USING IDENTITY-BASED PRIVACY-PROTECTED ACCESS CONTROL FILTER (IPACF) TO AGAINST DENIAL OF SERVICE ATTACKS AND PROTECT USER PRIVACY

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Chun-Ching Andy Huang

Certificate of Approval:

J. David Irwin Professor Electrical and Computer Engineering	Chwan-Hwa Wu, Chair Professor Electrical and Computer Engineering
Fa Foster Dai Associate Professor Electrical and Computer Engineering	Stephen L. McFarland Acting Dean Graduate School

USING IDENTITY-BASED PRIVACY-PROTECTED ACCESS CONTROL FILTER (IPACF) TO AGAINST DENIAL OF SERVICE ATTACKS AND PROTECT USER PRIVACY

Chun-Ching Andy Huang

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
August 7, 2006

USING IDENTITY-BASED PRIVACY-PROTECTED ACCESS CONTROL FILTER

(IPACF) TO AGAINST DENIAL OF SERVICE ATTACKS

AND PROTECT USER PRIVACY

Chun-Ching Andy Huang

____
Signature of Author

____
Date of Graduation

VITA

Chun-Ching Andy was born in December 1978, in Taipei, Taiwan. He graduated with Bachelor of Science degree in Computer Science from University Of British Columbia, Vancouver, Canada in 2003. Chun-Ching Andy Huang then entered Graduate School, Auburn University, in January 2004.

THESIS ABSTRACT

USING IDENTITY-BASED PRIVACY-PROTECTED ACCESS CONTROL FILTER (IPACF) TO AGAINST DENIAL OF SERVICE ATTACKS AND PROTECT USER PRIVACY

Chun-Ching Andy Huang

Master of Science, August 7, 2006
(B.S., University Of British Columbia, Canada, 2003)

75 Typed Pages

Directed by Chwan-Hwa Wu

Denial of service (DoS)/Distributed DoS (DDoS) attack is an eminent threat to an authentication server, which is used to guard access to firewalls, virtual private networks and wired/wireless networks. The major problem is that an authentication server needs to verify whether a request is from a legitimate user and if intensive computation and/or memory resources are needed for verifying a request, then DoS/DDoS attack is feasible. In this thesis, a new protocol called Identity-Based Privacy-Protected Access Control Filter (IPACF) is proposed to counter DoS/DDoS attack. This protocol is an improvement of IDF (Identity-Based Dynamic Access Control Filter). The proposed protocol is stateless because it does not create a state for an authentication request unless the request is from a legitimate user. Moreover, the IPACF is stateless for both user and authentication server since a user and responder authenticate each other. A filter value, which is generated by pre-shared secrets, is sent in a frame and checked to see if the request is legitimate. Note that the process of checking filter value is not intensive computation. The filter value is tabulated in a table with user identity so that a filter value represents a user's identity and only the legitimate user and authentication server can figure out the identity. When a filter value is from a legitimate source, a new filter value will be generated for the next frame. Consequently, the filter value is changed for every frame. Thus the privacy of both user and server are protected.

The IPACF is implemented for both user and authentication server. The performance of the implementation is reported in this thesis. In order to counter more DoS/DDoS attacks that issue fake requests, parallel processing technique is used to implement the authentication server, which is divided into server 1 and server 2. Server 1 only checks the validity of the request filter value against the filter value table. If the request is legitimate, the request will be passed to server 2 for generating a new filter value; otherwise, the fake request is rejected by server 1. The performance comparison of dual server and single server is also reported.

ACKNOWLEDGMENTS

This thesis could not be done without the following individuals. First, I would like to thank my major advisor Dr. Chwan-Hwa Wu. Dr. Wu gives me great learning experiences. I could not finish this thesis with his guidance during my research. He also turns me into a more mature person. Second, I would like to thank Dr. J. David Irwin for giving me the great opportunity to be here and full support of my Master study. He always encourages me on my researches. I would also like to thank Dr. Fa Dai. He always concerns about my research every time we meet and gives me some feedbacks.

For the past two years of study in Auburn University, it's really a pleasure to be part of the research lab. Finally, I would like to dedicate this thesis to my parents Shun-Ming Huang and Chin-Hui Kao, my brother Chun-Ju Huang, and my fiancée Szu-Hua Chen for their continuously support and endless love.

Style manual or journal used: Bibliography conforms to the Institute of Electrical and Electronics Engineers

Computer software used: Microsoft Word 2003, Microsoft Visio 2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

Authentication server is widely used to guard wireless access points, virtual private networks, firewalls and so on. Denial of service (DoS) or distributed DoS (DDoS) attacks floods the authentication server with fake packets and causes the server to exhaust its resources for processing fake packets. When the resources of the authentication server is exhausted, legitimate user's authentication requests can not be responded.

In this thesis, we propose a stateless authentication protocol, which allows the authentication server to check if the request is from a legitimate user and to commit computational resources only if the request is legitimate. This protocol protects the resources of an authentication server so that the resource consumption of DoS/DDoS attacks is minimized.

1.1 Current Problem

The advancement in wireless network technology provides wireless access networks to the Internet. Authentication servers are used to defend the wireless access networks. Attackers can easily launch DoS/DDoS attacks to authentication servers and can disable the wireless access networks. The current 802.1x [1] and 802.11i [2] standards both still lack the ability to prevent the DoS/DDoS attacks to authentication servers and wireless access points. Firewalls and virtual private networks suffer the same drawback that DoS/DDoS can easily halt the authentication functions [3].

The major problem of DoS/DDoS attack is that the authentication server (or responder) needs to validate that the request is from a legitimate user (or initiator). However, the authentication server only has a limited CPU computation power and restricted amount of memory. When attackers initiate sufficient requests, the authentication server cannot respond to legitimate requests. It is necessary to minimize the resource committed to a request before verifying the request is from a legitimate source.

Ingress and egress filters based on IP address have been suggested to counter DoS/DDoS attacks. But a legitimate source IP address can be spoofed to launch DoS/DDoS attacks by sniffing the communication frames between the legitimate user and authentication server.

1.2 Motivation of the Proposed IPACF protocol

To address the current problems, a protocol called Identity-Based Privacy-Protected Access Control filter (IPACF) is proposed. An identity based filtering is used instead of an IP address-based filtering. The IPACF is based on pre-shared secrets that are known by a user (initiator) and an authentication server (responder). Both user and responder generate a unique (one-time) filter value for each communication frame using pre-shared secrets. Only the user and responder have the necessary secrets to calculate the filter values. The filter value can be checked to make sure that the value is from a legitimate source. If the filter value is correct, then a new filter value is generated for the next frame; otherwise, the received frame is rejected.

Follow the concept of a stateless protocol "*Do not create any state or do expensive computation before you can ensure that the received frame is legitimate.*" the concept of stateless server and stateless connection was proposed to defend against DoS. If the server authenticates the user and verifies the user's filter value without keeping states, the authentication scheme will be stateless. Moreover, the protocol is stateless since a frame with an incorrect filter value is only checked and no other computation resource is committed. Only when the frame has a legitimate filter value, then resource will be committed for calculating next filter value.

The privacy of the identity relies on the one-time filter value and the pseudo ID which are an indication of the user identity. Furthermore, the IPACF can prevent session hijacking, dictionary attacks and man-in-the-middle attacks using the secure master key exchange.

1.3 Organization of the Thesis

This thesis is divided into six chapters. The first chapter provides the background information that is necessary to understand the rest of the work. The remaining chapters are organized as follows.

In Chapter 2, we present the related work and current research to show the importance of proposed IPACF protocol. In Chapter 3, we propose a concept and the design goals for the new IPACF protocol. We also point out a couple of disadvantage and drawbacks about Wang's IDF protocol [4]. In Chapter 4, we explain the details of how the proposed IPACF protocol creates a truly stateless protocol environment and defends against DoS/DDoS attacks. In Chapter 5, we implemented and simulated

Wang’s IDF protocol in the real-world implementation. We also describe the implementation and performance of the IPACF protocol and compare it with Wang’s IDF protocol; the compatibility of the IPACF protocol in IPv6 network is also described. Chapter 6 contains the conclusions and suggestions for future work.

CHAPTER TWO

RELEATED WORKS

2.1 Denial of Service (DoS)

The objective of DoS attack is to degrade services by flooding a network with faulty beacons, preventing legitimate traffic and causing systems not to respond. Denial of service relies on methods that exploit the weaknesses of the network and attempts to reduce the ability of a responder to service users [5].

DoS can be achieved by either overloading the ability of the target network, that causing a responder to neglect incoming traffic, or by sending network packets that causing target networks to behave unpredictably and crash. For example, one common form of DoS is Ping of Death, which generates and sends certain kinds of network messages that are technically unsupported but known to cause problems for systems that receive them. Other DoS attacks may simply "flood" a network with useless data traffic, rendering systems incapable by pretending as legitimate users. The main problem is: when a responder receives a request that requires verifying the request using system resources, the computation and memory storage gives an adversary a chance to launch a DoS attack.

