functions.

A different yet important application of one-way hash functions for authentication is their application in multi-server environments. With the wide spread of the Internet, users usually have several sets of passwords and identities to access different services online. In fact, according to a recent survey about users' password habits among American customers, 61% of people use a single password to access multiple websites and 54% of users have 5 passwords or less [11]. This reality increases the chances for passwords being forgotten and/or results in users choosing easy to remember uncomplicated passwords. For that reason, many applications are now providing opportunities for users to use a single identity to access a variety of services (i.e. Single Sign-On SSO). One-way hash functions are typically involved in the authentication of SSO. The challenge here is different. Many of the SSO environments suffer from weaknesses pertaining to the communication channel between communication nodes. More specifically, the fact that this channel is uni-directional, thereby lacking mutual authentication, necessitate looking for more novel technique to convert this channel into a bi-directional one to guarantee stronger authentication.

1.3   Contributions

In this dissertation, we investigate the possibility of optimizing the one-way hash constructions to achieve better handling of session cookies as well as user and broadcast authentication in WSNs. In addition, we investigate the utilization of hash calendars and Merkle hash trees to improve the authentication of Single Sign-On environments and radio frequency identification schemes in toll/traffic management. Therefore, seven solutions in which one-way hash functions are the core of authentication are presented. Although employed differently in

each of these solutions to fit each individual environment, one-way hash functions have proven useful.

In the first contribution [12], we propose a solution geared towards session hijacking in the general context and mobile devices in particular. We evaluate the use of sparse caching techniques to reduce the overhead of one-way hash-chain authentication. Sparse cache schemes with uniform spacing, non-uniform spacing and geometric spacing are designed and analyzed. A Weighted Overhead formula is used to obtain insight into the suitable cache size for different classes of mobile devices. Additionally, the scheme is evaluated from an energy consumption perspective. We show that sparse caching can also be effective in the case of uncertainty in the number of transactions per user session. Our extensive performance tests have shown significant improvement achieved by the sparse caching schemes.

In the second contribution [13], we try to lower the computational overhead of OHC by dividing the one-way hash chain to multiple smaller hash chains. In doing so, we present a two-tier one-way hash chain (TTOHC) protocol to secure cookie-based Internet transactions. By utilizing different cryptographic hash functions arranged in two tiers, our hierarchical TTOHC protocol gives significant performance improvement over previously proposed solutions for securing Internet cookies. This solution also succeeds in lowering the computational overhead significantly, if compared to the straight forward implementation of the one-way hash chain protocol.

Another area of interest to this dissertation is the collaborative computing environment and social networking. Collaborative computing is a relatively new computing paradigm whose success is contingent upon the success of the means used to facilitate it. Similar session hijacking threats do actually exist for collaborative networking, and, therefore, in the third contribution

[14] we design a protocol which mitigates the risks by adopting a position-indexed hashing scheme. Similar to the TTOHC protocol, this protocol utilizes multi-dimensional one-way hashing, but benefits from the idea of position-indexed hashing. The idea basically uses the position of the hashing operation within the chain as an additional argument in the hashing function input adding rigor and complexity to the hashed value. We also run evaluative experiments to verify the benefits of the proposed protocol, and the results indicate the superiority of the proposed technique over the straight-forwardly configured hashing protocol. An analytical model to compute the optimal lengths of the chains is presented.

In the fourth contribution [15], we explore the possibility of applying the one-way hash function protocol to the user and broadcast authentication of wireless sensor networks. In order to do so, and to make the one-way hash protocol more attractive for this context, where energy and memory of wireless sensor networks are scarce as they are, we design a mini one-way hash chain protocol that is computationally efficient, thereby preserving the memory and energy of WSN nodes.

The fifth contribution [16] examines the potential benefits of a cache supported hybrid two-dimensional one-way hash construction to handle social networks' user sessions authentication in collaborative applications. The proposed scheme is based on utilizing two-dimensional OHC chains equipped with sparse caching capabilities to carry out authentication during users' social networks sessions.

The sixth contribution focuses on the context of mutli-server environments. In particular, a keyless signature scheme to handle single sign-on authentication is proposed. The proposed solution attempts to mitigate impersonation attacks commonly experienced in such environment. Merkle hash trees and hash calendars are employed to generate session keys without completely

9

relying on public/private keys to achieve authentication. In doing so, the proposed scheme provides bi-directional authentication.

The final contribution [17] targets the radio frequency identification systems utilized in toll/traffic management environments. To address authentication in this setting, a scheme based Merkle hash trees is proposed. The main benefit of MHTs is its scalability compared to traditional schemes. Specifically, we try to improve traditional MHTs by a simple division mechanism where MHTs are split into smaller ones with the premise of improving efficiency.

1.4   Dissertation Organization

The dissertation is organized as follows: in Chapter Two, we survey and discuss the relevant literature. In Chapter Three, we introduce our first contribution where we utilize the concept of sparse caching to enhance the security of cookie-based sessions in mobile devices. We demonstrate the main benefits of the proposed solution by presenting the results of our evaluation experiments. In Chapter Four, we present the second contribution, the two-tier one-way hash chain protocol, along with the experiments conducted to measure its benefits. In Chapter Five, we investigate the idea of position-indexed hashing and its potential improvement of one-way hash chains constructions. The investigation is supported by an analytical model and evaluative experiments. In Chapter Six, we propose ideas to reduce the computational overhead of one-way hash chains to appropriate it in the context of wireless sensor networks. Chapter Seven presents the next contribution where a cache supported hybrid two-dimensional one-way hash construction scheme is employed to authenticate users' session in collaborative and social networks environments. The scheme is evaluated mathematically to determine the cost of authentication and develop a quartic equation to check the optimal configuration of the two dimensions. We also evaluate the hybrid scheme with simulation experiments of different

configurations and scenarios. In Chapter Eight, we discuss the context of multi-server networks where single sign-on solutions are employed to provide a more convenient Internet experience. A combination of hash calendars and Merkle hash trees is devised to produce a keyless signature to achieve an updated and bi-directional authentication in this setting. Like the previous chapters, Chapter eight presents the evaluation and simulations experiments results. Chapter Nine introduces the tree-based authentication scheme for a cloud based toll/traffic RFID systems and the relevant evaluation experiments. Finally, Chapter Ten concludes the dissertation and draws directions for future research in the area of one-way hash chain authentication.

CHAPTER 2: REVIEW OF THE LITERATURE

In this chapter, a review of the relevant literature is surveyed to try to bring into prominence the direction to which research in this area is heading, and situate the current research within the larger paradigm of cryptography and authentication. First, we present the literature pertaining to the session cookies illustrating how they evolved. We then introduce one-hashing techniques and their application. The main objective of this chapter is to highlight the great potential of the one-way hashing schemes which made them applicable in many scenarios and contexts, and to review the work in the literature related to each of our proposed schemes.

2.1   Introduction

The past few years have witnessed an exponential growth in the use of smartphones and tablets to gain wireless access to the Internet. This growth has been accompanied by a similar growth in cyber attacks over wireless links to steal session cookies and hijack private users' accounts. In a 2010 Open Web Application Security Project (OWASP) [18], broken authentication and session management attacks were identified among the top ten list along with injection, cross-site scripting and cross-site forgery attacks. In their 2013 Release, OWASP still lists these attacks, which lead to session hijacking, among the top 10 security threats of the Internet [19]. Additionally, the broken authentication and session management attacks have moved up in prevalence to be second in the list. Many tools have been shown to aid in such attacks for empirical purposes. For example, the Android application FaceNiff [20] makes it easy to hijack other people's sessions using an Android smartphone with root access. The application is claimed to work over any private Wi-Fi network using any of the common protocols, including WEP, WPA-PSK, WPA2-PSK, or no security at all. FaceNiff can be used to intercept web

session profiles and easily hijack sessions for Facebook, Twitter, YouTube, Amazon, MySpace, Nasza-Klasa, blogger, etc. The only exception that disables FaceNiff is when the session is protected by EAP or SSL. In a similar vein, Firesheep [21], an extension for Firefox, clearly revealed that many sites fail to protect users against session hijacking attacks. To alert users of these vulnerabilities, a Firefox plug-in extension has been developed in [22] which notifies users if the server they are visiting is susceptible to cookie hijacking. This extension only gives warning for users to avoid risky web sessions but does not provide protection.

Wireless networks have more types of security threats and are much more prone to malicious attacks than wired networks. Serious security vulnerabilities exist in all types of wireless networks including 802.11 wireless LANs, wireless ad hoc networks, multihop wireless mesh networks, and wireless sensor networks [23], [24], [25], [26]. Of particular interest to the current research is the security threat of hijacking user' private sessions over wireless links. This threat is increasingly on the rise due to two reasons: i) the worldwide proliferation of smartphones, tablets and other handheld mobile devices as a primary tool for Internet access and ii) the increasing use of HTTP cookies by web applications to speed up responses to users and offer a better web experience that is more personalized and richer in interactivity. Since cookies are stored on the client's machine, they are one of the most popular means available to web developers to create better web applications without requiring significant server resources. Session cookies are being increasingly used for many purposes including session and transaction authentication, tracking of shopping cart contents, identification of user's preferences, and tracking browsing behavior. Cookies usually exist as plain text on the client machine and can be tampered with by attackers if they succeed to compromise this machine.

13

The best way to overcome the issue of session hijacking is to use HTTPS (i.e., HTTP over SSL or TLS) throughout the lifetime of the session. This guarantees a fully secure session by encrypting users' login information and securing all transactions between login and logout. However, due to the costly nature of this approach, many developers use HTTPS only in the login and initial authentication then revert to the approach of using cookies stored locally at the client's side to carry out authentication throughout the remainder of the session. Session cookies store information about the identity of the client and a shared secret which indicates that the user has been authenticated in the initial login. Since these cookies are transmitted over unsecure links using the unsecure HTTP protocol, they are susceptible to malicious attacks which can sniff out session cookies and, consequently, take over a client's identity and allow the attacker to gain the same level of access over the web application that the original user has.

The use of cookies as a cheaper alternative to the wide utilization of the secure HTTPS protocol has been also attractive to many collaborative websites and social media networks. One cannot deny the fact that the unprotected nature of cookies can compromise the collaborative environment. Evidently, the availability of social networks and collaboration websites where access to the website is extended to long durations has made this issue even more pressing. Although using a secure protocol (e.g. HTTPS) to connect to the web provides higher levels of security, it is not always applied by many web servers and is replaced by cookie protection. The nature of cookies as plain text stored at the client's side makes it not too complicated for an adversary to hack these cookies and steal the Internet session leading to a compromise in the users' overall Internet experience.

2.2 Securing Session Cookies

As mentioned above the most secure way to handle an internet session is to carry out this session using either The Secure Sockets Layer (SSL) protocol [27] and its successor, the Transport Layer Security Protocol (TLS) [28], or the Secure-Hypertext Transfer Protocol (HTTPS) [29]. The encryption in the SSL/TLS protocol is achieved via utilizing asymmetric cryptography allowing applications between client-server to communicate without worrying about eavesdropping and tampering. The communication via the SSL/TLS protocol is carried out using an encrypted, secure channel that is established by using authentication certificates. Note that these authentication certificates can be mutually exchanged, thereby enabling mutual authentication between the server and clients for maximum protection. The HTTPS, on the other hand, is not considered a protocol by itself, but it achieves its security by working on top of SSL/TLS. However, since the SSL/TLS is the backbone encryption mechanism used in HTTPS, the same can be said about the HTTPS.

The security of the SSL/TLS protocols has resulted in their wide utilization in many applications as they are well-suited for finance, e-business, healthcare etc. However, these advantages do not come without their prices. One of the major drawbacks of the secure SSL/TLS and HTTPS protocols preventing their wide utilization in Internet is their demands for computational resources. Given the fact that they rely on authentication certificates to achieve their encryption and decryption, the computation required to transmit the certificate is an additional burden both on the client and the server sides. Furthermore, SSL incurs additional costs which many companies prefer to spare for other expenses. According to [30], web servers that employ SSL incur performance penalty that ranges from a factor 3.4 to 9 compared to web servers not utilizing SSL. For these reasons many websites prefer to use SSL/TLS based

communication for the initial login stages, and then resort to the less expensive, both computationally and economically, stateless internet protocol HTTP. In order to keep state and avoid the need for the user to identify himself to websites every time he logs in to a web site, we revert to cookies as carriers of users' identity information. Imagine having to introduce yourself every time you meet someone. While this is not pertinent in real life, cookies are made with this imagination in mind. Unlike humans, this reality can pose serious threats to the Internet experience as these identifiers can be accessed and unlawfully exploited, thus, leading to attacks such as session hijacking.

2.3   What Are Cookies?

Cookies were originally invented to help facilitate the online e-commerce industry. The authors of [31] identify cookies as metadata that pertains to a pair of name/value information sent by an HTTP server to a user agent. Whenever the user agent tries to access URL that requires identification in the HTTP server, the cookies are used as identifiers. According to [32], cookies are the piece of information that the server and client pass back and forth. In addition, [32] indicates that the amount of information, whose content is at the discretion of the server, tends to be usually small. Although simply examining a cookie's value will not reveal what the cookie is for, or what the value represents as [32] points out, information about the user can still leak out exposing privacy and potentially infringing the security of the internet experience. This is evident in the mechanism with which the HTTP cookies operate. As [32] further explains, "a cooperating client returns the cookie information verbatim in a Cookie header, one of its request headers, each time it makes a new request to the same server". Thus, unless told by the server that the cookie is changed, the client will keep the previous cookie unchanged. Note that changing the cookie automatically supersedes the previous cookie [31]

16

Cookies can be of many types and flavors and can serve a variety of purposes. For example, there are session cookies, first party cookies, persistent cookies, secure cookies, and several more. Our main concern in this research is session cookies or more precisely, persistent cookies, and, thus, will be highlighted. As the name suggests, session cookies are mainly used to identify a user for a particular Internet session. One of the main features of sessions cookies is that they are subject to an expiration period or valid for a certain amount of time. First designed by [33] to keep state of the user in online shopping scenarios, session cookies are set up to have an end point set forth by the server. This end point, however, can be extended, as a convenient feature, to alleviate the need of users to enter their login credentials each time they log in, thereby resulting in a higher possibilities of them being sniffed out and exploited.

2.4 Current Practices

The security measures currently utilized to avoid the hijacking of session cookies are lacking both in depth and dearth. In fact, the authors of [34] point out that the use of cookies in web sessions has not evolved much since its first days indicating that they are susceptible to attacks such as eavesdropping. The fact that web sessions utilizes static tokens sent over a plaintext channel makes it possible for an eavesdropper to possibly take over the session [34]. This threat is on the rise due to the wide utilization of wireless networks. On the threat caused by wireless connection, the authors of [34] highlight that users freely connect to the wireless base stations and then are asked for login credentials of the web sites they are trying to access, allowing wireless operators to manage per-user password access-control. They further explain that wireless operators should not be doing this practice and limiting password exposure to that pertaining to the Wi-Fi service they are providing.

Several solutions have been proposed to address the problem of session cookie hijacking in wired and wireless networks. Liu et al. [35] proposed a secure cookie protocol for ensuring integrity of each transmitted cookie by applying HMAC on the concatenation (username | expiration time | data | session key). Their fundamental assumption, however, is that their secure cookie protocol would either encrypt the data using a session key or run on top of SSL. As mentioned above, the use of HTTPS throughout the internet session is not only computationally inadequate due to the large computation associated with key generation and handling, it is also very costly. Many websites have different servers that they rely on, and to provide HTTPS on their servers would incur extraneous costs that might be unnecessary.

Given that SSL is not always optimal for all applications, Adida [34] proposed a technique called SessionLock as an alternative to the wide utilization of SSL. Essentially, SessionLock works in two stages: the first sage is done using a secure connection on top of SSL and involves establishing a session key, and the second stage is where the remainder of the session is carried out and it is run on HTTP. Using HMAC as a cryptographic function, a shared session secret is generated and used to encrypt each subsequent request from the client to the server. While this solution offers potential in solving session hijacking especially, it is not robust against different types of attacks especially active attackers who can amend or even steal the session key and use it unlawfully. In HMAC based techniques, the compromise of the shared key compromises the whole session as all future values of HMAC can be easily calculated. Moreover, techniques such as URL re-writing and URL fragmentation, which are essential in the SessionLock solution, might not be suitable for different applications.

Another direction of research suggests the use of an external proxy where authentication and sensitive information management are carried out completely at the proxy or some other

external device (e.g. a user's cell phone) [36],[37]. However, this solution's implementation can pose difficulties as it might not be optimal in all situations. Specifically, if a user does not have access to the proxy for any reason or in case the external device is not available at the time the service is desired (e.g., cellphone battery is dead, no coverage...etc.), he will not be able to use the service provided by the web application.

2.5   New Direction

We have seen in the previous section that for the most part current practices to secure session cookies are either based on the wide utilization of SSL/TLS based connection, or on solutions that are not as rigorous. Alternatively, a possible candidate to address the problem of session hijacking caused by the lacking security of Internet cookies is the Lamport's well-known one-way hash chain technique for one-time passwords [38] which was later formulated by Haller to the S/Key standard [39]. Lamport's ideas were very attractive to many researchers. For example, the attack-resilient security architecture ARSA [40], proposed for multihop wireless mesh networks, uses a hierarchical one-way hash chain to authenticate beacons transmitted by mesh routers. The Ariadne secure on-demand Adhoc network routing protocol [41] uses hash chains in its Route Discovery phase and in later phases to thwart the effects of routing misbehavior. The SEAD protocol [42] proposed for Adhoc networks also uses a one-way hash chain for authenticating important routing information such as the routing metric and the sequence number. This prevents any malicious nodes from falsely advertising a better route or tampering with the critical routing information contained in the packet that it received from the source.

Several proposals have tried to address the problem of session cookies' exploitation by adopting schemes which rely on Lamport's one-way passwords. For instance, [43] proposed a

solution that targets the read-only property of the session cookies in the website's databases. They achieve their protection by leveraging the read property so that it becomes hard for an attacker to correctly guess the cookie value. Conceptually, they suggest including an iterated hashed value of the user's password and its pre-image in the session cookies. These two values are compared each time a communication between the server and client is desired. To strengthen the cookies, they add a *salt* value which they claim makes it even harder for an adversary to detect the users' private information.

We, therefore, address the potential benefits of applying the one-way hash chain as a technique to secure session cookies. Our objective is to propose schemes which not only preserve the cryptographic strength of one-way hash chains, but also use it in a computationally efficient manner. Unlike HMAC based schemes discussed above, using the one-way hash chains to secure cookies minimizes the chances of them being sniffed out and abused for unlawful utilization by entities other than the respective parties.

The one-way hash chain technique has been recently used to protect against cookie hijacking in wireless networks. The One-Time-Cookies (OTC) protocol proposed in [10] is a straightforward implementation of Lamport's hash chain technique for one-time passwords. The authors implemented OTC as a plug-in for Firefox and Firefox for mobile browsers. OTC uses a hash chain construction to generate a sequence of values that can be used as one-time authentication tokens. These tokens, once verified by the web application, cannot be reused due to the pre-image resistance property of cryptographic hash functions. The Rolling Code protocol proposed in [44] is an attempt to reduce the computational overhead of the OTC protocol for mobile devices with constrained memory. The protocol replaces the hash chain performed by OTC in each transaction by two hash operations: one to update and randomize the value of a

variable $d = hash(d)$ and the other to produce a one-time authentication token by applying a hash function on the Exclusive-OR of a secret seed and the new value of d. In essence the protocol is much like the rolling code technology used to prevent perpetrators from recording a code and replaying it to open a garage door. The Rolling code protocol is less robust than the one-way hash chain approach (e.g., the OTC protocol) but is lightweight and more suitable for mobile phones and PDA's. Although the seed is guaranteed to be fresh during each iteration due to the monotonic function used to increase the value of d, there is a high risk of discovering this value and consequently compromising the internet session.

2.6   The One-way Hash Chain Model

We first present the notations for the one-way hash chain technique that we will use in this dissertation. We will refer to this **ha**sh **ch**ain technique as the HACH technique, which can be described as follows:

HACH uses an initial secret s which is the seed of the hash chain. We apply a cryptographic hash function H() successively to obtain the following values

$$
\begin{aligned}
V_0 &= s \\
V_1 &= H(V_0) = H^1(s) \\
V_2 &= H(V_1) = H^2(s) \\
&\dots \\
V_j &= H(V_{j-1}) = H^j(s) \\
&\dots \\
V_N &= H(V_{N-1}) = H^N(s)
\end{aligned}
$$

Due to the pre-image resistance property of the hash function (e.g., SHA-1), the values V_i are distinct and can therefore be used to represent one-time authentication tokens in successive user transactions after the initial login, i.e., the value of the appropriate V token is stored in the session cookie transmitted by the client to the server in each transaction. To properly use the one-way hash chain, these values must be transmitted in reverse order. In the first transaction

21

after login, the client browser transmits the value V_N. Similarly, the client transmits V_{N-1} in the second transaction and V_{N-i+1} in the ith transaction.

During the initial HTTPS authentication, a shared secret value s and a value chain length N representing the number of transactions are exchanged between the client and the server. After the initial login, the costly HTTPS protocol is replaced by HTTP and authentication is done by sending the one-time authentication tokens in the session cookies. Upon receiving the value of an authentication token, the server computes a similar value based on the values of N and s, and accepts the transaction if the received value matches the computed value.

The variable N is the chain length and represents an upper bound on the number of transactions that can be handled by the above hash chain. One difficulty with the HACH technique is that the efficient use of the chain requires an accurate estimation of the value of N, i.e., the number of transactions expected during the lifetime of the session. If the number of transactions is overestimated, the authentication in the early steps will suffer from an unjustified large computational overhead. If the number of transactions is underestimated, there will be the undesirable synchronization overhead of establishing a new secret and a new number for the remaining transactions. For our initial presentation of the sparse caching technique, we will assume that the number of transactions in a session, N, is accurately known. Later in the dissertation, we will show how the sparse caching technique can be used to effectively deal with the case when the value of N is not accurately known.

We will use the number of hash operations executed per transaction as the metric to evaluate the execution overhead of the process of verifying this transaction using the authentication token transmitted by the client. We denote this metric for the ith transaction by $HCost_i$. Smaller values of this metric indicate faster speeds for the authentication process. The

total cost of the entire session is the sum of the costs of the N transactions and is denoted by *TotalHCost*.

In its most basic format, the 1st transaction in the HACH protocol computes $H^N(s)$ and has a cost $HCost_1 = N$. In general, the ith transaction computes $H^{N-i+1}(s)$ and has a cost $HCost_i = N-i+1$. We have

$$TotalHCost = \sum_{i=1}^{N}(N - i + 1) = \frac{N(N+1)}{2} \qquad\qquad (2.1)$$

The maximum and average value of the transaction execution overhead are given by

$HCost_{max} = N$
$HCost_{avg} = (N+1)/2$

It is important to stress that the authentication tokens are generated by successive hashing from V_0 to X_N but are exposed (i.e., presented as authentication tokens) from V_N to V_0. The reverse presentation of the values of the hash sequence is a fundamental feature that must be enforced in order to attain the good security properties of one-way hash chains. Exposing (presenting) the hash sequence values in the same forward order in which they are generated makes the scheme much less secure because if one hash value is compromised, the attacker can compute all the hash values that will be used in the future.

The HACH protocol in the above description is the backbone of our solutions. In the four proposed schemes, we try to optimize the HACH by reducing its computational overhead to suit different scenarios and contexts. The different techniques utilized try to benefit as much as possible from the advantages of the HACH protocol maintaining its cryptographic strength, while at the same time minimizing the drawback inherently associated with its construction.

2.7   User and Broadcast Authentication in Wireless Sensor Networks

Another area of interest that we investigated in this dissertation is user and broadcast authentication in wireless sensor networks. A wireless sensor network (WSN) is a collection of tiny sensor nodes that work collaboratively to gather information. Typical shortcomings of wireless sensor nodes include their limited storage and power resources. Versatile as they are, WSNs have gained acceptance and are being widely used in various arenas such as military sensing and tracking, traffic monitoring, industrial quality control, medical monitoring, etc. With the fast rate of acceptance of such networks, they are expected to grow both in capacities and capabilities.

One of the important issues slowing down this growth in deployment lies in their security vulnerability. Attacks on WSNs can be as easy as intercepting messages, falsifying data and impersonating users to gain access. Such attacks can only be avoided by exercising robust and effective security measures that prevent unlawful access of unauthorized entities. The fact that sensor nodes in WSNs are not well-endowed in terms of memory and energy makes it necessary for solutions addressing such attacks to be lightweight, efficient and memory/energy preserving.

Solutions proposed to address the problem of security issues in WSNs are still work in progress. Specifically, authenticating the data broadcast and users over a WSN is very crucial. Researchers such as [45], [46], [47] suggest using public-key based solution. However, with limited energy and memory capabilities, such solutions can pose difficulties in deployment due to high cost of calculating public keys. Alternatively, solutions based on the keyed-hash chains have been proposed [48], [49]. However, if used in the basic configuration where a shared key is hashed with no limits on the length of keyed-hash chain, two major problems arise. Similar to session cookies security issues discussed above, the shared key in the WSN can be compromised,

and, hence, data might be at risk putting the whole WSN in danger. Second, the length of the keyed-hashed chain can grow considerably leading to an unjustified overhead that cannot be handled by a sensor node with limited memory and energy. In this dissertation, we investigate these two problems and try to address them by adapting the HACH protocol to achieve broadcast and user authentication for WSNs.

2.8 Authentication in Single Sign-On Environments

Due to the proliferation of social networking, cloud computing and the Internet of things, Single Sign-On schemes are now becoming more popular. Social networking giants such Facebook, Google Twitter and several others are providing SSO services to the public. Protocols utilizing Browser-based Single Sign-On (SSO) provide convenience and security for users and service providers alike. By employing SSO protocols, users are relieved the burden of having to possess multiple sets of IDs and passwords for different web services they wish to use. Similarly, service providers are relieved the burden of a variety of login-related tasks such as clients contacting them for stolen passwords, recovering of forgotten passwords, revocation of possibly compromised passwords etc. As such, by substituting multiple identities corresponding to multiple service providers connected via a central identity provider with a single identity, SSO protocols are promising a more secure Internet experience, especially given that clients typically use easy to remember weak passwords and usually use the same passwords to access different applications.

An SSO authentication scheme rely on three main components: the users wishing to be granted access to certain services, the service provider SP (also known as the relying party) and the Identity Provider (IdP) who delegates identity assertions to the SP. Several researchers have discovered critical security concerns related to the communication between these three

components (e.g. [50], [51]). In an extensive analysis of SSO, the authors of [50] detected several serious logic flaws in commonly deployed SSO systems and RP's. Similarly, the authors of [51] have identified flaws in two of the commonly utilized SSO frameworks, namely, SAML SSO and OpenID that lead to hijacking a user's authentication attempt and accessing resources without the user's consent. Such impersonation attacks can be detrimental to the utilization of SSO schemes, as they can lead to exposing of critical information about the users thereby threatening their privacy.

SSO authentication schemes generally aim to provide the users with the convenience of singing in to one portal and gaining access to several services. Therefore, they inherent many of the password related features present in common authentication systems such as the Lamport authentication protocol [38]. After successful login, session authentication is typically achieved via either a secure channel such an SSL or in a cookie supported HTTP connection. While they have been relatively successful in securing communication in conventional computing environments, they fell short in the context of SSO due to their inability to provide robustness and rigor in authentication required for this setting.

Therefore, researchers have tried to come with solutions particularly designed to suit the SSO environment. The majority of schemes designed to suit SSO fall within two main categories: hash-based authentication schemes and public-key based authentication schemes. Among the early attempts to design a hash-based authentication architecture for SSO is reported in [52] who proposed an authentication protocol that guarantees users' anonymity by employing a combination of public key and session token infrastructure. However, this scheme was shown to have vulnerabilities due to the public nature of the session token, thereby leading to impersonation and identity disclosure attacks as shown in [53]. The authors of [53], alternatively,

26

proposed utilizing timestamping to improve the strength of the session key-based approach. Yang et al. in [54] identified a weakness in the previous protocol resulting in the possibility of the service providers getting a hold of the clients session ID, and they proposed an improvement which relies on public-key cryptographic infrastructure. However, as they point out, their scheme is prone to DoS attacks. These three schemes all rely on the use of smart cards as an additional authentication factor. Although there followed several improvements on the smart card solutions (e.g. [55], [56], [57] [58]), relying on a central agency to issue smart card credentials for the whole network can be problematic as it increases single point of failure which can be detrimental to the whole scheme.

Another group of SSO authentication protocols that have gained popularity and became widely adopted in today's SSO environments is the Security Assertion Markup Language SAML [59], [60], and the OpenID Project [61]. One of the essential reasons why the two SSO authentication mechanisms have gain popularity can be attributed to the fact that both are browser-based solutions. In other words, unlike the schemes discussed earlier which require additional resources such as a smart card issuing entity, the SAML and the OpenID simply require a standard web browser as a user agent. The SAML authentication protocol relies on sets of public and private keys for verification, whereas the OpenID depends on session cookies as a verification mechanisms. However, both schemers were found to have critical vulnerabilities pertaining to the logic with which they are designed. In particular, the fact that both are uni-directional and the possibility of using multiple SSL connections render them susceptible to attacks such as impersonation and hijacking attacks. In this dissertation, we propose an authentication protocol that is bi-directional and guarantees the recentness of authentication

assertions. Specifically, we utilize the idea of keyless signatures proposed in [62] in conjunction of Merkle Hash Trees [63] to authenticate communication in the SSO network.

2.9   Authenticating Cloud-based Toll/Traffic RFID Systems

Due to its capacity to wirelessly and remotely identify tagged objects, RFID (Radio Frequency Identification) technology has long been used in numerous applications to track assets, enable supply chain management, track commodities in retail stores, payment, security and access control and many more. Recently, vehicular and traffic management applications have adopted this technology in road tolls administration and enforcement. Many municipalities around the world depend on RFID for electronic toll tracking/ payment through devising RFID tags mounted in the cars or plates of their customers. TransCore, one of the biggest RFID providers, indicate that over 3.5 million customers travel their electronic tolling systems at highway speeds and simply pay their tolls using RFID tags or their vehicle license plates [64]. The Express Way Authority in Central Florida, for example, says that in 2013, the number of transponders used to track their electronic toll operations is estimated at around 530.000 units; a number that has grown steadily throughout the past 10 years [65]. According to a report by E-ZPass Group, the number of transponders in their road toll system has more than doubled in less than ten years from around 13 million in 2005 to more than 26 million transponders 2013 [66]. The RFID technology is moving toward being adopted in more traffic related applications such as tag registration and number plate tracking. Such applications are promising since they can solve a lot of traffic issues experienced today such as congestion, car theft, and rental fleet management; all of which can bring about better utilization of current infrastructures.

As part of its thriving evolution to become a more enabling technology, the RFID schemes are presenting architectures which benefit from cloud computing to provide more

resilient and catering solutions for companies. The context of traffic and road toll management is one of the beneficiary applications of this trend, as managing data that grows exponentially throughout the years can be very costly and can hinder the future of RFID in traffic management and application. Therefore, moving to a cloud-assisted traffic management scheme definitely reduces some of the costs associated with storing and handling vehicles' information. However, this movement is not without cost. Authenticating cloud-based RFID schemes is challenging because such schemes inherent security flaws of both the RFID technology and cloud computing.

The majority of RFID schemes presented in the literature targeting road tolls and traffic contexts have been concerned with the engineering and conceptual aspects of the issue (e.g. [67], [68]). However, little research has been conducted to investigate the issue of authentication in RFID based toll/traffic management systems. This dissertation also tries to address this issue in addition to the previous ones. In this section, we overview some of the related literature to the RFID in general.

The authors of [69] stress the importance of authentication in RFID systems. They indicate that RFID systems are susceptible to security threats due to the fact that they work non-line-of-sight in a contactless fashion, thereby, allowing attackers to remotely and passively launch attacks without being noticed. The resulting security concerns can be unwanted consumer tracking, tag forgery and unauthorized access of tags' memory content. As a result, any RFID scheme should address this issue. Strong authentication of RFID systems, according to [69], are efficient in combating such attacks. Many authentication schemes proposed thereafter attempt to achieve this property. Authentication can be achieved through cryptographic based approaches such as symmetric block ciphers and one-way hash based schemes. Examples of schemes that adopt symmetric block ciphers include [70], [71], [72]]. The main drawback of symmetric block

cipher solutions is the high computational overload associated with the calculation scheme. Alternatively, solutions based on the one-way hash chain constructions such as [73], [74], [75] have been proposed. However, with a network of RFID readers and RFID tags as big as the traffic/ road toll management that can go up to several hundreds of thousands of tags and readers, these solutions will suffer from a scalability problem. Both symmetric block cipher schemes and one-way hash chains solutions suffer from this problem, as a linear search is needed in the database each time the reader tries to identify a tag. This problem is more prominent in the above described scenario where many readers and tags alike can require access to the database at the same time; which renders such protocols unsuitable for traffic/ road toll applications.

Schemes based on binary trees have shown benefit in improving the scalability by reducing the search complexity from $O(N)$ in the linear search schemes to $O(logN)$, where N represents the number of RFID tags in the network.

The traffic/ road toll scenario presents challenging dimensions in the employment of RFID schemes. First, the majority of RFID readers in the system are stationary making physical capture of them far-fetched. However, readers in this context are typically given one chance to read the RFID tags since RFID-equipped vehicles typically pass through the RFID readers range only once. Thus, the reading window is limited and readers cannot afford a complex search, such as the one achieved by a linear search. As stated above, a tree-based solution achieves acceptable and relatively low complexity searches.

In this dissertation, we introduce a tree-based solution that reinforces the RFID authentication process by predefining the tree upper bound, thereby, spreading the data of the RFID system into multiple trees. Our aim from doing so is to reduce the computational overhead while at the same time improving scalability.

CHAPTER 3: HASH CHAIN (HACH) WITH SPARSE CACHING TECHNIQUES

3.1   Introduction

In this chapter, we present and evaluate the use of sparse caching techniques to reduce the overhead of one-way hash-chain authentication. Sparse cache schemes with uniform spacing, non-uniform spacing and geometric spacing are designed and analyzed. A Weighted Overhead formula is used to obtain insight into the suitable cache size for different classes of mobile devices. Additionally, the scheme is evaluated from an energy consumption perspective. We show that sparse caching can also be effective in the case of uncertainty in the number of transactions per user session.

3.2   Sparse Caching: Basic Idea

The main drawback of the HACH protocol, as elaborated in Chapter Two, is the need to perform the hashing operation N times for the first transaction, N-1 times for the second transaction, and so on. For large values of N, the Hash Chain (HACH) technique is costly and is not generally suitable for most mobile devices that have scarce memory resources. In this chapter, we propose using a sparse caching approach to attain the full level of security of one-way hash chains but at a much reduced computational overhead. An earlier proposal for sparse caching was given by Gupta et al. [76] as a way to reduce directory memory requirements. The hallmark of this approach is that a memory block is allocated for each active entry and invalidated data is discarded as they are no longer needed. Because of this feature, sparse caching techniques have been appealing in a variety of environments and applications. The authors of [77], for instance, proposed a method to improve users' perceived performance in wireless networks using sparse infrastructure. Also, the authors of [78], [79] suggested using

sparse caching in multimedia applications. As we describe our proposed solution, we also introduce several cache spacing configurations that are deployable in different scenarios.

$cache[0] = s$
$cache[1] = H^{20}(s),$
$cache[2] = H^{40}(s),$
$cache[3] = H^{60}(s),$
$cache[4] = H^{80}(s).$

The above values can be computed using a simple loop that computes the hash function 80 times. The availability of the above cache values reduces the execution overhead of transactions considerably. For example, the first transaction can now start by fetching the value of *cache[4]* then performing 20 more hashes to obtain $H^{20}(H^{80}(s)) = H^{100}(s)$. The value $HCost_1$ for the first transaction has been reduced from 100 to 20. Basically the sparse caching scheme has divided the initial hash chain into five mini-chains, each with length 20.

The above example uses sparse caching with uniform (equal) spacing. We will examine non-uniform spacing later, but we will first formally define the uniform spacing model and examine its characteristics.

3.2.1   Sparse Caching with Uniform Spacing

This scheme is defined by two parameters, the size of the cache, *cache_size,* and the uniform space interval, *minichain_len*. In the above example, *cache_size* = 5 and *minichain_len*= 20.

The pseudo code to compute the authentication token of the ith transaction is presented below. Notice that in the ith transaction, the client transmits the value $V_{N-i+1} = H^{N-i+1}(s).$

Authentiaction_Token(i);

$k = (N-i) / minichain\_len$ // integer division

$m = N - (k \times minichain\_len)$

$V_{N-i+1} = H^m(cache[k])$

Using the uniform sparse caching scheme, the total cost of a session including the cost of the initial cache setup is given by

$$TotalHCost = C \times \frac{M(M+1)}{2} = \frac{N(M+1)}{2}$$ (3.1)

Where $C = cache\_size$, $M = minichain\_len$, and $N=C \times M$.

The above formula is derived under the simplifying assumption that N is divisible by C. The cost *TotalHCost* is the sum of two components: a cost of $(C-1) \times M$ for the initial filling of the cache values and a cost of $(C-1) \times M \times (M+1)/2$ for the N transactions. Notice that $(C-1)$ of the N transactions will not need to perform any hashing since the required value is already in the cache. For the above example of $N = 100$ and $C = 5$, transaction # 21 will simply read V_{80} from *cache[4]*. The maximum and average value of the transaction execution overhead are given by

$HCost_{max} = M$ // for the 1st transaction

$$HCost_{avg} = \frac{0.5 \times (C-1) \times (M-1)}{C}$$ (3.2)

In section 3.4, we evaluate the tradeoffs of the sparse caching scheme and examine policies for selecting the cache size C for different values of N. It is important to notice that the sparse caching scheme is applied only to the user mobile device, not to the server. The server can compute the same authentication tokens using any caching scheme; for example it may use a complete caching scheme that stores all authentication tokens in the cache during initialization.