2.2 Current research

DoS attacks continue to be a critical threat. They can intervene critical services, prevent data transfer between devices, and decrease overall productivity. Because DoS attacks can be extraordinarily costly and harmful to internetworking environment, networks must proactively counteract these attacks. Authentication is a significant issue in establishing secure communication between the users and responders by identifying each other prior to accepting each other's frames. Several works [8-14] have been proposed to prevent DoS attacks and to improve the security or the computational performance based on public-key signatures authentication scheme. Two password-based integrated schemes for user authentication and access control was proposed by Jan and Tseng [13], defined as the JT-1 and the JT-2 schemes, to administer the security administration functions with respect to both the computational cost and the communicational overhead efficiency. But both the JT-1 and the JT-2 schemes are not secure against an impersonation attack, an adversary can successfully fool the system to act as any other legitimate user, and take over all access rights granted to that user without being detected [14]. W. Aiello *et al* [8] provides a capability for perfect forward secrecy in order to efficiently defend against DoS attacks. Although the public-key signatures scheme can provide a certain level of security to resist a DoS attack, it is still vulnerable to DoS attacks on the first and third message flows [8][10]. Since the first flow is sent in clear text, the adversary can sniff the legitimate user's identity and then spoof it to become a legitimate user and pass the identity check from the server. This process could lead to a DDoS attack. If the server need not check the identity of the user like J. Leiwo's scheme [10], the adversary can randomly generate a request nonce to the server for launching a DoS attack. This will cause the responder very busy in processing the spoofed requests sent from an adversary and has limited ability for legitimate users. In the third flow, the public-key signatures authentication scheme must verify the authenticity of the returned data that is sent from the legitimate uses. This step is vulnerable to a DoS attack, because an adversary can send out a plethora of faked signatures to the server for verification that will require system resources and cause buffer overflow. Zhiguo *et al* [12] proposed a PKC (Public Key Cryptosystem) based protocol which was combined with [8]'s key exchange protocol JFKi. The user identity protection has been added but this protocol is susceptible to DoS attacks on the first message flow that the nonce generated by the client and sent in clear text to the server. This message is not authenticated and could be sent by attackers to perform DoS attacks. DoS detection techniques has been brought up to reduce the threats [15][16]. These techniques and testing results provides insight into our ability to successfully identify DoS flooding attacks. But none completely solve the detection problem.

The client puzzle is another technique that users are required to do a considerable amount of computation before consuming resources. The first client puzzle was used to defend against connection depletion attacks proposed by Juels and Brainard [6]. A user must solve the puzzle correctly in order to get service from a responder. Client puzzles were also proposed to similarly protect authentication protocols [7], which combined the stateless authentication protocols [19][20] with a client puzzle to address a DoS attack. In general, cryptographic puzzles have been employed for key agreement [21] and address the problem of junk e-mail [22]. The only implementation of a client puzzle in the context of transport layer security (TLS) was proposed by Dean [23]. Wang and

Reiter [24] proposed a puzzle auction that allows the client to determine the difficulty (bid) of the puzzle using their implementation within transmission control protocol (TCP).

While the deployment of client puzzles in attack scenarios seems promising, but most proposed systems of this type have one basic shortcomings found by Waters [25]. The client puzzle mechanism itself can become the target of a denial-of-service attack. In most systems either the puzzle creation or verification operations (or both) require the server to perform a cryptographic hash computation [7][26][27]. This opens the possibility that the puzzle verification mechanism itself will be the target of a denial of service attack, in which an attacker floods the server with bogus puzzle solutions that the server has to process.

Secure key exchange is also a critical issue in authentication scheme. Internet Key Exchange (IKE) is an Internet Protocol Security (IPSec) standard protocol used for establishing and maintaining security associations, and to ensure security for virtual private network (VPN) using secure key exchange [28]. Several works [8], [29], [30], [31], and [32] have reported that IKE is vulnerable to DoS attacks. The user identity can not be protected in IKE [8] and IKE is very vulnerable to untraceable DoS attacks against both computational and memory resources [32]. IKEv2 [33] was then proposed to replace the original IKE. Some work has been done towards addressing, or at least examining, the DoS problems found in IKE [9][34]. Other protocol design that defend against DoS attacks include stateless cookies[36], forcing clients to store server state, rearranging the order of computations in a protocol [37], and the use of a formal method framework for analyzing the properties of protocols with respect to DoS attacks [38].

Any protocol for a server commits expensive computations or to store the protocol state prior to the client authentication is vulnerable to DoS attacks [7][19]. The advantages of being stateless, at least in the beginning of a protocol run, were recognized in the security protocol context in [19] and [20].

CHAPTER THREE

PROTOCOLS DESIGN

A new protocol, called the identity-based, privacy-protected access control filter (IPACF), has been proposed to defend DoS/DDoS attacks. The design of the proposed protocol is described as follows.

3.1 The Basic Concept of the Filter

The IP address-based filter will not prevent DDoS attacks from coming into a network with a valid source IP address. The IP address-based filter contains a fixed set of IP addresses and cannot change dynamically to protect the IP address from being sniffed and spoofed by an adversary. An adversary can spoof source IP addresses from legitimate users or a subnet's valid address range in order to pass through the IP address-based filter and launch a DoS attack. Using an identity-based filter, all users have their unique filter values and pseudo ID generated from their master pre-shared secret keys. The master key is protected by two-factor feature keys. The user's ID and password are memorized by the user, and nonce and timestamp are generated by the user's system [4]. Only legitimate users or responders have exact secrets to generate their filter values so that subsequent frames can pass through the access control filter. Otherwise, the prohibited frames will be rejected by the filter. To avoid sniffing and spoofing, the filter values cannot be reused, and thus the IPACF filter value should be changed dynamically. For the IPACF protocol, only the received frames that match the users' or responders' filter values will be allowed to use system resources as shown in Figure 1. The system performs computations only after the frames have passed the privacy protected access control filter, $\boldsymbol{F}(t,\ \boldsymbol{P}_{id(t)})$, which is a time-dependent function that changes every frame ($\boldsymbol{P}_{id(t)}$ is the pseudo ID). If received frames do not match the users' or responders' filter value, they will be rejected by the privacy protected access control filter. At this point, the computer commits only the resources necessary to compare filter values. Similar access control techniques are being used in packet filters, routers and firewalls and are well-known for their processing speed [4].

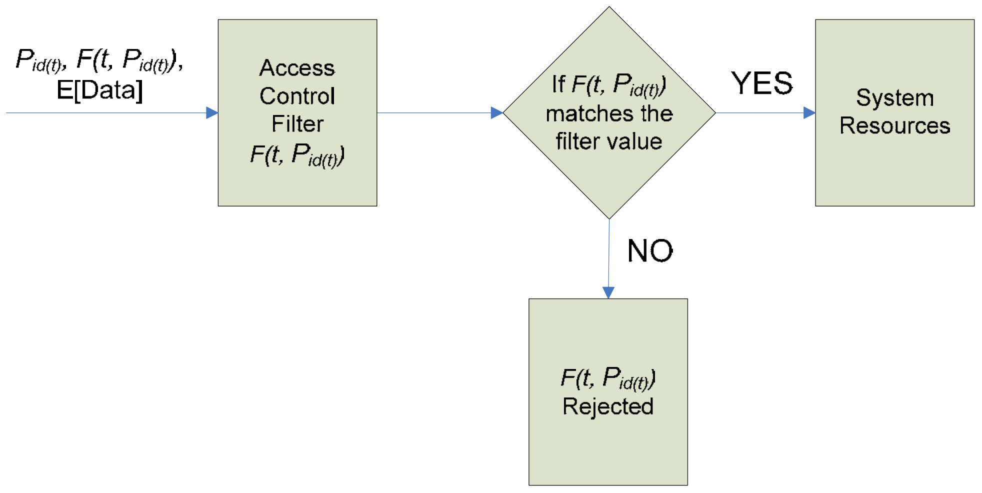

$P_{id(t)}$, $F(t, P_{id(t)})$, E[Data]
Access Control Filter $F(t, P_{id(t)})$
If $F(t, P_{id(t)})$ matches the filter value
YES
System Resources
NO
$F(t, P_{id(t)})$ Rejected

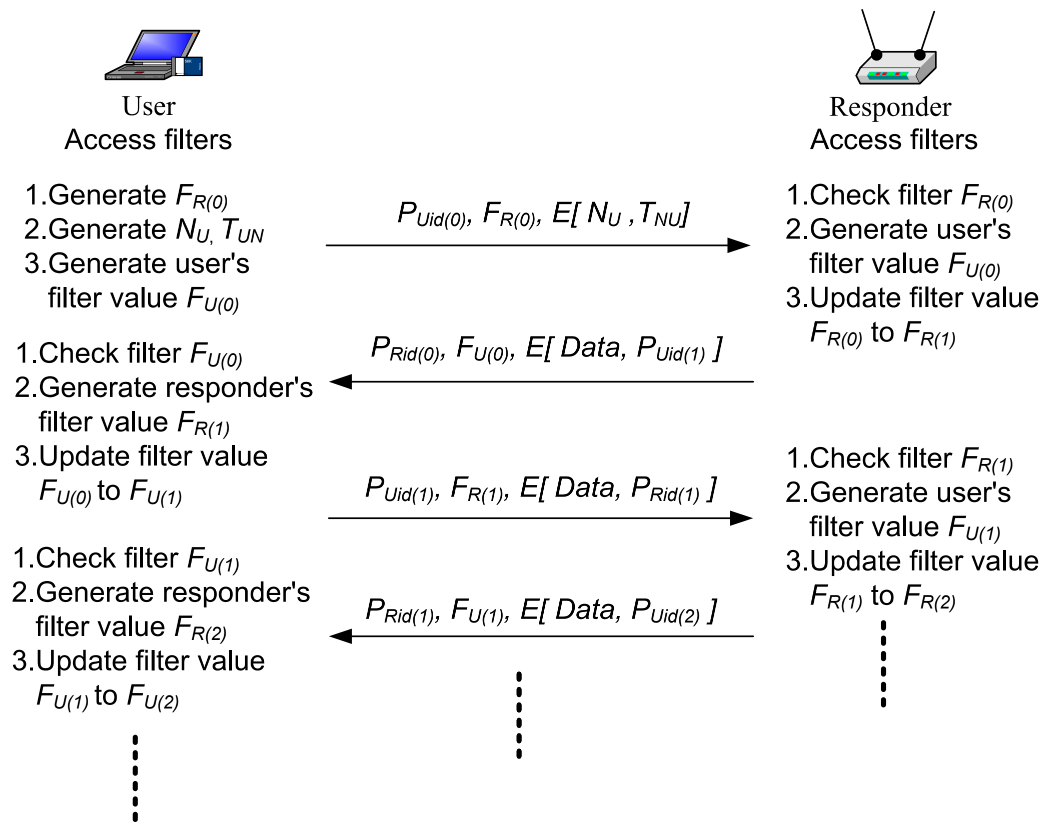
Figure 1: The Concept of Identity-Based Privacy-Protected Access Control Filter: if $\boldsymbol{F}(t,\ \boldsymbol{P}_{id(t)})$ matches the filter value, then E[Data] (encrypted data) is allowed to use system resources for decryption.

3.2 Design principle

With the current designed protocol, it is clear that Wang's IDF [4] has several weaknesses that are related to the server system resources. The following design principles were improved in the thesis.

- Efficient two frames used in the session mutual authentication stage (SMAS)
- Stateless protocol for both the user and the responder in SMAS
- User privacy is protected
- Capable of being implemented on a firewall, router, access point and authentication server
- Prevention of resource (memory and CPU) exhausting DoS and DDoS attacks, session hijacking, dictionary attacks, and men-in-the-middle attacks on both sides

The hash based message authentication codes (HMAC) is used here because the computational cost for a keyed hash function is less than that for Public Key Infrastructure (PKI) based or signature-based schemes, both of which are more vulnerable to DoS attacks [7]. Confidentiality for packet transmissions can be provided using techniques that utilize symmetric key cryptography. Encrypting the message with the secret keys shared among the users and the responders. The Advanced Encryption Standard (AES) with HMAC [39] is used to guarantee the confidentiality and integrity of data during communication.



User
Access filters
Responder
Access filters
1.Generate $F_{R(0)}$
2.Generate $N_{U,}$ T_{UN}
3.Generate user's filter value $F_{U(0)}$
$P_{Uid(0)}$, $F_{R(0)}$, $E[N_U, T_{NU}]$
1.Check filter $F_{R(0)}$
2.Generate user's filter value $F_{U(0)}$
3.Update filter value $F_{R(0)}$ to $F_{R(1)}$
$P_{Rid(0)}$, $F_{U(0)}$, $E[$ Data, $P_{Uid(1)}]$
1.Check filter $F_{U(0)}$
2.Generate responder's filter value $F_{R(1)}$
3.Update filter value $F_{U(0)}$ to $F_{U(1)}$
$P_{Uid(1)}$, $F_{R(1)}$, $E[$ Data, $P_{Rid(1)}]$
1.Check filter $F_{R(1)}$
2.Generate user's filter value $F_{U(1)}$
3.Update filter value $F_{R(1)}$ to $F_{R(2)}$
$P_{Rid(1)}$, $F_{U(1)}$, $E[$ Data, $P_{Uid(2)}]$
1.Check filter $F_{U(1)}$
2.Generate responder's filter value $F_{R(2)}$
3.Update filter value $F_{U(1)}$ to $F_{U(2)}$

Figure 2: Identity-Based Privacy-Protected Access Filter (IPACF) Overview

The Identity-Based Privacy-Protected Access Filter is derived from a user's and responder's pre-shared secrets. Each user has a unique filter associated with a particular responder. It is a time-dependent filter that changes with every frame. Only legitimate users and responders can update their filters and confirm corresponding filter values, as illustrated in Figure 2. The ID and password for user and responder are protected by human memory, never stored on a device, or sent in traffic. These precautions theoretically guarantee IDs and passwords will not be lost. When a responder receives the first frame from a user, the IPACF can determine the user ID by comparing the received filter value to the responder's filter table using the pseudo ID. Then the

Identity-Based Privacy-Protected Access Control Filter is triggered for a particular user. This time-dependent scheme involves both the user's secret and a timestamp sent from the user. The anonymity of the user's ID protects the user's privacy in wired/wireless networks.

CHAPTER FOUR

THE IDENTITY-BASED PRIVACY-PROTECTED ACCESS CONTROL FILTER (IPACF) PROTOCOL

The Identity-Based Privacy-Protected Access Control Filter (IPACF) protocol consists of three stages. The first stage is the initial configuration stage (ICS). During this stage, the master secret keys of a user and responder are generated and exchanged via a secure channel as pre-shared secrets. Using the pre-shared secrets, the initial access control filter values for all legitimate users will be generated and stored on a responder filter table for authenticating legitimate users. The second stage is the session mutual authentication stage (SMAS) used by users and responders to identify each other. The third stage is the dynamic data communication stage (DDCS). The computational for all stages should be designed with the stateless requirement in order to prevent DoS/DDoS attacks. To keep the notations simple and easy to understand, the discussions of the IPACF protocol are based on one user with a responder.

4.1 Notations for Identity-Based Privacy-Protected Access Control Filter

$\boldsymbol{P}_R$	A responder's secret key memorized by an administrator
$\boldsymbol{T}_R$	A timestamp generated by responder's system when an administrator assigns the $\boldsymbol{P}_R$
$\boldsymbol{N}_{Ri}$	An initial nonce generated by responder's system when $\boldsymbol{P}_R$ is assigned

$\boldsymbol{K}_R$	Master secret key of a responder generated by a keyed hash function during the ICS
$\boldsymbol{U}_{id}$	User's identifier
$\boldsymbol{U}_{id}$	Responder's identifier
$\boldsymbol{P}_{Uid}$	Pesudo identifier for user
$\boldsymbol{P}_{Rid}$	Pesudo identifier for responder
$\boldsymbol{P}_u$	User password, typed in for each login session and not stored in the device
$\boldsymbol{T}_u$	A timestamp generated by a user's system when a user is assigned a password
$\boldsymbol{N}_{ui}$	An initial nonce generated by a user's system when $\boldsymbol{P}_u$ is assigned
$\boldsymbol{K}_U$	Master secret key of a user generated by a keyed hash function during the initial stage
$\boldsymbol{N}_i$	A random number is chosen by a user. This number remains the same during the user's login session. It will be updated automatically after a new $\boldsymbol{K}_U$ is generated and used for the next login session.
$\boldsymbol{T}_{KU}$	Timestamp when $\boldsymbol{K}_U$ is generated
$\boldsymbol{N}_R$	A nonce is generated by a responder. It is encrypted using the AES when sent to challenge a user during session mutual authentication stage.
$\boldsymbol{T}_{NR}$	Timestamp when a responder's nonce $\boldsymbol{N}_R$ is generated
$\boldsymbol{f}_{R(t)}$	A 128-bit seed from a responder's truncated function used to update filter $\boldsymbol{F}_{\boldsymbol{R(t)}}$
$\boldsymbol{f}_{U(t)}$	A 128-bit seed from a user's truncated function used to update filter $\boldsymbol{F}_{\boldsymbol{U(t)}}$
$\boldsymbol{F}_{R(t)}$	Access filter stored in a responder's filter table to filter the frames from users
$\boldsymbol{F}_{U(t)}$	Access filter stored in a user's filter table to filter the frames from a responder
$\boldsymbol{MAC}_{R(t)}$	The output of HMAC-SHA-512, which is used to generate the filter for a responder
$\boldsymbol{MAC}_{U(t)}$	The output of HMAC-SHA-512, which is used to generate the filter for a user
$\boldsymbol{S}_{Ri}$	A responder's initial seed during the initial configuration stage
$\boldsymbol{S}_{Ui}$	A user's initial seed during the initial configuration stage
$\boldsymbol{S}_{R(t)}$	A responder's seed used to update the frame key $\boldsymbol{K}_{RF(t)}$

$\boldsymbol{S}_{U(t)}$	A user's seed used to update the frame key $\boldsymbol{K}_{US(t)}$
$\boldsymbol{N}_{U}$	A nonce is generated by a user. It is encrypted using the AES when sent to challenge a server during the mutual authentication stage.
$\boldsymbol{T}_{NU}$	Timestamp when a user's nonce $\boldsymbol{N}_{U}$ is generated
$\boldsymbol{K}_{RF(t)}$	A responder's frame key for AES encryption
$\boldsymbol{K'}_{RF(t)}$	A responder's HMAC key for integrity check
$\boldsymbol{K}_{UF(t)}$	A user's frame key for AES encryption
$\boldsymbol{K'}_{UF(t)}$	A user's HMAC key for integrity check
$\boldsymbol{MAC}_{RF(t)}$	A 256-bit MAC that contains $\boldsymbol{K}_{RF(t)}$ and $\boldsymbol{K'}_{RF(t)}$ that are used for responder as AES encryption key and HMAC key, respectively
$\boldsymbol{MAC}_{UF(t)}$	A 256-bit MAC that contains $\boldsymbol{K}_{UF(t)}$ and $\boldsymbol{K'}_{UF(t)}$ that are used for user as AES encryption key and HMAC key, respectively
$\boldsymbol{T}_{RF(t)}$	A timestamp generated by a responder when updating its filter value
$\boldsymbol{T}_{UF(t)}$	A timestamp generated by a user when updating its filter value
$\boldsymbol{N}_{INI}$	A nonce is generated by a responder. It is exchanged in initial configuration stage via secured channel for generating the first hash value of mutual authentication stage.
$\boldsymbol{T}_{INI}$	A timestamp generated by a responder when nonce $\boldsymbol{N}_{INI}$ is generated
$\boldsymbol{K}_{R(j)}$	Updated master secret key of a user for session *j* generated by a hash function for the next session
$\boldsymbol{K}_{U(j)}$	Updated master secret key of a responder for session *j* generated by a hash function for the next session
$\boldsymbol{h}$	HMAC operation.