3.2.2 Weighted Overhead Formula

For a session with *N* transactions and a user with a certain mobile device, what would be the best *cache_size C*? In answering this question, we first recognize that wireless users can use mobile devices having a wide range of capabilities. Some users may use high-end laptops that have plenty of storage resources while others may use mobile phones with limited memory. We use the following Weighted Overhead (*WO*) formula to obtain insight into the suitable value of the *cache_size C*.

$$WO = W \times CacheSize + TotalHCost \qquad\qquad (3.3)$$

Where *w* is the weight assigned to the cost of using memory in the mobile device. The Weighted Overhead formula is simply a pragmatic approach to select the size of the cache for the different categories of mobile devices. The Weighted Overhead *WO* obtained from the formula can be viewed as the combined cost of memory and execution overhead where *w* is the cost of memory relative to a unit cost of execution. The value of *w* may be assigned based on classes. For example, we may have three classes of mobile devices with the following weights.

Class 1: w_1 is used for high-end laptops with plenty of memory.

Class 2: w_2 is used for mobile devices, e.g., high end smartphones and tablets, with reasonable but constrained memory resources.

Class 3: w_3 is used for mobile devices, e.g. low-end mobile phones, with very limited memory resources.

We have the obvious relationship $w_3 > w_2 > w_1$. To choose the best cache size for a mobile device, we simply minimize the value of the Weighted Overhead. For Class 1, the value of w_1 is nearly zero and the minimization problem reduces to minimizing the second term

TotalHCost. Minimizing *TotalHCost* is achieved by choosing the largest possible value of cache-size, which is simply *N*. This means that for high-end laptops, the sparse caching scheme is replaced by complete caching in which the authentication token value V_j is obtained by fetching *cache[j]* without performing any hash computation. For Class 2 and 3, plotting the value of Weighted Overhead versus *cache_size* could reveal the best size that minimizes the combined cost.

In section 3.4.4, we present performance results that show how the Weighted Overhead formula can be used to gain insight into selecting the size of cache.

3.2.3 Uncertainty in the Number of Transactions

In all previous discussions, we assumed that the number of transactions in a session, *N*, is accurately known. In real-life applications, however, the value of *N* is usually only approximately known. If the application developer chooses a value for *N* that is too small, the hash chain will be exhausted before finishing all of the transactions of the session. If the selected value of *N* is too large, the transactions will have large execution overhead. It is difficult for an application developer to consistently strike a good balance of *N* for the different sessions of the different users. The sparse caching approach presented earlier comes into play to elegantly solve this problem. Below, we elaborate on this issue.

Suppose that we know that a session will have approximately 1000 transactions but there is some likelihood that it could have up to 2000 transactions. Without any caching, the developer will be tempted to choose a value of *N* between 1000 and 2000 to strike a balance between reducing *HCost* for the individual transactions and avoiding the scenario of exhausting the chain and resorting to costly HTTPS authentication and additional hash chain setup. By placing a single cached value *cache[1]* = $H^{1000}(s)$ at the middle of the range, the developer can safely

select $N = 2000$ with a guarantee that $HCost_{max}$ will be 1000 and the costly HTTPS re-initialization will not be needed. By placing a second cached value at 2000, the developer can extend the value of N to 3000 with the same guarantee that $HCost_{max}$ will be 1000 and the costly HTTPS initialization will not be needed. The sparse caching scheme can be used with few cached units to extend the range of N to a large safe value without incurring an increase in the execution overhead of individual transactions or running the risk of additional HTTPS setup.

In section 3.4, we present test results that illustrate the application of sparse caching for the case of approximate values of N.

3.2.4 Sparse Caching with Non-uniform Spacing

The case of uncertain values of N motivates the use of sparse caching with non-uniform distribution. We elaborate on this by an example.

Suppose it is highly probable that the number of transactions in a user session will be 100 or less but there is some small probability that this number could go up to 500. As shown earlier, we could use the sparse caching scheme to set the value of N to 500 without increasing the value of $HCost_{max}$ above 100. We can actually do better than this by choosing non-uniform cache spacing to significantly improve the execution overhead of the first likely 100 transactions. For example, we can distribute 9 cache values non-uniformly as follows:

$cache[0] = s$
$cache[1] = H^{100}(s)$
$cache[2] = H^{200}(s)$
$cache[3] = H^{300}(s)$
$cache[4] = H^{400}(s)$
$cache[5] = H^{420}(s)$
$cache[6] = H^{440}(s)$
$cache[7] = H^{460}(s)$

$cache[8] = H^{480}(s)$

The above scheme gives priority to the first 100 transactions with a guaranteed value of $HCost_{max} = 20$. The other less likely 400 transactions will be guaranteed a value of $HCost_{max}=100$.

Schemes for the forward generation and reverse presentation of one-way hash chains can use different topologies including tree topologies. But the simplicity of the proposed sparse caching authentication scheme and its flexibility in dealing with different scenarios (such as the non-uniform spacing discussed in this section or the geometric spacing discussed in the next section) make the scheme more practically appealing than other schemes for implementing one-way hash chains [80].

In section 3.5, we present the results of our tests to evaluate non-uniform spacing for sparse caching.

3.2.5   Caching with Geometric Spacing

In the previous section, the non-uniform spacing of cached values was achieved by creating two groups: the high priority group (first 100 transactions in the above example) and the low priority group (the remaining less likely 400 transactions). Within each group, the cached values are distributed uniformly. This scheme is suitable when the value of N is not accurately known but there is knowledge about the minimum value of N, i.e., the number of transactions that are most likely or are guaranteed to occur. If this knowledge is not available (i.e., the value of N could range from a small number to a large number), it would be better to distribute the cached values at progressively increasing intervals. One possible progressive strategy is the geometric distribution scheme. We illustrate this scheme using the previous example in which

the value of *N* could be as small as 1 but could go up to 500. We use 9 cached values geometrically distributed as follows:

cache[0] = s
cache[1] = $H^{246}(s)$
cache[2] = $H^{374}(s)$
cache[3] = $H^{438}(s)$
cache[4] = $H^{470}(s)$
cache[5] = $H^{486}(s)$
cache[6] = $H^{494}(s)$
cache[7] = $H^{498}(s)$
cache[8] = $H^{500}(s)$

The first transaction uses *cache[8]= $H^{500}(s)$* and will not need to perform any hash operations. The second and third transactions use *cache[7]= $H^{498}(s)$*, resulting in performing one hash operation for the second transaction and no operation for the third transaction. The transactions numbered 4, 5, 6, and 7 use *cache[6]= $H^{494}(s)$* resulting in performing 3, 2, 1, and 0 hash operations, respectively for these four transactions. Notice that the cached values are anchored at points that are geometrically spaced apart. The difference between the number of hash operations performed for *cache[8]* and *cache[7]* is 2, between *cache[7]* and *cache[6]* is 4, between *cache[6]* and *cache[5]* is 8, between *cache[5]* and *cache[4]* is 16, and so on. As the number of transactions in the real session increases, the range of the number of hash operations will increase geometrically and high-numbered transactions will need to perform larger number of hash operations on average.

The result of the geometric distribution scheme is to give smaller *HCost$_{avg}$* value for earlier transactions. Shorter sessions will benefit more from the geometric distribution.

In section 3.7, we present the results of our tests to evaluate sparse caching with geometric spacing.

3.3 Energy Consumption

When designing any authentication protocol for mobile devices, it is important to reduce the energy expended by this protocol. According to [81], there are at least three approaches to preserving battery life in mobile devices: efficient hardware, accurate knowledge of energy consumption of different cryptographic approaches and light weight security mechanisms. In designing our protocol, one of our major goals was to come up with a light security mechanism while ensuring the highest cryptographic strength available.

Energy consumption is largely influenced by the cryptographic hash function used in the authentication scheme as different hash functions have different energy consumption levels. The authors of [82] conducted an extensive analysis of energy characteristics of various cryptographic approaches and found that energy varies according to the cryptographic approach utilized. For SHA, SHA1 and HMAC, the energy required to conduct a single operation is 0.75, 0.76 and 1.16 microjoule/byte, respectively (for a complete list of energy consumption characteristics of different cryptographic approaches, please refer to [82]). The level of energy consumption by our authentication protocol in a user session is correlated with the value of *TotalHashCost* described in section 3.2.1.

In section 3.8, we compare the energy consumption of our sparse caching protocol and compare it with the case of no caching. It should be noted though that the initialization phase is not included in this comparison because it is conducted using an HTTPS.

3.4   Evaluation and Performance Results

To experiment with the sparse caching HACH scheme, we developed a benchmark which was written using Java. The benchmark fully implements the one-way hash chain model and the different sparse caching configurations.

In the tests and experiments to evaluate the performance of HACH, we considered several situations and different scenarios. Our objective was to simulate real life connections which are characterized by differing needs as far as storage and performance are concerned. Therefore, we used the benchmark to evaluate the HACH performance with numbers of transactions having different ranges: from 1 to 200 for short sessions, from 500 to 2500 for long sessions and up to 4000 and 5000 transactions in some tests. We also varied the storage availability to make sure that we address different users' needs. The storage spaces used to evaluate performance ranged from 20 to 500 spaces.

3.4.1   Impact of Cache Size on HACH Performance

Figure 3.1 shows the impact of cache size on the performance of HACH. The execution overhead of HACH is measured by *TotalHCost*, the total number of hash operations in a session including the initial cache setup. The spacing scheme used in Figure 3.1 is the uniform sparse caching scheme described in section 3.2.1. The results in Figure 3.1 are obtained from the Java testbed and they agree with the cost estimation derived in section 3.2.1. The results clearly show that the more cache we have, the less the *TotalHCost* would be.
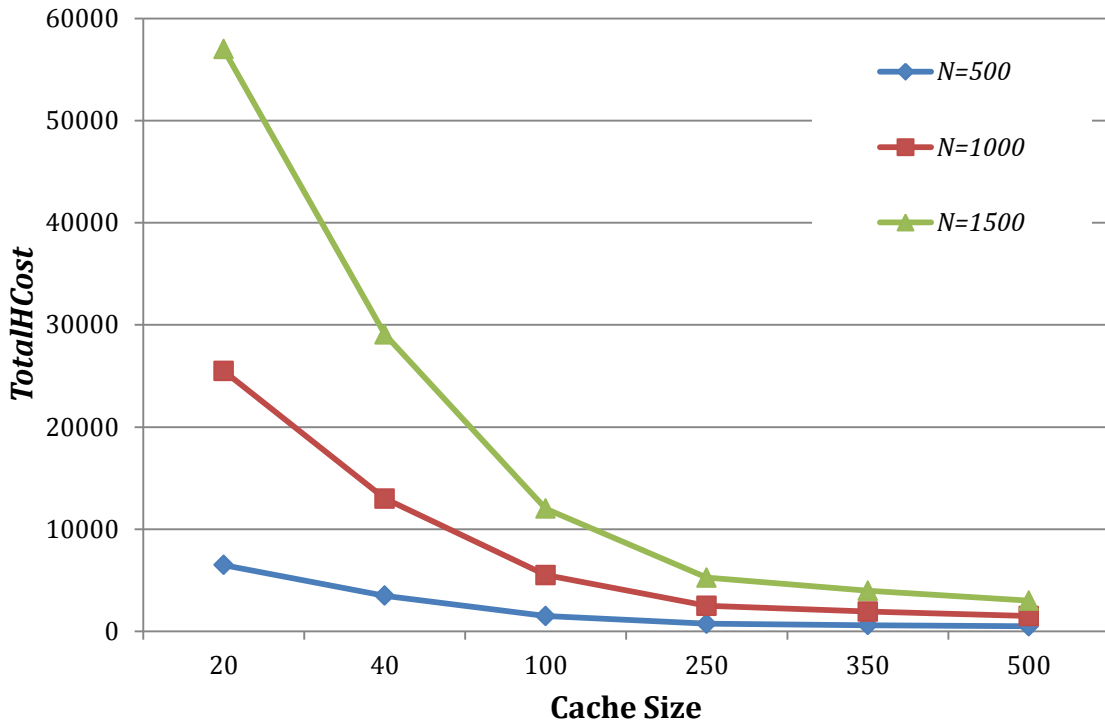
Figure 3-1 Impact of cache size on *TotalHCost*

Figure 3.2 demonstrates the impact of cache size on the average value of the execution overhead of one transaction (i.e. $HCost_{avg}$). Again we see that the more cache we have, the less the value of $HCost_{avg}$. The sparse caching scheme speeds up transaction authentication and helps in reducing the turnaround time of user requests.
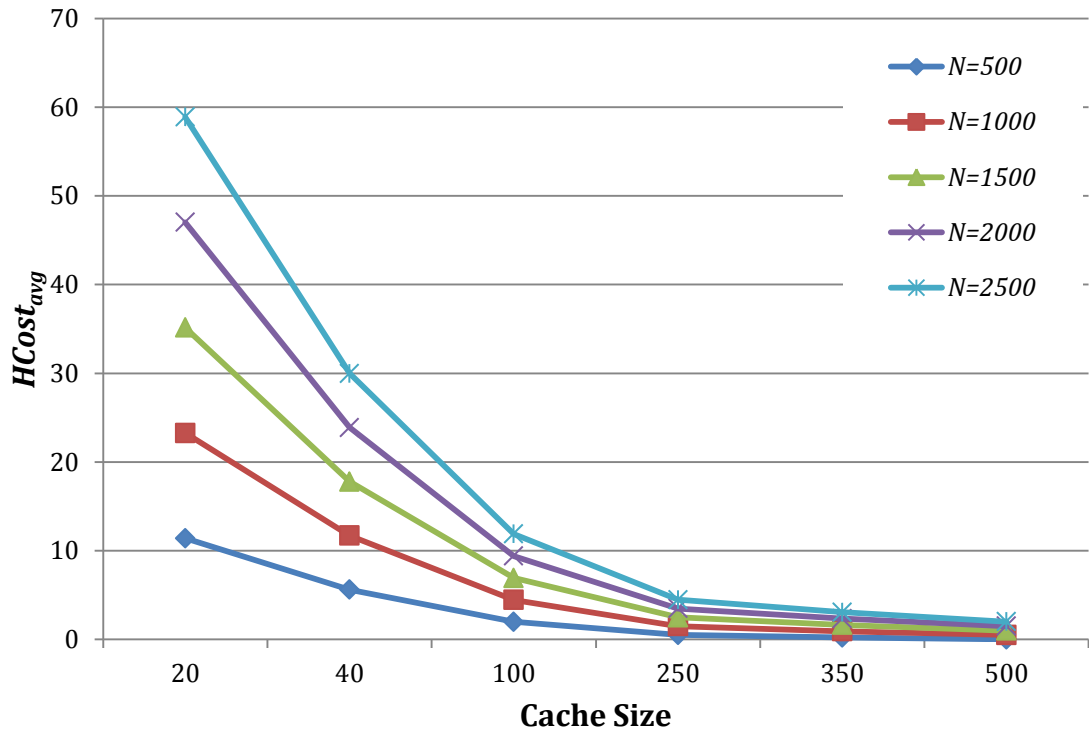
Figure 3-2 Impact of cache size on *HCostavg*

3.4.2 Cache Space Allocation Policies

In section 3.2.2, we proposed a Weighted Overhead formula and gave an example of three classes for mobile devices used in wireless networks. For Class 1 (high-end laptops with plenty of memory), we can afford to allocate large cache memory to get the best benefit of HACH. For Classes 2 and 3 (mobile phones with limited capability), the memory allocated to cache will be at a lesser level. To investigate the performance of sparse caching on the different mobile devices, we used the following two policies for allocating cache memory to a user session with *N* transactions:

1- The *square root (sqrt)* policy which allocates *cache_size* $= (N)^{0.5}$ for a session with N transactions.

2- The *logarithm (log₂)* policy which allocates *cache_size* $= log_2(N)$ for a session with N transactions.

We have tested a few other policies but we select the above two policies for presentation in this dissertation because they nicely suit the classification of mobile devices described in section 3.2.2. The log_2 policy is an aggressive policy which reserves more memory and therefore can be used for Class 1 devices. The *sqrt* policy, on the other hand, exhibits a more conservative behavior and reserves smaller amount of cache memory, which is suitable for low end mobile devices of Class 2 and 3. Both policies can be multiplied by a scale factor (ranging from a small fraction to a large number) to adapt the rate of memory allocation based on the capability of the mobile device, e.g., to differentiate between Class 2 and Class 3.

Figure 3.3 shows the HACH execution overhead and Figure 3.4 shows the HACH storage overhead of the *sqrt* and log_2 cache allocation policies. The log_2 policy allocates more cache space and consequently incurs a much less *TotalHCost*. The *sqrt* policy allocates less cache space and incurs much higher *TotalHCost*.
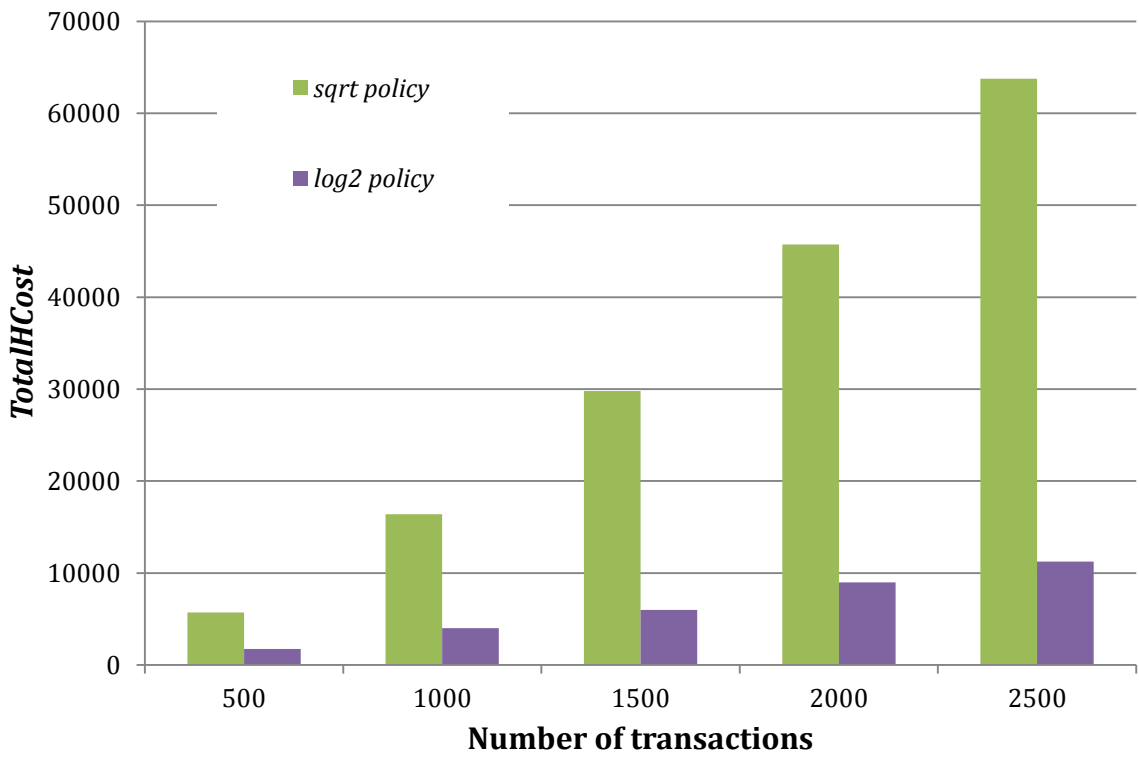
Figure 3-3 HACH performance with two policies for sparse caching
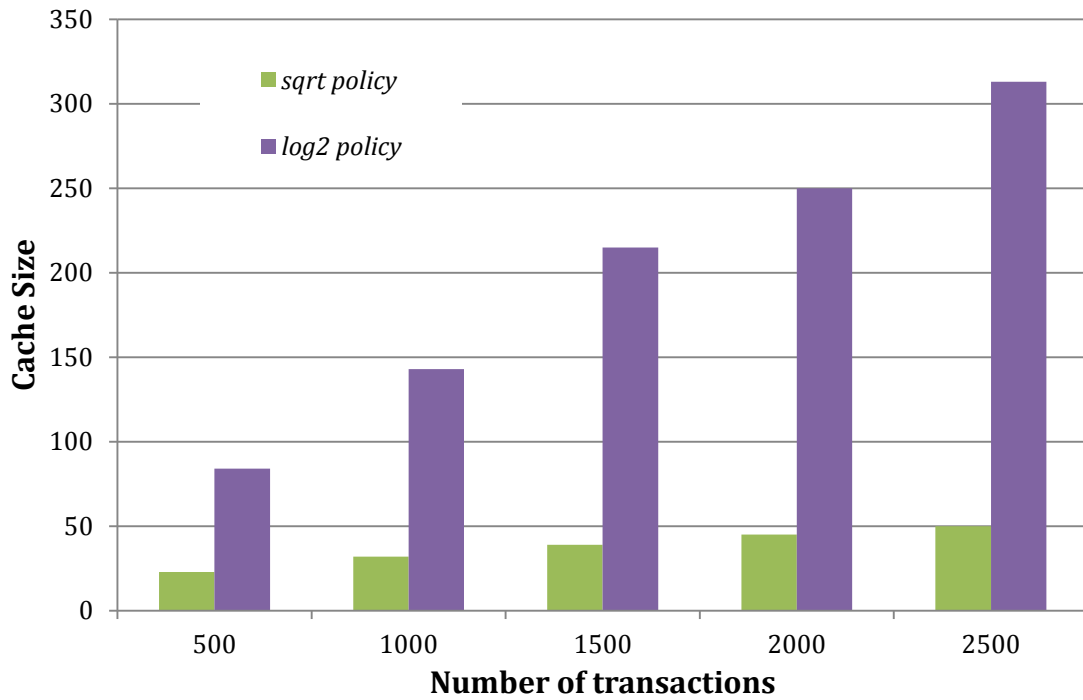
Figure 3-4 HACH storage requirements with two policies for sparse caching

3.4.3   Effectiveness of Sparse Caching

In this section, we present the simulation results when we compared the performance of HACH with and without sparse caching. Table 3.1 summarizes the results we obtained from tests using the *square root (sqrt)* policy. The Performance Ratio in the last column is the ratio *TotalHCost with no caching/TotalHCost with sparse caching*. The sparse caching scheme significantly improves performance of HACH. Sparse caching is able to decrease *TotalHCost* of transactions by an average of approximately 41 times over the six values of *N* shown in Table 3.1.

Table 3-1 HACH performance with/without sparse caching using the sqrt policy

Number of Transactions	*TotalHCost* No Caching	*TotalHCost* Sparse Caching	Performance Ratio
500	125250	5702	21.97
1000	500500	16404	30.51
1500	1125750	29811	37.76
2000	2001000	45750	43.74
3000	4501500	83625	53.83
4000	8002000	127506	62.76

In Figure 3.5, we further illustrate the performance improvement ratio of the HACH using sparse caching. The x-axis represents the number of transactions and the left y-axis represents the performance improvement ratio associated with every value of *N*. The right y-axis shows the storage requirement for each value of *N*. We notice that each added cache unit decreases the value of *TotalHCost*. This is intuitive since if we can afford more memory, we

would definitely improve performance. However, in the case of mobile phones and other low-end mobile devices, the storage may not always be readily available and we need to find a compromise between performance and storage.
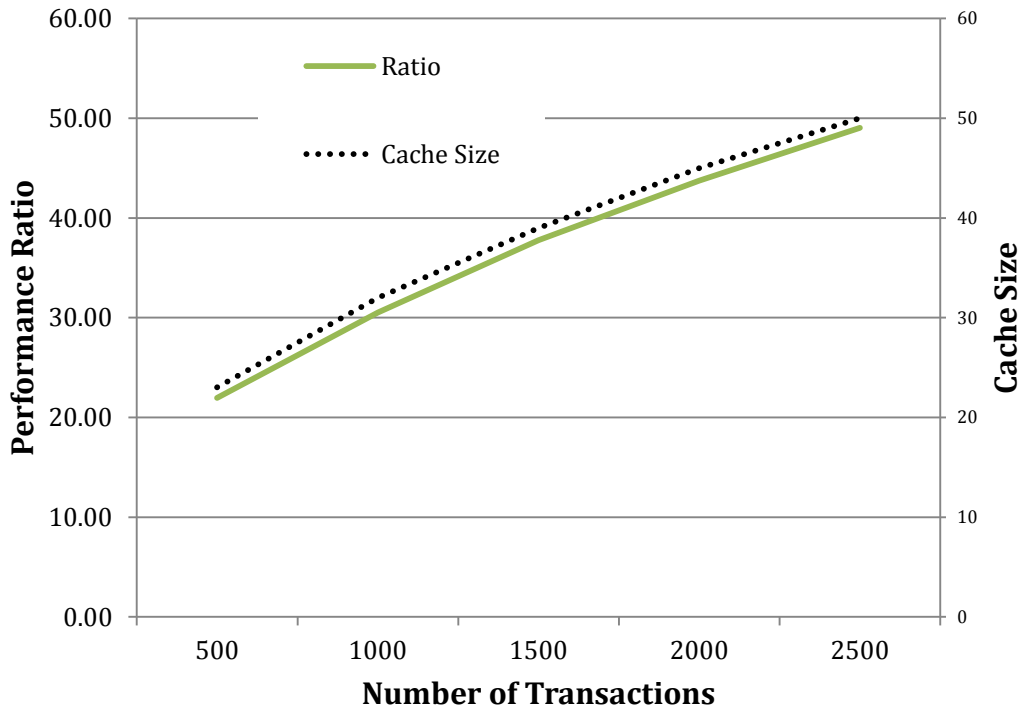
Figure 3-5 HACH performance improvement ratio

3.4.4 Selecting Cache Size for Mobile Devices

In order to find the best tradeoff between cache size and performance (measured in *TotalHCost*), we introduced the Weighted Overhead formula

$$WO = W \times CacheSize + TotalHCost \qquad\qquad (3.4)$$

In the simulation tests, we experimented with different values of *w* to represent the different classes. For Class 1 devices, the value of *w* was a small value close to zero because memory consumption is not a substantial issue with high-end devices. Higher values of *w* are

used for Class 3 devices and lower values are used for Class 2 devices. We ran tests for different values of N.
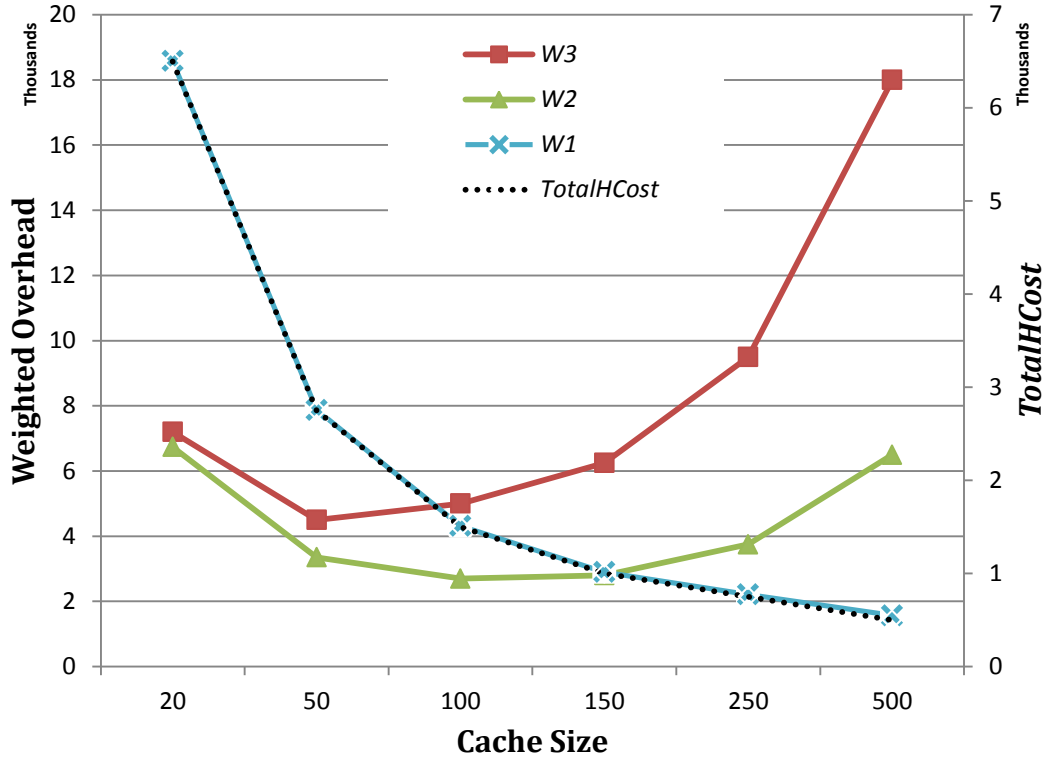
Figure 3-6 Weighted Overhead results for N=500

Figure 3.6 shows the best cache size when the number of transactions is 500. The left y-axis represents the value of the Weighted Overhead and the right y-axis is the value of *TotalHCost*. The values of weights are $w1 = 0.1$ representing Class 1 devices, $w2 = 12$ representing Class 2 devices and $w3$=35 representing Class3 devices. The dashed curve is used to indicate *TotalHCost*. We notice that for this particular number of transactions (N=500), a cache size of 50 units is the most suitable for Class 3 devices as it exhibits the lowest Weighted Overhead value which strikes an acceptable tradeoff between *TotalHCost* and memory. For Class 2 devices, the best cache size is 100. We notice for Class 1 devices represented by $w1$, the

Weighted Overhead curve is very close to the *TotalHCost* curve, indicating that we should select the highest possible cache size *C=N*, i.e., complete caching of size 500.
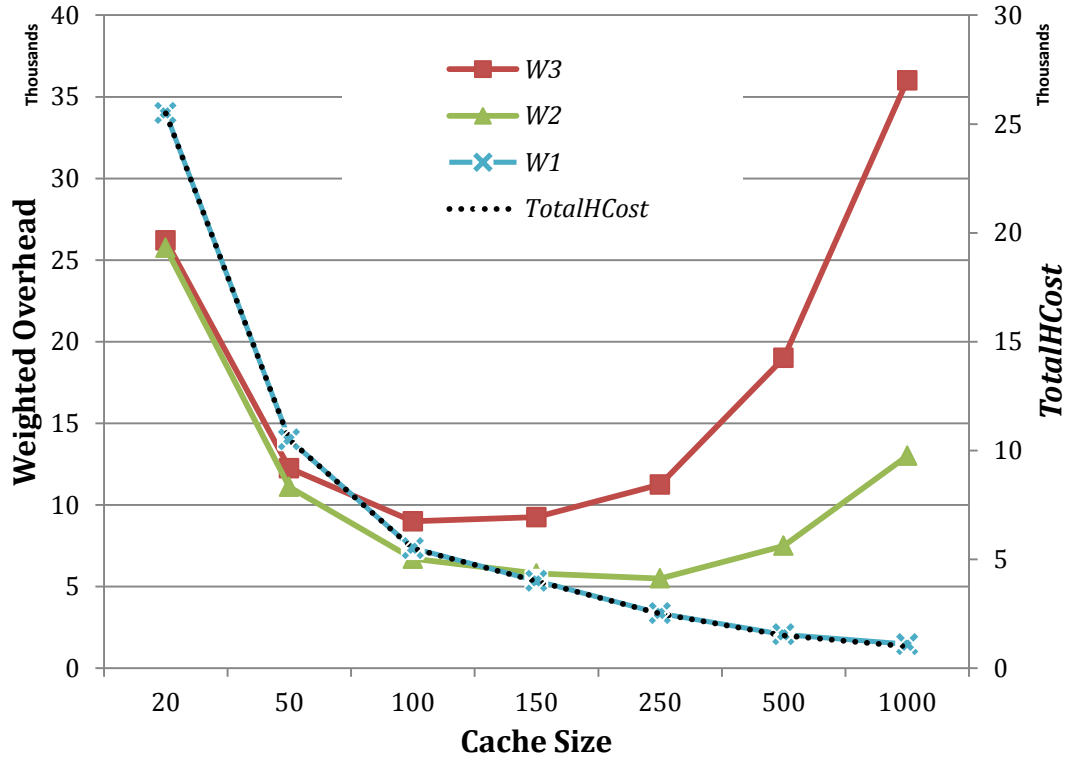
Figure 3-7 Weighted Overhead results for *N*=1000

Figure 3.7 shows the same test but with *N*=1000. The same *w* values reported for Figure 3.6 were used here. For Class 3 devices (*w3* curve), the best cache size is slightly over 100. For Class 2 devices (*w2* curve), the best size of cache is 250. For Class 1 devices (*w1* curve), it is best to use complete caching of size 1000.

We further analyzed the performance of sparse caching for HACH using a metric called the *speedup factor per unit cache* (or simply the *speedup factor*) defined as follows:

$$Speedup\ factor = \frac{H1}{(H2 \times C)}$$

Where

H1 = the value of *TotalHCost* without caching, and

H2 = the value of *TotalHCost* with sparse caching of size *C*

Table 3.2 gives the results for the *speedup* factor for Class 3 (weight *w3=35*) when the cache size used is the best cache size selected by the Weighted Overhead formula. Table 3.2 shows that the contribution of one cache unit is captured by a *speedup* value of approximately 0.9. As an example for $N = 500$ transactions and cache size $C = 50$, each cache unit improves the performance by a magnitude of 0.91 and the total cache of size 50 increases performance by a magnitude of 45.5 resulting in decreasing the total number of hashes from 125250 to 2750.

Table 3-2 Speedup factor per unit cache for Class 3

Number of Transactions	Cache Size	*H2*	*H1*	*Speedup*
500	50	2750	125250	0.91
1000	100	5500	500500	0.91
1500	150	8250	1125750	0.91
2000	250	9000	2001000	0.89
3000	250	18000	4501500	1.00
4000	500	18000	8002000	0.89

Table 3.3 shows similar results for Class 2 devices (*w2* = 12). The cache sizes used for these devices are bigger than those of Class 3. It should be mentioned that for values of *N* equal to 2000 and higher, the weight *w2* = 12 gave multiple best cache sizes (giving the same Weighted Overhead value). We therefore used a weight value *w2* = 15 to pick the specific cache size shown in Table 3.3.

Table 3-3 *Speedup* factor per unit cache for Class 2

Number of Transactions	Cache Size	*H2*	*H1*	*Speedup*
500	50	2750	125250	0.91
1000	100	5500	500500	0.91
1500	150	8250	1125750	0.91
2000	250	9000	2001000	0.89
3000	250	18000	4501500	1.00
4000	500	18000	8002000	0.89

For Class 1 devices (*w1* ≈ 0), the Weighted Overhead formula suggests using complete caching. Table 3.4 gives the results for the *speedup* factor for Class 1 (weight *w1* ≈ 0) when the cache size used is equal to the value of *N*. The *speedup* factor for Class 1 with complete caching is 0.50.

Table 3-4 *Speedup* factor per unit cache for Class 1

Number of Transactions	Cache Size	*H2*	*H1*	*Speedup*
500	500	500	125250	0.50
1000	1000	1000	500500	0.50
1500	1500	1500	1125750	0.50
2000	2000	2000	2001000	0.50
3000	3000	3000	4501500	0.50
4000	4000	4000	8002000	0.50

3.5   HACH with Non-uniform Cache Spacing

Tables 3.5 and 3.6 give comparisons between sparse caching with uniform spacing and with non-uniform spacing. For non-uniform spacing, we allocated 50% of the cache for the first 20% of transactions, which we call "high priority transactions". The other 50% of the cache is used to serve the remaining 80% transactions, which we call the low priority (uncertain) transactions.   Table 3.5 analyzes the average authentication time (average number of hash operations) of a high priority transaction. The non-uniform spacing gives superior (much smaller) authentication time compared to the uniform spacing. The last column in Table 3.5 gives the value of speedup for a high priority transaction. For example when $N = 500$, the average authentication time for a high priority transaction using non-uniform spacing is 1.5 hash operations whereas the corresponding figure for uniform spacing is 4.5. This means that high priority transactions enjoy a three-fold *speedup* under the non-uniform spacing scheme compared to the uniform spacing scheme. As the number of transactions goes up, the speedup factor decreases slightly and becomes equal to 2.55 at $N = 4000$

Table 3-5 Average *speedup* for a high-priority transaction

Number of Transactions	High Priority	Non Uniform Spacing	Uniform Spacing	*Speedup*
500	100	1.5	4.5	3.00
1000	200	3.5	9.5	2.71
1500	300	5.5	14.5	2.64
2000	400	7.5	19.5	2.60
3000	600	11.5	29.5	2.57
4000	800	15.5	39.5	2.55

Table 3.6 analyzes the average authentication time (average number of hash operations) of a low priority transaction. The non-uniform spacing gives larger (slower) turnaround time compared to the uniform spacing. The last column in Table 6 gives the value of the *slow-up* for a low priority transaction. Notice that the *uniform-spacing* scheme gives the same speed for both high priority transactions (Table 3.5) and low priority transactions (Table 3.6). From Tables 3.5 and 3.6, we see that the non-uniform caching scheme has positively impacted high-priority transactions by a *speedup* factor of 2.5 or higher and has negatively impacted low-priority transactions by a *slow-up* factor of only 1.67 or less.

Table 3-6 Average *slow-up* of a low-priority transaction

Number of Transactions	Low Priority N	Non Uniform Spacing	Uniform Spacing	*Slow-up*
500	400	7.5	4.5	1.67
1000	800	15.5	9.5	1.63
1500	1200	23.5	14.5	1.62
2000	1600	31.5	19.5	1.62
3000	2400	47.5	29.5	1.61
4000	3200	63.5	39.5	1.61

3.6   Approximate Knowledge of N

In the previous sections, we performed our simulation tests assuming that the value of N is accurately known. The value of N in most real life applications is not accurately known. A minimal level of sparse caching can help the developer apply hash chains of larger length

without the fear of any increase in the average or maximum authentication time for a transaction.

Figure 3.8 shows the impact of using sparse caching with *minichain_len* set at 1000 for sessions with unknown number of transactions exceeding 1000. It can be seen that sparse caching reduces $HCost_{avg}$ of transactions when N is not precisely known. For example, when $N = 4000$ transactions, the value of $HCost_{avg}$ without sparse caching is approximately 1500. By placing only three cache units, the value of $HCost_{avg}$ is reduced to 375. We observed that the reduction in $HCost_{avg}$ due to sparse caching increases as the number of transactions increases.
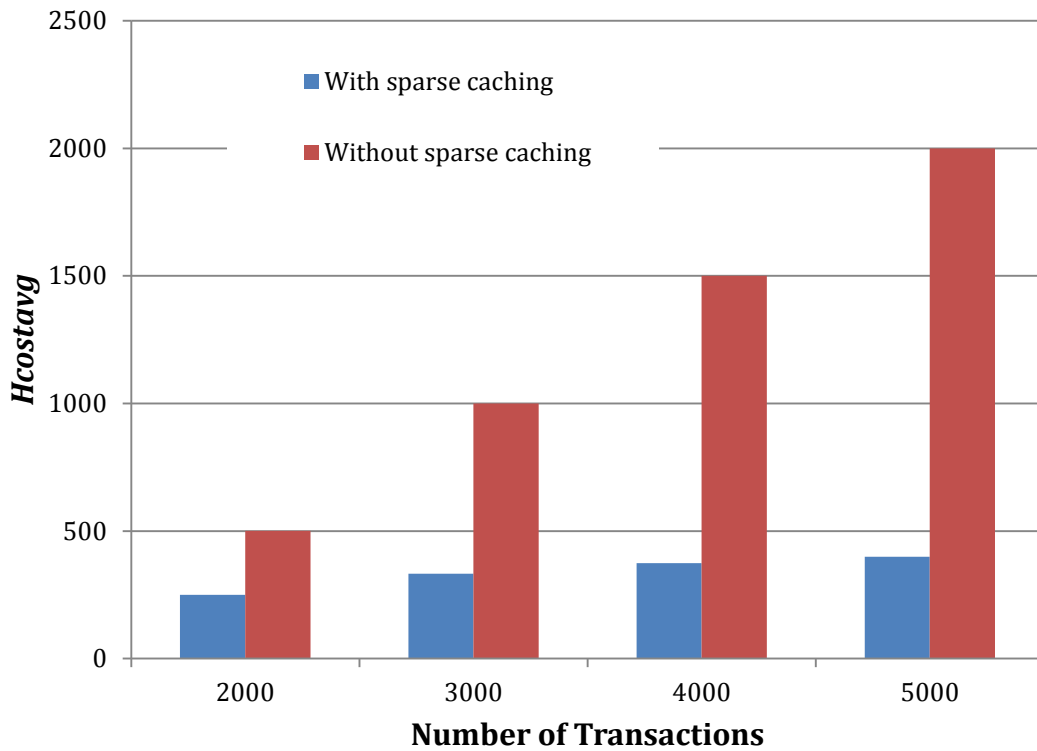
Figure 3-8 Handling unknown large values of *N*: comparison between *HCostavg* with and without sparse caching

3.7   Geometric Spacing

In section 3.2.5, we introduced the idea of geometric spacing of cache which gives smaller $HCost_{avg}$ value for earlier transactions in cases when the exact number of transactions N

53

is not known and could be as small as one and as large as N. The idea involves increasing the cache spacing intervals progressively as the real number of transactions increases.
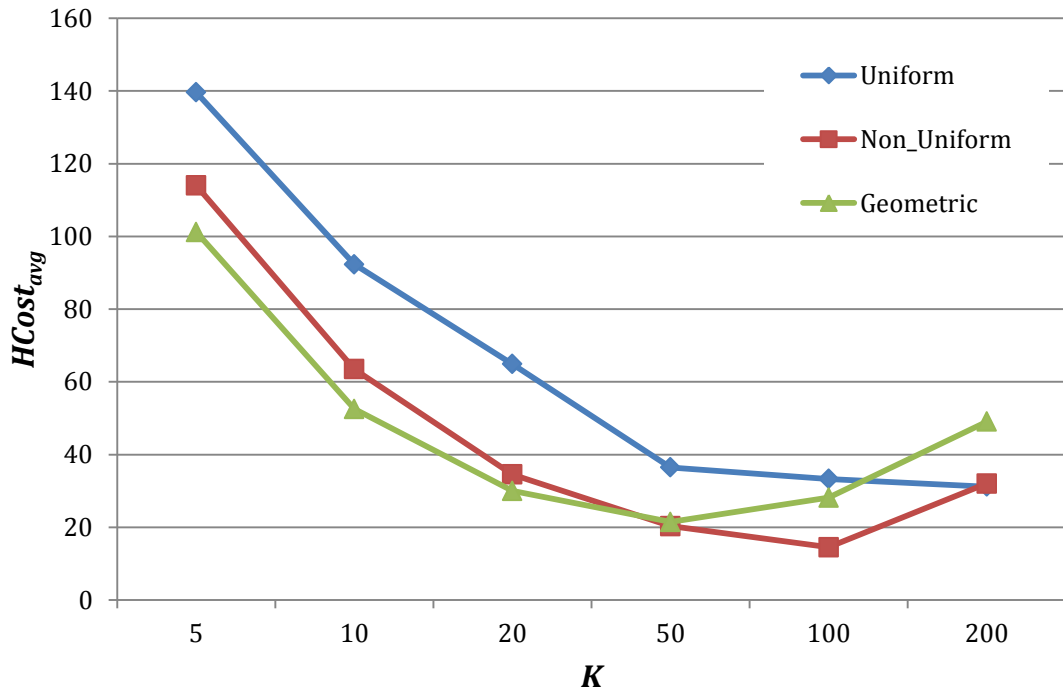
Figure 3-9 Comparison of *HCostavg* between Uniform, Non-Uniform and Geometric Spacing

Figure 3.9 shows the test results for the case when N is not known and the developer has chosen the value $N = 500$ to be the length of the hash chain. The horizontal axis represents the real number of transactions, denoted K. The figure compares the performance of the following three schemes for values of $K = 5$, 10, 20, 50, 100 and 200.

- Uniform caching with equal spacing as described in section 3.2.1.

- Non-uniform caching with two groups as described in section 3.2.4.

- Geometric spacing as described in section 3.2.5.

54

As seen in Figure 3.9, using the geometric spacing policy improves the performance of HACH compared to the other two policies when the real number of transactions K is lower than 50. For the case $K = 50$, the geometric spacing performs better than uniform spacing and has the same performance as the non-uniform spacing policy. For $K = 100$ transactions, geometric spacing still performs better than uniform spacing but worse than non-uniform spacing. The geometric spacing scheme does not benefit HACH when $K \geq 120$ transactions.

3.8 Energy Consumption

In section 3.3, we introduced the energy consumption metric used to evaluate the performance of our sparse caching protocol. Since the energy consumption is largely influenced by the cryptographic hash function used, the type and amount of hashing operations required to carry out the authentication of the internet session translate into the energy consumption of the authentication scheme.

Figure 3.10 illustrates the energy consumption comparison between our sparse caching based HACH protocol and its counterpart without sparse caching. It should be noted that the sparse caching in this comparison is conducted using the *sqrt* policy and SHA-1 hashing. As can be seen in the figure, our sparse caching scheme tremendously improves energy consumption of the HACH authentication protocol. It is also noted that the HACH protocol suffers from a huge increase in energy consumption making it far from ideal especially for platforms with limited energy capacities.
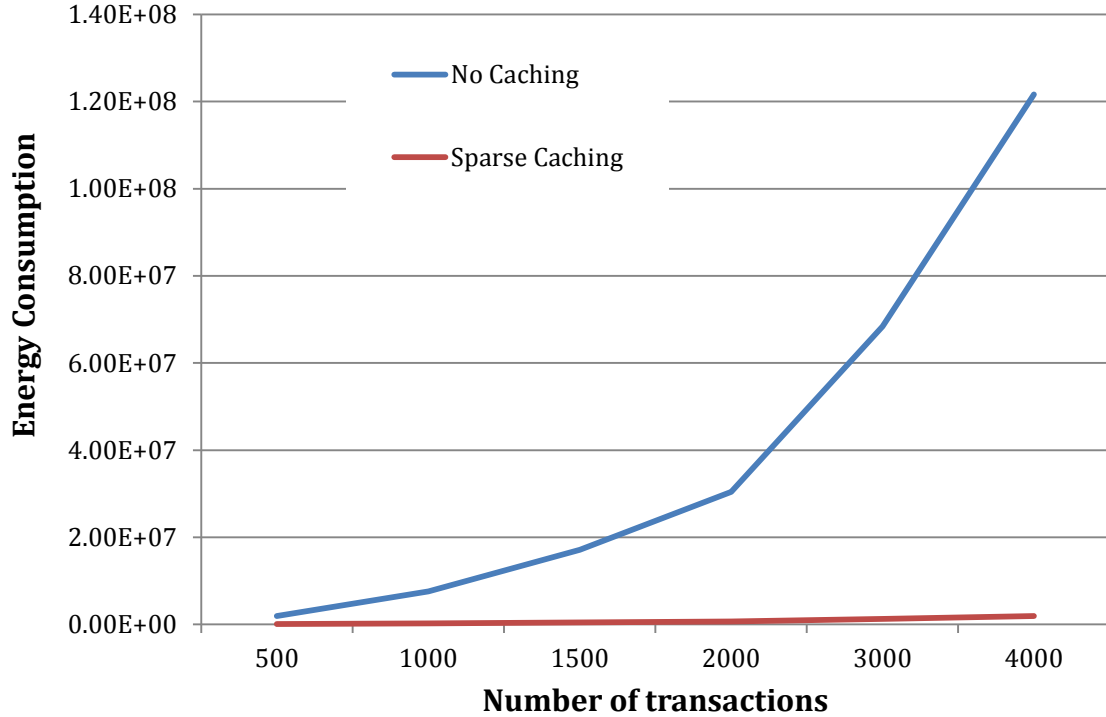
Figure 3-10 Energy consumption comparison of HACH with and without sparse caching

3.9   Conclusion

In this chapter, we have shown that the use of lightweight easy to implement sparse caching approach can significantly improve the performance of the widely used cryptographic one-way hash chain technique to secure session cookies. We introduced a memory-times-computation complexity metric to help select a best cache size depending on different users' storage requirements. Different cache spacing techniques have been investigated to demonstrate different connection behaviors.

We presented the results of extensive performance tests that have shown the significant reduction in authentication cost achieved by the sparse caching schemes. We have also shown how to deal with real-life situations in which the number of transactions per user session is largely unknown and cannot be accurately estimated.

CHAPTER 4: HACH WITH A HIERARCHICAL TWO-TIER CONSTRUCTION (TTOHC)

In this chapter, we present our second contribution to solve the computational overhead of the HACH protocol discussed in Chapter Two. The two-tier one-way hash chain protocol revolves around utilizing different cryptographic hash functions arranged in two tiers. The TTOHC protocol gives significant performance improvement over previously proposed solutions for securing Internet cookies. A detailed Java testbed has been used to evaluate alternative configurations for the hierarchical scheme and investigate the optimal set up of the two tiers. Detailed performance results obtained from this testbed are presented and analyzed.

4.1　Proposed Scheme

A. The Two-Tier One-way Hash Chain (TTOHC) Protocol

Our TTOHC protocol is based on the idea of utilizing two levels of hashing that use different cryptographic hash algorithms, e.g., SHA-1 and HMAC. The OHC (OTC and SCRHC [83] are examples of OHC) scheme has a single one-way hash chain and the length of this chain represents the number of transactions N. The TTOHC scheme, on the other hand, has a single first-tier chain and multiple second-tier chains. The variable N is divided into two components: the length of the first-tier chain N_f and the length of the second-tier chain N_s. Although we will primarily focus on the static case where all second-tier chains have the same length N_s, we will also present results on the dynamic case in which the length of the second-tier chain changes linearly.

The TTOHC protocol is composed of three routines: *Initialization, Authenticate_Transaction,* and *Adjust_Seed.* In the *Initialization* routine, a shared secret value S_0, the number of transactions N and length of the second-tier chain N_s are selected and exchanged

between the server and the client. The authentication tokens transmitted in the cookies are generated by the *Authenticate_Transaction* routine using the second-tier hash function *HS*. These authentication tokens are denoted $V_{i,j}$ where V represents the authentication token, i refers to the first-tier index and j refers to the second-tier index. Each time the length of the second-tier hash chain is exhausted, the *Adjust_Seed* routine is used to compute the new seed for the next second-tier chain. The new seeds are computed using the first-tier hash function *HF*. In what follows, we give the notations and more details of the TTOHC protocol.

N = number of transactions in the session

N_f = length of the first-tier chain

N_s = length of the second-tier chain assuming the static case

$N \leq N_f \times N_s$; ideally $N = N_f \times N_s$

HF is the hash function used in the first-tier

HS is the hash function used in the second-tier

S_0 = Initial seed used by the first-tier chain

In our tests we used a message authentication code HMAC for *HF* and a hash function SHA-1 for *HS*. The values generated in the first-tier are denoted S_1, S_2, S_3, etc., and are used as seeds for the second-tier chains as shown below.

$$S_1 = HF^{Nf}(S_0), S_2 = HF^{Nf-1}(S_0), \ldots, S_{Nf-1} = HF^2(S_0), S_{Nf} = HF(S_0)$$

In our testbed, the HMAC algorithm used to implement *HF* uses two arguments: a seed S_0 and a message g; therefore, we created another argument *g* during initialization (*g* is the string "first-tier"). The values generated in the second-tier chain are the authentication tokens used in the authentication step for the successive transactions of a user's session. User transactions

numbered 1 to N_s use the tokens generated by the first second-tier chain based on the seed S_1, transactions numbered N_s+1 to $2N_s$ use the tokens generated by the second second-tier chain based on S_2 and so on. Specifically, the first transaction uses the token value $V_{1,1} = HS^{Ns}(S_1)$. The second transaction uses $V_{1,2} = HS^{Ns-1}(S_1)$ and the transaction numbered N_S uses the token $V_{1,Ns} = HS(S_1)$. The token $V_{1,Ns}$ is the last token generated using the seed S_1. For the next transaction, numbered N_S+1, the routine *Adjust_Seed* is invoked to compute the new seed S_2 for the second second-tier chain. After calling the *Adjust_Seed* routine, transaction numbered N_S+1 uses the authentication token $V_{2,1} = HS^{Ns}(S_2)$. The pseudo codes for the three routines are given below.

Initialization:

The initialization stage takes place using an HTTPS connection. During the HTTPS authentication, the initial value of the shared secret S_0, a message g, the length of the hash chain N and the length of the second-tier chain N_s are selected and exchanged between the server and the client. The following code is executed at both the client and the server.

$N_f := N \div N_s$ // length of the first-tier chain

$L := N_s$ // L is the global index for the second-tier chain

$J := N_f$ // J is the global index for the first-tier chain

$S := HF^J(S_0)$ // S is now S_1=seed for the first second-tier chain

The routine *Authenticate_Transaction* generates the authentication token V at the client side. The authentication token is generated by the second-tier hash function HS. The token V is then transmitted in the cookie of the transaction to the server where a similar routine is executed to compute the value V at the server. The server compares the received value and the computed value to authenticate the transaction.

Authenticate_Transaction(S,L)

Begin

$V := HS^L(S)$ // L is the global index for the second-tier chain

$L := L - 1$

if ($L == 0$) then

    $S := $ *Call Adjust_Seed( )*

    $L := N_s$; end_if

Return (V);

End

For example, if $N = 500$ transactions and $N_s = 100$, then $N_f = 5$ and the authentication tokens will be computed as follows:

1st transaction uses token $V_{1,1} = HS^{100}(S_1)$,

2nd transaction uses token $V_{1,2} = HS^{99}(S_1)$,

………,

100th transaction uses token $V_{1,100} = HS(S_1)$,

101st transaction uses token $V_{2,1} = HS^{100}(S_2)$,

102nd transaction uses token $V_{2,2} = HS^{99}(S_2)$,

……..,

200th transaction uses token $V_{2,100} = HS(S_2)$,

……..,

401st transaction uses token $V_{5,1} = HS^{100}(S_5)$,

……....,

500th transaction uses token $V_{5,100} = HS(S_5)$,

The *Adjust_Seed* routine generates the seeds for the second-tier chains using the first-tier hash function *HF* as given blow.

Adjust_Seed( )
Begin
$J := J$-1   // J is the global index for the first-tier chain
$S := HF^J(S_0)$;
Return (S)
End

4.2   Performance Results

To evaluate the proposed TTOHC protocol and investigate its optimal setup, we wrote a detailed benchmark in Java. We used a metric, *SessionCost,* which is the total cost of the hash operations in both the first-tier chain and all second-tier chains during the entire lifetime of the session. In the *SessionCost* metric, the unit cost is normalized to be the cost of a hashing operation computed by SHA-1 (the cost of HMAC is higher). We use this metric as an indicator of the computational overhead. The higher the *SessionCost* is, the higher the computational overhead associated with transmission of cookies. We conducted extensive tests to account for different real life scenarios including long sessions in which the user opts for the 'remember me' option offered by many web applications. In our experiments, we took into consideration the discrepancy of session length in real life and so approximated the results. The results are an average of 1000 simulations. Below we present and analyze our performance results.

A. Comparison between TTOHC and OHC

Figure 4.1 compares the performance of the TTOHC protocol with the non-hierarchical OHC scheme for different values of *N*. In Figure 4.1, the TTOHC protocol used the same second-tier chain length N_s=20 for all values of *N*. It is easy to see that TTOHC outperforms

OHC by a wide margin. The cost comparison indicates that the TTOHC protocol decreases the computational overhead represented by the metric *SessionCost* tremendously. For example when the session has *N*=500 transactions, the session cost of OHC is 250, whereas the corresponding cost of TTOHC is approximately 13. The difference in performance grows as the number of transactions increases showing the benefit of our proposed scheme.
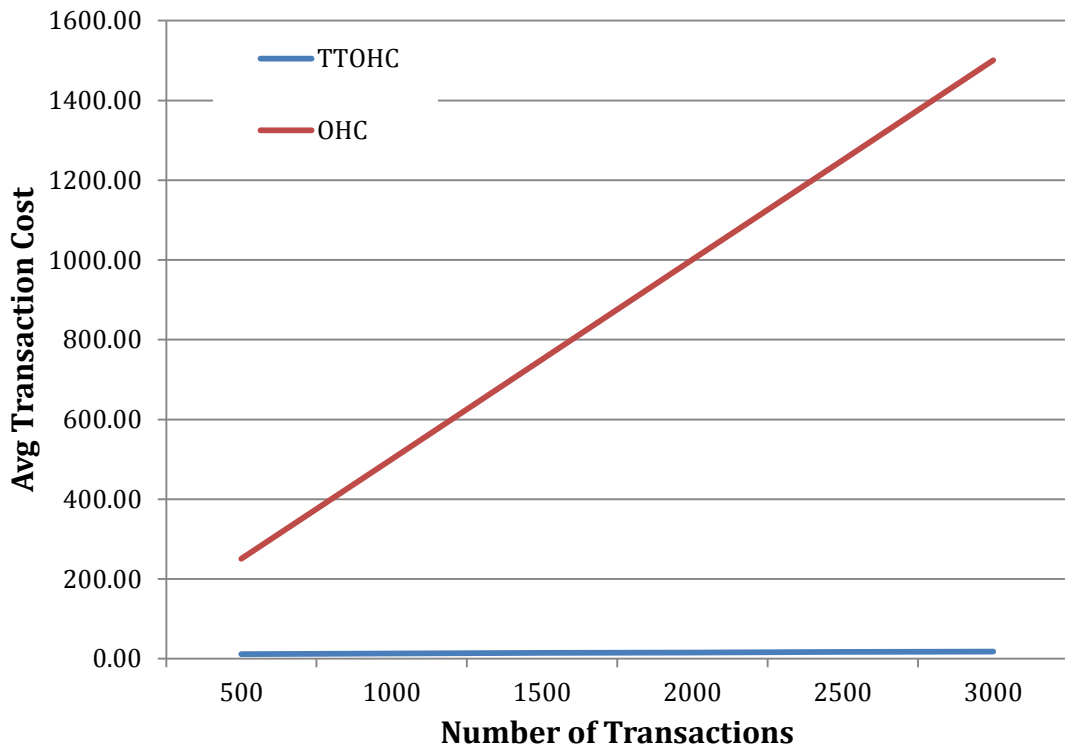
Figure 4-1 Comparison between OHC and TTOHC (*Ns*=20)

To further illustrate the benefit of the TTOHC over OHC, we plotted (in Figure 4.2) the performance improvement ratio between the session cost of TTOHC and the session cost of OHC. The performance improvement ratio ranges from 21 at *N*=500 to 83 at *N*=3000. The average improvement ratio of TTOHC over OHC in Figure 4.2 is 56. The two-tier one-way hash chain scheme achieves great performance gain and significantly reduces the computational overhead of the OHC scheme.

We next investigate the problem of how to select the value of the length of the second-tier chain N_s. The value of N_s, selected during initialization, affects the length of the first-tier chain N_f and significantly impacts performance of the TTOHC protocol.
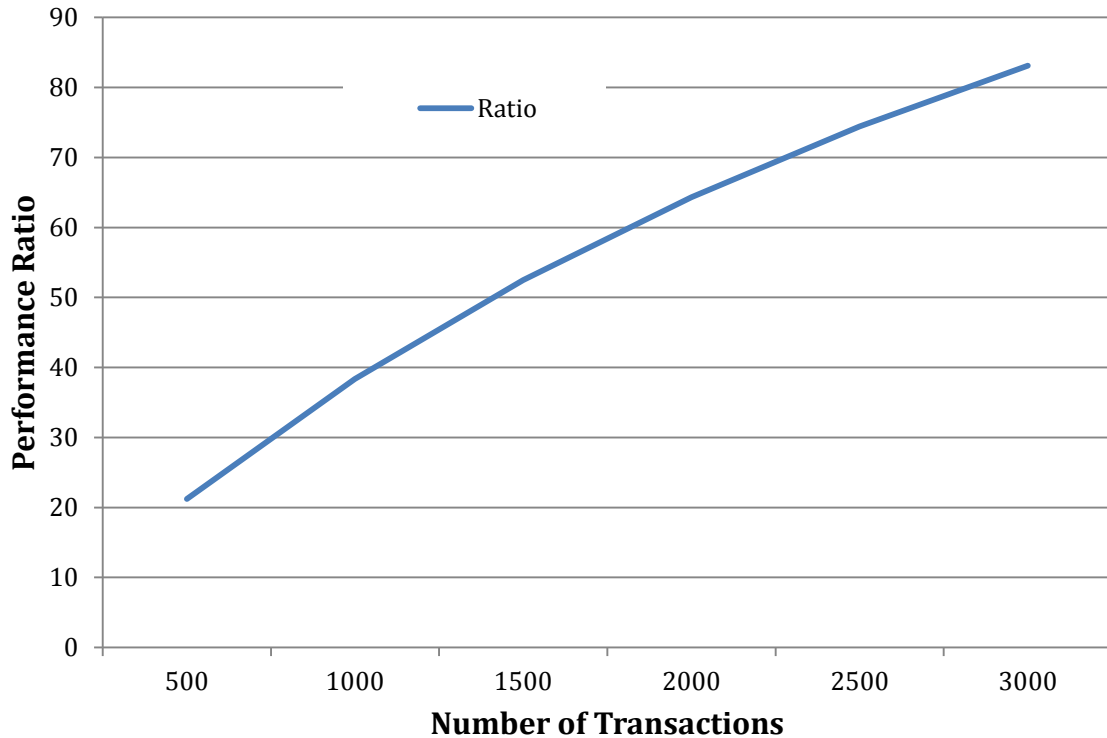
Figure 4-2 TTOHC to OHC Performance Improvement Ratio in the Static case

B. Selecting N_s in the Static Case

In the static case, the value of N_s is set up during initialization and is not changed throughout the duration of the session. Once the first second-tier chain is exhausted and the *Adjust_Seed* is called to generate the seed for the next second-tier chain, the value of N_s is reset to its original value selected during initialization. This is repeated for all second-tier chains until the session is completed.

The performance of the TTOHC protocol when N_s is static is illustrated in Figure 4.3 which plots *SessionCost* versus N_s.
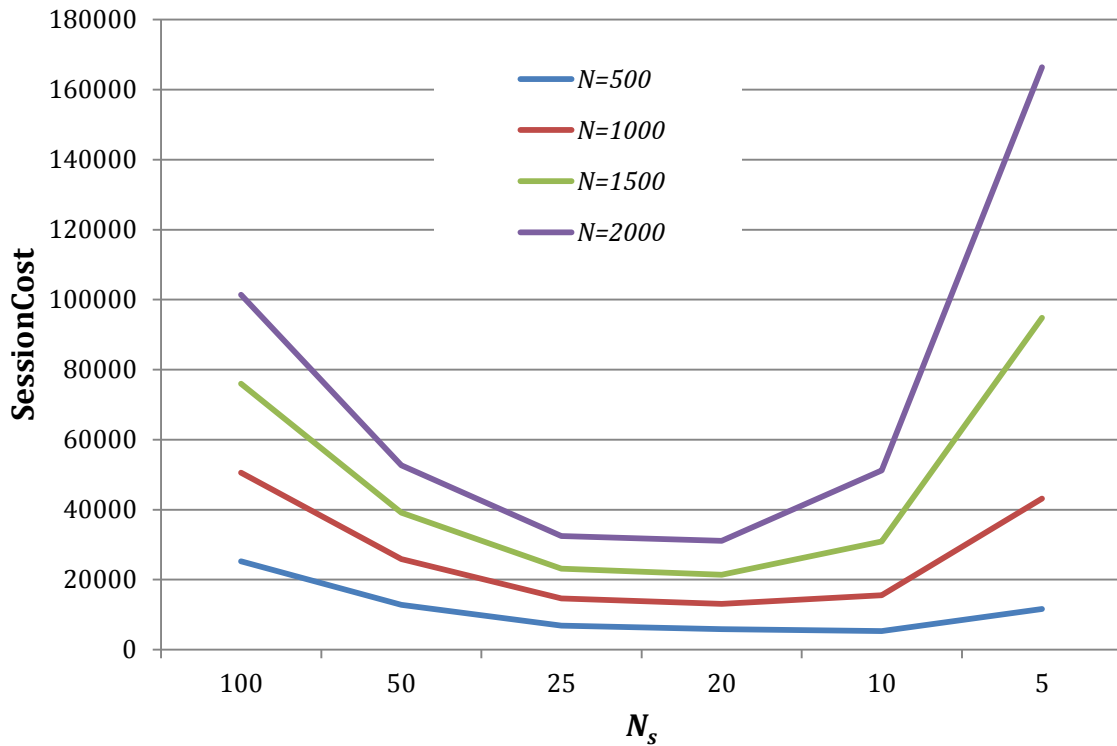
Figure 4-3 TTOHC Performance for static *Ns*

Figure 4.3 shows that the performance of TTOHC is optimal when the value of N_s is between 20 to 25. This means that we should select the hash chain length of the second-tier chain to be in the range 20-25. In fact, the value $N_s = 20$ tends to strike a good balance for minimizing *SessionCost* over all values of *N*. It should be noted that the value $N_s = 20$ is only optimal for the particular setup we used in our tests, i.e., when HMAC is used to implement *HF* in the first-tier chain and SHA-1 is used to implement *HS* in all second-tier chains. Choosing different hash function will certainly change the optimal chain length value. For example, when we swapped the two functions and used the more expensive HMAC in the second-tier chains and SHA-1 in the first-tier chain, the optimal value of N_s was found to be around 10.

C. Dynamic N_s

We also investigated the use of dynamic values for N_s, i.e., the length of the second-tier chain is not always the same. In this case, the value of N_s dynamically changes each time the *Adjust_Seed* routine is called. There are several options of dynamic N_s configurations, but we will report the results of the case when N_s increases linearly. The linear increase of the value of N_s means that an initial value N_s is selected for the first second-tier chain and is increased by a certain amount in the successive chains. In our tests we used an initial value $Init\_N_s = N_s$ as the length of the first second-tier chain, $2*N_s$ as the length of the second second-tier chain, $3*N_s$ as the length of the third second-tier chain and so on until the session is completed. In order to account for this increase, we modified the code of the *Initialization* routine so that we can compute the length of the first-tier chain N_f needed to produce the seeds for the second-tier chains in the dynamic case. We also modified the *Adjust_Seed* routine by letting it handle the dynamic N_s update as follows:

Adjust_Seed( )

Begin
$N_s := N_s + Init\_N_s$ // N_s increases linearly
$J := J-1$
$S := HF^J(S_0);$
Return (S)
End

To illustrate this, let us examine how the code works to handle $N = 500$ transactions when N_s is linearly updated. Below we give the authentication token values produced for the successive transactions when $N = 500$, $Init\_N_s = 2$ and the length of second-tier chain increases linearly as explained above. The *Initialization* routine uses the values $N = 500$ and $Init\_N_s = 2$ to

compute the maximum possible value of N_f (i.e. the number of times the seed is going to be updated). The maximum value of N_f in this case is 22 (i.e., there are 22 second-chair chains needed to cover all 500 transactions.) The following are the authentication tokens for the dynamic case when the initial value of N_s is 2.

$$V_{1,1} = HS^2(S_1)$$
$$V_{1,2} = HS^1(S_1)$$
$$V_{2,1} = HS^4(S_2)$$
$$V_{2,2} = HS^3(S_2)$$
$$V_{2,3} = HS^2(S_2)$$
$$V_{2,4} = HS\ (S_2)$$
$$....$$
$$V_{22,38} = HS^7(S_{22})$$

In the above example, the first transaction (with token $V_{1,1}$) performs two hash operations. The second transaction (with token $V_{1,2}$) performs one hash operation. Then, the value N_s has to be updated by the *Adjust_Seed* routine to $N_s = 4$ and a new seed S_2 is computed. By the 500th transaction, we will have updated the value of N_s twenty two times.

Figure 4.4 shows how linearly increasing N_s impacts the performance of the TTOHC protocol. Here, we can see that if we start N_s from a low value (e.g. *Init_N_s* = 1 or 2), we achieve better performance. If the initial value of N_s starts at a relatively high value, say 20, the protocol suffers from a large overhead as the *SessionCost* tends to be high. So, a low initial value of *Init_N_s* = 1 or 2 would be the best choice for the dynamic case with linear increase.
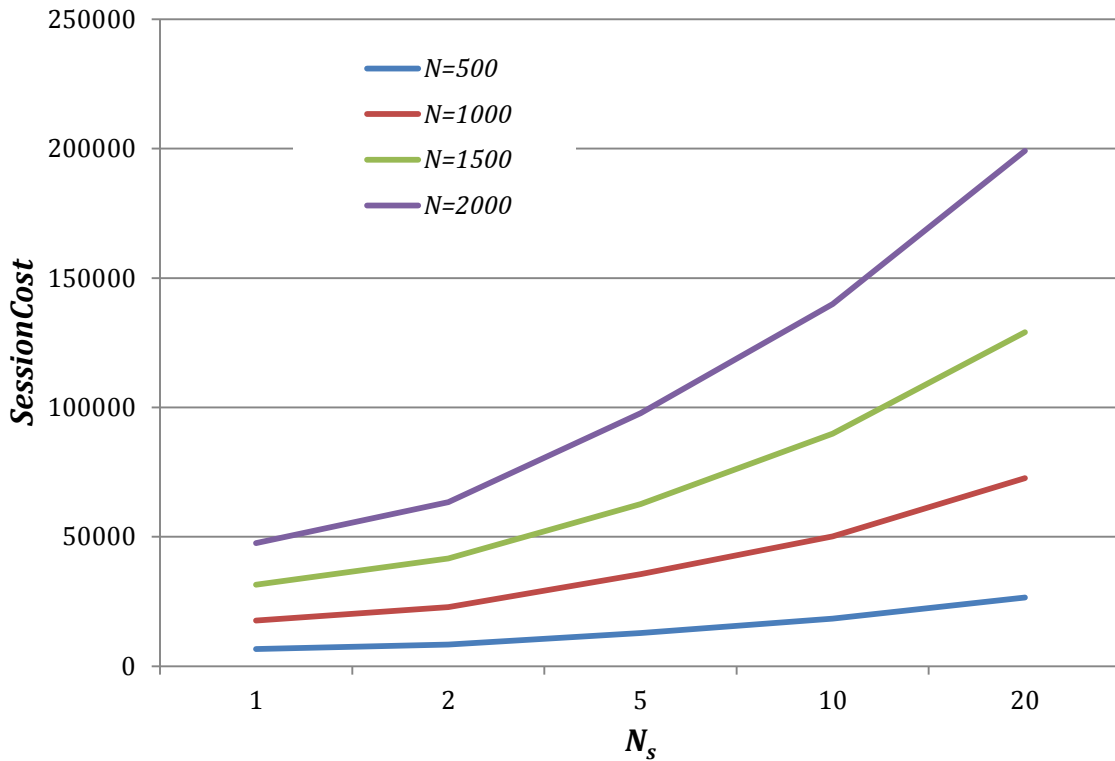
Figure 4-4 TTOHC performance when *Ns* changes linearly

The performance (*SessionCost*) of TTOHC with the dynamic configuration for N_s is relatively higher than the static configuration. Nevertheless, this dynamic configuration still results in a much better performance than the OHC scheme. Figure 4.5 demonstrates how the TTOHC scheme with dynamic configuration outperforms the OHC by a large ratio. Comparing Figure 4.2 and Figure 4.5, we see that the static configuration gives better performance improvement ratio than the dynamic configuration, especially at high values of *N*. Figure 4.6 gives a comparison between the two TTOHC configurations. In Figure 4.6, we used the optimal value of N_s for both configurations, i.e., $N_s = 20$ for the static configuration and *Init_Ns* = 1 for the dynamic configuration. As shown in the figure, the performance of the two configurations is comparable when the number of transactions is low, but the static configuration tends to outperform its dynamic counterpart at higher number of transactions.

Although the dynamic configuration does not match the performance of the static one, it has some practical merit when the value of N is not known and has a wide range that extends from a very small number to a large number. In this case, the dynamic TTOHC configuration would be practically acceptable because the early (more certain) transactions would be authenticated by second-tier chains of small lengths (e.g., 1, 2, 4, 8) and are therefore given good performance.
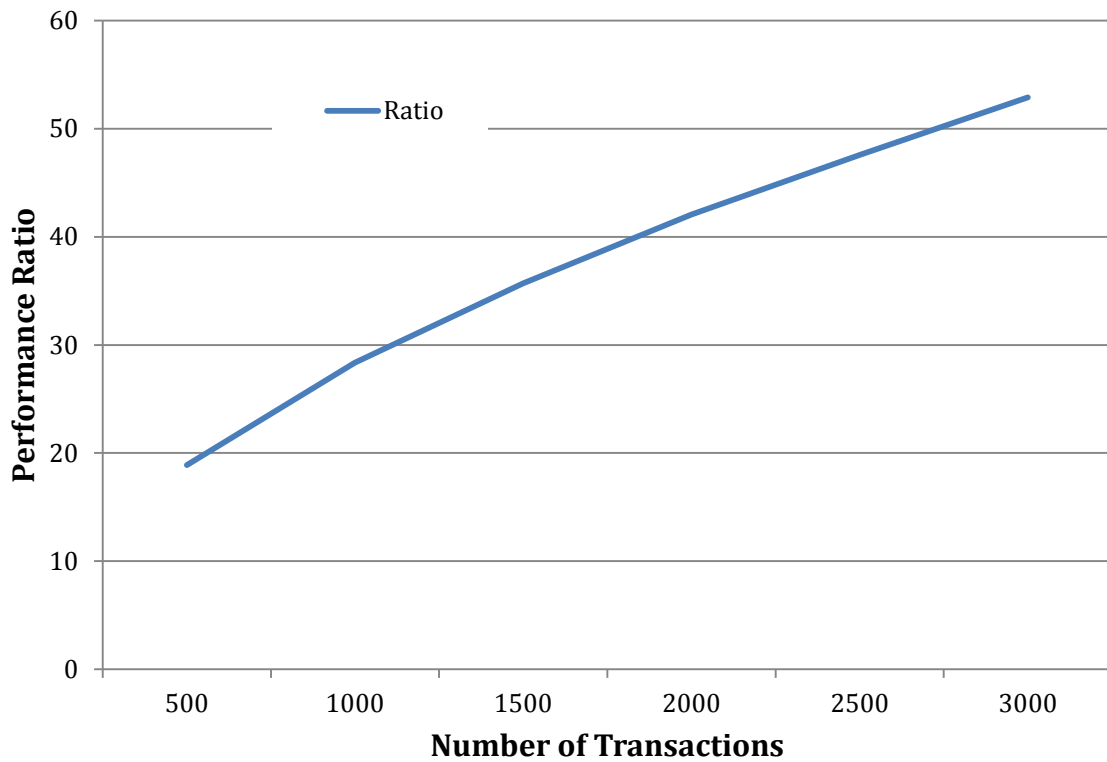
Figure 4-5 Dynamic TTOHC Performance Improvement Ratio
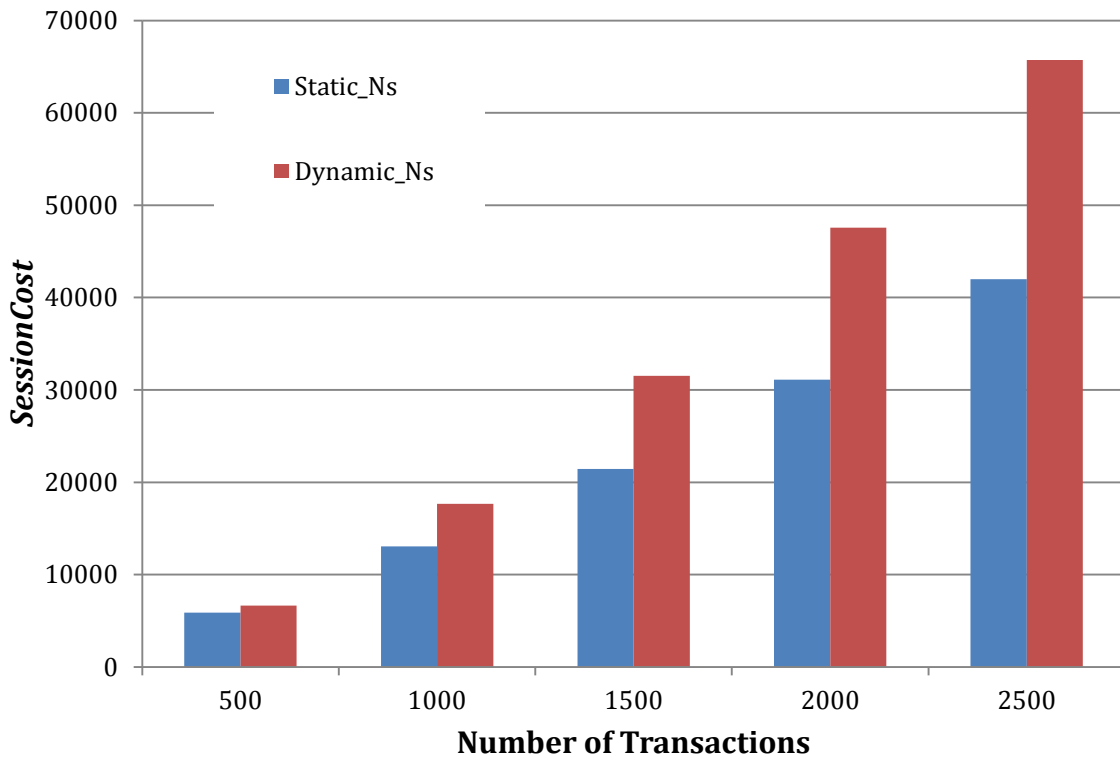
Figure 4-6 TTOHC performance with static and dynamic *Ns*

4.3 Conclusion

In this chapter, we presented a protocol for the efficient authentication of session cookies. Our protocol uses two tiers of one-way hash chains to reduce the overhead of the flat one-way hash chain significantly. Our testbed for the proposed scheme used two good cryptographic hash functions (HAMC for first-tier hashing and SHA-1 for second-tier hashing). The use of cryptographically strong hash functions is recommended to increase the resistance of the scheme to collision and per-image attacks.