4.2 Initial Configuration Stage

4.2.1 Master keys and pre-shared secrets generation

$\boldsymbol{K_U}$, $\boldsymbol{K_R}$, and $\boldsymbol{N_i}$, are the pre-shared secrets of a user and a responder and are stored on both systems. The pre-shared secrets $\boldsymbol{K_U}$, $\boldsymbol{K_R}$, and $\boldsymbol{N_i}$ could be delivered between a user and responder via a secure channel or could be used by a trusted third party to secure the delivery. The actual techniques are beyond the scope of this paper. $\boldsymbol{K_U}$ and $\boldsymbol{K_R}$ are the master secret keys of a user (initiator) and responder, respectively.

$$\boldsymbol{K_U = h_{(N_{ui})}[\, U_{id} \,||\, P_u \,||\, T_u \,||\, N_{ui} \,]} \quad (1)$$

$$\boldsymbol{K_R = h_{(N_{Ri})}[\, R_{id} \,||\, P_R \,||\, T_R \,||\, N_{Ri} \,]} \quad (2)$$

$\boldsymbol{U_{id}}$ and $\boldsymbol{P_u}$ are the user's ID and password, respectively, required for each login session, and should not be stored on any system. $\boldsymbol{N_{ui}}$ and $\boldsymbol{T_u}$ are generated automatically by the user's system when a user types in the $\boldsymbol{U_{id}}$ and $\boldsymbol{P_u}$ during the system ICS. The user system generates a different nonce $\boldsymbol{N_{ui}}$ and timestamp $\boldsymbol{T_u}$ to ensure a user's master key $\boldsymbol{K_U}$ is refreshed. $\boldsymbol{N_{ui}}$ is only known by the user's system and will be the input key for the HMAC function in equation (1). $\boldsymbol{K_U}$ is a user's master key generated by the HMAC function and must be stored in both the user's and responder's system. The $\boldsymbol{R_{id}}$ and $\boldsymbol{P_R}$ are the responder's ID and password, respectively, and assigned by a system administrator during the ICS. A responder's initial nonce $\boldsymbol{N_{Ri}}$ and timestamp $\boldsymbol{T_R}$ are known only by the responder's system. They are generated automatically by a system when the system administrator assigns the $\boldsymbol{P_R}$ and $\boldsymbol{R_{id}}$ for a responder. $\boldsymbol{N_{Ri}}$ is an input key for the HMAC function and used to generate the responder's master key $\boldsymbol{K_R}$ as shown in equation (2). Both the user's password $\boldsymbol{P_u}$ and the responder's password $\boldsymbol{P_R}$ are dictionary attack resistant because they are protected by the nonce and timestamp that are known only by the systems. The user or responder knows half ($\boldsymbol{U}_{id}$, $\boldsymbol{P}_u$ and $\boldsymbol{R}_{id}, \boldsymbol{P}_R$) of the master secret and their systems know the other half ($\boldsymbol{N}_{ui}$, $\boldsymbol{T}_u$ and $\boldsymbol{N}_{Ri}$, $\boldsymbol{T}_R$). This mechanism is called a two-factor feature and widely used to protect the real user's password and ID from dictionary attacks [4].

$\boldsymbol{N}_i$ is a random number that must be chosen by a user during the ICS and stored in both the user's and responder's systems; this number remains constant during the same session for the user. After the ICS, $\boldsymbol{N}_i$ can be updated locally by equation (3) while the old $\boldsymbol{N}_i$ will be used in a HMAC function to generate a new $\boldsymbol{N}_i$ for a new login session [4].

$$\boldsymbol{N}_i \leftarrow \boldsymbol{h}_{(K_u \oplus N_i)} [\, \boldsymbol{U}_{id} \,||\, \boldsymbol{K}_U \,||\, \boldsymbol{T}_{KU} \,||\, \boldsymbol{N}_i \,] \quad (3)$$

The nonce $\boldsymbol{N}_{INI}$ and the time stamp $\boldsymbol{T}_{INI}$ will be generated by the responder while a user and responder are exchanging the pre-shared secrets $\boldsymbol{K}_R$, $\boldsymbol{K}_U$, and $\boldsymbol{N}_i$ via a secured channel. $\boldsymbol{N}_{INI}$ and $\boldsymbol{T}_{INI}$ are used in ICS for a user and responder to generate the initial access control filter value $\boldsymbol{F}_{R(0)}$. The pseudo identifier $\boldsymbol{P}_{Uid}$ and $\boldsymbol{P}_{Rid}$ are assigned by the user and responder respectively to hide their real identities. $\boldsymbol{P}_{Uid}$ and $\boldsymbol{P}_{Rid}$ will be changed for each frame to ensure the privacy protection in IPACF protocol. During the ICS, [$\boldsymbol{R}_{id}$, $\boldsymbol{N}_i$, $\boldsymbol{K}_U$, $\boldsymbol{K}_R$, $\boldsymbol{N}_{INI}$, $\boldsymbol{T}_{INII}$, $\boldsymbol{P}_{Uid}$, $\boldsymbol{P}_{Rid}$] is stored on the user's system and [$\boldsymbol{U}_{id}$, $\boldsymbol{N}_i$, $\boldsymbol{K}_U$, $\boldsymbol{K}_R$, $\boldsymbol{N}_{INI}$, $\boldsymbol{T}_{INII}$, $\boldsymbol{P}_{Uid}$, $\boldsymbol{P}_{Rid}$] is stored in the responder's system as pre-shared secrets, which are used to generate the filter value and frame key.

4.2.2 Initial Filter Setup in a Responder as a Filter Table

After a user and responder have saved the pre-shared secrets in their systems, the responder must generate a filter table for all of its legitimate users. The user will also be able to create the initial access control filter value using the equations (3), (4), and (5). To create the initial filter table, the random nonce $\boldsymbol{N_{INI}}$ and timestamp $\boldsymbol{T_{INI}}$ will be used. To create the responder's access control filter, two calculation steps are required as outlined in equations (4) and (5).

$$\boldsymbol{h}_{(N_i \oplus K_R)}[\, \boldsymbol{N_i} \,||\, \boldsymbol{N_{INI}} \,||\, \boldsymbol{T_{INI}} \,] \equiv \boldsymbol{S_{Ri}} \quad (4)$$

$\boldsymbol{K_R}$ is the master key of the responder and is a group key known by all of its legitimate users in IDF protocol proposed by Wang [4]; however, $\boldsymbol{K_R}$ is a pairwise key shard by a user and a responder in this thesis. Then the input key for the HMAC function is formed by $\boldsymbol{K_R}$ XOR with $\boldsymbol{N_i}$, which is chosen by a particular user. Different users have different $\boldsymbol{N_i}$ and thus different $\boldsymbol{S_{Ri}}$. $\boldsymbol{S_{Ri}}$ is a 512-bit output of the HMAC function as shown in equation (4). It is a responder's initial seed and used only at the ICS. This seed $\boldsymbol{S_{Ri}}$ is used as an input key for the HMAC function in equation (5). The purpose of the formula in equation (4) is to generate different seeds for a particular user, given the user's $\boldsymbol{N_i}$, and use $\boldsymbol{N_i}$, $\boldsymbol{N_{INI}}$, and $\boldsymbol{T_{INI}}$ to avoid reusing the seed. Because the input keys for the HMAC function are never reused, the output of the HMAC function in equation (5) is made very secure by avoiding dictionary attacks [18][35].

$$\boldsymbol{h}_{(S_{Ri})}^{[N_i \oplus N_{INI}]}[\, \boldsymbol{U_{id}} \,||\, \boldsymbol{K_U} \,||\, \boldsymbol{N_i} \,||\, \boldsymbol{N_{INI}} \,||\, \boldsymbol{T_{INI}} \,] \equiv \boldsymbol{MAC_{R(0)}} \quad (5)$$

[$N_i \oplus N_{INI}$] is the number of rounds used to conduct the hash function. In consideration of performance, the number of rounds is recommended to be truncated to 10 bit, which means that the number of rounds is limited to between 0 and 1023. $MAC_{R(t)}$ is a 512-bit keyed hash value and is truncated in three parts, including the 128 most significant bits $\boldsymbol{f_{R(t)}}$ and the 128 least significant bits $\boldsymbol{S_{R(t)}}$, as shown in Figure 3. t is the index of a time-dependent function, and $t=0$ is the responder's first seed used to generate the first access control filter and frame key [4].

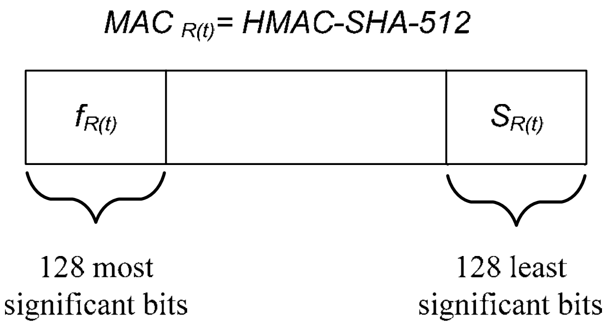


Figure 3: The Responder's Truncated Function

$\boldsymbol{f_{R(0)}}$ is a seed used to generate a 160-bit responder's access control filter $\boldsymbol{F_{R(0)}}$ using equation (6). The length of the HMAC function used to generate the access control filter is recommended to be a 160-bit HMAC-SHA-1, but different applications may vary. $\boldsymbol{F_{R(0)}}$ is the initial access control filter value stored in a responder's device. The first frame sent from the user must match this value so that subsequent parts of this frame can be admitted into the responder's system. $\boldsymbol{S_{R(0)}}$, which is the 128 least significant bits of $MAC_{R(0)}$, is a seed used to update the frame key and then generate the new access filter for the next frame.

$$\boldsymbol{h}_{(S_{Ri})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{MAC}_{R(0)} \,||\, \boldsymbol{T}_{INI} \,||\, \boldsymbol{f}_{R(0)}] \equiv \boldsymbol{F}_{R(0)} \quad (6)$$

The responder can then create a table and store the identity-based filters and all the necessary key information as shown in Table 1.