CHAPTER 5: HACH FOR COLLABORATIVE AND SOCIAL MEDIA NETWORKS

5.1 Introduction

The use of insecure cookies as a means to authenticate web transactions in collaborative and social media websites presents a hazard to users' privacy. In this chapter, we propose and evaluate a novel protocol for protecting transmitted cookies using two dimensional one-way hash chains. In the first dimension, there is a hash chain that computes secret values used in the second dimension hash function. Multiple hash chains use the secret values created by the first dimension to authenticate session cookies in the second dimension. For improved security, the hashing operations in the second dimension use a concatenation of the secret values and the position index of the hash function within the hash chain. The performance of the scheme is evaluated using a detailed simulation testbed and an analytical model. The optimal lengths of the chains are derived when the number of transactions in the session is known. The protocol is extended to efficiently handle the case when the number of transactions is not known. The evaluation of the proposed scheme reveals that it achieves tremendous improvement over straightforwardly configured one-way hash chain schemes. Also, by adopting the position-indexed hashing protocol, energy consumption is reduced significantly especially with longer sessions making our protocol ideal for battery operated devices.

In designing our protocol, we took into consideration different cryptographic approaches. While one of our main objectives is reducing the computational overhead of one-way hash chains based cookies, we wanted our protocol to benefit from the features of current cryptographic approaches especially their strength and resistance to replay attacks, collision

attacks, pre-image attacks and second pre-image attacks. Hence, our protocol is designed with the state-of-the art cryptographic approaches in mind.

5.2   The Proposed Protocol

A. Conceptualization

Conceptually, the one way hash chains in our protocol are arranged in two dimensions (Figure 5.1). In the first dimension (i.e. horizontal axis), there is a single hash chain that computes the seeds for the second dimension chains (i.e. vertical axis). In the second dimension, multiple hash chains use these seeds to generate authentication tokens. The authentication tokens are derived by hashing the seeds and the position of the hashing functions in the hash chains (e.g. via a concatenation process $\|$). Given the cryptographic hash function used is resistant to attacks (e.g. SHA-1, SHA-2 or SHA-3), a slight change in the argument to be hashed is expected to result in a significantly different output. Figure 5.1 provides a conceptual view of how the proposed protocol functions.
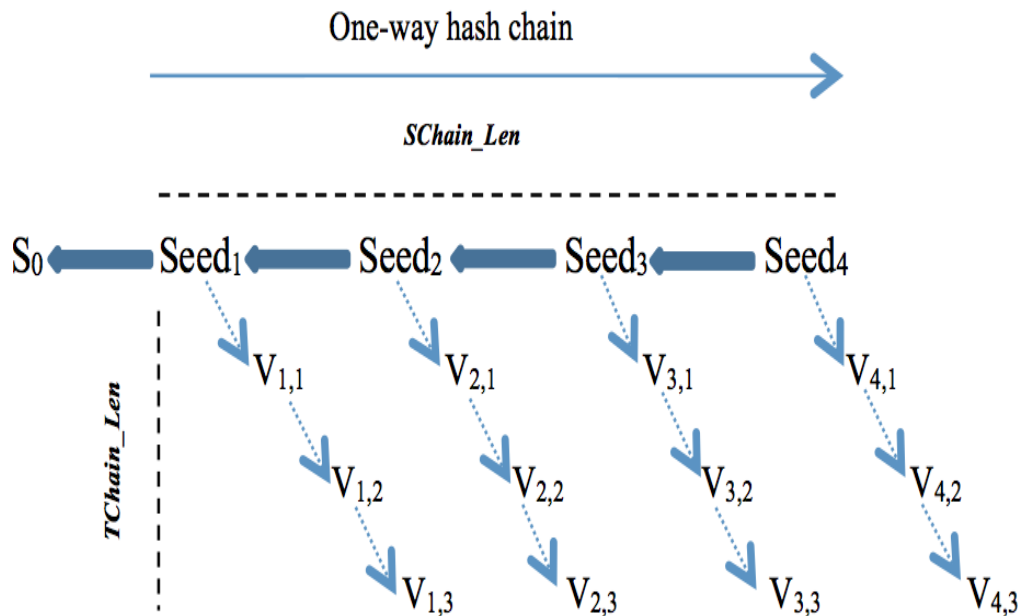
Figure 5-1 Position-indexed hashing for 12 transactions *TChain_Len*= 3, *SChain_Len*= 4

71

The proposed protocol is composed of three main stages: the *Initialization stage, Authenticate_Token stage* and *Next_Seed stage*. The notations we use in our scheme are summarized in Table 5.1.

1) Initialization stage

During the initialization stage, which is done using an HTTPS protocol, information about the session length (i.e., number of transactions N), an initial secret S_0 and *TChain_Len* is exchanged between the server and the client. Once this information is exchanged, the *SChain_Len* is determined by dividing N by *TChain_Len.* The result of this division will give us the number of seeds that will be needed during an internet session. Our definition of a session refers to the communication activities between the web application and the client during the login time (i.e. between log-in and log-out). A transaction on the other hand is a set of request and response between the web application and the client. The session is composed of N transactions. Information about the session length, *TChains* are predetermined and exchanged during this stage.

Table 5-1 Notations used in the proposed scheme

Notations	Description
N	Number of transactions to be handled during an internet session.
SChain	The chain where seeds are generated.
TChain	The chain where authentication tokens are generated.
S_0	The initial seed used by the *SChain*
SChain_Len	Length of the *SChain*.
TChain_Len	Length of the *TChain*.
H	Hash function used to generate seeds or authentication tokens.
V	Authentication token.

2) Authenticate_Token stage

The next stage *Authenticate_Token*, is where the authentication tokens are actually produced. The authentication tokens are denoted $V_{i,j}$ where the variable i represents the current *TChain* and j represents the current transaction number within the *TChain*. The tokens are created by hashing the seed concatenated || with a variable indicating the position of the hash function in the *TChain*. This position indexing technique is a well-known technique for boosting security because Birthday Attacks can be avoided if all hash functions used are indexed by their position in the chain [80]. As will be explained in the *Next_Seed stage,* we also update the seed several times during the session. The number of times the seed is updated depends on the number of transactions and the value of *TChain_Len.* This number is used to indicate how many *TChains*

we will have during the session. In other words, each updated seed is only used by the transactions of one *TChain* and then discarded and never used again.

3) Next_Seed stage

The third component of the protocol is the *Next_Seed* routine. This routine is responsible for updating the seeds used in the *TChains* to generate authentication tokens. It should be noted that each *TChain* has its own seed. This routine is invoked once the authentication tokens of the first *TChain* are created and transmitted. Based on the number of transactions and *TChain_Len* exchanged in the initialization stage, we know the number of times the seed is expected to be updated. The length of the seed chain, *SChain_Len,* is a result of dividing the number of transactions *N* by the value *TChain_Len.* Once the authentication tokens have all used a seed once (i.e. *TChain_Len* is exhausted), the *Next_Seed* routine is invoked to produce an updated seed for the next authentication token chain; *TChain*. We illustrate in the following section how our protocol works with a pseudo code and detailed examples. The performance evaluation results of the proposed scheme are presented in sections 5.3 and 5.4.

B. Selecting a Cryptographic Hash Function

A cryptographic hash function is an algorithm which changes a certain set of data into a string of a fixed size, called the block size. Examples of cryptographic hash functions include MD4, MD5, SHA-1 and SHA-2. It was proven that the MD5 hash function is prone to collision attacks [84] , [85] as well as pre-image attacks [86], and therefore, we did not consider it in our scheme. While SHA-1 is resistant to pre-image attacks, it was proven by [87] that it is theoretically prone to collision attacks. However, since it is not practically susceptible to collision attacks, we have used it in our protocol for the purpose of illustration.

In our implementation, the original block size is 160-bit corresponding to SHA-1, but it can easily be expanded to accommodate stronger cryptographic techniques that require larger block sizes such as SHA-2 (in all its sizes) and SHA-3 once it is released by NIST.

5.3   Case of Known Number of Transactions

Accurate statistics about network traffic related to social networking sites can be helpful in identifying the length of the one-way hash chain. However, it is not always the case that these are readily available. Dacosta et al [10] conducted basic traffic analysis of the social networking site "Facebook" and concluded that a typical session requires hundreds of transactions, and thus they set their chain length at 1000. In our study, we have varied this chain length since different social networking sites might have different requirements.   Following are the steps of the protocol when the number of transactions is known.

5.4   The Proposed Protocol's Steps

The *initialization* stage takes place using an HTTPS connection. During the HTTPS authentication, the initial value of the secret key S_0, the number of transactions N and the length of *TChain* (i.e. *TChain_Len*) are selected and exchanged between the server and the client. The following code is executed at both the client and the server sides.

```
SChain_Len:= N ÷ TChain_Len   // length of the SChain
I:= SChain_Len                // I is the global index for the SChain
J:= TChain_Len                // J is the global index for the TChain
Seed:= H^I(S_0)               // Seed is now Seed_1=seed for the first TChain
```

The routine *Authenticate_Token* is executed once for each transaction to compute the authentication tokens that will be transmitted with the transaction cookie.

Authenticate_Token(Seed, J)

Begin

$V := HJ(Seed||J)$ // *J* is the global index for the *TChain* where *||* is a concatenation of

the seed   with the hash function position in the chain

$J := J - 1$

if (*J*==0) **then**   // *TChain* length is exhausted

  $Seed := Call\ Next\_Seed(\ )$ // *Seed* has to be updated

  $J := TChain\_Len$ // *TChain* length is reset

end-if

Return (*V*)

End

Next_Seed( )

Begin

$I := I-1$ // *I* is the global index for the *SChain*

$Seed := H^I(S_0)$ // updating the *Seed* value

Return (*Seed*)

End;

Let us now illustrate how the protocol works with an example. In case the number of transactions is known to be *N* =200, and the *TChain_Len* =4, the seed is going to be updated 50 times (i.e., *SChain_Len = 50*) to carry out the hashing functions for 200 transactions.

We have I=50,   J=4, $Seed_1$= $H^{50}(S_0)$

The first *TChain* of four transactions will create the following authentication tokens.

$V_{1,1} = H^4(Seed_1||4)$

$V_{1,2} = H^3(Seed_1||3)$

$V_{1,3} = H^2(Seed_1||2)$

$V_{1,4} = H^1(Seed_1||1)$

Once these authentication tokens have been transmitted, the *Seed* has to be updated to $Seed_2 = H^{49}(S_0)$ and *J* has to be reset to 4.

The next step is to generate the second set of four transactions which will be:

$V_{2,1} = H^4(Seed_2 \| 4)$

……..

$V_{2,4} = H^1(Seed_2 \| 1)$

The code continues to calculate the authentication tokens in each *TChain* until we reach the 50th *TChain*. The 50th *TChain* will have the $Seed_{50} = H(S_0)$ and its authentication tokens will be:

$V_{50,1} = H^4(Seed_{50} \| 4)$

……..

$V_{50,4} = H^1(Seed_{50} \| 1)$

5.5   Protocol Evaluation

1) The Testbed

In this section, we present the protocol evaluation results when the number of transactions in a session is known. We developed a detailed benchmark in Java which allowed us to test different session scenarios. An important metric used in our tests is *SessionCost* which is the total number of hash operations performed during the lifetime of the session. The metric *SessionCost* represents the overall execution overhead of the protocol including the overhead of the *Initialization stage* and the overhead of the *Authenticate_Token* routine for all transactions as well as the overhead of the *Next_Seed* routine.

Figure 5.2 shows the performance of the protocol for different values of the number of transactions *N* and the length of *TChain.* It is interesting to see that the value of *SessionCost*

decreases as the value of *TChain_Len* increases until a certain point then starts to increase again. For each value of *N*, there is a certain value of *TChain_Len* that minimizes the value of *SessionCost*. We validate this behavior by an analytical model.
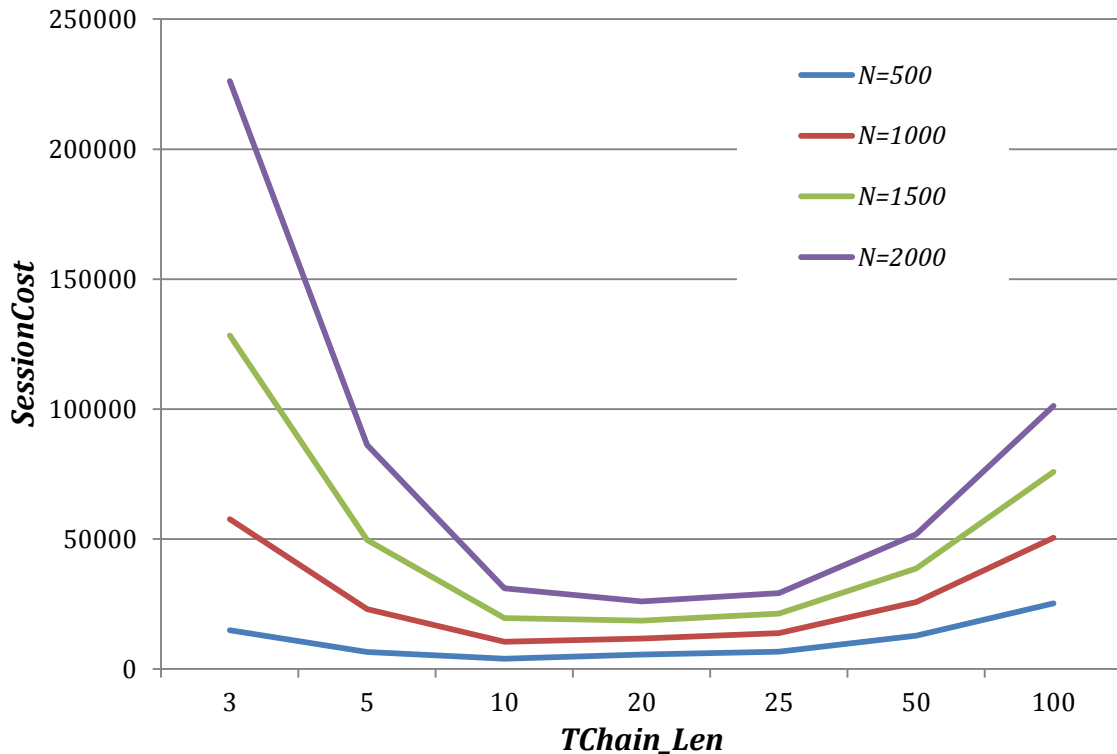
Figure 5-2 Protocol Evaluation (known number of transactions)

2) Analytical Model

As we described earlier our protocol is composed of two chains: i) the seed generating chain *SChain* represented by the horizontal axis in Figure 5.1, and ii) the authentication generating chain *TChain* represented by the vertical (slanted) axis in Fig 5.1. For simplicity, we assume that the cost of a single hash operation used in the *SChain* and *TChain* is the same because they both use the same hashing algorithm (i.e. SHA-1); we will examine this assumption later at the end of this section. The cost of a single session *SessionCost= C* is the sum of the hashing operations required to generate authentication tokens in the vertical chains, C_V, and the

hashing operations required to update the seeds in the horizontal chain, C_H. Here is how

SessionCost is calculated.

N= number of transactions

M= *SChain_Len*

K= *TChain_Len*

Cost of one vertical chain $= \frac{K(K+1)}{2}$ $\qquad$ (5.1)

$$C_V = \frac{MK(K+1)}{2} = \frac{N(K+1)}{2} \qquad (5.2)$$

$$C_H = \frac{M(M+1)}{2} \qquad (5.3)$$

$$C = C_V + C_H$$

$$= \frac{K(K+1)}{2} + \frac{M(M+1)}{2} \qquad (5.4)$$

$$C = \frac{N(K+1)}{2} + \frac{M(M+1)}{2} = \frac{NK}{2} + \frac{N}{2} + \frac{M^2}{2} + \frac{M}{2} \qquad (5.5)$$

The next formula can be used to plot C as a function of N and K.

$$C = \frac{NK}{2} + \frac{N}{2} + \frac{N^2}{2K^2} + \frac{N}{2K} = (K + 1 + \frac{N}{K^2} + \frac{1}{K}) \qquad (5.6)$$

To find the optimal value of K which minimizes the cost C, we differentiate formula 5.6

and equate to 0

$$\frac{\partial C}{\partial K} = \frac{N}{2}(1 - \frac{2N}{K^3} - \frac{1}{K^2}) \qquad (5.7)$$

$$\left(1 - \frac{2N}{K^3} - \frac{1}{K^2}\right) = 0 \qquad (5.8)$$

$$K^3 - K - 2N = 0 \qquad (5.9)$$

Equation 5.9 can be used to derive the optimal value of *K* which corresponds to *TChain_Len*. Table 5.2 gives the optimal value of *TChain_Len* obtained by solving the above cubic equation numerically.

Table 5-2 *TChain_Len* OPTIMAL VALUE

Number of Transactions	Optimal *TChain_Len*
500	10.03
1000	12.625
1500	14.445
2000	15.895

Comparing Table 5.2 with Figure 5.2, we can see that the *TChain* optimal values which we obtained from the simulation are very close to the optimal values obtained from the analytical solution.

It should be mentioned that the above analytical solution was derived based on the assumption that the hash operation used in the *SChain* and *TChain* have the same cost because both use the same hash function SHA-1. A more accurate model can be easily developed to account for the extra overhead of the concatenation operation used in the *SChain*. Since the cost of the concatenation operation is much smaller than the cost of the hash operation, the slight increase due to concatenation can be accurately modeled by multiplying the cost of C_H by a factor r which is slightly larger than 1. This will cause Equation 2 to be slightly modified as follows: the second and third terms will be multiplied by the factor r. Solving this modified equation gives optimal values very close to those given in Table 5.2.

5.6   Protocol Comparison with OHC

In order to validate the efficiency of our protocol, we compared its performance with the OHC protocol proposed in [10].
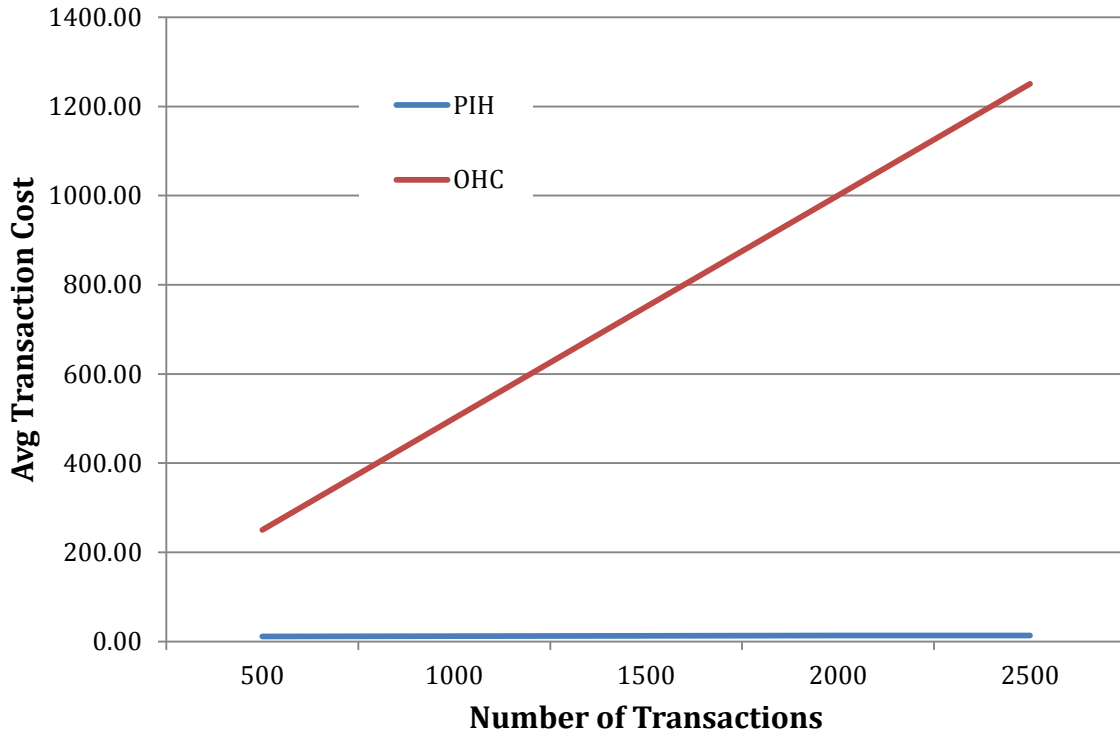
Figure 5-3 Performance comparison between position-index hashing (PIH) and OHC

In Figure 5.3, we demonstrate the performance comparison between our protocol and the OHC protocol. We compared the average cost of a single transaction in both protocols. In our protocol, we were able to lower the average cost of the transaction tremendously. For instance, for 500 transactions the average transaction cost of the OHC protocol is 250, whereas our protocol lowers this average to a little over 11. While this average tends to increase significantly with the increase in transaction numbers for the OHC protocol, our average stays relatively low even with N= 2500 transactions where the average cost is 13.65.

In order to further gain insight in the benefits of adopting our protocol, we measured the performance improvement ratio when our protocol is chosen over the OHC. The performance improvement ratio is defined as the ratio *SessionCost* of OHC : *SessionCost* of our PIH protocol. Figure 5.4 displays the significant improvement our protocol achieves over OHC.
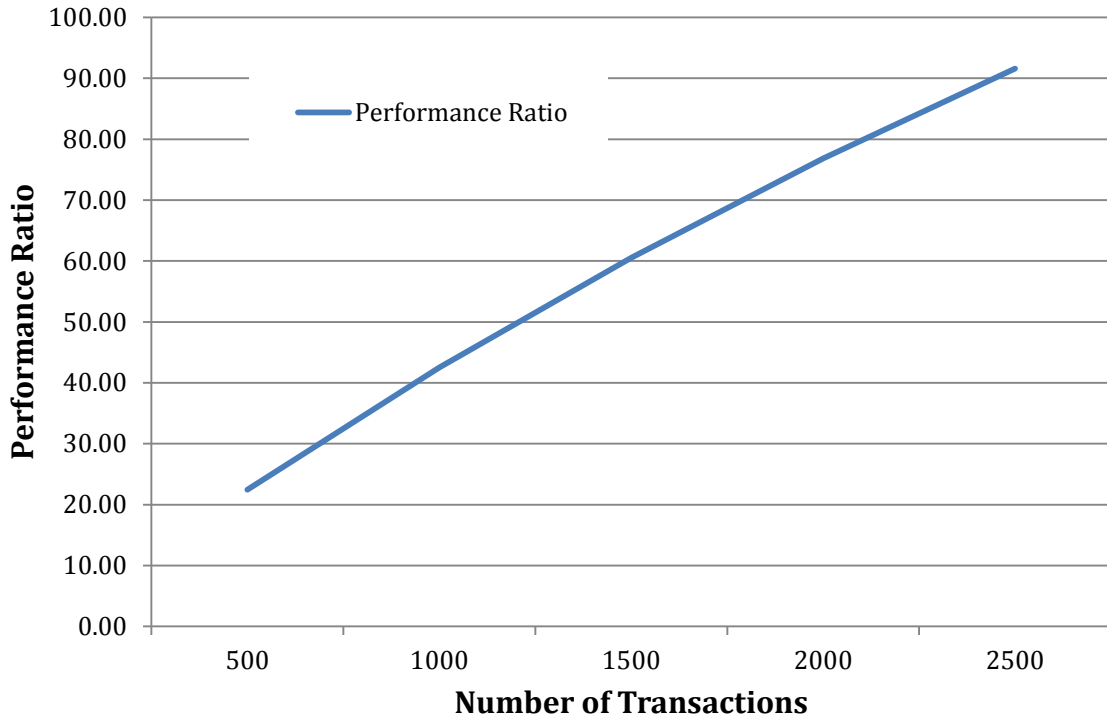
Figure 5-4 Performance improvement ratio of OHC

Looking at Figure 5.4, we can easily gain insight on how choosing our protocol is beneficial. Our protocol outperforms the OHC protocol by a little over 22 times when the number of transactions is 500. This improvement ratio is much higher with higher transaction numbers. When we have 2500 transactions to be handled in a session, our protocol outperforms the OHC by over 91 times. This is a relatively wide margin and makes our protocol plausible. The ratio is expected to be higher for longer sessions with higher transaction numbers.

5.7   Unknown Number of Transactions

A. Certainty versus Uncertainty in Transaction Number

In designing the protocol, we took into consideration the possibility of certain versus uncertain number of transactions. More often than not, it is very hard to estimate the exact number of transactions to be handled in a single session in a social networking site. As we have seen above in our discussion of Facebook session length statistics introduced in [10], the length can range from a few hundred transactions to several hundreds. This has led us to devise two different versions of the position-indexed hashing protocol to accommodate the two scenarios: *known number of transactions* and *unknown number of transactions*.

We have already accounted for the case of known number of transactions in the previous section, and this section is devoted to explicating the case of unknown number of transactions. When the number of transactions is unknown, there is no way to calculate the *SChain_Len* and hence the number of *TChains* in a session. Therefore, we needed to change the code slightly to account for this discrepancy. During the *Initialization stage* instead of exchanging the number of transactions *N* and the *TChain_Len*, the client and server exchange the *TChain_Len* and another value representing the *SChain_Len*. The importance of *SChain_Len* specification comes from the need to update the seed during the session multiple times. Since the transaction number is unknown, we have no way of determining how many times the seed is going to be updated. Given this scenario, we are faced with another problem. If the specified *SChain_Len* is not long enough (i.e., the actual number of times we will have to update the seed is more than the value of *SChain_Len*), we will need to repeat using one or more seeds, which could compromise the security of the session. To solve this problem, we utilize the number of *TChains* in a session as an index to be attached to the updated seed via a concatenation process ||. The index for *TChain*

number can be derived from the *TChain_Len* and the *Next_Seed* routine. The first *TChain* in a session is the one that uses the first seed and once the *Next_Seed* routine is invoked, the *TChain* index is incremented and the new value is attached to the hashed seed.

B. The Modified Protocol

Here, we introduce how we modified the protocol to account for the case of unknown number of transactions. We still have the three stages we had in the unknown number of transactions case.

1) Initialization

The initialization stage takes place using an HTTPS connection. During the HTTPS authentication, the initial value of the secret key S_0, the length of the authenticate-token chain *TChain_Len* and the length of the next-seed chain *SChain_Len* are selected and exchanged between the server and the client. The following code is executed at both the client and the server sides.

```
I:= SChain_Len          // I is the global index for the SChain
J:= TChain_Len          //J is the global index for the TChain
index:= 1               //a global variable indicating TChain
                          number where 1 refers to first TChain
Seed:= H^I(S_0||index)   // Seed is now Seed_1=seed for the first
                          TChain
```

```
Authenticate_Token(Seed, J)
Begin
V:= H^J(Seed||J )                    // J is the global index for the TChain
J:= J -1
if (J==0) then                       // TChain_Len length is exhausted
    index:= index + 1                // index incremented for the next TChain
```

84

Seed:= *Call Next_Seed( )*   // *Seed* has to be updated

J:= *TChain_Len*   // *TChain* length is reset

end-if

Return (*V*)

End

Next_Seed( )

Begin

I:= I-1

if (*I*==0) **then**   // *SChain* length is exhausted

  I:= SChain_Len   // *I* is reset to *SChain_Len*

end-if

Seed:= $H^I(S_0 || index)$;

Return (*Seed*)

End

Here is an example to help illustrate how the protocol handles the transmission of authentication tokens when the number of transactions is unknown. During initialization, the values of *TChain_Len*=4 and *SChain_Len*=10 will be selected and exchanged between the server and client.

We have *I*=10, *J*=4,   *index*=1. We assign an index which represents the first *TChain*, and therefore the first seed will be:

$Seed_1 = H^{10}(S_0 || 1)$

The first *TChain* will use $Seed_1$ in the hashing function to derive the first set of authentication tokens as follows…

$V_{1,1} = H^4(Seed_1 || 4)$

$V_{1,2} = H^3(Seed_1 || 3)$

$V_{1,3} = H^2(Seed_1 || 2)$

$V_{1,4} = H^1(Seed_1 \| 1)$

Now *index* becomes 2 which represents the second *TChain*, $I = 9$, *Seed* has to be updated to $Seed_2 = H^9(S_0 \| 2)$ and J has to be reset to 4. The second *TChain* will have the following authentication tokens:

$V_{2,1} = H^4(Seed_2 \| 4)$

……..

$V_{2,4} = H^1(Seed_2 \| 1)$

After finishing ten *TChains* (i.e. 40 transactions) *index* becomes 11 which represents the 11th *TChain*, *I* has to be reset to 10, *Seed* has to be updated to $Seed_{11} = H^{10}(S_0 \| 11)$ and *J* has to be also reset to 4. If we did not use the *TChain* number as a value attached to S_0, we would have been forced to recycle $Seed_1$ as $Seed_{11} = Seed_1 = H^{10}(S_0)$. This could potentially compromise our protocol as it becomes easier to detect the initial seed. By indexing the *TChain* number and using its value in the hashing function, we are able to solve this problem.

Therefore, the 11th *TChain* (11th set of four transactions) will have authentication tokens which will be:

$V_{11,1} = H^4(Seed_{11} \| 4)$

……..

$V_{11,4} = H^1(Seed_{11} \| 1)$

The protocol goes on according to this routine until the user or the server terminates the session.

C. Protocol Evaluation (unknown number of transactions)

In this section, we present the evaluation of our protocol when the number of transactions is unknown. Figure 5.5 illustrates the results when the *SChain_Len* is fixed at 5, while Figure 5.6 demonstrates the protocol's performance when the *SChain_Len* is fixed at 20. Our main goal from this is to determine the best value of *TChain_Len* where the protocol performs relatively well.

In both Figure 5.5 and Figure 5.6, regardless of the *SChain_Len*, the protocol seems to perform well when the *TChain_Len* is set at 3. Unlike the case of known number of transactions where we noticed some kind of correlation between *TChain_Len* and performance, in the case of unknown number of transactions it is better to set *TChain_Len* at a relatively low value.
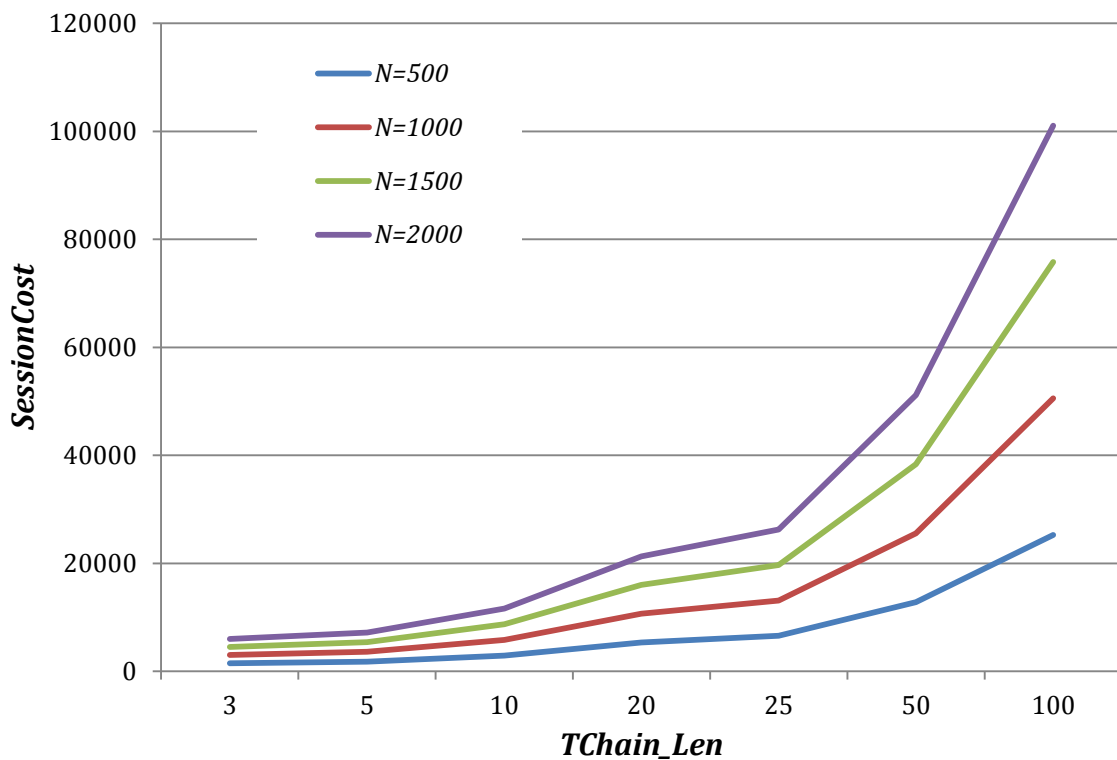
Figure 5-5 Protocol Evaluation (Unknown number of transactions) *SChain_Len= 5 all the time*
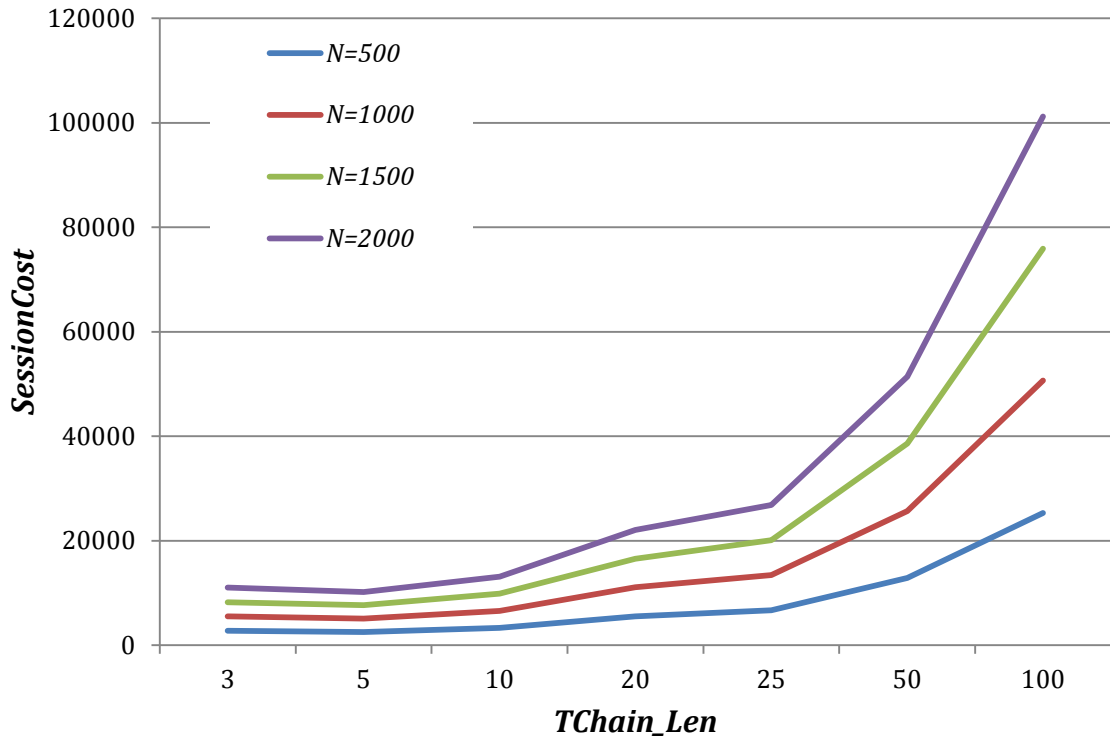
Figure 5-6 Protocol Evaluation (Unknown number of transactions) *SChain_Len= 20 all the time*

Our next task was to see what the best value of *SChain_Len* is when the *TChain_Len* is fixed at 3. The next graph (Figure 5.7) represents the session cost of different number of transactions with *TChain_Len*= 3 and different *SChain_Len* values.
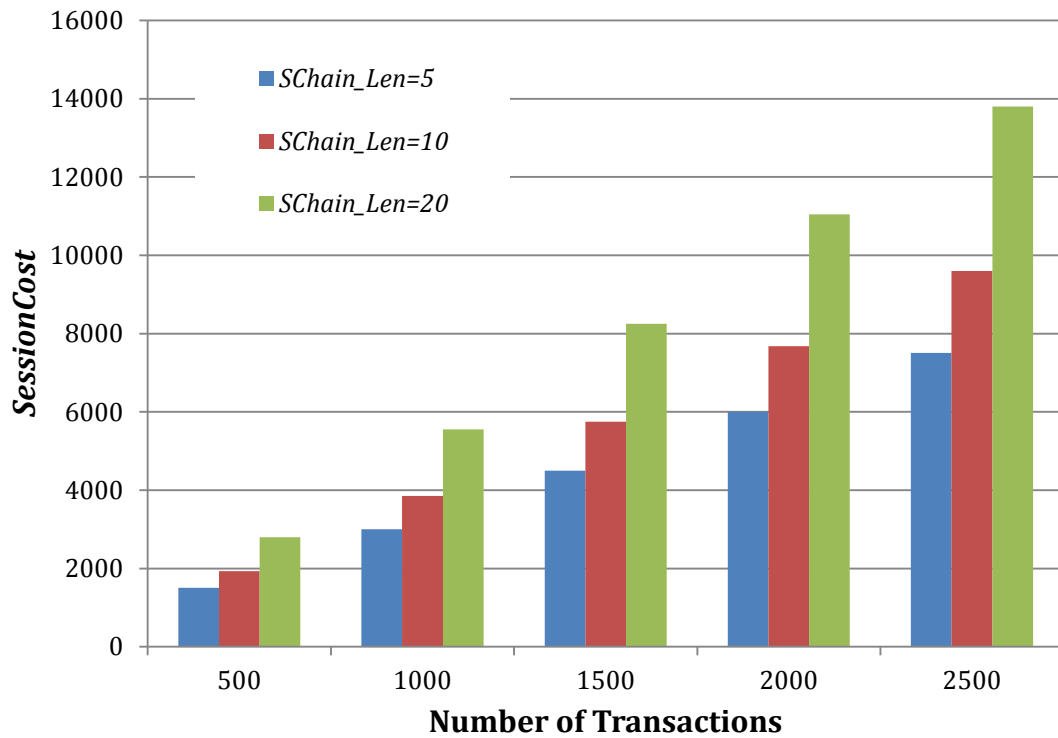
Figure 5-7 Protocol performance comparison when *TChain_Len*=3 and different *SChain_Len*

Figure 5.7 indicates that there is a steady and direct relationship between *SChain_Len* and performance measured in *SessionCost*. The lower the value *SChain_Len*, the better performance we can achieve when we do not know the number of transactions during a session. In other words, we need to start with relatively short chains in both the authenticate-token chain *TChain* and the next-seed chain *SChain*.

5.8   Conclusion

One-way hash chains can be efficiently used in collaborative and social media networks to overcome the problem of session hijacking in Internet sessions caused by stealing cookies. In this chapter, we proposed a one-way hash chain protocol to address the problem of overestimating the number of transactions during a session in the straightforwardly configured

one-way hash chains. Our solution achieves its goal by utilizing two one-way hash chains; one is responsible for updating the secret and the other for creating the authentication tokens attached to the cookies using the secrets produced by the first chain. We also employ the position of the hashing function in the chain in order to strengthen our protocol against attacks such as Birthday attacks.

Our extensive evaluation of the protocol and comparison with other protocols yielded encouraging results. We have been able to improve the performance of one-way hash chains significantly while keeping the same levels of security.

CHAPTER 6: HACH FOR WIRELESS SENSOR NETWORKS

6.1   Introduction

Different aspects of WSN security have been addressed in the literature. Among the problems are user authentication [45], [46], [48], [49], and broadcast authentication [47], [88]. WSN security solutions have a general goal of improving WSN standards in terms of *authenticity, confidentiality* and *integrity*. Conventional user and broadcast authentication mechanisms are not as practical in WSNs due to the scarcity of memory and energy capacities of sensor nodes.

Conversely, one of the best solutions for user and broadcast authentication is the use of public-keys to secure communication. For example, Benenson et al. proposed a solution in which user authentication is achieved by adopting public-key cryptography [45]. They basically use a certificate/signature generated by a base station. What renders this technique impractical for WSN is the high computation cost and large signature size. In addition, they make WSN susceptible to DOS attacks draining energy resources. To remedy this problem, shortened public-key based solutions designed for WSNs have been suggested. Addressing the problem of expensive user authentication if public-keys are used, Wang et al. suggested using a short public key whose lifetime is much shorter than regular public keys [47]. This solution aims at reducing computation, but it still suffers from vulnerability to DOS attacks.

The HACH protocol has been considered by several researchers as an efficient mechanism to achieve user authentication in wireless sensor networks. One of the seminal papers to take this route is the μTESLA scheme introduced in [89] which aimed at achieving broadcast authentication. In μTESLA, the basic idea is to have a base station, assumed to be trusted all the time, that acts as a user of sensor nodes services, i.e. the base station is where sensed information

is collected and authenticated. Authentication is achieved by using one-way hash function $h()$ and using the hash preimages as keys in the Message Authentication Code (MAC). In the initialization stage, the base station calculates a hashed value of a secret x based on the number of nodes in the network and distributes it among relative sensor nodes in a unicast fashion. In the subsequent transactions, a preimage of the hashed value is used as a key in MAC. In the next stage, the sensors verify whether or not the hashed value and its preimage are consistent. The problem with this technique is low scalability and the high computational cost due the unicast nature of key distribution. To further improve μTESLA, Liu and Ning [90] suggested replacing the unicast technique for distributing keys with a broadcast based technology.

One-way hash based solutions have also been attractive in user authentication schemes of WSNs. One of the earliest implementations of user authentication employing one-way hash is introduced in [91]. The one-way hash operation is used in the initial stages to verify the user requesting access to the WSN is an authorized user. Subsequent user authentication schemes which try to overcome the shortcoming of [91] include [92], [93] and [94]. Unlike the one-way hash chains used in broadcast authentication where computational overhead is an issue of concern, user authentication schemes do not suffer from such overhead since we only need one-way hash operations in the login stages.

In this chapter, we propose a novel scheme where the HACH approach is extended to achieve broadcast authentication in addition to user authentication in WSN's. The proposed scheme uses a mini one-way hash chain protocol for user and broadcast authentication that is as effective, while at the same time significantly reducing the computational overhead. We avoid the potential security breach by using easily computed one time authentication tokens to secure communication. Similar to the previous contributions, the proposed protocol is energy

preserving, light and efficient. Our simulation and evaluation tests reflect its benefit over straightforwardly configured one-way hash chains.

6.2   Network Model

For our protocol, we envision a star topology network where communication between the user and the WSN is mediated by a central entity, called *coordinator*. Essentially, the *coordinator* is a sensor node with special capabilities. The role of the coordinator is to manage communication in the WSN by receiving queries from the users, communicating with sensor nodes to get the sensed information, and fetching the results of users' queries back to the user. With this in mind, we assume the coordinator to be responsible for authentication in both directions. In the user's direction, the coordinator is expected to only allow identified users holding the right credentials to access the network. In the sensor nodes direction, on the other hand, the coordinator(s) delegates only authenticated sensors that hold the correct authentication tokens at the time of the communication.
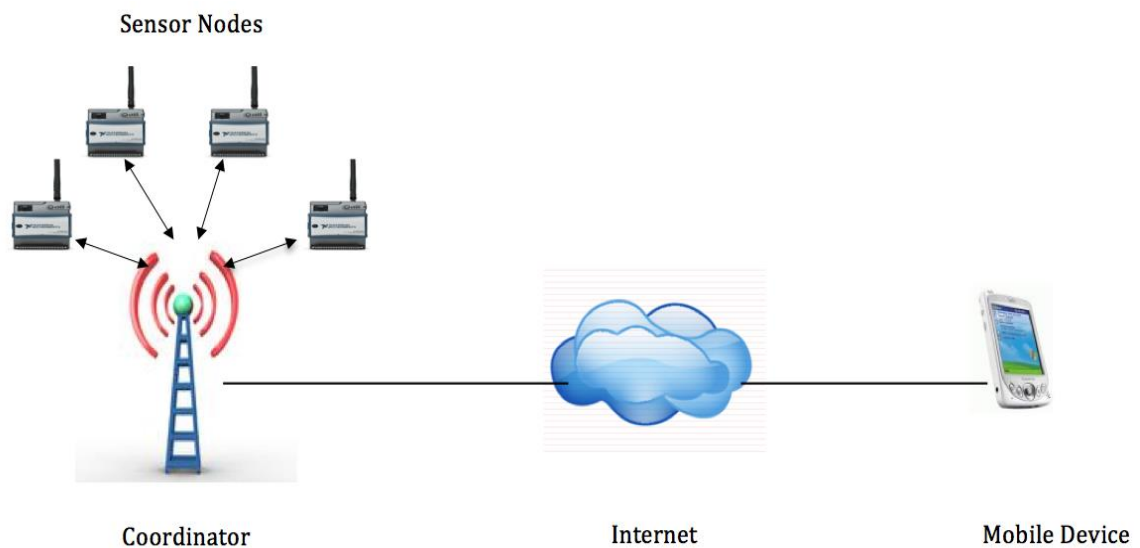
Figure 6-1 Network Model External Structure

93

Figure 6.1 captures the main design aspects of the network external structure. Such network is ideal in a medical or industry setting where the proximity of the sensor nodes makes it almost impossible for physical capture of sensor nodes to occur. Moreover, given coordinators are administered by network administrators who assign users such as physicians, nurses, quality control workers, etc. access to the WSN, we can make sure that only authorized personnel can access the network.

6.3   The Proposed MOHC Authentication Scheme

In this section, we introduce the formal description of our mini OHC (MOHC) authentication scheme. There are three components of the proposed solution: the user's mobile device, the coordinator CO, and the sensor nodes SN. The user's mobile device can be configured by the network administrator and is equipped with network access capabilities in the form of a username and a password. The user is anyone who is allowed to have access to the data (e.g. in the case of WSN used in the medical field, it could be the physician, the patient or the nurse etc.).

The network communication scenario consists of the following steps:

1) The network administrator provides WSN users with temporary login credentials that are changed by the respective users after initial login. This step ensures that only authorized personnel are allowed to have access to the WSN. By changing the password, we make sure that administrators' role is just to establish service.

2) Once logged in, the user communicates with the coordinator by sending a query asking for data from the sensor nodes.

3) The coordinator sends requests to the respective sensor nodes.

4) Once gathered, the sensor node(s) delivers the data to the coordinator who checks that each replying sensor has the correct credentials, accepting data if this condition holds, otherwise, data will be denied.

5) The coordinator replies to the user with the requested data.

6) Steps 2-5 are repeated for each query generated by the user in the current user session.

With this communication scenario in mind, we designed the protocol to achieve user and broadcast authentication throughout the WSN session. Therefore, the protocol works in two directions: the user direction and the SNs direction. In the user direction, once successfully logged in to the network, login credentials will be substituted by a secret value (*Secret*), which will be generated by the coordinator. This *Secret* automatically expires once the user logs out. Each time the user logs in, there will be a new generated *Secret*. This secret will be used to generate *authentication tokens* used to secure subsequent communication. The first authentication token V_1 will be the result of applying a hash function *h(.)* to the *Secret//C,* where *C* is the current number of communications between CO and the user; initially *C=0*. Thus, the first authentication token will have the value $V_1 := H^K(Secret|| C)$, where *K,* the length of the hash chain, is decremented after each iteration. Before each communication between the coordinator and the user, the *C* index is incremented. In the subsequent transactions, and to avoid exposing the *Secret* and potentially compromising the WSN, we replace the *Secret* in the hash function with the preceding authentication token used in the previous communication. Thus, the second authentication token *V* has the value $V_2 := H^{K-1}(V_1|| C+1)$, then V_1 is disposed, and V_2 is used instead to generate the third authentication token which is $V_3 := H^{K-2}(V_2|| C+2)$ and so on. The length of the chain *K* used in our implementation is 10. In [13], we proposed a scheme where one-way hash chains are divided into smaller hash chains. The best length of a chain striking a

good balance in terms of hash count falls between 10 and 25. Each time the chain length is exhausted, the index K is reset.

In the sensor nodes direction, the protocol works a little differently. It actually works in phases: the first phase occurs during deployment when sensor nodes identify themselves to the coordinator. The coordinator subsequently distributes $Seed_0$ among the sensor nodes which will be used in a hash function at the sensor node's side. The value of $Seed_0$ is never exposed, but used in a way similar to what we did with the *Secret* for the user authentication; i.e., as input for a hash function *h(.)* to generate subsequent *authentication tokens*. The second phase is when the actual communication begins when the coordinator relays a user's query to SNs. In the first round of communication, we use $Seed_0$ to generate the first *authentication token* at the sensor node's side $V_1 := H^K(Seed_0 \| C_i)$, where K is the hash chain length, decremented after each round, and C_i is the number of communication rounds between sensor_node [i] and the coordinator; initially $C_i = 0$. The value of C_i is incremented before each round of communication. When C_i reaches a certain value, *C_limit*, we reset its value to 0 to avoid large numbers, and also update and broadcast $Seed_0$ to sensor_node [i]. In the subsequent rounds, and to avoid exposing of $Seed_0$ and potentially compromising the WSN, we replace $Seed_0$ in the hash function with the preceding authentication token. Thus, the second authentication token will be $V_2 := H^{K-1}(V_1 \| C_i +1)$, then, V_1 is discarded. The third authentication token will be $V_3 := H^{K-2}(V_2 \| C_i +2)$, V_2 is discarded, and so on. Once the hash chain length is exhausted, the index K is reset. Below a high level pseudo code of the authentication token scheme described in this section is presented.

<u>Authentication token</u>

Phase 1: Deployment Phase

<u>Input:</u> number of sensor nodes

Coordinator creates $Seed_0$

Attach $Seed_0$ to each Sensor node

Set number of communications C_i between Sensor node [i] and coordinator to 0

Phase 2: Operational Phase

<u>Input:</u> sensor_node [i] or coordinator

Increment C_i

If Authentication token in sensor_node [i] or coordinator is the first authentication

   K:= 10

  Create 1st token $V := H^K(Seed_0 \parallel C_i)$

Else

  Prev_V= V

  $V := H^K(Prev\_V \parallel C_i)$, End if

Decrement K

If K is equal to 0

  K:= 10; End_if

If C_i is equal to *C_limit*

  Set C_i to 0

  If input is coordinator

  Create random $Seed_0$

  Broadcast $Seed_0$ to sensor_node [i]

  End If

End If

Return (V)

The above description is for the protocol when utilized in lenient environments. We also developed the code with the potential of being deployed in harsh environments where more stringent authentication is desired. We achieve this functionality by determining and presetting an expiration to the *Seed*. The expiration could be as simple as a period of time, or after a certain number of rounds of communication between the coordinator and sensor node. Once the *Seed* is updated, the communication count has to be reset to 0.

6.4 Simulation and Evaluation

In order to evaluate the performance of our proposed scheme, we designed a testbed where the network model described in section 6.2 is depicted. The whole network structure has been translated in a Java code and run without the authentication protocol. The performance of the simulated WSN is measured and reported. To test how our scheme impacts the simulated WSN network, we also wrote a Java code implementing the authentication protocol described in section 6.3. As a benchmark, we also simulated the authentication scheme if a straightforward OHC was used. A comparison between these scenarios is made and will be presented in the next section.

In our simulation, the design consisted of two configurations: serial and parallel communication. One possible example where the serial mode can be found is in industry/manufacturing quality control where a product stays on the assembly line for a period of time and moves from one point to another for different phases of processing. In each phase, there is a sensor to measure the product temperature and report it to the coordinator. For that purpose, a set of serial transactions is useful. In this configuration, the coordinator sends a request message to the first sensor node, and receives a response message from it. Then, the coordinator sends a request message to the second sensor node, and receives a response message from it, and

so on until all sensors reply with their data. The coordinator, then, sends the response message to the mobile device or the quality control manager. In the parallel mode, the coordinator sends a request message to all sensor nodes at the same time by creating multiple threads. After all response messages are received, it sends a response message to the mobile device. An example where parallel configuration is ideal is a medical setting where a physician, for instance, needs to monitor a patient's electrocardiogram (EKG) by placing the sensor nodes around the body. Thus, multiple threads are initiated between the coordinator and sensor nodes.

To measure the effectiveness of our proposed WSN authentication scheme, we used the following performance metrics in our simulations:

- Time delay: the time was measured based on rounds of communication between the coordinator and N sensor nodes. Figure 6.2 shows one round of communication for N sensors in the serial mode. For example a round of communication between the coordinator and sensor node 1 is the difference between the time that coordinator sends a request message to sensor node (t_1) and receives a response from the sensor node t_2. We measured the WSN performance, in the serial and parallel modes, once without the proposed authentication scheme and another time when the authentication scheme is plugged either in the OHC or our MOHC format.
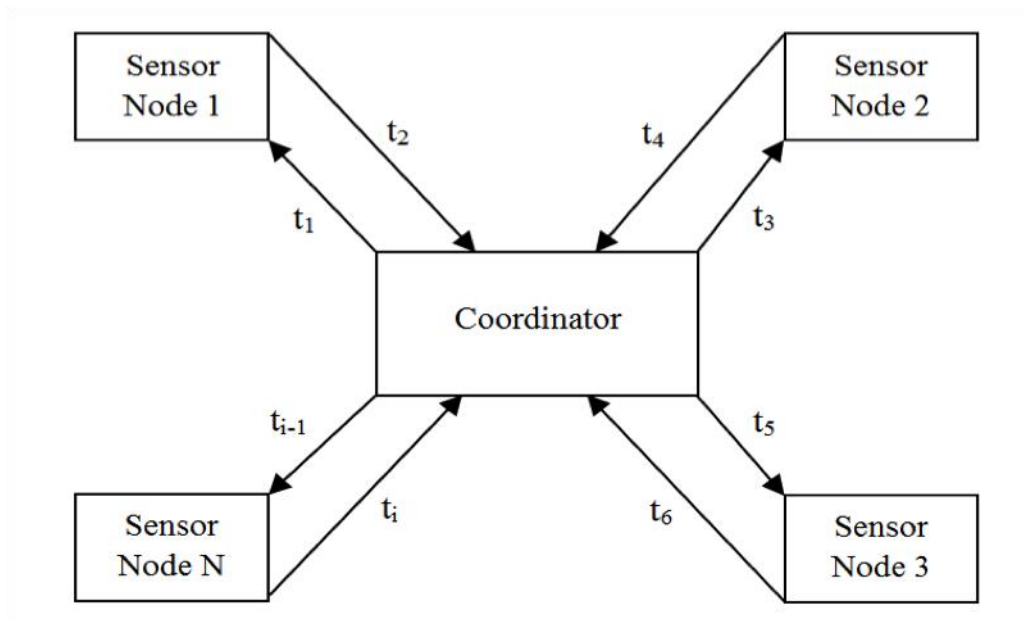
Figure 6-2 Example of time calculation for serial mode

- Average number of hashes: the average number of hash operations performed during a round of communication. Given the fact that each component of the WSN (coordinator and sensor nodes) is going to calculate the same number of hashes in each round of communication, we used an average number of hashes for each of these components and presented comparisons based on them.

6.5   Performance and Results

In this section, we present the performance evaluation results of our protocol. Let us first look at how the MOHC protocol performed when compared with the case of no authentication at all, and with the straightforward OHC. Table 6.1 illustrates the difference in milliseconds between these three cases in the serial mode. Here, we compared the time in the case of no authentication, straight forward OHC and our protocol based on the number of rounds of communication between the coordinator and the sensor nodes. In this case, the coordinator sends a query to the first sensor node and waits for its response before sending the next request to the

second sensor node. The results indicate that adding authentication increases the time. This increase tends to correlate linearly with the duration of the WSN session which is not unexpected. However, our focus was to compare our idea of mini hash chains on the performance of WSN with the straightforward implementation of OHC. When the number of rounds of communication is small, as in the case of 50 rounds, it can be seen that the difference between OHC and MOHC protocol is approximately 2 milliseconds. However, this difference becomes more salient with higher rounds of communication numbers. In real WSN traffic, communication is expected to be higher, and, therefore, rounds of communication number increases, which makes our solutions an obvious better choice as it outperforms its counterpart by a margin.

Table 6-1 Time comparisons showing equal execution overheads for MOHC and OHC (Serial Mode)

Rounds of communication	No-Authentication[*]	OHC[*]	MOHC[*]
50	1192	1201	1199
150	2513	2531	2526
300	5011	5047	5029
600	9767	9874	9793

* In milliseconds

Similarly, Table 6.2 summarizes the performance comparison in the parallel mode. As described above, the coordinator initiates multiple threads to send a request message to all sensor nodes. The response messages are received in no particular order from the sensor nodes. Here, we can see the time required to run the simulation without authentication is much less. However, the time required for authentication, either in the serial or parallel mode, is approximately the

same. What is important for our comparison is the time difference between OHC and our

MOHC. The difference in time shows our protocol still outperforms the OHC as the table shows.

Table 6-2 Time comparisons showing equal execution overheads for MHOC and OHC
(Parallel Mode)

Rounds of communication	No-Authentication*	OHC*	MOHC*
50	297	306	304
150	939	957	951
300	1545	1581	1563
600	3104	3211	3131

* In milliseconds
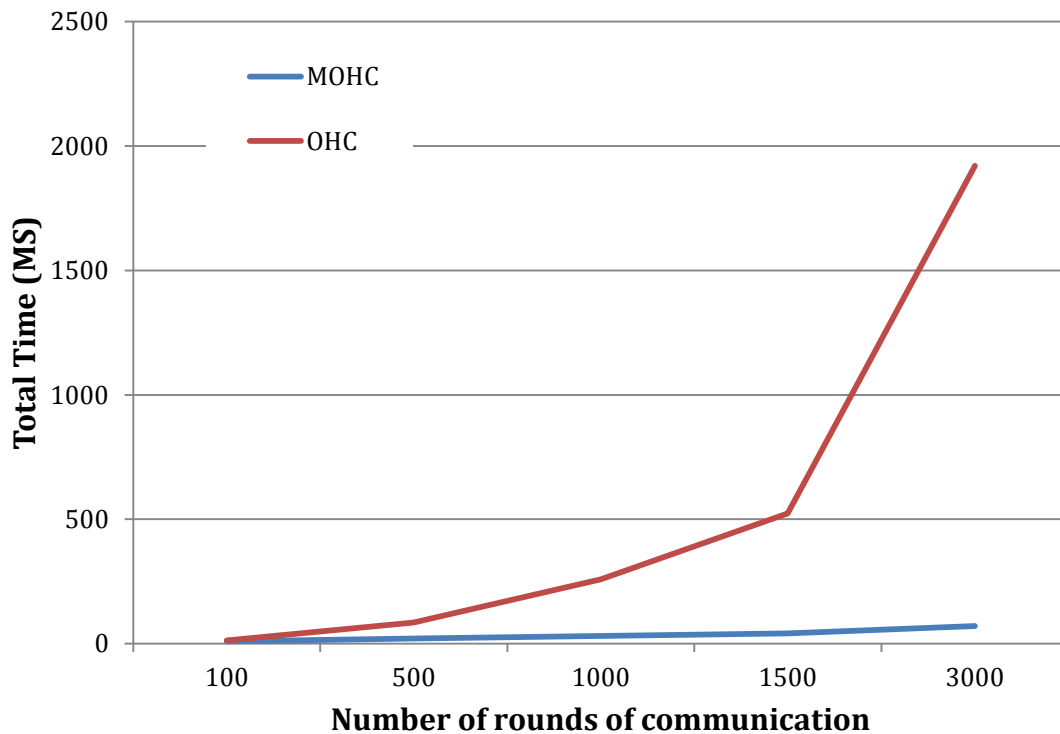
Figure 6-3 Time comparisons between OHC and MOHC protocol with higher rounds of
communication (Serial Mode).

In order to further demonstrate the improvement we achieved in our protocol over OHC, we measured the time difference between the performance of the two schemes and present it in Figure 6.3. This figure represents the performance in the serial mode.

As can be seen Figure 6.3, the higher the number of rounds of communication, the wider the difference margin is. This is another indication that using our MOHC protocol definitely improves performance and makes deploying a one-way hash chain based protocol more attractive in WSN.

When the average number of hashes required for a round of communication is compared as in Figure 6.4, we still can see how our MOHC scheme outperforms the traditional OHC. Similar to the time delay compared above, as the number of rounds of communication increases, so does the average hash count. This is indicative of the high computation cost of the one-way hash chain schemes if configured as is. With minor modifications, like the ones we have proposed in this chapter, we were able to achieve noticeable improvements. It should be noted though that regardless of configuration options, the number of hashes is going to be the same. Each component of the WSN will calculate the same number of hashes.
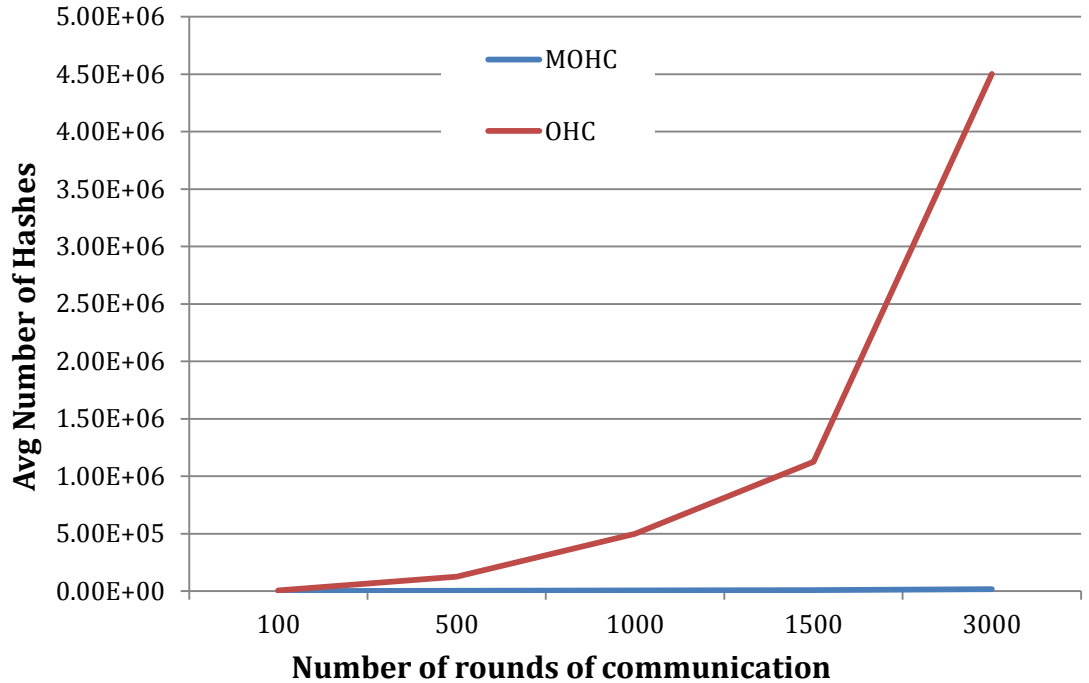
Figure 6-4 Average number of hashes comparison between OHC and MOHC

6.6   Conclusion

In this chapter, we have presented a mini one-way hash chain based solution for WSN user and broadcast authentication. Our objective of the proposed solution was to mitigate the high computational cost of the straightforwardly configured one-way hash chain and avoid exposing the initial secret. We have shown through simulation and evaluative experiments that with economic modifications to one-way hash chains, we were able to achieve lower computational overhead measured by computation time and number of hash operations required to provide protection. Additionally, by substituting the initial secret with easily computed authentication tokens, we were able to maintain the initial secret less exposed. We have also shown that our protocol is attractive form an energy consumption perspective as it lowers

consumption of sensor nodes' battery lives. In the future, we plan to improve our solution and

apply it in alternative WSN topologies.

CHAPTER 7: HYBRID CACHE-SUPPORTED HACH

7.1   Introduction

In the previous chapter, we have proposed four schemes utilizing different configurations of one-way hash constructions to deliver efficient authentication schemes that are attractive to developers.   Our main goal has been to achieve efficiency while utilizing the minimum computing resources possible. In this chapter, we explore the possibility of utilizing the two-dimensional HACH equipped with sparse caching capabilities to carry out authentication. While the scheme is designed with the collaborative and social media network in mind, it can be easily employed in other environments where session cookies based authentication is used.

7.2   The proposed Hybrid Scheme

We propose a hybrid solution that maximizes efficiency and minimizes the cost of memory resources. To achieve this, we divide the one-way hash chain into multiple chains and support them with caching units where authentication tokens are stored and fetched as needed. To measure efficiency, we use the number of hash operations needed in a session. In the next section, we define the configuration of the system's components and determine the cost based on these configurations. Before this overview is presented, we introduce the notations used in the scheme. We refer to the proposed scheme as the hybrid scheme.

Scheme Notation

I = mini OHC scheme

J = OHC Caching scheme

K = Hybrid scheme

Common Notation

N = number of transactions

X = horizontal chain for seeds

Y = vertical chain for authentication tokens

M = space interval between cache units

$N = X \times Y$ // simplified assumption

We will introduce a more detailed description of these notations in our description of the schemes below.

7.2.1   The mini OHC Scheme:

The conventional OHC scheme has one dimension where one seed is used to generate authentication tokens by a single one-way hash chain for the whole session. However, the mini OHC scheme is arranged into two dimensions (see Figure 7.1). In the first dimension (i.e. the horizontal axis X_i), there is a single hash chain that computes the seeds for the second dimensions chains (i.e. vertical axis Y_i). In the second dimension, we have multiple hash chains that use these seeds to generate authentication tokens. Authentication tokens are generated by hashing the seeds using cryptographic hash functions (e.g. SHA-1, SHA-2 or SHA-3). These cryptographic hash functions are known for their resistance against attacks.
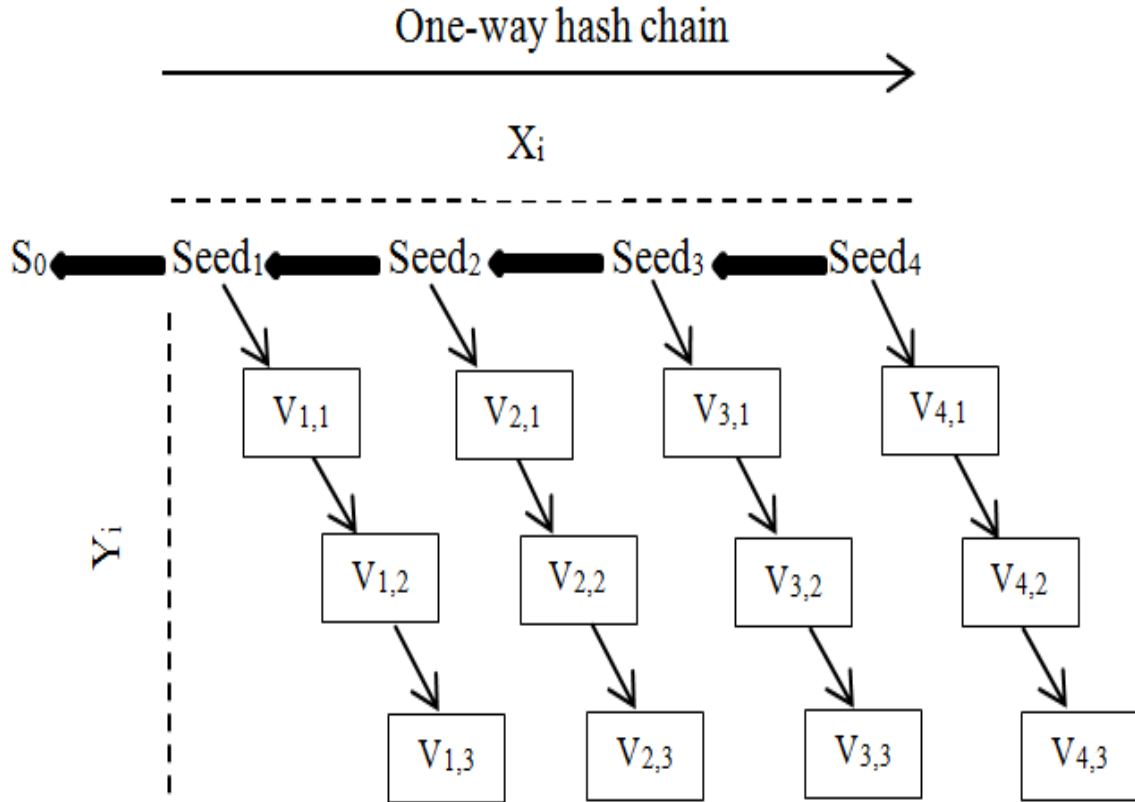
Figure 7-1 mini OHC Scheme

Authentication in the mini OHC is done in three steps:

- *Initialization*:

The server and the client utilize an HTTPS channel to exchange the number of transactions in a session N, an initial value of the shared secret S_0, and the length of the authentication token chain Y_i. Based on these variables, the number of seeds is determined, and $Seed_1$ is calculated to be used in the first authentication token chain by applying one-way hash function on S_0.

- *Authentication:*

In this step, the scheme generates the authentication token V at the client side. The authentication tokens are derived in the vertical chains by applying the one-way hash function on

Seed₁. The authentication token is then attached to the transaction cookie and sent to the server. A similar authentication routine is done at the server's side to check the authenticity of the authentication token. If authentication is verified, the transaction is accepted; otherwise, it will be rejected. Once the first vertical chain is exhausted, the next routine (i.e. *Seed Update*) is called to calculate seeds for the following vertical chain.

- *Seed Update*:

Once the first vertical chain is exhausted, the seed is updated for the next chain by applying a one-way hash function on the initial S_0. Note that each seed is only used in a single vertical chain to generate authentication tokens for that chain. Furthermore, the authentication tokens once used are discarded and never used again. As such, the cost of authentication in the mini OHC scheme is a result of calculating the number of hash operations in the horizontal chain and the multiple vertical chains. The following is how we calculate the cost of the scheme:

$$\text{Cost of one vertical chain} = \frac{Y_i \times (Y_i + 1)}{2} \tag{7.1}$$

$$\text{Total cost of all vertical chains} = C_V = X_i \times \frac{Y_i \times (Y_i + 1)}{2}$$

$$= N \times \frac{(Y_i + 1)}{2} \tag{7.2}$$

$$\text{Cost of the horizontal chain} = C_H = \frac{X_i \times (X_i + 1)}{2} \tag{7.3}$$

$$\text{Total Cost} = C = C_V + C_H$$

$$= N \times \frac{(Y_i + 1)}{2} + \frac{X_i \times (X_i + 1)}{2} \tag{7.4}$$

7.2.2   The OHC Caching Scheme:

Unlike the previous mini OHC scheme, the OHC caching scheme utilizes storage and only one-dimension chain, to reduce the computation overhead of the OHC. During the

initialization step, and in addition to the initial S_0 a few authentication tokens are pre-calculated and stored. Figure 7.2 demonstrates how the cache units are placed for this scheme. The highlighted blocks are where the authentication tokens are stored. For ease we assume the interval between caches is one. It should be noted that since we only have one dimension in the OHC caching scheme, the X parameter is considered the cache size (i.e. number of cache units utilized). Also, given we do not have vertical chains, we consider Y to be the interval between cache units. Therefore, X_j= size of cache Y_j= interval between two cache units.
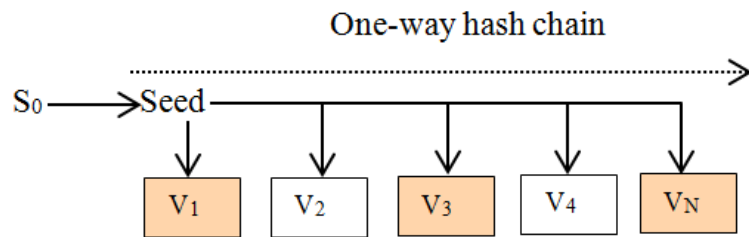
Figure 7-2 OHC Caching Scheme

To reiterate, let us revisit an example for a session of 100 transactions authenticated using the OHC caching scheme. For a session of this size, five storage units (of length 160 bits for SHA-1) can be reserved. Thus five authentication tokens are calculated in the initial steps and stored at 20 transaction interval as follows. Note that X_j in this example is 5 and Y_j is 20.

cache[0] = s

cache[1] = $H^{20}(s)$,

cache[2] = $H^{40}(s)$,

cache[3] = $H^{60}(s)$,

cache[4] = $H^{80}(s)$.

Compared to the mini OHC scheme, the OHC with caching has the advantage of low computation cost. Two important parameters are used to guide the calculation of computation overhead in the OHC with caching: the cache size X_j and the cache spacing interval Y_j. In the above example, the cache size= 5 and the cache spacing interval= 20. Based on these assumptions, we can determine the cost of the OHC caching scheme according to the following formulas:

Cost of authentication tokens between two cache units $= \frac{Y_J \times (Y_J+1)}{2}$ (7.5)

$$\text{Total Cost} = C = X_j \times \frac{Y_J \times (Y_J+1)}{2}$$

$$= N \times \frac{(Y_J+1)}{2} \qquad (7.6)$$

It should be noted that the total cost includes the sum of the cost of $(X_j\text{-}1) \times Y_j$ for the initial filling of the cache values and a cost of $\frac{(X_j-1) \times Y \times (X_j+1)}{2}$ for the N transactions. Also, notice that $(X_j\text{-}1)$ of the N transactions will not need to perform any hashing since the required value is already in the cache. For the above example of $N = 100$ and $X_j = 5$, transaction # 21 will simply read V_{80} from cache[4].

In order to handle more transactions efficiently, we either need to increase the number of storage units allocations. Or, we have to increase the cache spacing interval.