P_{Uid}	Flag	U_{id}	P_{Rid}	Access Filter	Hash Value	Frame Key	Packet counter	Idle Time
				$F_{R(0)}$	$MAC_{R(0)}$			

Table 1: Responder's Dynamic Access Filter Table

Each row specifies the information for a particular user. The packet counter will accumulate all the received frames that pass the access control filter for a user's login session. The packet counter will repeat the process for each new login session. This feature aids the responder in monitoring any abnormal traffic from the user and thus helps prevent a DDoS attack. The idle time is represented by the time that the packet counter does not increase while the user remains connected. The responder should ask the user to re-authenticate if the idle time is too long. On the other hand, the users can generate the initial access control filter value $\boldsymbol{F}_{R(0)}$ using equations (4), (5), and (6) that introduce a complete stateless configuration.

In Wang's IDF protocol [4], the user side was not completely stateless in the mutual authentication stage. The user's access control filter will be created after the user receives the beacon from the responder and can derive the responder's nonce $\boldsymbol{N}_R$, and timestamp $\boldsymbol{T}_{NR}$, by using the responder's pre-shared master key $\boldsymbol{K}_R$. The IDF protocol requires one more frame in comparison to IPACF in the mutual authentication stage for the responder to broadcast the beacon to the legitimate users who can decrypt the beacon encrypted with AES encryption. Nonce $\boldsymbol{N_R}$ and timestamp $\boldsymbol{T_{NR}}$ will be the contents of the beacon. $\boldsymbol{N_R}$ and $\boldsymbol{T_{NR}}$ will be the secrets for the user to generate filter value $\boldsymbol{F_{R(0)}}$.

The responder will pre-generate the filter value ahead of time in Wang's IDF protocol [4]. $\boldsymbol{N_R}$ and $\boldsymbol{T_{NR}}$ are broadcast in the beacon by the responder; a user receives and generates the same filter value as the responder. But there is one problem: what happens if the users do not log in during this period of time? The responder will have to regenerate $\boldsymbol{F_{R(0)}}$ for every user if $\boldsymbol{N_R}$ and $\boldsymbol{T_{NR}}$ are changed, and this is a computationally expensive process. It is desirable that a protocol does not require every user to calculate $\boldsymbol{F_{R(0)}}$ periodically. Using equations (4), (5), and (6) will make the IPACF protocol for both the user and the responder stateless after ICS. In addition, IPACF protocol has one less frame to complete in SMAS in comparison to IDF since there is no need of the beacon frame. Because of the complete stateless configuration in IPACF, the user protocol becomes stateless in the SMAS, the user will not have to depend on the beacon broadcasted by the responder.

At this point, the ICS is completed. The pre-shared secrets and initial filter value $\boldsymbol{F_{R(0)}}$ are stored in both user and responder, as well as the information for all of its legitimate users, in a filter table.

4.2.3 User Login and Master Key Renew

Session Login

$\boldsymbol{K_U}$, $\boldsymbol{K_R}$ and $\boldsymbol{N_i}$, are the pre-shared secrets stored on a user's system. When a user wants to log in, the user must type in the user ID and password. The user's system has stored $\boldsymbol{N_{ui}}$ and $\boldsymbol{T_u}$ from the initial setup; therefore, the user's system can generate a $\boldsymbol{K_U}$

based upon the password and ID that the user entered using equation (1). If this $\boldsymbol{K}_U$ matches the pre-stored $\boldsymbol{K}_U$, then the system identifies that the user as a legitimate one. If this $\boldsymbol{K}_U$ does not match the pre-stored $\boldsymbol{K}_U$, after a reasonable number of trials, the user's system should be locked. In this case, no filter values will be generated for the particular user to prevent an on-line password guessing attack.

Master Key Renewal

The $\boldsymbol{K}_U$ and $\boldsymbol{N}_i$ should be refreshed for every login session. The $\boldsymbol{N}_i$ will be updated using equation (3). During each login session, the pre-stored $\boldsymbol{K}_U$ is matched to ensure that he is a legitimate user, and the system will generate a new nonce and timestamp. Given the new nonce and timestamp, the system performs a calculation using equation (7) to obtain the new $\boldsymbol{K}_{U(j)}$ for session j. The new $\boldsymbol{K}_{U(j)}$ and $\boldsymbol{N}_i$ for the next login session will be encrypted using the AES and then sent to the responder during communication.

Wang's IDF [4] on a fixed master key $\boldsymbol{K}_R$ means that the responder must uses the same key to encrypt the packet. All users must use the same master key $\boldsymbol{K}_R$ to decrypt packets. Since all users have the same $\boldsymbol{K}_R$, they not only have the ability to encrypt packets, but also have the capability to pretend to be a responder, and perform malicious actions. Therefore, $\boldsymbol{K}_R$ does not provide the essential security. Furthermore, the master key $\boldsymbol{K}_R$ will not be updated unless ICS is performed: this problematic feature is the most serious drawback in Wang's IDF protocol. If ICS is necessary to perform updating master key $\boldsymbol{K}_R$, all legitimate users will have to perform ICS at the same time, which means all legitimate users need to be synchronized to change $\boldsymbol{K}_R$ while ICS is performed. A legitimate user will not be able to update the master key $\boldsymbol{K}_R$ unless the user creates a secured channel and updates the master key $\boldsymbol{K_R}$ with the responder individually; otherwise, the legitimate user will not be able to decrypt any beacon sent by the responder in the mutual authentication stage.

An entire group of users should not be responsible for updating master key $\boldsymbol{K_{R(j)}}$. Legitimate users should have the flexibility of performing ICS, whenever necessary. On the other hand, having a pairwise master key $\boldsymbol{K_{R(j)}}$ for each legitimate user will avoid malicious actions from other legitimate users. IPACF protocol will be able to set up pairwise master keys with every legitimate user, and the master secret keys will be exchanged between the user and the responder in every session.

$$\boldsymbol{K_{U(j+1)}} = \boldsymbol{h}_{(N_{ui})}[\, \boldsymbol{U_{id}} \,||\, \boldsymbol{P_u} \,||\, \boldsymbol{N_u} \,||\, \boldsymbol{T_{NU}} \,||\, \boldsymbol{K_{U(j)}}]\;\; j = 1, 2, 3\ldots \quad (7)$$

$$\boldsymbol{K_{R(j+1)}} = \boldsymbol{h}_{(N_{Ri})}[\, \boldsymbol{R_{id}} \,||\, \boldsymbol{P_R} \,||\, \boldsymbol{N_R} \,||\, \boldsymbol{T_{NR}} \,||\, \boldsymbol{K_{R(j)}}]\;\; j = 1, 2, 3\ldots \quad (8)$$

In equations (7) and (8), $\boldsymbol{K_U}$, $\boldsymbol{K_R}$, and $\boldsymbol{N_i}$ are the pre-shared secrets of a user and a responder and are stored on both systems and j is the index of session number. $\boldsymbol{U_{id}}$ and $\boldsymbol{P_u}$ are the user's ID and password, respectively, required for each login session and should not be stored on any system. $\boldsymbol{N_u}$, $\boldsymbol{T_{NU}}$, $\boldsymbol{N_R}$, and $\boldsymbol{T_{NR}}$ are nonce and time stamp generated for each session master key update and stored in both the user's and the responder's systems. $\boldsymbol{K_{R(j)}}$ and $\boldsymbol{K_{U(j)}}$ are set for each session during which a user logs in and logs out. For each login session, the system generates a different nonce $\boldsymbol{N_u}$, $\boldsymbol{N_R}$ and timestamp $\boldsymbol{T_{NU}}$, $\boldsymbol{T_{NR}}$ automatically to ensure that both a user's master key $\boldsymbol{K_U}$ and a responder's master key $\boldsymbol{K_R}$ are not reused before. $\boldsymbol{K_R}$ and $\boldsymbol{K_U}$ are used as input key for the HMAC function in equation (7) and (8). The purpose of master key renewal is to provide authenticated keying material in a protected manner.

Each user will have a pairwise master key $\boldsymbol{K}_R$ to avoid malicious actions from other legitimate users and the master key $\boldsymbol{K}_{R(j)}$ will be exchanged between the user, and the responder in every session. We introduced the secured master secret key exchange (SMSKE) in ICS. The idea of SMSKE is to negotiate and provide authenticated keying material for security associations in a protected manner. SMSKE will be performed in the last frame of the SMAS and the first frame of the DDCS. After the secured authentication has been initiated, the user and the responder have identified each other; a secured channel is created after SMAS using Advanced Encryption Standard (AES) encryption and HMAC message authentication code for integrity check. The SMSKE will not only decrease the communication cost using a secured channel, but also improved the security of Wang's IDF protocol [4] by using dynamic master key $\boldsymbol{K}_{R(j)}$ and $\boldsymbol{K}_{U(j)}$ for each session while IDF must used a fixed master secret key $\boldsymbol{K}_R$.

The IPACF protocol is based on the legitimate user's pre-shared secret and not their IP address in order to avoid an IP spoofing attack. The IPACF filter is not a fixed value filter. The filter values vary with every frame by both the user and the responder to prevent sniffing and replay attacks. Perform SMSKE will make the result of secure master key update on both user and responder after every single success SMAS. All the legitimate users will have a unique responder master key $\boldsymbol{K}_{R(j)}$ that will increase the randomness of both master keys for the future SMAS.

4.3 Session Mutual Authentication Stage (SMAS)

Following the initial configuration stage, two processes will be accomplished by a user before the user initiates the SMAS. First, both the responder and the user will be able to generate the same $\boldsymbol{F}_{R(0)}$ using equations (4), (5), and (6). $\boldsymbol{F}_{R(0)}$ is a 160-bit responder's filter value and will be sent to the responder to check its access filter after the second step. Second, before a user initiates SMAS, a user will generate a user's nonce $\boldsymbol{N}_U$ and a timestamp $\boldsymbol{T}_{NU}$ for that nonce to authenticate the responder and also create the user's initial dynamic access filter. To create this filter, two rounds of calculation are required as outlined in equations (9) and (10).

$$\boldsymbol{h}_{(K_U \oplus N_i)}[\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{N}_U \,||\, \boldsymbol{T}_{NU} \,] \equiv \boldsymbol{S}_{Ui} \quad (9)$$

$$\boldsymbol{h}_{(S_{Ui})}^{[N_i \oplus N_U]}[\, \boldsymbol{U}_{id} \,||\, \boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{N}_U \,||\, \boldsymbol{T}_{NU} \,] \equiv \boldsymbol{MAC}_{U(0)} \quad (10)$$

The message authentication code will be generated with HMAC-SHA1. SHA1 is considered cryptographically stronger than MD5 even it takes more CPU cycles to compute. HMAC-SHA1 is also recommended in IPSec where the slightly superior security of SHA1 over MD5 is important [17]. $\boldsymbol{S}_{Ui}$ is a 512-bit output of the HMAC function in equation (9) and will be used as an input key for the HMAC function in equation (10). It is a user's initial seed and is only used in the initial configuration stage. This seed will be changed for every login session by the user because of varying properties of $\boldsymbol{K}_U$, $\boldsymbol{N}_i$, $\boldsymbol{N}_U$, and $\boldsymbol{T}_{NU}$. The primary aim of the formula in equation (9) is to generate different seeds for every login session to avoid reuse of the filter values.

$[N_i \oplus N_U]$ is the number of rounds employed in conducting the hash function, and from a performance perspective, should be truncated to 10 bits. Therefore, the number of rounds is limited to between 0 and 1023. $MAC_{U(t)}$ is a 512-bit keyed hash value truncated in three parts, including the 128 most significant bits $\boldsymbol{f}_{U(t)}$ and the 128 least significant bits $\boldsymbol{S}_{U(t)}$, as shown in Figure 5. t is an index for a time-dependent function, and $t=0$ is the user's first seed that is used to generate the first access control filter and frame key [4].

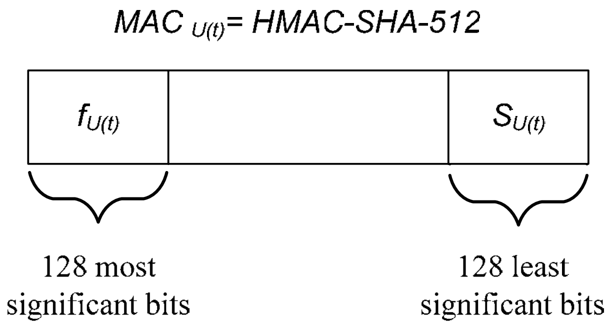


Figure 4: The User's Truncated Function

$\boldsymbol{f}_{U(0)}$ is a seed used to generate a 160-bit user's access control filter $\boldsymbol{F}_{U(0)}$, as illustrated in equation (11). $\boldsymbol{F}_{U(0)}$ is a user's first access control filter value stored on the user's system and is used for checking the frames sent from a responder. If the frame does not match the filter, it will be blocked by the user's system. $\boldsymbol{S}_{U(0)}$ is a user's first seed used to update the frame key and generates the new access filter for the next frame. Once the user generates $\boldsymbol{F}_{U(0)}$, an access filter table will be created as shown in Table 2, to store the filter value and other information.

$$\boldsymbol{h}_{(S_{Ui})}[\, \boldsymbol{K}_U \,||\, \boldsymbol{N}_U \,||\, \boldsymbol{MAC}_{U(0)} \,||\, \boldsymbol{T}_{NU} \,||\, \boldsymbol{T}_{INI} \,||\, \boldsymbol{f}_{U(0)}] \equiv \boldsymbol{F}_{U(0)} \quad (11)$$

P_{Rid}	Flag	R_{id}	P_{Uid}	Access Filter	Hash Value	Frame Key	Packet counter	Idle Time
				$F_{U(0)}$	$MAC_{U(0)}$			

Table 2: User's Dynamic Access Filter Table

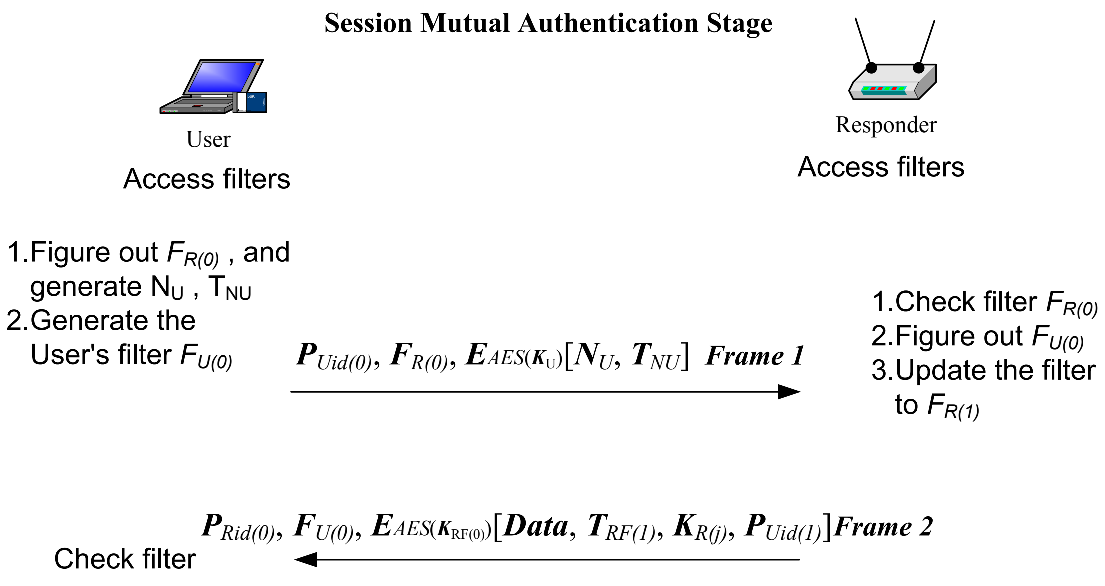

Session Mutual Authentication Stage
User
Access filters
Responder
Access filters
1.Figure out $F_{R(0)}$, and generate N_U , T_{NU}
2.Generate the User's filter $F_{U(0)}$
$P_{Uid(0)}$, $F_{R(0)}$, $E_{AES(K_U)}[N_U, T_{NU}]$ Frame 1
1.Check filter $F_{R(0)}$
2.Figure out $F_{U(0)}$
3.Update the filter to $F_{R(1)}$
$P_{Rid(0)}$, $F_{U(0)}$, $E_{AES(K_{RF(0)})}[Data, T_{RF(1)}, K_{R(j)}, P_{Uid(1)}]$ Frame 2
Check filter

Figure 5: Session Mutual Authentication Stage Overview

Frame 1 ***U → R :*** $\boldsymbol{P_{Uid(0)}}$, $\boldsymbol{F_{R(0)}}$, $\boldsymbol{E_{AES(K_U)}}[\boldsymbol{N_U}, \boldsymbol{T_{NU}}]$

A legitimate user will be able to perform SMAS as the user desires. After the responder receives ***Frame 1***, the responder will perform three steps: First, the responder will check $\boldsymbol{F_{R(0)}}$ to see if $\boldsymbol{F_{R(0)}}$ matches one of the filter values in the filter value table. If there is a match, the responder will know the user's ID and $\boldsymbol{K_U}$ in accordance with the received filter value. The responder will also identify the user as legitimate and then pass the encrypted data to the system for decryption to obtain $\boldsymbol{N_U}$ and $\boldsymbol{T_{NU}}$. If ***Frame 1*** matches the value in the responder's filter table, the responder has completed half of the SMAS for identifying a legitimate user. If the received filter value $\boldsymbol{F_{R(0)}}$ does not match the responder's filter values, the responder should block this frame and stop the session. Second, equations (9), (10), and (11) are used to derive $\boldsymbol{S}_{Ui}$, $\boldsymbol{MAC}_{U(0)}$, $\boldsymbol{f}_{U(0)}$, $\boldsymbol{S}_{U(0)}$, and $\boldsymbol{F}_{U(0)}$. Third, generate a timestamp $\boldsymbol{T}_{RF(1)}$, and update the responder's access filter table for the next frame using the previous secret $\boldsymbol{MAC}_{R(0)}$ to generate a new $\boldsymbol{MAC}_{R(1)}$ from equation (12). $\boldsymbol{f}_{R(1)}$ and $\boldsymbol{S}_{R(1)}$ will be truncated from $\boldsymbol{MAC}_{R(1)}$. $\boldsymbol{f}_{R(1)}$ is a new seed that is used to update the responder's access control filter from $\boldsymbol{F}_{R(0)}$ to $\boldsymbol{F}_{R(1)}$ using equation (14). $\boldsymbol{S}_{R(1)}$ is a new seed that is used to update the responder's frame key generation function $\boldsymbol{K}_{RF(t)}$ for the next frame [4].

$$\boldsymbol{h}_{\left(MAC_{RF(0)}\right)}[\,\boldsymbol{K}_R \,||\, \boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{MAC}_{R(0)} \,||\, \boldsymbol{T}_{RF(1)}] \equiv \boldsymbol{MAC}_{R(1)} \quad (12)$$