7.3 HACH Hybrid Scheme

We can alleviate the need for extra storage units and increase efficiency by equipping the mini OHC with sparse caching components to benefit from the advantages of caching in a two-dimensional configuration. Figure 7.3 is a general view of how the hybrid scheme looks with the caching units added to the mini OHC in the vertical chain Y_k. In the Simulation Results Section,

we discuss why we prefer to equip the scheme with caching at the vertical chain. The highlighted blocks are where the sparse caching units are placed, and the authentication tokens in these locations are computed and stored. The proposed scheme uses two dimensions, each of which generates a set of values. In the first dimension—the horizontal dimension denoted X_k, is a single one-way hash chain responsible for generating seeds to be used in producing the authentication tokens in the second dimension Y_k—the vertical multiple one-way hash chains.
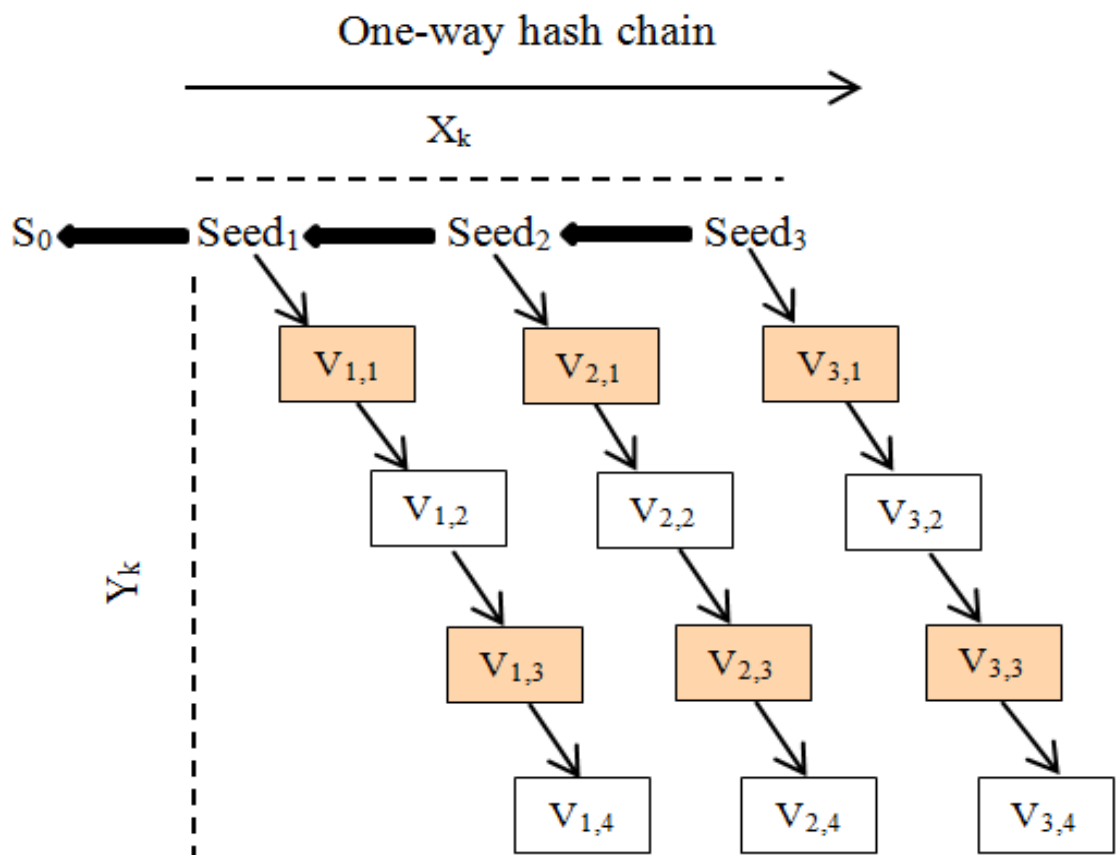
Figure 7-3 HACH Hybrid Scheme

Here is a high level description of the protocol.

Initialization:

$X_k := N \div Y_k$ // length of the Seed_Chain

$K := Y_k$ // K is the global index for the Token_Chain

$J := X_k$ // J is the global index for the Seed_Chain

$Seed := H^J(S0)$ // Seed is now $Seed_1$ for the first Token_Chain

$Interval := Y_k \div Cache\_Size$ // # of hash operations between cache units

Call Fill_Cache(Seed)

Fill_Cache (Seed)

Begin

$i := 0$

$HNum := 1$ // number of hashes to be calculated

While(i not equal to Cache_Size)

$Cache[i] := H^M(Seed)$ // authentication tokens stored

$HNum := HNum + Interval$

$i := i+1$

End-While

End

Authentication ( )

Begin

$L := (K/Interval)-1$ //L is the cache locator to fetch the token value

$HNum := K-(L*interval)-1$

$V := H^{HNum}(Cache[L])$

$K := K -1$

if (K==0) then

  Update_Seed( )

  K:= Y_k;

  end_if

Return (V);

End

Update_Seed( )

Begin

J := J-1   // J is the global index for the first-tier chain

Seed:= $H^J(S0)$;

Call Fill_Cache(Seed) // update the next authentication tokens

Return (Seed)

End

The protocol is composed of four main procedures: the *Initialization*, the *Authentication,* the *Update_Seed* and the *Fill_Cache (Seed)* routines. Each of these routines is responsible for some part of the protocol.

The *Initialization* procedure works the same way as described in the mini OHC described in Section 3. An additional step in the *Initialization* entails filling the cache with authentication tokens based on the cache size. This is achieved by invoking the *Fill_Cache (Seed)* procedure.

The next step is when the session actually starts. It is where authentication tokens are used to protect session cookies. The *Authentication* procedure is responsible for generating the authentication tokens. This step works by locating the closest cache, fetching the respective stored authentication token and performing the additional hash operations if needed. Once the

114

first vertical chain is exhausted, the *Update_Seed* step is invoked and a new seed is calculated and handed over to *Fill_Cache (Seed)* so that authentication tokens for the next Token Chain are stored. The protocol works in this manner until the session is complete.

Compared to the single dimension OHC caching scheme described in Section 3, where the number of cache units devised either grows proportionately with the number of transactions, or is configured to handle more transactions by increasing the cache spacing interval, the hybrid scheme is more efficient. In other words, to achieve good performance with higher number of transactions, the OHC caching scheme will need to devise more cache units. In the hybrid configuration, however, we efficiently handle this scenario, but with much less space by emptying storage after each Token Chain is exhausted.

While the mini OHC performance is influenced by the length of the Token Chain and the OHC with caching by the cache size, we need to investigate the optimal configuration of the hybrid scheme by comparing the costs of the previous two schemes and identifying the factors that influence the performance. In the following section, we introduce our evaluation of these factors and present an analytical model to find the best tradeoff between cache employment and performance.

7.4   Comparison and Tradeoffs

Essentially, using the number of hash operations in a session as a measurement metric, the mini OHC has higher computation cost as opposed to the OHC caching scheme. The difference between the two schemes is $\frac{X_i \times (X_i + 1)}{2}$. However, there is the expense of extra storage units associated with OHC caching scheme. If the number of cache units devised is relatively small, the performance is comparable. However, if more cache units are added, the OHC caching

scheme outperforms the mini OHC. Figure 7.4 shows how the two schemes give different performance for different size of cache for 500 transactions.
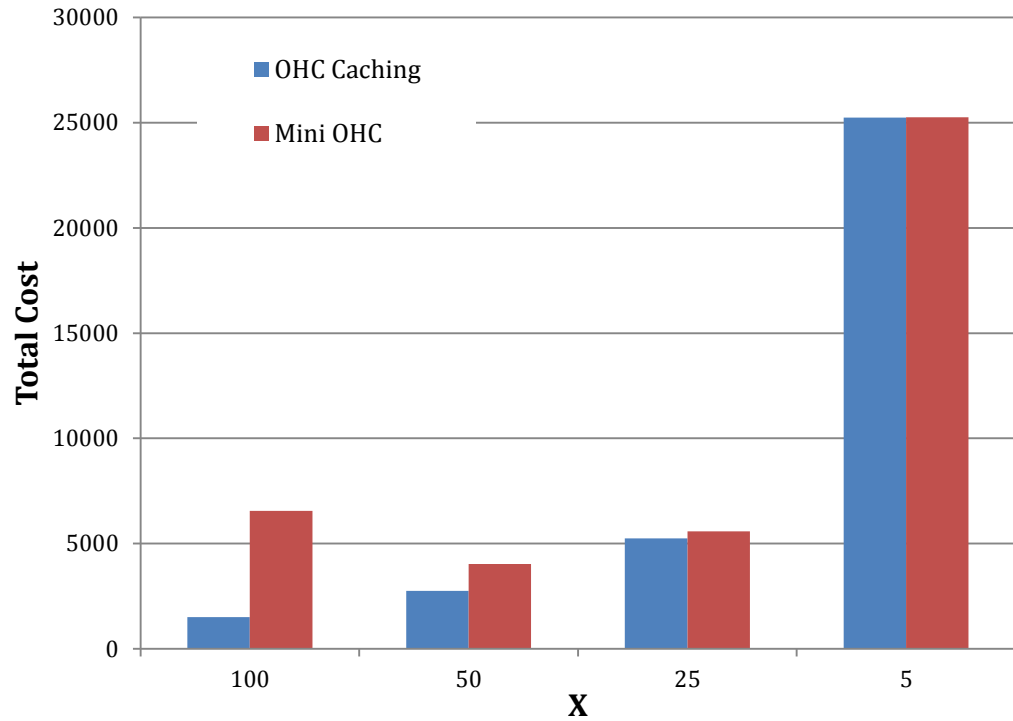
Figure 7-4 Total Cost of 500 transactions with different x values

In the hybrid scheme, our goal is to utilize the minimum storage requirements while efficiently handling authentication. First, we present how the total hash cost of the hybrid scheme is calculated. The analytical model below is used to obtain the optimal setup of the scheme; we try to achieve a configuration that strikes a balance between efficiency and memory requirements.

$$N = X_K \times Y_K \tag{7.7}$$

Assuming $X_K \ll Y_K$ and using the same number of sparse storage units X in the vertical chains, we get:

Let Space_Interval $M = \frac{Y_K}{X_K}$ for simplifying assumption M integer

$$Y_K = M \times X_K \qquad (7.8)$$

$$N = X_K \times Y_K = M \times X^2 \qquad (7.9)$$

Cost of authentication tokens between 2 cache units $= \frac{M \times (M+1)}{2}$ $\qquad (7.10)$

Cost of one vertical chain $= X_K \times \frac{M \times (M+1)}{2} = Y_K \times \frac{(M+1)}{2}$ $\qquad (7.11)$

Cost of all vertical chains $= C_V = X_K \times Y_K \times \frac{(M+1)}{2} = N \times \frac{(M+1)}{2}$ $\qquad (7.12)$

Cost of horizontal chain $= C_H = \frac{X_K \times (X_K + 1)}{2}$ $\qquad (7.13)$

$Total\ Cost = C = C_V + C_H = N \times \frac{(M+1)}{2} + \frac{X_K \times (X_K + 1)}{2}$ $\qquad (7.14)$

The above formula can be used to plot C as a function of N and M where $X = \sqrt{\frac{N}{M}}$

$$C = 0.5NM + 0.5N + 0.5\,\frac{N}{M} + 0.5\sqrt{\frac{N}{M}} \qquad (7.15)$$

To find the optimal value of M which minimizes the cost C, we differentiate the above formula with respect to M and equate to 0. On differentiation we have

$$\frac{dC}{dM} = 0.5\,N - 0.5\,\frac{N}{M^2} + 0.5\,\sqrt{N}\left(-0.5\,M^{-3/2}\right)$$

$$= 0.5N - 0.5\,\frac{N}{M^2} - 0.25\,\frac{1}{M}\sqrt{\frac{N}{M}} \qquad (7.16)$$

After equating $\frac{dC}{dM}$ to zero, we get the following quartic equation:

$$4NM^4 - 8NM^2 - M + 4N = 0 \qquad (7.17)$$

To give an example, the Plot of the function C when the number of transactions is 500 is given below in Figure 7.5.
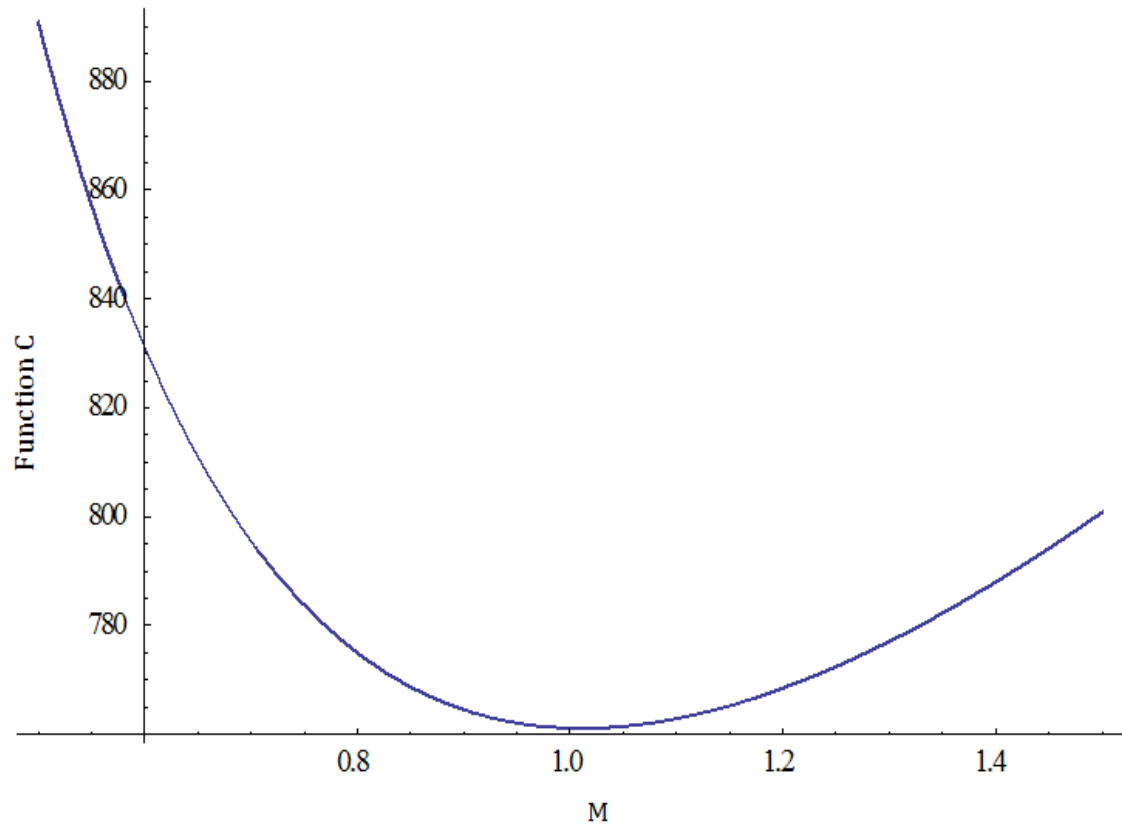
Figure 7-5 The minimum value of M when N= 500

Thus, the minimum value occurs near the value $M = 1$. Table 7-1 summarizes the optimal values of M for the different numbers of transactions.

Table 7-1 Optimal values of M

N	Optimal value of M by equating $\frac{dC}{dM} = 0$
500	1.01118
1000	1.00791
1500	1.00645
2000	1.00559
2500	1.00500
3000	1.00456

Based on the analytical modeling presented above, we can determine the optimal cache spacing in the hybrid scheme to be 1. Therefore, given the optimal Token Chain length Y_k obtained in [14], we run our simulation with the assumption that the optimal cache spacing is 1. In the next section the simulation results for the three schemes are presented.

7.5   Simulation and Performance Results

The performance of the HACH hybrid scheme is evaluated using a detailed Java benchmark. Our goal was to measure the performance of the three schemes. We compared and contrasted the results measured in terms of efficiency (number of hash operations in a session) and in terms of storage units required to complete an internet session.

7.5.1   Caching options

The hybrid scheme can benefit from caching in a number of ways. Our first option is to use caching in the Token Chain Y_k. In other words, we only store the authentication tokens or a subset of them in the vertical dimension of the mini OHC. Caching can be either full or partial.

In the full caching option, all the authentication tokens are calculated and stored before authentication, whereas only a subset of authentication tokens are calculated and stored in the partial caching option. Below is a description of both options.

In the full caching

The number of cache units required is equal to the optimal Token Chain length Y_k. Since all authentication tokens are going to be calculated and stored before the start of the session, the full caching approach indicates that the authentication tokens do not require any hash operation in the Y_k. The only cost incurred when full caching is utilized would be hash operations used to derive the seeds in the X_k.

In the partial caching

A subset of authentication tokens in the Token Chain Y_k is stored. We use the optimal cache spacing obtained above (i.e. $M = 1$) and the optimal Token Chain Y_k as the basis for our spacing. As a result, each authentication token will cost either one hash operation or none (i.e. fetching the authentication token form the cache). Here is an example to illustrate this:

Suppose we have an Internet session of length 500 transactions. According to [15], the optimal length of Y_k in the mini OHC is 10. The length of Seed Chain X_k is going to be 50.

Cache [0]=$H^1(S_1)$

Cache [1]=$H^3(S_1)$

Cache [2]=$H^5(S_1)$

Cache [3]=$H^7(S_1)$

Cache [4]=$H^9(S_1)$

Given the number of cache units and the spacing interval, the following are the first ten authentication tokens along with their cost in terms of hash operations.

1st Transaction $= V_1 = H^1$(Cache [4])

2nd Transaction $= V_2 = H^0$(Cache [4])

3rd Transaction $= V_3 = H^1$(Cache [3])

…….

10th Transaction $= V_{10} = H^0$(cache [0])

Therefore, for every Y_k only 5 hash operations are required given we have 5 cache units uniformly distributed. The maximum number of hash operations for the authentication token transaction is 1 if partial caching with cache spacing of 1 is used.

These caches are used in the first Y_k. Once this chain is exhausted, the cache is emptied and a new seed is generated for the next Y_k authentication tokens. New values are calculated and the cache is filled again with new authentication tokens. If partial caching was used in the OHC with caching scheme only, we would need 250 memory spaces to carry out an Internet session of length 500 transactions to achieve comparable results. With adding partial sparse caching to the mini OHC, we can bring this number down to just 5 memory spaces.

7.5.2 Full vs. Partial Caching Performance

Figure 7.6 demonstrates the session cost measured by the number of hashes in the full caching configuration and the partial caching operation. We perform this test when caching is only performed at the Token Chain Y_k. Later, we test the caching option in the Seed Chain X_k. Here we can see that the full caching does not have a tremendous improvement over partial caching.
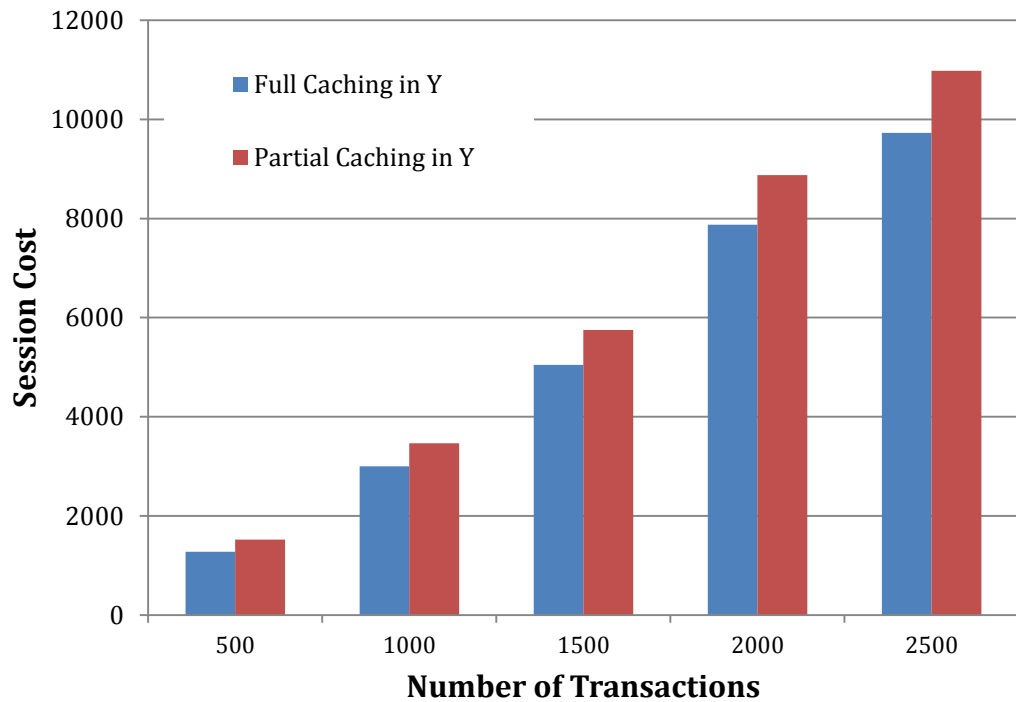
Figure 7-6 Session cost comparison between full and partial caching in g in Y

Thus, the next step is to compare the storage requirements in the full and partial caching configurations in the Token Chain Y_k to see whether it is worth to employ full caching or partial caching. The comparison is presented in Figure 7.7. While the partial caching requires half the storage of the full caching, it can still achieve good results. Therefore, the partial caching can be a better option as the memory requirement is half without sacrificing performance.
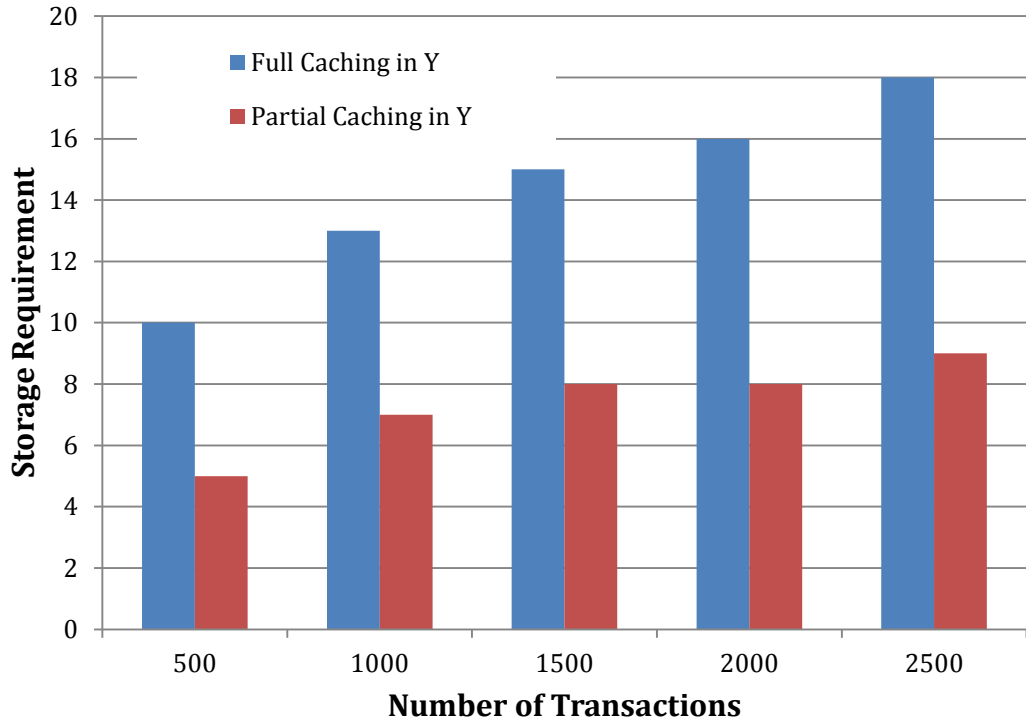
Figure 7-7 Storage requirement comparison between full and partial caching

7.5.3   Performance Comparison between the three schemes

Since the partial caching option strikes a good balance between memory requirement and efficient performance, we compare the performance of the hybrid scheme in the partial caching option at the Token Chain Y_k with the HACH caching scheme and the mini HACH scheme. Table 7.2 summarizes the results of this comparison. Note that we use the same number of cache units in the simulation of each scenario except for mini HACH where caching is not supported. In terms of efficiency, the mini HACH scheme helps reduce the session cost significantly if compared to the HACH scheme. By utilizing very little storage in the hybrid scheme, we were able to lower this cost by approximately 65% as indicated in the table.

Table 7-2 Comparing session cost between the three schemes

Number of Transactions	HACH	mini HACH	Hybrid HACH
500	25250	4025	1525
1000	71930	10009	3465
1500	141382	12505	5750
2000	251000	24875	8875
2500	348471	33496	10980

7.6 Caching in Token Chain Y_k or Seed Chain X_k

Our next task is to see whether equipping the proposed scheme with caching capabilities in both dimensions can have better outcome. Figure 7.8 and 7.9 demonstrate the storage and session cost requirements if either the Seed Chain or the Token Chain is equipped with caching capabilities. It is obvious that adding caching in the Seed Chain does not benefit the scheme as storage requirement increases (Figure 7.8) while the session cost increases (Figure 8.9). Therefore, we have opted for equipping the hybrid scheme with sparse caching at the Token Chain to be the optimal setup.
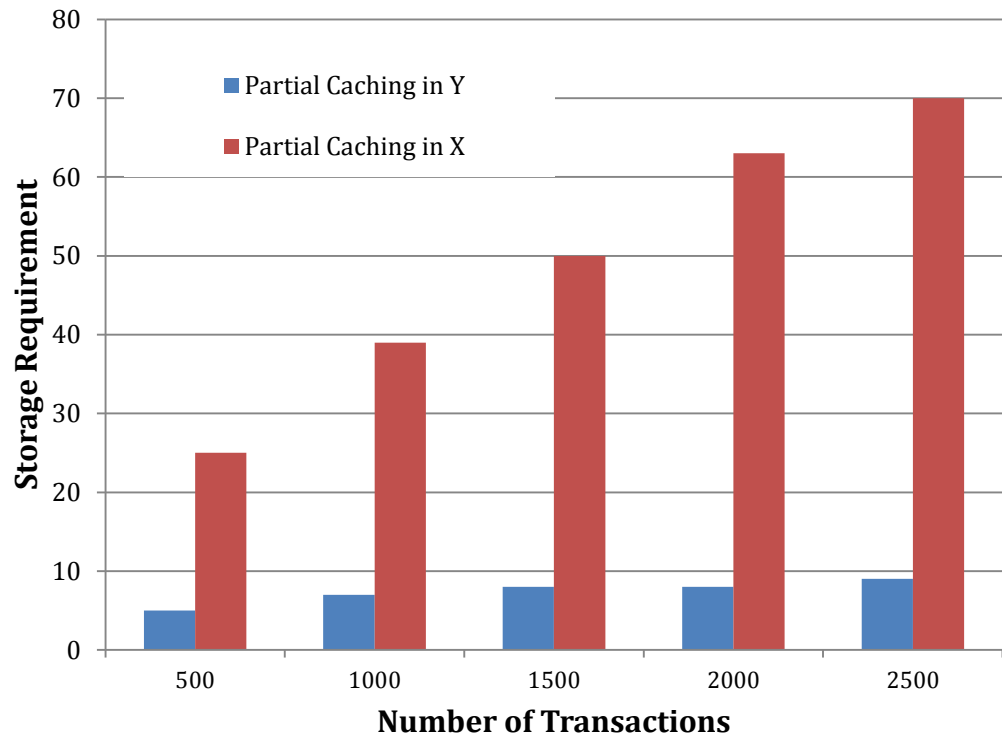
Figure 7-8 Storage requirement comparison with cache support in either Token_Chain (Y) or Seed_Chain (X)
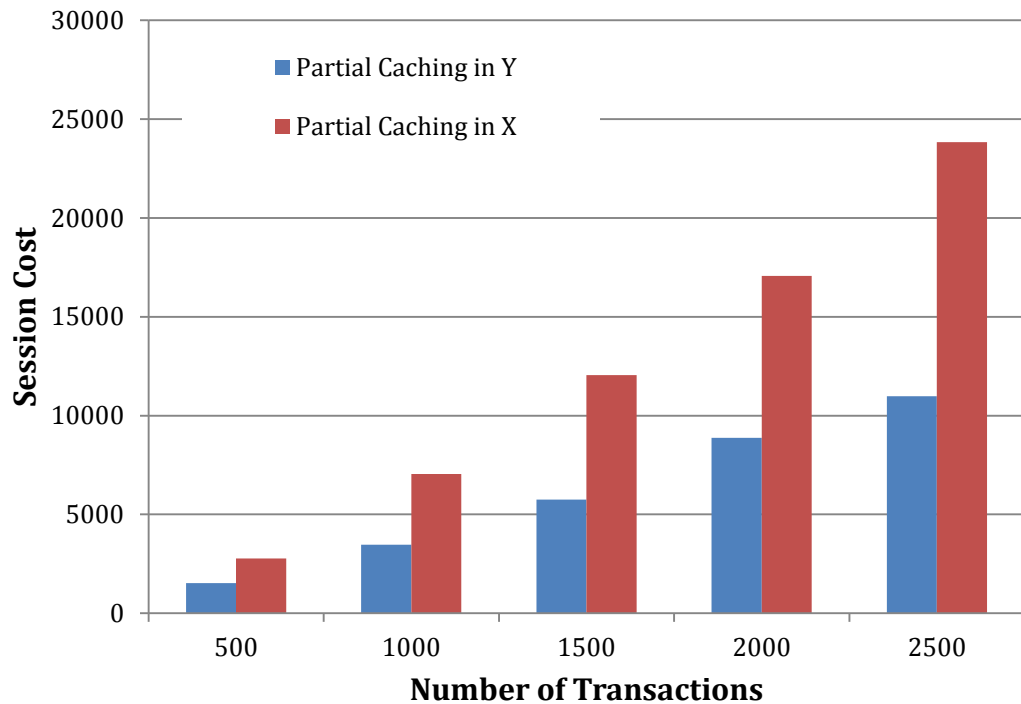
Figure 7-9 Session cost comparison with cache support in either Token_Chain (Y) or Seed_Chain (X)

7.7   Conclusion

This chapter addresses the security threat of session hijacking attacks facing collaborative applications especially when mobile and wireless applications are utilized to access collaborative services. Common HTTPS based solutions do not usually suit mobile devices especially those with limited computation and storage capacities. One-way hash chain based solutions have been proposed to replace the current cookie based session management techniques but due to their inherent nature requiring recursive computation of hash values, they do not suit some mobile devices. This is particularly because of the high computational overhead associated with OHC in Internet session.

This chapter proposed and analyzed the potential of a hybrid solution where divided one-way hash chains are equipped with caching capacities to store pre-computed hashed values and

fetch them once needed to authenticate a user session. We presented an analytical model which aimed at measuring the cost of the hybrid scheme compared to the straightforward HACH with caching and the two-dimensional HACH. We also used this analysis to derive a quartic equation with which we were able to identify the optimal cache spacing configuration in the hybrid scheme. The evaluation and experimentation reveal major improvements and highlight advantage of adding sparse caching to the mini one-way hash chains to achieve economic and efficient authentication for mobile devices that suits collaborative applications and other Internet applications.

CHAPTER 8: IMPROVED AUTHENTICATION IN SINGLE SIGN-ON ENVIRONMENT AND RFID IN TRAFFIC MANAGEMENT SYSTEMS

8.1 Introduction

In this chapter, we propose an authentication scheme tailored for the growing SSO environment. Our scheme relies on the utilization of hash calendars and Merkle Hash trees in addition to HACH to provide for a keyless signature environment. Our main motivation for adopting a keyless signature scheme is that the majority of SSO schemes that suffer from impersonation attacks rely heavily on public/private key cryptosystems to achieve their authentication. Our goal is to add rigor and robustness to current SSO systems without sacrificing functionality and effectiveness.

8.2 Problem Statement

Before we provide a formal definition of the authentication problem in SSO, let us introduce the typical components of SSO systems and their functionalities. Figure 8.1 depicts the SSO overall architecture. As can be seen in the figure, the communication and authentication of the SSO follows the steps illustrated. In the first step, a client requests a URI (unified resource identifier) from a service provider. Second, the service provider contacts an identity provider, who manages and delegates authentication in the scheme, to check if the client holds the required authentication privileges (an active user session established via the exchange of user's credentials), and, if not, asks the user to create one and provide it to the identity provider. If the user provides the necessary authentication credentials, the identify provider provides the requesting services provider and the other participating service providers with authentication assertion that entail that the user is a verified one.
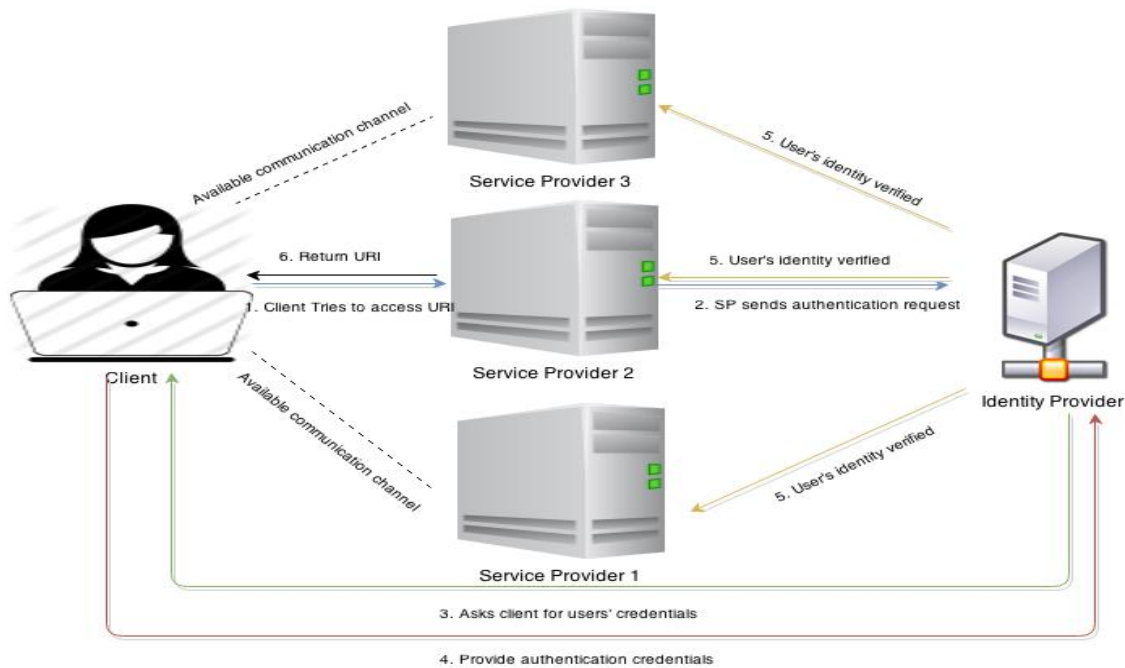
Figure 8-1 SSO overall structure

The authors of [95] indicate that the main cause of SSO impersonation attacks are related to the design of the communication channel between the service providers and the identity provider which lacks bi-directional authentication. In fact, the SAML 2.0 specification repeatedly indicate that the communication between the client, who is typically a browser guided by a user, and the SP is carried over a unilateral SSL connection. Similarly, the communication between the client and the SP starts as a unilateral SSL connection and becomes bilateral once the client is authenticated through the exchange of valid certificates from the IdP and appropriate credentials from the clients. These two assumptions are, according to [51], problematic due to two essential reasons: 1) the lack/ambiguity of authentication recentness which is a logical requirement in any authentication scheme, and 2) the unidirectional nature of authentication between the SP and the client. As far the first reason is concerned, the SAML 2.0 specification states that the SP and client mutually authenticate and agree on the value of URI, but fails to

guarantee whether this assertion is recent or not. This feature is particularly important given the fact that the SAML 2.0 protocol does not guarantee authentication to be carried out using a single SSL connection. Due to difficulties pertaining to the nature of SSL such as the difficulty to resume an already established connection after it has been terminated, it is natural to assume that SAML 2.0 protocol is carried out using different SSL connections. Therefore, the recentness of authentication assertion is not possible to achieve unless the protocol is run with each run of the SSL connection, which is necessarily against the whole nature of SSO. In terms of the second reason, the SAML 2.0 does not provide the client with guarantees of the authenticity of the SP's after he/she has been granted access by the IdP. Similar problems have also been detected in the OpenID environment.

The lack of recentness of the SSL protocols utilized in the SSO environment and the uni-directional nature of the SSO schemes makes it possible for a malicious SP to impersonate a client and cause it to access certain resources without its consent. Therefore, it is imperative for any authentication scheme to be suitable for SSO to satisfy these requirements in addition to the common authentication requirements.

8.3   The Proposed SSO Authentication Scheme

Our main goal in this paper is to develop a solution that efficiently handles the security of SSO. In the proposed protocol, we pay special attention to the possible causes of certain SSO vulnerabilities. From a cryptographic point of view, access to the SSO requires the service providers, clients as well as identity providers to carry their communication based on public and private keys as seen in the majority of the schemes discussed in the Literature Review Chapter of this dissertation. The security of the whole environment is contingent upon these keys. If for any reason these keys are compromised, there is the risk of stolen identities, password revocation and

their likes. In this chapter, we propose a keyless signature scheme that is based on utilizing a hash calendar and Merkle Hash Trees in addition to our HACH protocol. In Table 8.1, we introduce the notations used in the chapter.

Table 8-1 Keyless SSO Scheme Notations

Symbol	Description
Idp	The central Identity Provider
C	An SSO client
SPs	List of Service Providers
K	Length of one-way hash chain to generate authentication tokens
Seed	Secret value dispensed by *SP* to *C* to be used in *V*
N	The activity between client's login and logout
V	Authentication Token
H()	One way hash function
CO	The total computational cost during *N*
I	Computational cost in SSL connection when *Seed* is exchanged between *C* and *SP*

8.3.1 Adversary Model

We summarize the adversary model envisioned in the current scheme in the following steps. Steps of the attack involving a malicious service provider, which can be conducted by malicious code insertion:

1. C contacts a malicious SP (thinking it's an honest SP) requesting a URI

2. Impersonating C, the malicious SP relays the request received from C to an honest SP

3. The honest SP replies to the malicious SP to authenticate himself to IdP.

4. The malicious SP relays the authenticate message to C.

5. C communicates with IdP through HTTPS for authentication and provide its credentials.

6. IdP provides C with authentication assertions to be sent to SP.

7. SP receives C's authentication assertions and set cookies for further communication.

8. When C tries to access a service at SP and provides session cookies, the malicious SP will steal the cookies because the connection is in http.

9. Malicious SP will freely use services at SP as long as cookies are active.

Let us before describing the details of our keyless protocol touch on the concepts of Merkle hash trees and hash calendars.

8.3.2   Merkle Hash Trees

A Merkle hash tree is a binary tree whose leaf nodes are pieces of data. Neighboring nodes are concatenated and hashed. The resulting hash value is the first level of the tree. An example of a Merkle Hash Tree is shown in Figure 8.2. At lowest level of the tree, we have the data that is hashed and placed in a leaf node.