$\boldsymbol{MAC}_{RF(t)}$ is the generation function for the responder's frame key. When $t=0$, as shown in equation (13), $\boldsymbol{MAC}_{RF(0)}$ is the secret used by the responder to generate $\boldsymbol{MAC}_{R(1)}$ to update the access filter from $\boldsymbol{F}_{R(0)}$ to $\boldsymbol{F}_{R(1)}$ for the next frame using equations (12) and (14). $\boldsymbol{K}_{RF(0)}$ and $\boldsymbol{K'}_{RF(0)}$ will be truncated from $\boldsymbol{MAC}_{RF(0)}$ with 128 most significant bits and 128 least significant bits, respectively. $\boldsymbol{K}_{RF(0)}$ is a frame key used by the AES to encrypt the data. $\boldsymbol{K'}_{RF(0)}$ is a new key used in HMAC for data integrity. The encrypted data, with filter value $\boldsymbol{F}_{U(0)}$, will be sent to the user after the responder updates the filter table.

$$\boldsymbol{h}_{\left(K_{R(j)}\right)}[\,\boldsymbol{S}_{Ri} \,||\, \boldsymbol{S}_{R(0)} \,||\, \boldsymbol{T}_{INI} \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{K}_U \,||\, \boldsymbol{K}_R] \equiv \boldsymbol{MAC}_{RF(0)} \quad (13)$$

$\boldsymbol{F}_{R(1)}$ is a new access control filter value, which is stored in the responder system and used for checking the next frame that is sent from the user.

$$\boldsymbol{h}_{\left(MAC_{RF(0)}\right)}[\,\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{MAC}_{R(1)} \,||\, \boldsymbol{T}_{RF(1)} \,||\, \boldsymbol{f}_{R(1)}] \equiv \boldsymbol{F}_{R(1)} \quad (14)$$

Frame 2 $\boldsymbol{R} \rightarrow \boldsymbol{U}$ **:** $\boldsymbol{P}_{Rid(0)}$, $\boldsymbol{F}_{U(0)}$, $\boldsymbol{E}_{AES(\boldsymbol{K}_{RF(0)})}$ [$\boldsymbol{Data}$, $\boldsymbol{T}_{RF(1)}$, $\boldsymbol{K}_{R(j)}$, $\boldsymbol{P}_{Uid(1)}$]

After the responder has authenticated the user in ***Frame 1***, the master secret key for the next session, $\boldsymbol{K}_{R(j)}$ will be sent to the user in this frame. When the user receives ***Frame 2***, three steps are performed: First, check $\boldsymbol{F}_{U(0)}$ to see if it matches the filter value. If there is a match, most likely this frame was sent from the responder, and then the encrypted data, $\boldsymbol{T}_{RF(1)}$ and $\boldsymbol{K}_{R(j)}$ can be decrypted using the key $\boldsymbol{K}_{RF(0)}$. Since the user knows the same secrets $\boldsymbol{S}_{Ri}$, $\boldsymbol{S}_{R(0)}$, $\boldsymbol{T}_{INI}$, $\boldsymbol{K}_U$, and $\boldsymbol{K}_R$ as the responder, the user can perform the same calculation to derive secret $\boldsymbol{MAC}_{RF(0)}$ by using equation (13). $\boldsymbol{K}_{RF(0)}$ (128-bit AES key) and $\boldsymbol{K'}_{RF(0)}$ (128-bit HMAC key) can be truncated from $\boldsymbol{MAC}_{RF(0)}$ and the user is able to decrypt the data using frame key $\boldsymbol{K}_{RF(0)}$ to obtain the timestamp $\boldsymbol{T}_{RF(1)}$ and $\boldsymbol{K}_{R(j)}$. Second, use this secret $\boldsymbol{MAC}_{RF(0)}$ and the timestamp $\boldsymbol{T}_{RF(1)}$ to derive $\boldsymbol{MAC}_{R(1)}$, $\boldsymbol{f}_{R(1)}$, $\boldsymbol{S}_{R(1)}$ and $\boldsymbol{F}_{R(1)}$ by using equations (12) and (14). $\boldsymbol{F}_{R(1)}$ is a filter value that will be sent to the responder for checking. $\boldsymbol{S}_{R(1)}$ is a seed used to update the responder's secret from $\boldsymbol{MAC}_{RF(0)}$ to $\boldsymbol{MAC}_{RF(1)}$. The user must also store this seed to update and synchronize the responder's secret in the future. In addition, the user must update the $\boldsymbol{K}_R$; original $\boldsymbol{K}_{R(j-1)}$ will be replaced by $\boldsymbol{K}_{R(j)}$. Third, the user will update the access filter from $\boldsymbol{F}_{U(0)}$ to $\boldsymbol{F}_{U(1)}$ to check the next frame. Two rounds are required to update the access filter:

First Round: Generate the user's frame key as follows:

$$\boldsymbol{h}_{\left(K_{U(j)}\right)}[\,\boldsymbol{S}_{Ui} \,||\, \boldsymbol{S}_{U(0)} \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{T}_{NU}] \equiv \boldsymbol{MAC}_{UF(0)} \quad (15)$$

$\boldsymbol{MAC}_{UF(t)}$ is the generation function for the user's frame key. When $t = 0$, as shown in equation (15), $\boldsymbol{MAC}_{UF(0)}$ is the secret employed by the user to generate $\boldsymbol{MAC}_{U(1)}$ to update the access filter from $\boldsymbol{F}_{U(0)}$ to $\boldsymbol{F}_{U(1)}$ for the next frame using equations (16) and (17). $\boldsymbol{K}_{UF(0)}$ and $\boldsymbol{K'}_{UF(0)}$ will be truncated from $\boldsymbol{MAC}_{UF(0)}$ with 128 most significant bits and 128 least significant bits, respectively. $\boldsymbol{K}_{UF(0)}$ is a frame key used by the AES to encrypt the data. $\boldsymbol{K'}_{UF(0)}$ is a key used in HMAC for data integrity. The encrypted data, with filter value $\boldsymbol{F}_{R(1)}$, will be sent to the responder after the user updates the filter table.

Second Round: Calculate the new MAC value and update the access filter as follows:

$$\boldsymbol{h}_{(MAC_{UF(0)})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{MAC}_{U(0)} \,||\, \boldsymbol{T}_{UF(1)}] \equiv \boldsymbol{MAC}_{U(1)} \quad (16)$$

$\boldsymbol{f}_{U(1)}$ and $\boldsymbol{S}_{U(1)}$ will be truncated from the $\boldsymbol{MAC}_{U(1)}$ in the same manner. $\boldsymbol{f}_{U(1)}$ is a seed used to generate a 160-bit user's access control filter $\boldsymbol{F}_{U(1)}$ using equation (17). $\boldsymbol{F}_{U(1)}$ is a new user's access filter and is stored in the user's access filter table for checking the next frame. $\boldsymbol{S}_{U(1)}$ is a new seed that is used to update the user's secret $\boldsymbol{MAC}_{UF(t)}$, from $\boldsymbol{MAC}_{UF(0)}$ to $\boldsymbol{MAC}_{UF(1)}$.

$$\boldsymbol{h}_{(MAC_{UF(0)})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{MAC}_{U(0)} \,||\, \boldsymbol{T}_{UF(1)} \,||\, \boldsymbol{f}_{U(1)}] \equiv \boldsymbol{F}_{U(1)} \quad (17)$$

After the user identifies the responder, the SMAS for the IPACF protocol is completed. The user will start Dynamic Data Communication Stage (DDCS).

4.4 Dynamic Data Communication Stage (DDCS)

In the DDCS, the filter value is changed with every frame. Because the responder must process a high volume of traffic, the cost of computation at this stage should be less than other stages. This stage is called the dynamic data communication stage because at this stage, the user and the responder need not generate the nonce to identify each other. The amount of system computation is also less than that required in the SMAS because fewer rounds of the hash function are performed for every frame. To enhance security, however, a time-dependent scheme is used at this point to refresh the frame key and update the filter for both the user and the responder. The master key will be also updated in the first frame of DDCS.

In the DDCS, both the access control filter values of the responder and the user are changed dynamically in every frame. The advantage of this approach is that an adversary is unable to predict the dynamic access filter value for the next frame. Only the legitimate user and responder that can match access filter values are allowed to use system resources. During the DDCS, the user and responder identify each other continuously to match the access filter. This mechanism prevents both the middleman attack and session-hijack because an adversary is unable to guess the filter values and frame keys. If a frame is lost in this stage, a re-transmission by UDP will maintain the connectivity. If a connectionless link is being used, then the SMAS must be carried out again to maintain high security for the re-established link.