Let us now explain how authentication is conducted in a Merkle Hash Tree. At the lowest level of the tree, the values of the eight leaf nodes are calculated by applying a hash function $h(i)$ $=h(Ti)(i = 1, . . . , 8)$, where T is a piece of data. At the tree nodes level, the internal nodes are derived from their child nodes. For example, the value of the node $h_{3,4} = h(h_3||h_4)$ and so on until we reach the root node $h_{1,8}=h(h_{1,4}||h_{5,8})$. To verify the authenticity of a node, the leaf nodes can be authenticated with the root node $h_{1,8}$ and the corresponding authentication path information. For example, to verify the node T1, the server, in which $h_{1,8}$ (the root) is stored, checks the information sent from T1 (which also includes the corresponding authentication path information

= h_2, $h_{3,4}$, $h_{5,8}$). The server first computes and h_1, then $h_{1,2} = h(h_1\|h_2)$, $h_{1,4} = h(h_{1,2}\|h_{3,4})$, and last, $h_{1,8} = h(h_{1,4}\|h_{5,8})$. In the last step, the server checks whether the computed $h_{1,8}$ corresponds to the existing $h_{1,8}$. The server only accepts the data from T1, only if the two values are matching.

Our utilization of the Merkle hash tree does not only involve authentication of data as explained above. Rather, we use it to generate a value that constitutes the SSO key. As will be shown later in this section, the Merkle hash tree construction enables us to periodically update this key.
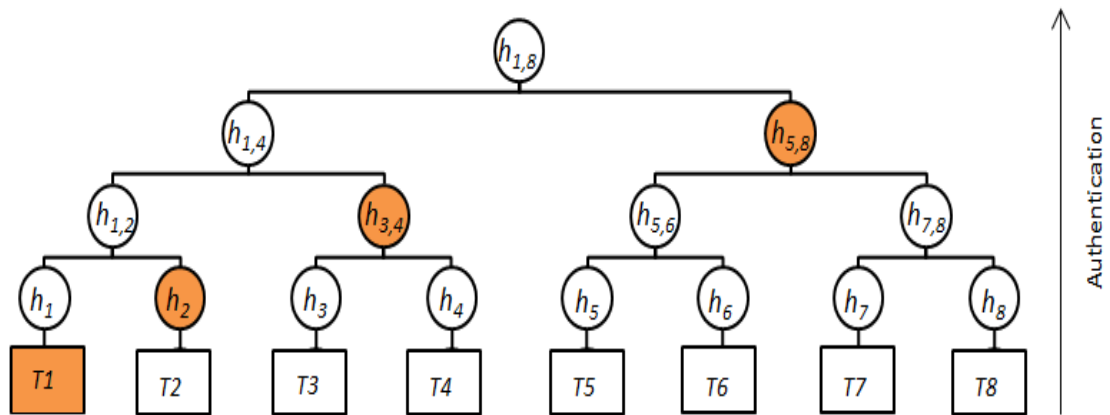
Figure 8-2 Merkle hash tree

8.3.3   Hash Calendars

The authors of [62] indicate that a hash calendar as a special type of a hash tree with leaf nodes representing every second since 1970-01-01 00:00:00 UTC. As such, at any given moment, the tree has a leaf node for each second starting from this time. The structure of the tree starts from left to right and each leaf node added is added to the right. They describe several ways the tree can be structured and hash chains for individual seconds are extracted from it. One such construction is as follows:

The tree is built with maximal number of nodes and the nodes representing future seconds are left empty as in Figure 8.3. In this particular figure, we have a tree for 11 seconds. Instead of the regular concatenation and hashing in a typical Merkle tree the, the empty nodes (grey nodes) are merged and copied to the next level. In case there is one empty node and another non-empty node, the non-empty node is copied to the next level (dashed lines). Therefore, in the chain extraction process, the copying processes are not included and only the copied nodes are used for the chain extraction.
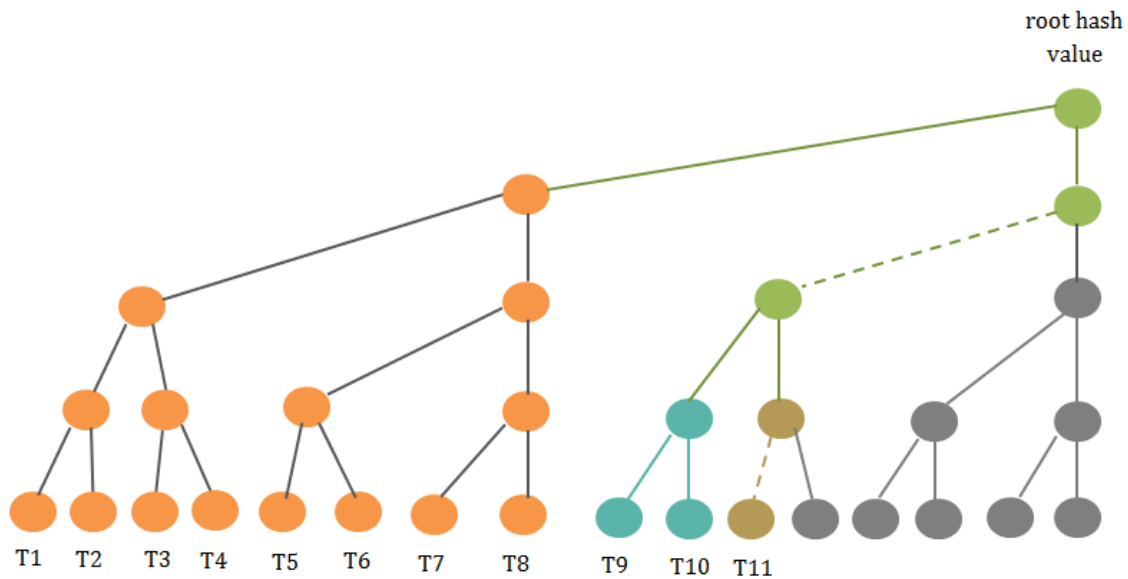
Figure 8-3 Hash calendar structure

The authentication mechanism using the hash calendar is to a great extent similar to that of the Merkle Hash Trees. That is the authentication is achieved by using two pieces of information: 1) the authentication path information and the root hash. The hash calendar is originally proposed to perform timestamping functionality [62] that is verifiable if needed. The root hash of the hash calendar is calculated at a certain point and published via easily accessible media. In our proposed scheme, we employ the hash calendar not only to provide timestamping,

but also to produce the authentication key. The published root is used as a key to verify the authenticity of the communicating parties. More importantly, the root is updated periodically to ensure recentness of the authentication key. In our proposed scheme, we employ the hash calendar in a two-phase manner to produce the authentication key. The published root is used as a key to verify the authenticity of the communicating parties. More importantly, the root is updated periodically to ensure recentness of the authentication key.

8.3.4 SSO Keyless Signature Scheme Components

In this section, we introduce the main components of the proposed scheme and their roles.

- *Identity provider*: the identity provider handles authentication and provides authentication assertions to the service providers and clients. The identity provider keeps track of a hash calendar which is pre-calculated and updated periodically (in the following step we will show when the hash calendar is updated), in addition to registering each log in by the client in a hash tree.

- *The client*: the first communication between the client and the identity provider involves the creation and exchange of authentication credentials (a user name, and a password). This step can be SSL/TSL secured. The first log in and each subsequent log in are used as input for the hash tree. Once the root of the hash tree is calculated, the root of the two trees are concatenated and hashed to produce the new key for the scheme.

- *The service provider*: like the client, the service provider contacts the identity provider to verify and get authentication assertions with reference to a certain client. Traffic

between the SP and C is secured by using a one-way hash chain which is meant to

produce a bidirectional authentication channel.

The identity provider is responsible for updating the hash trees and the hash calendar.

Each time the hash calendar is updated, the root value is handed over to the service providers and

the client as the fresh key for authentication. The key acts as a benchmark to validate the

communication between nodes. So, instead of sharing a public key between clients and servers

and utilizing private keys, the privacy of which is critical to the network, the key is used for

verification of authenticity of the communicating nodes. As such, if a malicious service provider

or client tries to get a hold of a client's credentials, they would need to have an updated and fresh

version of the key.

8.3.5   SSO Authentication Procedure

In this section, we provide a formal description of our SSO authentication procedure. The

following is a pseudo code:

C sends register request to *Idp*

    HashCalendar[C] ← *Idp* initializes a hash calendar for C   (*)

    Idp publish hash root to C and *SPs*   (**)

C sends a request to an *URI* at *SP*[i]

Idp verifies that C has permission to access *URI* by receiving a request from *SP[i]*.

LogIn (C, *SP*[i]) ← *Idp* authenticate if C is allowed to access *URI* at *SP*[i]

    Add *LogIn(C, SP[i])* to a hash tree   (*)

    If the hash tree of C is complete

        Calculate new hash root   (*)

        Append hash root to *HashCalendar*[C]   (*)

Idp publish hash root to C and SPs   (**)

SP[i] verifies *C* is valid and not malicious

    If the hash root matches

        Session[*C*] ← Active session for *C*

        SP[i] Generate *Seed* and send it along with *K* to *C*

        Transaction Authentication[*C*] ←Verify C's transaction

        $V \leftarrow H^k(Seed)$ (*)

        Decrement *K*   (*)

        If *K* is equal to 0

        Update *Seed*   (*)

        Reseat *K*

(*) Computation Overhead is measured

(**) Communication Overhead is measured

As the code above shows, the SSO authentication scheme is composed of two main routines; one is done at eh IdP side and the other is conducted at the SP side. While the goal of the routine at the IdP is to generate the SSO key (i.e. the hash value of the hash calendar and the hash tree) which satisfies the recent key requirement explained in section 2, the SP routine satisfies the unidirectional authentication requirement by utilizing and adapting our HACH scheme. In the following, we provide more explanation:

Routine at the Idp:

1. A client wants to access a service (URI) at an SP

2. SP checks with IdP to see if C is a registered client. If yes go to step 3 and, if not do the following:

> a. IdP returns to C for creating an account.
>
> b. Once the account is created, the IdP generates the hash root using a hash calendar and publishes to the client and SPs.

c. IdP initializes a hash tree and registers the first log in in the tree.

3. IdP checks if C is logged in:

> a. If yes, go to step 1 in the second routine (routine at SP).
>
> b. Otherwise, Idp requests C login information and add login info to the hash tree.

4. Once the hash tree is complete a new hash root is calculated and used as a new hash root.

Routine at the SP:

Each SP handles authentication independently by running the following routine (this routine prevents impersonation attacks):

1. After C tries to access a URI at SP, SP checks with IdP, which runs the first routine for authentication.

2. If there is an active log in, the SP requests the hash root from C. If it matches hash root at SP, a new session starts.

> a. SP generates and sends a Seed to be used in Transaction Authentication routine, along with the chain length K to C.

b. Each communication between C and SP is secured by authentication tokens generated by Transaction Authentication routine. If token V that came from C matches the stored one at SP, the transaction is accepted.

c. If session is exhausted the Seed is updated for the next authentication tokens.

4. Continue session until C logs out, or session terminates for lack of authentication.

Session termination causes:

Session termination requires the client to log in again: These can be changed depending on application needs:

1. Lack of authentication during an active session: when the client or SP fails to provide a current authentication token, the session terminates automatically and the user is required to log in again.

2. Time constraints: the user has been idle for a specific amount of time during an active session.

3. Changing machine: when the user logs in to the SSO using a different machine, the session in the first machine automatically terminates.

4. Changing connections: using a wireless connection that converts to an LTE connection requires the user to log in again.

8.4   Performance evaluation

This section introduces the performance evaluation and testing. We first introduce the testbed that was used as a benchmark to evaluate the performance of the proposed scheme.

8.4.1   Performance evaluation testbed

In order to evaluate the proposed SSO authentication scheme, a simulation based testbed was designed and developed. The simulator simulates communication between the components

139

described in section 4.4. The simulator functions as follows: 1) the simulator generates a number of requests for each client, and simulates the operation of the identity provider to record the communication overhead, computation overhead and number of hash operations in order to estimate the efficiency and performance of the simulated SSO authentication scheme. We provide more description of these metrics below.

8.4.2   Evaluation Metrics

The simulation is measured by counting each of measurement factors based on the logic of the authentication scheme. As such, the simulation records these following metrics which we use as our main evaluation metrics of the scheme:

- *Communication Overhead* is the number of communications required for the authentication scheme for services at service providers made by the client. The communication overhead entails communication between the client and the IdP in addition to communication between the client and SP's. Each of these overheads is measured separately.

- *Computation Overhead* is the number of computations required by the authentication scheme. Computation involves the creation and updating of the key from the IdP, in addition to the creation and update of the *Seed* at the SP's. This involves the hashing operations required to generate and update these variables.

The performance evaluation was run to find out the performance of the authentication scheme over a number of variables. The first configuration was when we varied the number of times a client logs in to the SSO. Second, we measured performance over variable number of service providers with fixed number of log ins. We then measure the optimal length of the one-way hash chain responsible for authenticating traffic between the client and SP's.

8.4.3 The Effect of Number of Log ins

The first test run to evaluate the SSO authentication scheme was when the number of log ins was varied. This number was set at 10 service providers. The graph in Figure 8.4 shows the existence of a linear relation between the authentication scheme performance over increasing number of log ins in terms of communication overhead. This is not surprising, as it is intuitive to have an increase in communication, computation overheads as well as the number of hash operations.
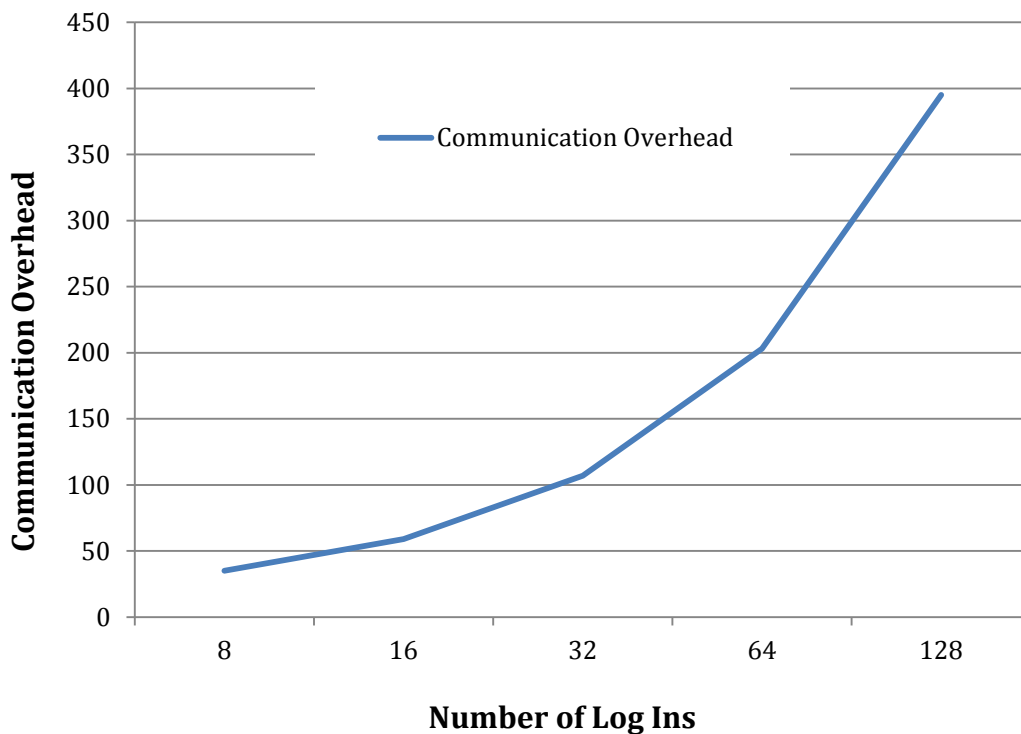
Figure 8-4 Communication Overhead over Variable Number of Log ins

The number of log ins also has a an effect on the computational overhead of the scheme. In Figure 8.5, this overhead shows that the more log ins we have, the higher the computational overhead incurred.
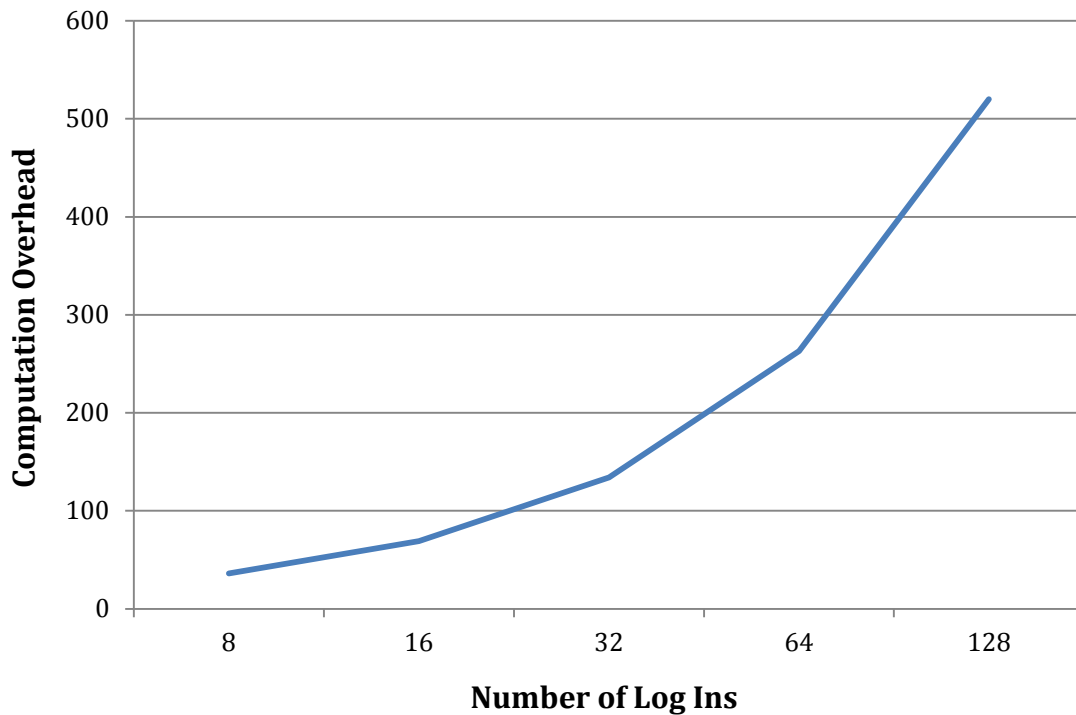
Figure 8-5 Computational Overhead with Variable Number of Log ins

8.4.4 Variable Number of Service Providers

The second evaluation environment was when the number of service providers increased. We increased the number of service providers incrementally by 10 until we simulated the authentication scheme's performance at 50 service providers. Note that we fix the number of log ins in this test at 8. The reason is that based on the previous two tests, 8 log ins appeared to have the least effect on both communication and computation. Figure 8.6 demonstrates the overhead in this setting. Similarly, Figure 8.7 shows the computation overhead with the different number of service providers.
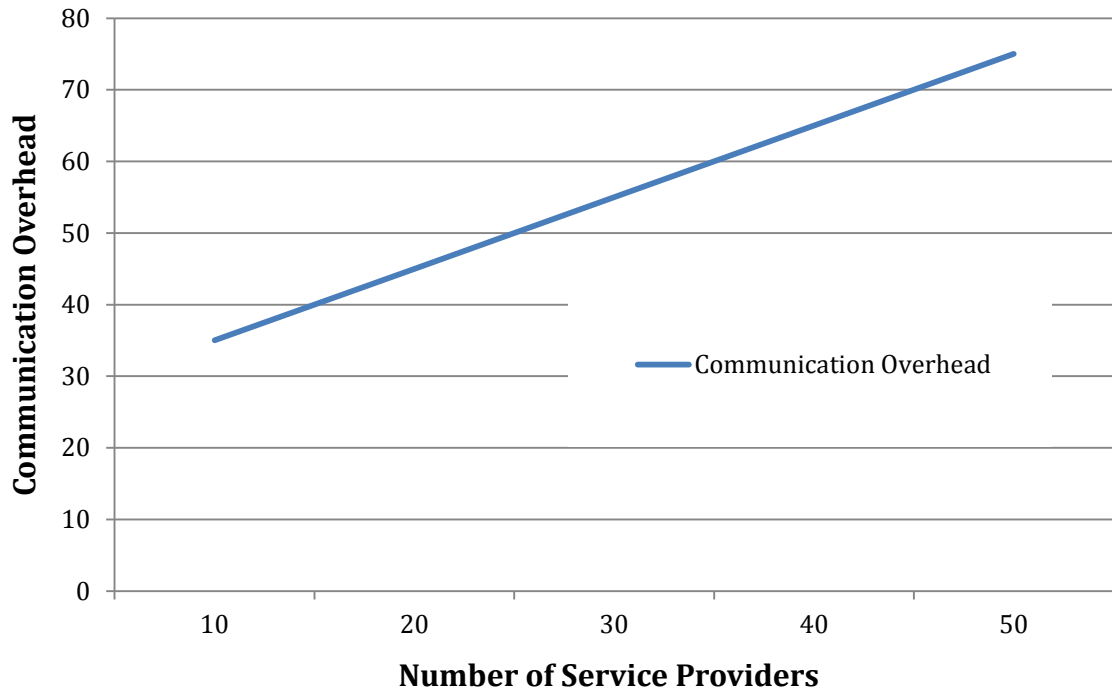
Figure 8-6 Communication Overhead with Variable Number of Service Providers
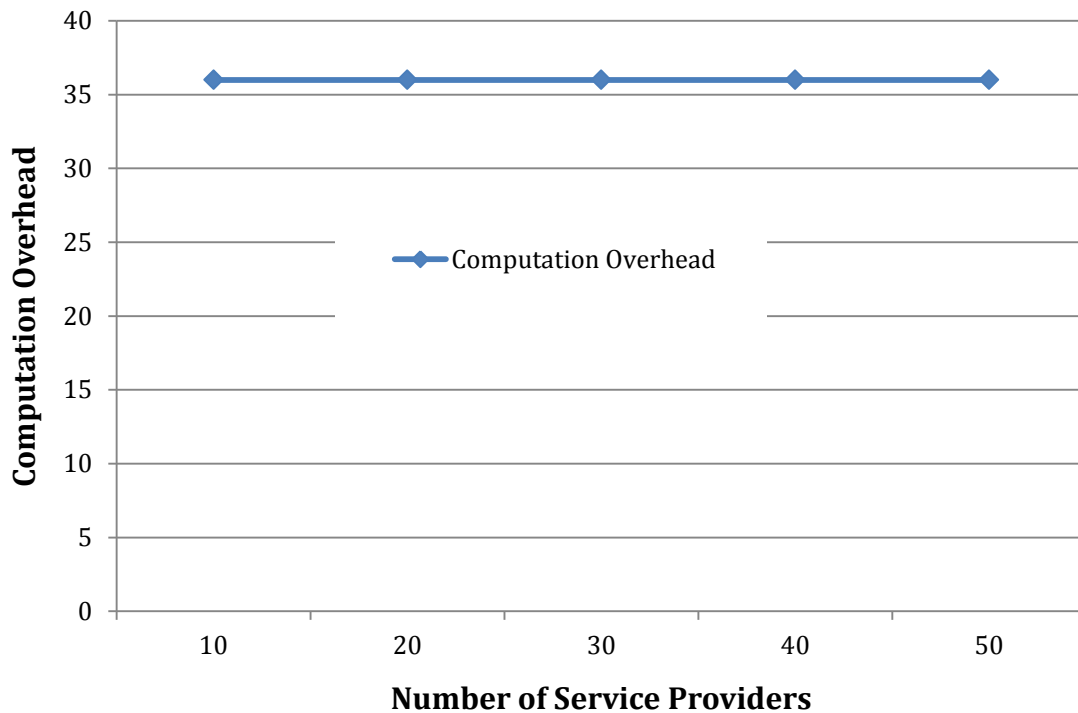
Figure 8-7 Computation Overhead with Variable Number of SPs

The main reason why this occurred is that each service provider is expected to compute the same number of hashes during the run of the authentication scheme. In other words, the work load is distributed equally among the service providers involved. In real life scenarios, this would be different, as the client might not use the services equally.

8.4.5 Optimal Token Chain Length in HACH

Although the computation and communication overheads seemed to be influenced by the increasing number of log ins and service providers, the length of K, which is the one-way hash chain length, appeared to have an unpredictable influence on the computation overhead at the SP's sides. More precisely, neither did the length of this chain yield a linearly increasing computation, nor did it lead to a decreasing effect. For that reason, we developed an analytical

model in which we derived an approximate optimal length of K by computing the cost of the HACH scheme at the SP.

In order to find this value, we had to account for the computations involved in the initialization and update of the *Seed* in an active Client/SP session. This cost is measured by adding the cost of the initial *Seed* generation, the hash operations to authenticate the traffic and the *Seed* updating once this chain is exhausted. While this cost is highly dependent on the number of transactions between the SP and C, which we gave a value of *N* in our notation Table 8.1, the *Seed* generation and update, which are done using an SSL based connection for which we used *I* in Table 8.1. Whereas *N* depends on users' behavior and differs from one application to another and, therefore, cannot be determined, we investigated the available literature to find an approximate cost of the SSL. The authors of [96] indicate that SSL is typically achieved through the SSL handshake protocol that usually involves three essential phases: 1) parameter negotiation, 2) mutual authentication and 4) secret key exchange. With this in mind, they measured the performance overload of SSL and found that approximately 80% of time is spent in SSL handshake and encryption/ decryption. Moreover, the authors of [97] analyzed SSL processing and found that about 70% of the total processing time of an HTTPS is spent in SSL processing. While computation cost of SSL is largely influenced by computing resources, an approximation can be made based on these statistics. In [97], they calculated the execution cost of the AES for key sizes of 128 and 256 and found them to be 562 and 747 cycles, respectively. For our derivation of the optimal chain length, we use both of these numbers for the different key sizes. Here is how we obtain the optimal length of K:

$$CO = (\frac{K \times (K+1)}{2} + I) \times \frac{N}{K} \qquad (8.1)$$

$$CO = \frac{NK+N}{2} + \frac{IN}{K} \qquad (8.2)$$

We Differentiate the above formula with respect to K and equate to 0.

$$\frac{\partial CO}{\partial K} = \frac{N}{2} - \frac{IN}{K^2} \qquad (8.3)$$

$$\frac{N}{2} - \frac{IN}{K^2} = 0 \qquad (8.4)$$

$$K^2 = 2I \qquad (8.5)$$

$$K = \sqrt{2I} \qquad (8.6)$$

We use this equation to plot the optimal value of K for 1500 transactions. Figure 8.8 demonstrates that the optimal value of k if an SSL key of either size 128 or 256 bits obtained by simulation. These are around 33 and 38 respectively. This means that the *Seed* needs to be updated approximately every 33, 38 transactions to achieve optimal performance depending on the AES key size used in the SSL connection.
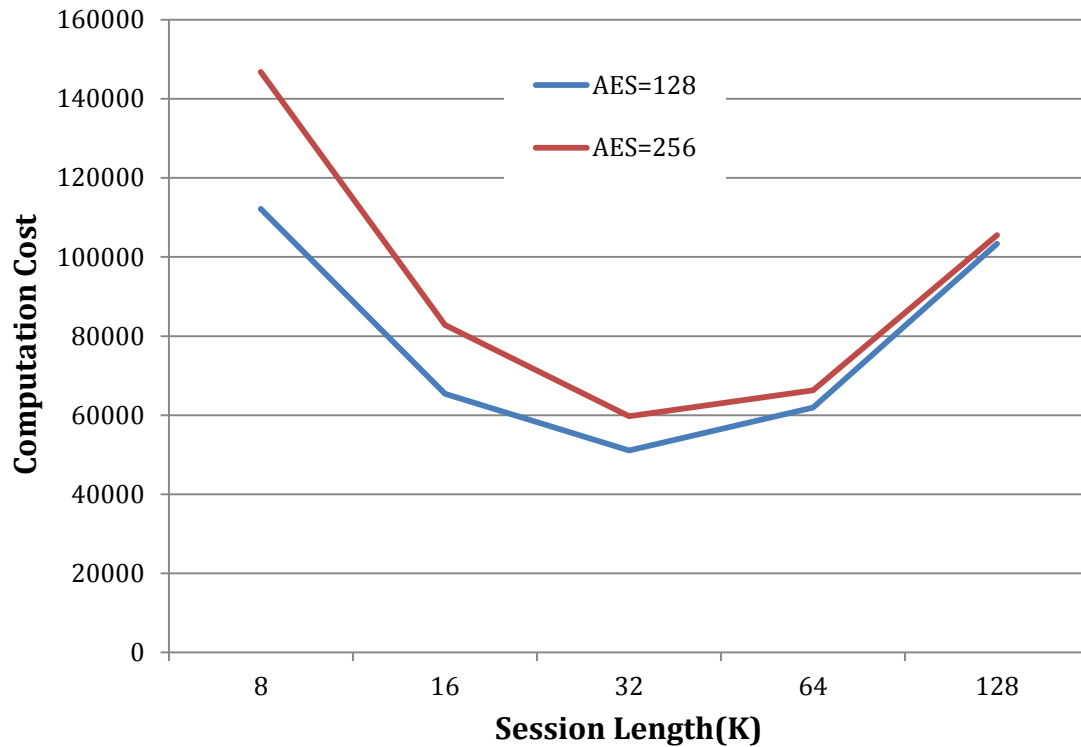
Figure 8-8 Optimal Chain Length for Different AES Key Sizes

8.5 Tree-based Authentication for Toll/Traffic Management Systems

After introducing our SSO authentication scheme, we devote this section of the chapter for an equally important context that requires rigorous authentication, namely, the context of Toll/Traffic management systems. We propose the utilization of a modified Merkle hash tree structure for better handling of RFID authentication in this particular context.

Existing RFID schemes targeting traffic management contexts are mainly conceptual and prototypical. Little attention has been given to the important issue of authentication. With the rapid migration to cloud-based technology, there is a pressing urgency for addressing authentication in toll/ traffic RFID solutions. We introduce a toll/ traffic RFID authentication scheme which benefits from the well-known Merkle hash trees. The main benefit of MHTs is its scalability compared to traditional schemes. In order to increase scalability, we propose a scheme

that significantly lowers the computation complexity of the traditional MHTs. In our proposed scheme, we achieve practical performance and improve traditional MHTs by a simple division mechanism. Initial evaluation results demonstrate these benefits and show promising direction for further research.

8.6   Problem Statement

A traditional RFID system is composed of three main components: RFID tags, RFID readers and a back-end server. The RFID readers, which are usually able to read multiple tags simultaneously, send encrypted radio signals interrogating RFID tags. The RFID tags in turn respond with their identification information along with the data relative to the application in which they are applied. For a road management RFID scheme, the main goal is to achieve Automatic Vehicle Identification (AVI). According to [98], AVI permits fully automatic and unique identification of vehicles at specific interrogation points and offers potential benefits in many fields such as fleet control, revenue collection, traffic operations, transportation planning, and safety and law enforcement. While primitive AVI techniques were based on technologies such as microwave and other dated technologies, more reliance on modern technologies such as RFID has become more common in recent systems.

Conventionally, RFID readers send, via radio signals, interrogative commands to RFID tags, which in turn respond with the interrogated information. Before this process of data transmission is incepted, an authentication process verifying the identity of the reader and the authenticity of the tags is conducted. During this process, both entities perform the security parameters set forth by the protocols deployed in the system. The reader first sends a nonce message to activate the RFID tags. The tags verify the identity of the readers by checking the readers' stored information. If verified, the tags respond with an encrypted message which

contains its identification information so that the reader verifies their authenticity. If this verification process is successful, the reader responds with more information containing queries about the data to be gathered.

Most authentication solutions addressing this scenario fall into two major categories: server-based solutions and server-less solutions. The authors of [99] point out that most RFID authentication schemes available in the literature (e.g. [69], [75]) are either backend-server-based or server-less (e.g. [100] and [101]). On the one hand, schemes utilizing backend servers entail communication between the RFID readers and the backend server to carry out the operation of the schemes. Essentially, readers transmit data which they gather from RFID tags to the backend servers which in turn verify the authenticity of the readers and tags according to their databases. This constant transmission of data back and forth with the backend server necessities the existence of communication channels between the readers and the backend servers throughout the connection, presenting an authentication problem. Essentially, authentication in these schemes typically targets the channels between the readers and the RFID tags.

The other category of RFID schemes is the server-less based solutions. Server-less RFID architectures are designed so that readers identify tags offline. While this alleviates the need for connecting to external resources via wireless connections, the readers are required to have an AL (Access List) downloaded from a CA (Certification Agency). Such solutions typically suffer from low scalability due to the limited capacities of the reader. In addition, they are known for their low privacy preservation due to the potential of readers to be tampered with either physically or via hacking.

Evolving trends in RFID solutions involve migration to cloud-based environments to further enhance their abilities, adding new challenging dimensions to the already complicated

authentication mechanisms. Neither server-based RFID schemes nor server-less schemes authentication cover the full spectrum of authentication requirements for a cloud-based RFID system. Therefore, our proposed authentication scheme tailored for cloud-based RFID with the traffic/road toll context as a basis of our scheme utilizes a Merkle hash tree to achieve authentication.

8.7   The Proposed RFID Authentication Scheme

Assuming there are several proposed schemes to address authentication between the readers and tags in conventional RFID schemes, the proposed scheme will be more focused on authenticating the communication between the RFID reader and the cloud server in the traffic and toll management/payment context.

Figure 8.9 is a high level depiction of the proposed scheme to solve the authentication problem between the RFID readers and the cloud server utilizing Merkle hash trees (MHT). The basic idea of the MHT scheme is that each node in the tree is a hash function derived by the computation of its children. Two pieces of information are important in the MHT scheme, namely the root node and the authentication path information (API). When the reader reads a tag, it will submit the tag's data along with the API. To ensure the authenticity of the node sending the data, the API is used to calculate the root node, and if the calculated root node matches the stored root node, the authenticity of the data source is verified. By doing this we can ensure authentication from the reader's direction. To ensure the data stored in the cloud is confidential, the data to be sent to the cloud is going to be encrypted using Advanced Encryption Standard (AES) for instance. Thus, T1 would be the Tag ID and the encrypted data.
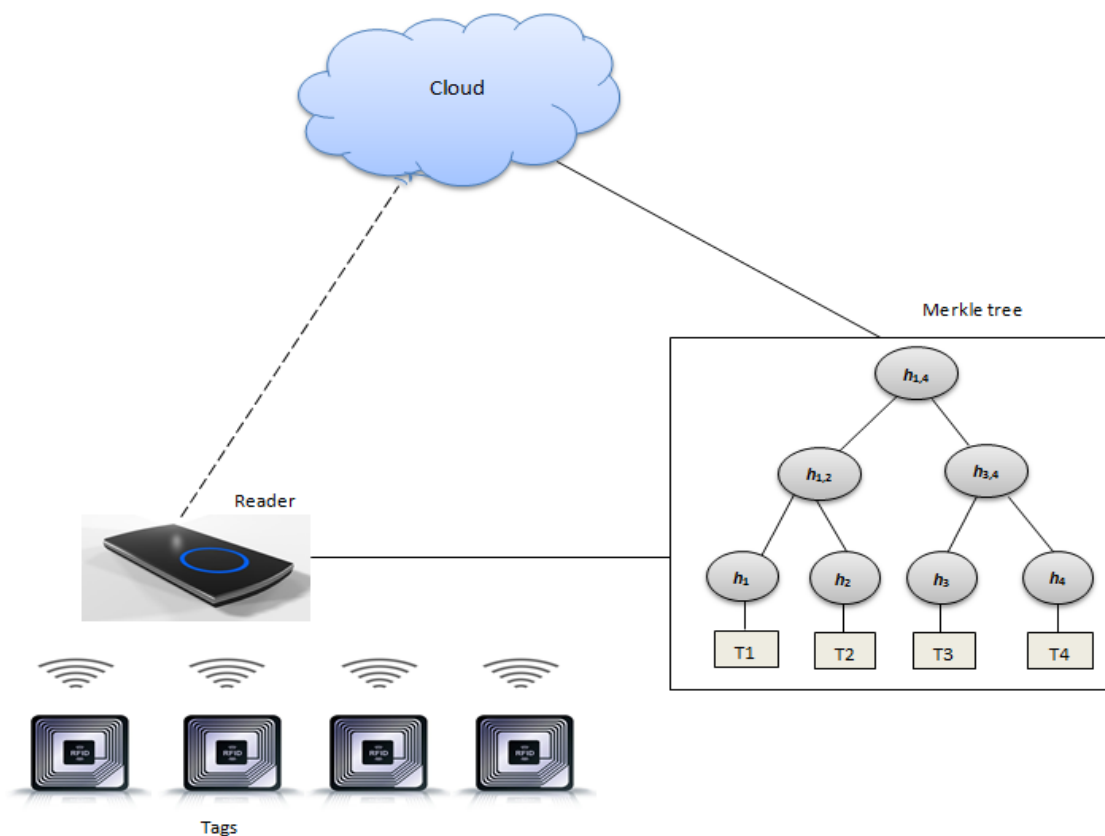
Figure 8-9 Cloud-based RFID tree system

In order to avoid the exponential growth of MHT's if the number of the tags increases, we can set an upper bound on the tree size, thereby reducing computational overhead. Depending on the number of tags, we will have multiple trees each with a unique ID. Each reading of the tags will be accompanied by the corresponding tree ID. For example, if we have a tree with 16 leaf nodes and we need to authenticate a piece of data sent from the reader to the server with API, we only need 4 hash operations to calculate the root node to see if it matches the stored one. By doing so, not only do we aim to lower the computation overhead incurred by the reading and authentication of tags, but we should be able to decrease the size of authentication tree making the search less expensive to perform by the reader. If a reader performs a search within a tree of size 16 as opposed to a tree of size N, where N is the total number of tags in the

database, the search complexity is lowered tremendously. This is especially beneficial in the context of traffic management and toll payment systems, where the readers typically have one chance of reading the RFID tags (i.e. when the vehicle passes through the toll gate at high speeds).

8.8   Protocol Components and performance

Commonly, RFID schemes are composed of three main components: 1) RFID tags, 2) RFID readers and 3) a back-end server, in case it is server-based. Our proposed scheme shares these components but instead of a back-end server owned by the company or a third party known to the respective service provider, we use the cloud as a back-end server. In this case, we aim to secure the communication not only on the readers and RFID tags side but also on the direction between the readers and the cloud. For that reason, the communication between the readers and the cloud is secured by the utilizing the AES protocol which is known for its security and robustness against different types of attacks. We chose AES despite its relatively high cost due to its strength and to the fact that the reader and the cloud can afford its high computation since both are well equipped with storage and powerful resources. In the RFID tags direction, however, and since they typically are much less powerful, as they are usually not capable of performing highly complex algorithms such as AES, we use a Merkle hash tree as depicted in Figure 8.2 (for more information about how authentication is achieved using a Merkle hash tree, please refer to section 8.3.2).

Given the nature of the toll/ traffic management scenario described in the introduction of this problem statement section, which necessitates RFID readers to identify tags on the go at vehicles' highway speeds, searching the whole tree to identify a certain RFID tag upon the passage of a vehicle in the readers' range can be cumbersome. As such, an MHT like this does

not provide scalability. That is because when the tree grows to accommodate more RFID tags, the readers' ability to search the tree to find the appropriate tag becomes difficult, thereby increasing the potential of denial of service attacks. Our proposed scheme addresses this issue by splitting the authentication tree into multiple smaller trees, each with its unique tree ID and its own set of RFID tags.

8.9   Performance Evaluation

In order to show the benefit of the proposed scheme, we show comparisons between the performance of one large MHT as opposed to multiple smaller MHTs. Let us first demonstrate how we calculate the cost of a single MHT in the following:

Number of RFID tags(Leaf nodes)= $N = 2^i$

Number of hash operations to authenticate one RFID tag = i-1

Number of hash operations to authenticate all RFID tags with one MHT= N*(i-1)

This formula is used to determine the cost of authentication in one tree. Note that we use the number of hash operation in a session as an indicator of the cost. Therefore, our main goal is to reduce this cost as much as possible. The computation cost of multiple hash trees is determined according to the following formula:

Using multiple hash trees(MMHT) with a maximum size of k leaf nodes:

Number of MHT=M =$\frac{Number\ of\ RFID\ tags(N)}{k}$

Number of RFID tags (Leaf nodes) in one MHT= $k = 2^j$

Number of hash operations to authenticate one RFID tag= j-1

Number of hash operations to authenticate all RFID tags in MMHT= M*k*(j-1)

Before presenting the comparison results, let us look at how the branching factor influnces the performance of the protocol. Figures 8.12 and 8.13 below demonstrate how the size of the tree impacts the number of hash operations when the numer of RFID tags in the system is 60,000. If we set the tree size at 32, we will end up having 1800 MHTs with in which 240,000 hash operations are required. Conversely, increasing the MHT size increases the number of hash operations slightly, with a much reduced number trees. For an MHT of 128 nodes, we will have 469 trees with a hash cost of 360,000. With an MHT of 64 nodes, we have 938 trees with a hash cost of 300,000.
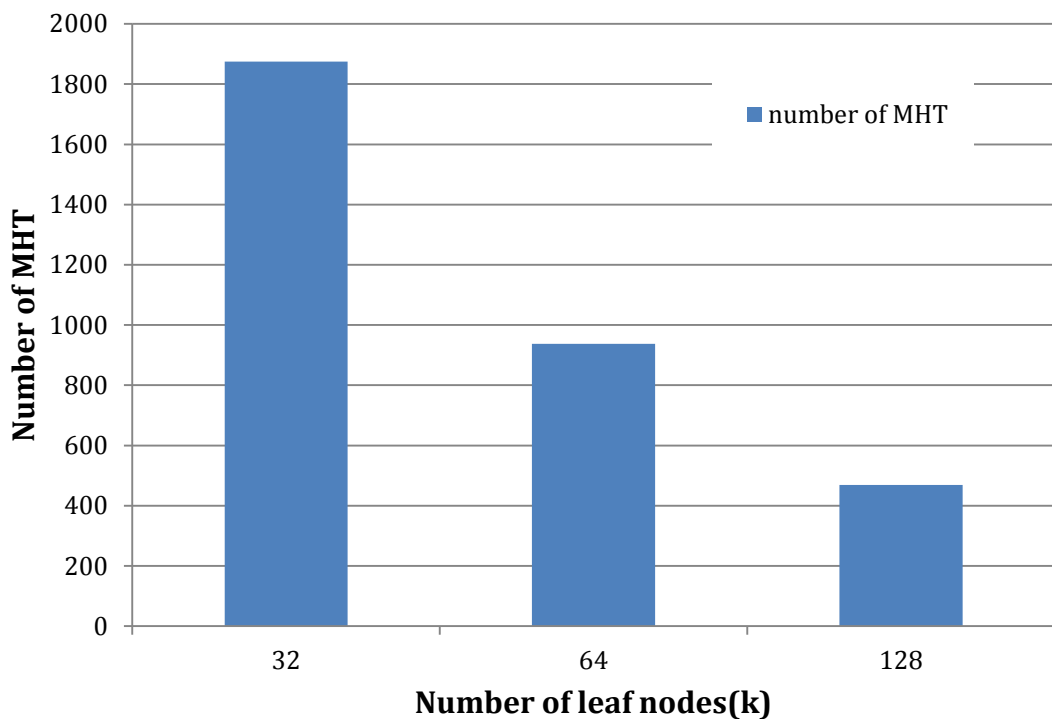
Figure 8-10 Impact of number of nodes on MHTs
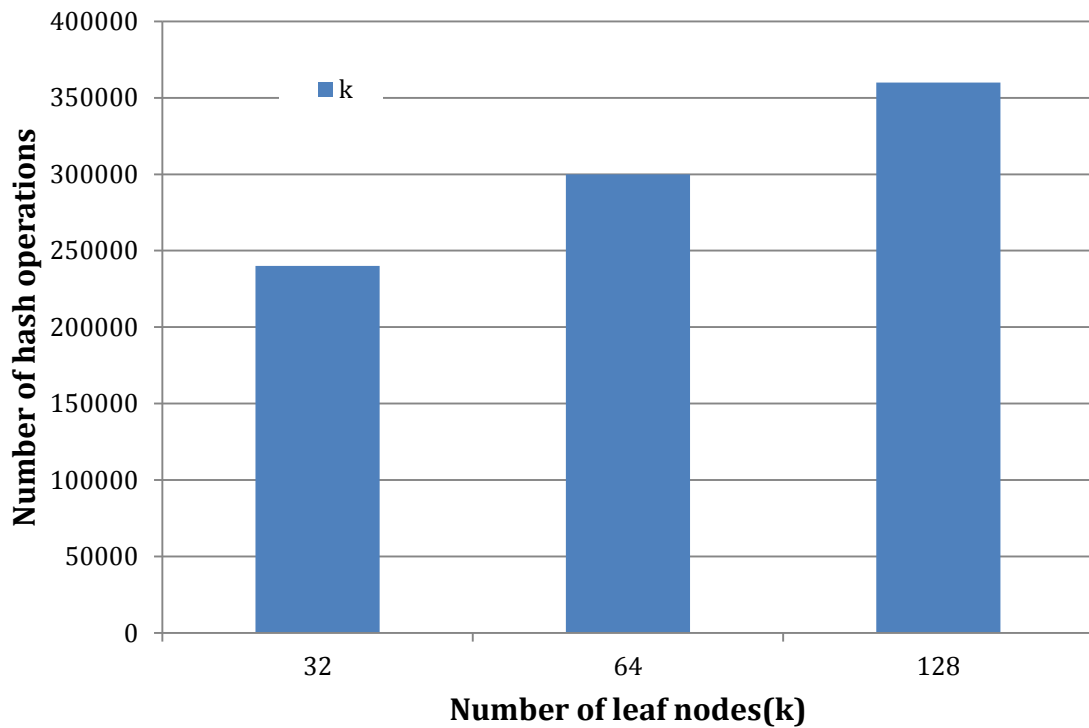
Figure 8-11 Impact of the tree size on hash count

Based on these results, we opted for the MHT size that yields the lowest number of hash operations (i.e. 32) to compare the performance of the proposed protocol with traditional MHT based solutions. In figures 8.14 and 8.15, we show the comparison results. The results below are the comparison of performance of one MHT versus multiple MHTs with 32 leaf node. The performance comparison assumes the whole dataset is to be authenticated. To authenticate a single tag, we will need only 4 hash operations as Figure 8.15 demonstrates.
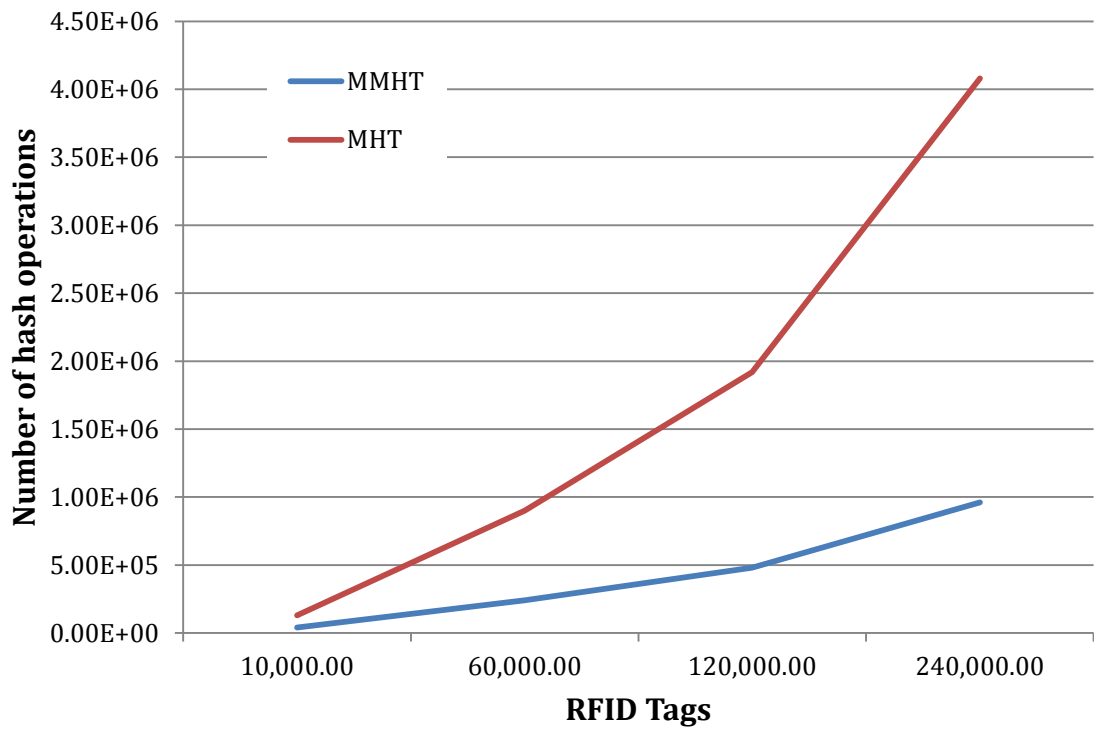
Figure 8-12 Performance comparison between a single MHT and multiple MHTs
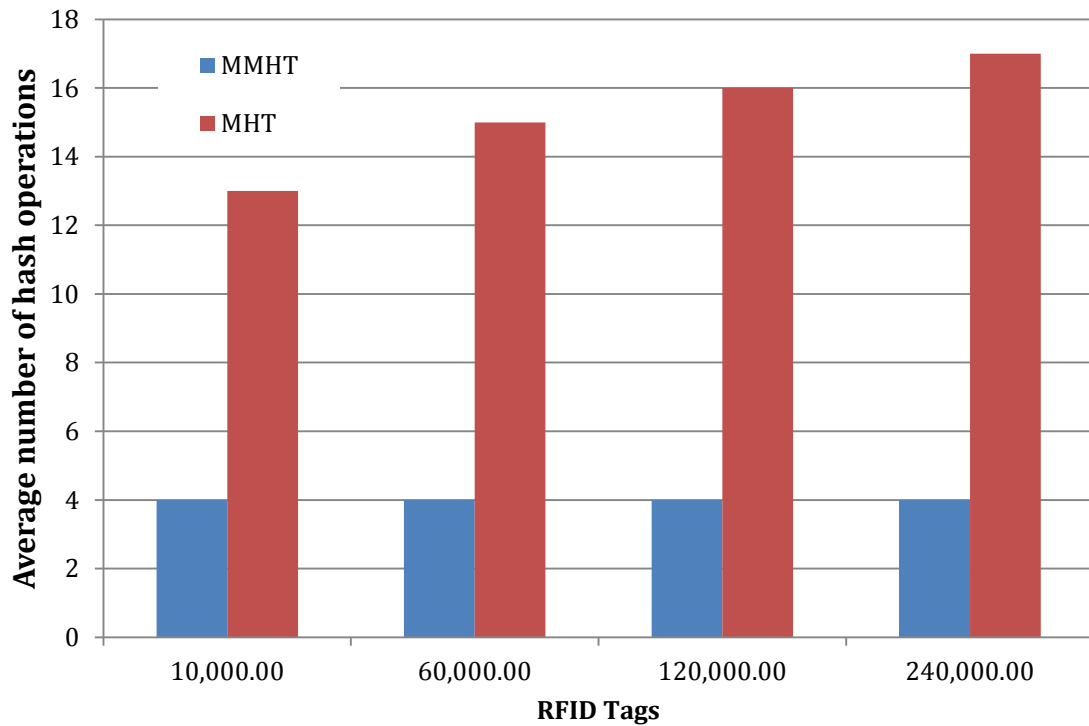
Figure 8-13 Average hash count comparison with different network size

As the comparison results demonstrate in figure 8.14, the divided MHT and the single MHT achieve similar performances when the number of tags in the network is small (i.e. 10,000 RFID tags in the experiment). Earlier this chapter, we have shown that toll/traffic management systems typically have a large number of vehicles and that networks constantly grow in size which means that the RFID tags to be authenticated is much higher. The results indicate that our proposed scheme achieves much lower computation as the number of tags increases when compared to traditional MHT based solutions.

8.10 Conclusion

In this chapter, we presented a keyless authentication scheme designed to suit multi-server authentication environments where single-sign on is adopted. Most previous schemes available in the literature suffer from the lack of bi-directional authentication channels between

service providers and clients. In addition, the fact that authentication can occur utilizing multiple SSL connection requires the authentication keys used to remain constant resulting in the lack of recentness of the keys.

The scheme presented in this paper addresses these issues by proposing the utilization of hash calendars and hash trees to derive authentication keys that can be bi-directionally verified and constantly changing to insure recentness. The proposed scheme was exposed to extensive experiments to measure communication, computation overheads and the total number of hash operations required to carry out the scheme. We also developed an analytical model to derive the optimal configuration of the one-way hash chain used for authentication between the client and service providers.

Also in this chapter, we explored the possibility and benefit of utilizing the Merkle hash tree construction to develop an RFID based solution for authenticating RFID tags in toll/traffic management environment. Due to the high cost associated with MHTs especially with large networks of RFID tags such as the case of toll/traffic management systems, traditional MHTs do not provide scalability. Therefore, we have proposed dividing the MHT into multiple trees to better handle environments like toll/traffic systems where ID identification is needed on the go at highway speeds. We have shown through comparison that the proposed RFID scheme outperforms the traditional MHT based protocols. The improvement in performance is much more significant as the number of RFID tags in the network increases. In our future work, we will focus on determining the ideal size of the tree and the optimal branching factor.

CHAPTER 9: CONCLUSION AND FUTURE WORK

The research in this dissertation has thus far shed some light on seven schemes that have been proposed to address the problem authentication in a variety of environments. In particular, we proposed five schemes to solve the computational overhead associated with one-way hash chain constructions when used to protect session cookies in HTTP connections as well as broadcast and user authentication in wireless sensor networks. In addition, it has introduced a novel solution to tackle the drawback of the uni-directional authentication channel and lack of key recentness in multi-server environments that utilize single sign-on mechanisms by employing a combination of a hash calendar, Merkle hash trees and one-way hash chains. Finally, an authentication scheme tailored for the context of toll/traffic management RFID systems was introduced.

Session cookies are one of the widely used mechanisms to keep browser state between the client and the server. They are, therefore, commonly used to substitute users' identity in Internet sessions as a cheaper alternative to the wide utilization of the secure HTTPS protocol or SSL/TSL based mechanisms which tend to be expensive and incur additional costs that service providers prefer to spare for other expenses. One of the major drawbacks of such practice is the fact that session cookies are stored and communicated in plaintext, thereby exposing critical private information about the user and about the services being used during that session. One common threat resulting from this practice is the hijacking of session cookies and other identifiers that belong to the user.

Much of the research addressing the issue of session hijacking is very much preliminary and up to this point the majority of session cookies are communicated in the air without proper

protection. A major recent security breach has resulted in exposing of critical financial and banking information of more than 40 million customers who have shopped at Target Stores in the United States during the months of November and December of 2013 [102]. Although it is early to determine whether this breach can be attributed to hijacking session cookies or not, the author of [102] suspects the incident took place because of debit personal identification numbers interception process that targeted the giant Target retailer stores. In a different incident, the author of [103] talks about an attacker who claims he was able to steal session cookies pertaining to Yahoo accounts and selling them to the public, and offers his services to hijack more accounts by launching cross-site scripting attacks (XSS). These attacks according to [103] let attackers steal cookies from Yahoo! Webmail users.

With attacks as easy as interception of cookies and consequences as severe as exposing an excess of 40 million customers' financial information, addressing issues of Internet, and especially session cookies security, is an ever growing necessity. A new direction of research in [10] suggested devising Lamport's one-time passwords [38] as a means with which cookies can be made secure. We described in the previous chapters that this promising solution offers a great potential and help avoid a lot of security threats seen in today's Internet. It is well known that the computational overhead resulting from the need to calculate the hash values of the one-way hash chain in the first transaction can be high and unmanageable by devices with modest memory and energy capabilities, and sometimes undesired by even the highly capable devices simply because we do not want to add burdens on our computers. Having this in mind, we designed five novel schemes that will aid in lowering this computational overhead while at the same time benefiting from the excellent security features of one-way hash functions. We have demonstrated with

extensive experiments that our solutions achieved better performance that the straightforwardly configured hash chains.

Another equally important issue that this dissertation has addressed is the context of multi-server environments where single-sign on mechanisms provide convenience for customers and cost effectiveness for providers at the potential cost of breached security. Scholars have pointed out that due to flaws encountered in the logic of common SSO solutions such as XAML 2.0 and OpenID, attackers can possibly launch attacks such as impersonation and cross-scripting targeting weaknesses that result from the unidirectional nature of the authentication channel between clients and service providers and the lack of guarantees of recent keys. With these weaknesses in mind, we have developed a keyless signature scheme that achieves authentication channel bi-directionality between the client and the service provider and a recent authentication key by periodically updating authentication credentials.

The last proposed scheme targets the context of toll/traffic management systems that utilize RFID tags. An authentication scheme employing a modified Merkle hash tree is suggested to address authentication of RFID tags in this contexts. The Merkle hash tree modification proposed is aims to achieve scalability and computational economy while not sacrificing strong authentication.

9.1   Future Work

The work in this dissertation can be extended and improved in a variety of ways. For example, in the context of the sparse caching solution proposed to avoid session hijacking attacks, a possible extension can be made by measuring real network traffic and finding accurate statistics about the number of transactions a typical user session is. This can be achieved by looking at more specific contexts such as medical settings where physicians and nurses typically

have a relatively tangible behavior. Another possibility to extend our work in this scheme is investigate the cost of session re-initialization as it is important to measure such construct to validate the result obtained.

While our solutions have been targeting cookies in HTTP connections, a possible extension of the works reported in this dissertation is to compare their performance with the utilization of SSL based connections throughout user session. However, in order to do that, it is imperative to implement and evaluate the proposed schemes in real networks before moving to the next step of comparing their potential benefits relative to SSL based channels.

As far as the SSO authentication scheme, our future work includes comparing the results obtained in our simulation with other schemes available in the literature. This will require analyzing and interpreting current SSO authentication schemes and comparing them with the proposed keyless signature scheme.

Finally, the work in the tree-based toll/traffic RFID management system can be extended by analyzing behavior in RFID networks and ultimately finding the appropriate setup of the Merkle hash tree suitable for this environment. This will result in a more scalable scheme; one that is attractive for developers in this arena.

REFERENCES

[1]   M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," 1979.

[2]   D. Liu and P. Ning, "Multilevel μTESLA: Broadcast authentication for distributed sensor networks," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 3, pp. 800-836, 2004.

[3]   H. Tan, J. Zic, S. Jha, and D. Ostry, "Secure multihop network programming with multiple one-way key chains," *Mobile Computing, IEEE Transactions on,* vol. 10, pp. 16-31, 2011.

[4]   I. Khalil, S. Bagchi, C. N. Rotaru, and N. B. Shroff, "UnMask: Utilizing neighbor monitoring for attack mitigation in multihop wireless sensor networks," *Ad Hoc Networks,* vol. 8, pp. 148-164, 2010.

[5]   M. Li, S. Yu, J. D. Guttman, W. Lou, and K. Ren, "Secure ad hoc trust initialization and key management in wireless body area networks," *ACM Transactions on Sensor Networks (TOSN),* vol. 9, p. 18, 2013.

[6]   T.-H. Chen, H.-C. Hsiang, and W.-K. Shih, "Security enhancement on an improvement on two remote user authentication schemes using smart cards," *Future Generation Computer Systems,* vol. 27, pp. 377-380, 2011.

[7]   C.-T. Li and M.-S. Hwang, "An efficient biometrics-based remote user authentication scheme using smart cards," *Journal of Network and Computer Applications,* vol. 33, pp. 1-5, 2010.

[8]   X. Dai and J. Grundy, "NetPay: An off-line, decentralized micro-payment system for thin-client applications," *Electronic Commerce Research and Applications,* vol. 6, pp. 91-101, 2007.

[9]   H.-T. Liaw, J.-F. Lin, and W.-C. Wu, "A new electronic traveler's check scheme based on one-way hash function," *Electronic Commerce Research and Applications,* vol. 6, pp. 499-508, 2008.

[10]   I. Dacosta, S. Chakradeo, M. Ahamad, and P. Traynor, "One-time cookies: Preventing session hijacking attacks with disposable credentials," *ACM Transactions on Internet Technology (TOIT),* vol. 12, 2012.

[11] CSID, "CONSUMER SURVEY: PASSWORD HABITS A study of password habits among American consumers," September 2012.

[12] A. Alabrah, J. Cashion, and M. Bassiouni, "Enhancing security of cookie-based sessions in mobile networks using sparse caching," *International Journal of Information Security - Springer Publishing*, Vol. 13, No. 4, July 2014, pp. 355–366, online version published December 2013, DOI: 10.1007/s10207-013-0223-8.

[13] A. Alabrah and M. Bassiouni, "A hierarchical two-tier one-way hash chain protocol for secure internet transactions," in *Proceedings of IEEE Global Communications Conference (GLOBECOM'12),* Anaheim, California, December 3-7, 2012, pp. 868-873.

[14] A. Alabrah and M. Bassiouni, "Robust and fast authentication of session cookies in collaborative and social media using position-indexed hashing," in *Proceeding of IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (COLLABORATECOM 2013), Austin, Texas, October 20–23, 2013, pp. 241-249.*

[15] A. Alabrah and M. Bassiouni, "Efficient User and Broadcast Authentication Scheme for WSNs," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'14)*, Istanbul, Turkey, April 6-9, 2014.

[16] A. Alabrah and M. Bassiouni, "Preventing session hijacking in collaborative applications with hybrid cache-supported one-way hash chains," in *Proceedings of IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (COLLABORATECOM 2014),* Miami, Florida, October 22–25, 2014.

[17] A. Alabrah and M. Bassiouni, "A Tree-Based Authentication Scheme for a Cloud Toll/traffic RFID System, submitted for publication," in *Proceedings of IEEE Vehicular Networking Conference (VNC'14)*, Paderborn, Germany 2014.

[18] T. OWASP, "10 2010," *The Ten Most Critical Web Application Security Risks,* 2010.

[19] D. Wichers, "The 2013 OWASP Top 10," in *AppSec USA 2013*, 2013.

[20] B. Ponurkiewicz, "FaceNiff—A new Android download application," ed, 2012.

[21] E. Butler, "FireSheep: Cookie Snatching Made Simple," in *ToorCon Conference, San Diego, CA*, 2010, pp. 22-24.

[22]   R. D. Riley, N. M. Ali, K. S. Al-Senaidi, and A. L. Al-Kuwari, "Empowering users against sidejacking attacks," in *ACM SIGCOMM Computer Communication Review*, 2010, pp. 435-436.

[23]   J.-C. Chen, M.-C. Jiang, and Y.-W. Liu, "Wireless LAN security and IEEE 802.11 i," *Wireless Communications, IEEE,* vol. 12, pp. 27-36, 2005.

[24]   C. Sreedhar, S. M. Verma, and N. Kasiviswanath, "A survey on security issues in wireless ad hoc network routing protocols," *International Journal on Computer Science and Engineering,* vol. 2, pp. 224-232, 2010.

[25]   M. S. Siddiqui and C. S. Hong, "Security issues in wireless mesh networks," in *Multimedia and Ubiquitous Engineering, 2007. MUE'07. International Conference on*, 2007, pp. 717-722.

[26]   Y. Zhou, Y. Fang, and Y. Zhang, "Securing wireless sensor networks: a survey," *Communications Surveys & Tutorials, IEEE,* vol. 10, pp. 6-28, 2008.

[27]   K. Hickman and T. Elgamal, "The SSL protocol," *Netscape Communications Corp,* vol. 501, 1995.

[28]   T. Dierks, "The transport layer security (TLS) protocol version 1.2," 2008.

[29]   E. Rescorla and A. Schiffman, "The secure hypertext transfer protocol," 1999.

[30]   C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of TLS Web servers," *ACM Transactions on Computer Systems (TOCS),* vol. 24, pp. 39-69, 2006.

[31]   D. M. Kristol and L. Montulli, "HTTP state management mechanism," 2000.

[32]   D. M. Kristol, "HTTP Cookies: Standards, privacy, and politics," *ACM Transactions on Internet Technology (TOIT),* vol. 1, pp. 151-198, 2001.

[33]   L. Montulli, "Method of on-line shopping utilizing persistent client state in a hypertext transfer protocol based client-server system," ed: Google Patents, 1998.

[34]   B. Adida, "Sessionlock: securing web sessions against eavesdropping," in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 517-524.

[35]   A. X. Liu, J. M. Kovacs, C.-T. Huang, and M. G. Gouda, "A secure cookie protocol," in *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, 2005, pp. 333-338.

[36]   G. Pujolle, A. Serhrouchni, and I. Ayadi, "Secure session management with cookies," in *Information, Communications and Signal Processing, 2009. ICICS 2009. 7th International Conference on*, 2009, pp. 1-6.

[37]   M. Wu, S. Garfinkel, and R. Miller, "Secure web authentication with mobile phones," in *DIMACS workshop on usable privacy and security software*, 2004, pp. 9-10.

[38]   L. Lamport, "Password authentication with insecure communication," *Communications of the ACM,* vol. 24, pp. 770-772, 1981.

[39]   N. Haller, "The S/KEY one-time password system," 1995.

[40]   Y. Zhang and Y. Fang, "ARSA: an attack-resilient security architecture for multihop wireless mesh networks," *Selected Areas in Communications, IEEE Journal on,* vol. 24, pp. 1916-1928, 2006.

[41]   Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," *Wireless Networks,* vol. 11, pp. 21-38, 2005.

[42]   Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks,* vol. 1, pp. 175-192, 2003.

[43]   S. J. Murdoch, "Hardened stateless session cookies," in *Security Protocols XVI*, ed: Springer, 2011, pp. 93-101.

[44]   J. Cashion and M. Bassiouni, "Robust and low-cost solution for preventing sidejacking attacks in wireless networks using a rolling code," in *Proceedings of the 7th ACM symposium on QoS and security for wireless and mobile networks*, 2011, pp. 21-26.

[45]   Z. Benenson, N. Gedicke, and O. Raivio, "Realizing robust user authentication in sensor networks," *Real-World Wireless Sensor Networks (REALWSN),* vol. 14, 2005.

[46]   I. Butun and R. Sankar, "Advanced two tier user authentication scheme for heterogeneous wireless sensor networks," in *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, 2011, pp. 169-171.

[47] R. Wang, W. Du, X. Liu, and P. Ning, "ShortPK: A short-term public key scheme for broadcast authentication in sensor networks," *ACM Transactions on Sensor Networks (TOSN),* vol. 6, p. 9, 2009.

[48] K. Arikumar and K. Thirumoorthy, "Improved user authentication in wireless sensor networks," in *Emerging Trends in Electrical and Computer Technology (ICETECT), 2011 International Conference on*, 2011, pp. 1010-1015.

[49] O. Cheikhrouhou, A. Koubaa, M. Boujelben, and M. Abid, "A lightweight user authentication scheme for Wireless Sensor Networks," in *Computer Systems and Applications (AICCSA), 2010 IEEE/ACS International Conference on*, 2010, pp. 1-7.

[50] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services," in *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, pp. 365-379.

[51] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti, "An authentication flaw in browser-based Single Sign-On protocols: Impact and remediations," *Computers & Security,* vol. 33, pp. 41-58, 2013.

[52] W.-B. Lee and C.-C. Chang, "User identification and key distribution maintaining anonymity for distributed computer networks," *Comput Syst Sci Eng,* vol. 15, pp. 211-214, 2000.

[53] T.-S. Wu and C.-L. Hsu, "Efficient user identification scheme with key distribution preserving anonymity for distributed computer networks," *Computers & Security,* vol. 23, pp. 120-125, 2004.

[54] Y. Yang, S. Wang, F. Bao, J. Wang, and R. H. Deng, "New efficient user identification and key distribution scheme providing enhanced security," *Computers & Security,* vol. 23, pp. 697-704, 2004.

[55] Y.-P. Liao and S.-S. Wang, "A secure dynamic ID based remote user authentication scheme for multi-server environment," *Computer Standards & Interfaces,* vol. 31, pp. 24-29, 2009.

[56] C.-C. Lee, T.-H. Lin, and R.-X. Chang, "A secure dynamic ID based remote user authentication scheme for multi-server environment using smart cards," *Expert Systems with Applications,* vol. 38, pp. 13863-13870, 2011.

[57] T.-T. Truong, M.-T. Tran, and A.-D. Duong, "Robust Secure Dynamic ID Based Remote User Authentication Scheme for Multi-server Environment," in *Computational Science and Its Applications–ICCSA 2013*, ed: Springer, 2013, pp. 502-515.

[58] Q. Pei and J. Yu, "A Secure Dynamic Identity based Single Sign-On Authentication Protocol," *Journal of Software,* vol. 9, pp. 154-161, 2014.

[59] P. Hallam-Baker, "Security assertions markup language," *May,* vol. 14, pp. 1-24, 2001.

[60] S. Cantor, I. J. Kemp, N. R. Philpott, and E. Maler, "Assertions and protocols for the oasis security assertion markup language," *OASIS Standard (March 2005),* 2005.

[61] O. Authentication, "2.0-Final," *OpenID Foundation website,* 2007.

[62] A. Buldas, A. Kroonmaa, and R. Laanoja, "Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees," in *Secure IT Systems*, ed: Springer, 2013, pp. 313-320.

[63] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology—CRYPTO'87*, 1988, pp. 369-378.

[64] L. TransCore. (07/01). *All Electronic Tolling* Available: https://http://www.transcore.com/tolling-systems - 0cc1c3a90eb92a65c3a914ae2f793446

[65] O.-O. C. E. Authority, "Saving Time, Connecting Communities for Fifty Years.," Annual Financial Report 06/30/2013 2013.

[66] E.-Z. I. Group. (2013, 07/01). *Statistics*

[67] K. Kamarulazizi and D. W. Ismail, "Electronic toll collection system using passive RFID technology," *Journal of Theoretical and Applied Information Technology,* vol. 2, 2010.

[68] Z.-h. Xiao, Z.-q. Guan, and Z.-h. Zheng, "The research and development of the highway's electronic toll collection system," in *Knowledge Discovery and Data Mining, 2008. WKDD 2008. First International Workshop on*, 2008, pp. 359-362.

[69] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems-CHES 2004*, ed: Springer, 2004, pp. 357-370.

[70]   W. Wu and L. Zhang, "LBlock: a lightweight block cipher," in *Applied Cryptography and Network Security*, 2011, pp. 327-344.

[71]   Z. Gong, S. Nikova, and Y. W. Law, "KLEIN: a new family of lightweight block ciphers," in *RFID. Security and Privacy*, ed: Springer, 2012, pp. 1-18.

[72]   K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: an ultra-lightweight blockcipher," in *Cryptographic Hardware and Embedded Systems– CHES 2011*, ed: Springer, 2011, pp. 342-357.

[73]   B. Song and C. J. Mitchell, "Scalable RFID security protocols supporting tag ownership transfer," *Computer Communications,* vol. 34, pp. 556-566, 2011.

[74]   G. Kapoor, W. Zhou, and S. Piramuthu, "Multi-tag and multi-owner RFID ownership transfer in supply chains," *Decision Support Systems,* vol. 52, pp. 258-270, 2011.

[75]   J.-S. Cho, S.-S. Yeo, and S. K. Kim, "Securing against brute-force attack: A hash-based RFID mutual authentication protocol using a secret value," *Computer Communications,* vol. 34, pp. 391-397, 2011.

[76]   A. Gupta, W.-D. Weber, and T. Mowry, *Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes**: Springer, 1992.

[77]   A. Deftu and A. Murarasu, "Optimization Techniques for Dimensionally Truncated Sparse Grids on Heterogeneous Systems," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, 2013, pp. 351-358.

[78]   W. Lau, M. Kumar, and S. Venkatesh, "A cooperative cache architecture in support of caching multimedia objects in MANETs," in *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, 2002, pp. 56-63.

[79]   C. C. Douglas, J. Hu, M. Iskandarani, M. Kowarschik, U. Rüde, and C. Weiss, "Maximizing cache memory usage for multigrid algorithms for applications of fluid flow in porous media," in *Numerical Treatment of Multiphase Flows in Porous Media*, ed: Springer, 2000, pp. 124-137.

[80]   Y.-C. Hu, M. Jakobsson, and A. Perrig, "Efficient constructions for one-way hash chains," in *Applied Cryptography and Network Security*, 2005, pp. 423-441.

[81]   R. Chandramouli, S. Bapatla, K. Subbalakshmi, and R. Uma, "Battery power-aware encryption," *ACM Transactions on Information and System Security (TISSEC),* vol. 9, pp. 162-180, 2006.

[82]   N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "Analyzing the energy consumption of security protocols," in *Proceedings of the 2003 international symposium on Low power electronics and design*, 2003, pp. 30-35.

[83]   J. Cashion and M. Bassiouni, "Protocol for mitigating the risk of hijacking social networking sites," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, 2011, pp. 324-331.

[84]   M. Stevens, "Fast Collision Attack on MD5," *IACR Cryptology ePrint Archive,* vol. 2006, p. 104, 2006.

[85]   S. Chen and C. Jin, "An improved collision attack on MD5 algorithm," in *Information Security and Cryptology*, 2008, pp. 343-357.

[86]   X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD," in *Advances in Cryptology–EUROCRYPT 2005*, ed: Springer, 2005, pp. 1-18.

[87]   X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology–CRYPTO 2005*, 2005, pp. 17-36.

[88]   Y. Liu, J. Li, and M. Guizani, "PKC Based Broadcast Authentication using Signature Amortization for WSNs," *Wireless Communications, IEEE Transactions on,* vol. 11, pp. 2106-2115, 2012.

[89]   A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. E. Culler, "SPINS: Security protocols for sensor networks," *Wireless networks,* vol. 8, pp. 521-534, 2002.

[90]   D. Liu and P. Ning, "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks," in *NDSS*, 2003.

[91]   K. H. Wong, Y. Zheng, J. Cao, and S. Wang, "A dynamic user authentication scheme for wireless sensor networks," in *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, 2006, p. 8 pp.

[92]   M. L. Das, "Two-factor user authentication in wireless sensor networks," *Wireless Communications, IEEE Transactions on,* vol. 8, pp. 1086-1090, 2009.

[93]   M. K. Khan and K. Alghathbar, "Cryptanalysis and security improvements of 'two-factor user authentication in wireless sensor networks'," *Sensors,* vol. 10, pp. 2450-2459, 2010.

[94]   B. Vaidya, D. Makrakis, and H. T. Mouftah, "Improved two-factor user authentication in wireless sensor networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2010 IEEE 6th International Conference on*, 2010, pp. 600-606.

[95]   Y. Cao, Y. Shoshitaishvili, K. Borgolte, C. Kruegel, G. Vigna, and Y. Chen, "Protecting Web-based Single Sign-on Protocols against Relying Party Impersonation Attacks through a Dedicated Bi-directional Authenticated Secure Channel," in *Research in Attacks, Intrusions and Defenses*, ed: Springer, 2014, pp. 276-298.

[96]   K. Kant, R. Iyer, and P. Mohapatra, "Architectural impact of secure socket layer on internet servers," in *Computer Design, 2000. Proceedings. 2000 International Conference on*, 2000, pp. 7-14.

[97]   L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan, "Anatomy and performance of SSL processing," in *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, 2005, pp. 197-206.

[98]   R. A. Hauslen, "The promise of automatic vehicle identification," *Vehicular Technology, IEEE Transactions on,* vol. 26, pp. 30-38, 1977.

[99]   W. Xie, L. Xie, C. Zhang, Q. Zhang, and C. Tang, "Cloud-based RFID authentication," in *RFID (RFID), 2013 IEEE International Conference on*, 2013, pp. 168-175.

[100]   C. F. Lee, H. Y. Chien, and C. S. Laih, "Server‑less RFID authentication and searching protocol with enhanced security," *International Journal of Communication Systems,* vol. 25, pp. 376-385, 2012.

[101]   B. Wang and M. Ma, "A server independent authentication scheme for RFID systems," *Industrial Informatics, IEEE Transactions on,* vol. 8, pp. 689-696, 2012.

[102]   B. Krebs. (2013, 31). *18 Sources: Target Investigating Data Breach*. Available: http://krebsonsecurity.com/2013/12/sources-target-investigating-data-breach/

[103]   B. Krebs. (2012, 31). *23 Yahoo Email-Stealing Exploit Fetches $700*. Available: http://krebsonsecurity.com/2012/11/yahoo-email-stealing-exploit-fetches-700/