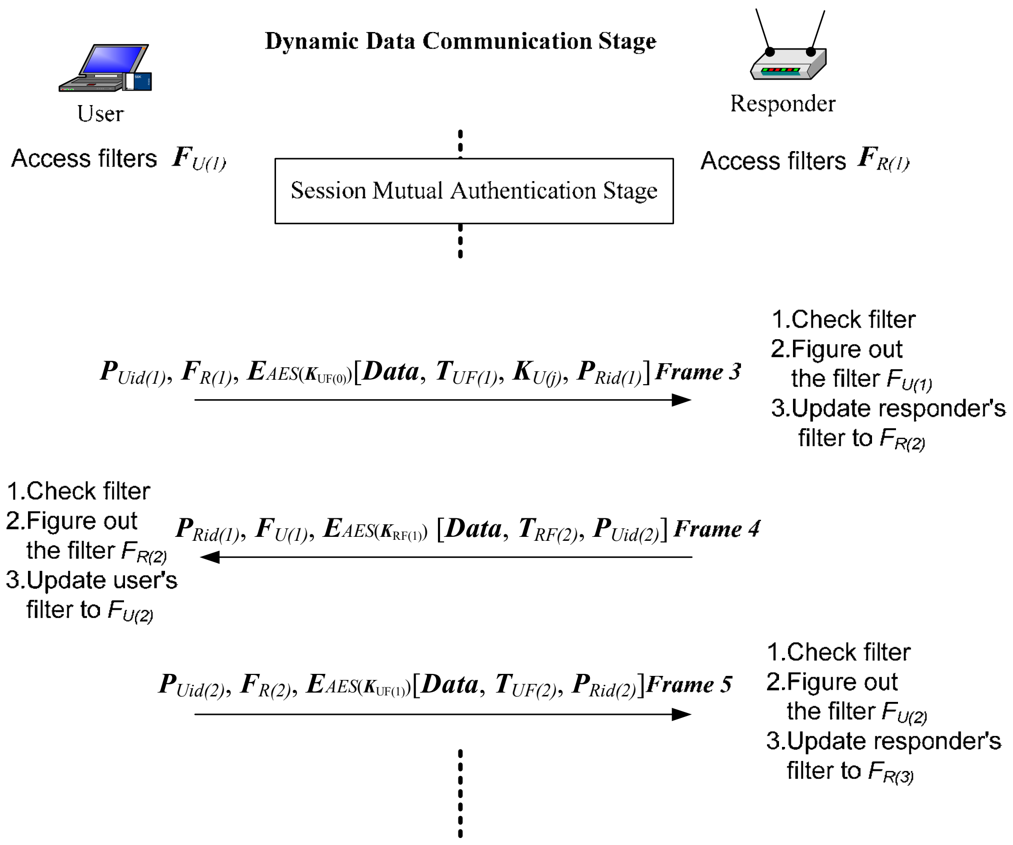

Dynamic Data Communication Stage
User
Responder
Access filters $F_{U(1)}$
Access filters $F_{R(1)}$
Session Mutual Authentication Stage
$P_{Uid(1)}, F_{R(1)}, E_{AES(K_{UF(0)})}[Data, T_{UF(1)}, K_{U(j)}, P_{Rid(1)}]$ Frame 3
1.Check filter
2.Figure out the filter $F_{U(1)}$
3.Update responder's filter to $F_{R(2)}$
$P_{Rid(1)}, F_{U(1)}, E_{AES(K_{RF(1)})}[Data, T_{RF(2)}, P_{Uid(2)}]$ Frame 4
1.Check filter
2.Figure out the filter $F_{R(2)}$
3.Update user's filter to $F_{U(2)}$
$P_{Uid(2)}, F_{R(2)}, E_{AES(K_{UF(1)})}[Data, T_{UF(2)}, P_{Rid(2)}]$ Frame 5
1.Check filter
2.Figure out the filter $F_{U(2)}$
3.Update responder's filter to $F_{R(3)}$

Figure 6: The Dynamic Data Communication Stage

In the DDCS, three steps are required on both sides for responders and users. First, check the access filter when a frame is received. Second, determine the filter values on the other side. Third, update the access control filter for the next frame. For example, the following ***Frame 3*** and ***4***, are the first and second frames for the DDCS as shown in Figure 6.

Frame 3 ***U*** $\rightarrow$ ***R*** : $P_{Uid(1)}, F_{R(1)}, E_{AES(K_{UF(0)})}[\ Data, T_{UF(1)}, K_{U(j)}, P_{Rid(1)}]$

In the first frame of DDCS (***Frame 3***), the responder will perform three steps: First verify the $F_{R(1)}$ from the user based on the pseudo ID $P_{Uid(1)}$. If there is a match, the responder will then decrypt data, $\boldsymbol{T}_{UF(1)}$, and $\boldsymbol{K}_{U(j)}$. Since the responder knows the same secrets $\boldsymbol{S}_{Ui}$, $\boldsymbol{S}_{U(0)}$, $\boldsymbol{T}_{NU}$, $\boldsymbol{N}_{i}$, and $\boldsymbol{N}_{INI}$ as the user, the responder can perform the same calculation with the equation (15), which to derive secret $\boldsymbol{MAC}_{UF(0)}$. Then, the responder is able to get the frame key $\boldsymbol{K}_{UF(0)}$ and decrypts the data to obtain the timestamp $\boldsymbol{T}_{UF(1)}$ and $\boldsymbol{K}_{U(j)}$. Second, using this secret $\boldsymbol{MAC}_{UF(0)}$ and the timestamp $\boldsymbol{T}_{UF(1)}$ to derive $\boldsymbol{MAC}_{U(1)}$, $\boldsymbol{f}_{U(1)}$, $\boldsymbol{S}_{U(1)}$, and $\boldsymbol{F}_{U(1)}$. The responder must also update the $\boldsymbol{K}_{U(j)}$, original $\boldsymbol{K}_{U(j-1)}$ will be replaced by $\boldsymbol{K}_{U(j)}$ to become the new $\boldsymbol{K}_{U}$. Third, the responder will generate a timestamp $\boldsymbol{T}_{RF(2)}$ and update the responder's access control filter from $\boldsymbol{F}_{R(1)}$ to $\boldsymbol{F}_{R(2)}$ and pseudo ID from $\boldsymbol{P}_{Uid(1)}$ to $\boldsymbol{P}_{Uid(2)}$, check the next frame sent from the user using equations (18), (19) and (20). Two rounds are required to update the access control filter:

First Round: the new responder's frame key generation

$$\boldsymbol{h}_{(K_{R(j)})}[\, \boldsymbol{S}_{R(0)} \,||\, \boldsymbol{S}_{R(1)} \,||\, \boldsymbol{K}_{R} \,||\, \boldsymbol{K}_{U} \,||\, \boldsymbol{T}_{RF(1)}] \equiv \boldsymbol{MAC}_{RF(1)} \quad (18)$$

$\boldsymbol{MAC}_{RF(1)}$ is a new responder's secret which replaces $\boldsymbol{MAC}_{RF(0)}$. $\boldsymbol{MAC}_{RF(1)}$ is used by the responder to generate $\boldsymbol{MAC}_{R(2)}$ to update the access control filter $\boldsymbol{F}_{R(1)}$ to $\boldsymbol{F}_{R(2)}$. This secret will be truncated into 128-bit $\boldsymbol{K}_{RF(1)}$ and 128-bit $\boldsymbol{K'}_{RF(1)}$. $\boldsymbol{K}_{RF(1)}$ is used as a new frame key for AES to encrypt the data when the frame is sent to the user; $\boldsymbol{K'}_{RF(1)}$ is a new key used in HMAC for data integrity.

Second Round: Calculate the new MAC value and update the access filter

$$\boldsymbol{h}_{(MAC_{RF(1)})}[\, \boldsymbol{K}_{R} \,||\, \boldsymbol{K}_{U} \,||\, \boldsymbol{N}_{i} \,||\, \boldsymbol{N}_{R} \,||\, \boldsymbol{MAC}_{R(1)} \,||\, \boldsymbol{T}_{RF(2)}] \equiv \boldsymbol{MAC}_{R(2)} \quad (19)$$

$\boldsymbol{f}_{R(2)}$ and $\boldsymbol{S}_{R(2)}$ will be truncated from $\boldsymbol{MAC}_{R(2)}$ using the method as shown in Figure 3. $\boldsymbol{f}_{R(2)}$ is a new seed used to update the responder's access control filter from $\boldsymbol{F}_{R(1)}$ to $\boldsymbol{F}_{R(2)}$, as illustrated in equation (20).

$$\boldsymbol{h}_{(MAC_{RF(1)})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{MAC}_{R(2)} \,||\, \boldsymbol{T}_{RF(2)} \,||\, \boldsymbol{f}_{R(2)}] \equiv \boldsymbol{F}_{R(2)} \quad (20)$$

$\boldsymbol{F}_{R(2)}$ is a new responder's access filter and is stored in the responder's access filter table for filtering the next frame. $\boldsymbol{S}_{R(2)}$ is a new seed that is used to update the responder's secret $\boldsymbol{MAC}_{RF(1)}$ to $\boldsymbol{MAC}_{RF(2)}$. When the responder receives further frames from the user, the responder will repeat the same three steps outlined for ***Frame 3***.

Frame 4 $\boldsymbol{R} \rightarrow \boldsymbol{U}$: $\boldsymbol{P}_{Rid(1)}$, $\boldsymbol{F}_{U(1)}$, $\boldsymbol{E}_{AES(K_{RF(1)})}$ [$\boldsymbol{Data}$, $\boldsymbol{T}_{RF(2)}$, $\boldsymbol{P}_{Uid(2)}$]

After the user receives ***Frame 4*** from the responder, the user then performs these three steps: First, checks $\boldsymbol{F}_{U(1)}$ to see if it matches the access control filter value based on pseudo ID $\boldsymbol{P}_{Rid(1)}$. If there is a match, the user can verify that this frame was sent from the responder, and then pass the encrypted frame to the system for decryption using the frame key $\boldsymbol{K}_{RF(1)}$. Since the user knows the same secrets $\boldsymbol{S}_{R(0)}$, $\boldsymbol{S}_{R(1)}$, $\boldsymbol{K}_U$, $\boldsymbol{K}_R$, and $\boldsymbol{T}_{RF(1)}$ as the responder, the user can perform the same calculation with the equation (18) as the responder to derive $\boldsymbol{MAC}_{RF(1)}$. Then, the user is able to get the frame key $\boldsymbol{K}_{RF(1)}$ to decrypt the encryption data and the timestamp $\boldsymbol{T}_{RF(2)}$. Second, use this secret $\boldsymbol{MAC}_{RF(1)}$ and timestamp $\boldsymbol{T}_{RF(2)}$ to perform the same calculation with equations (19) and (20) as the responder to derive $\boldsymbol{MAC}_{R(2)}$, $\boldsymbol{f}_{R(2)}$, $\boldsymbol{S}_{R(2)}$, and $\boldsymbol{F}_{R(2)}$. $\boldsymbol{F}_{R(2)}$ is a filter value that will be sent to the responder for checking. $\boldsymbol{S}_{R(2)}$ is a seed used to update the responder's secret from $\boldsymbol{MAC}_{RF(1)}$ to $\boldsymbol{MAC}_{RF(2)}$. Third, the user will generate a timestamp $\boldsymbol{T}_{UF(2)}$ and update the user's access control filter from $\boldsymbol{F}_{U(1)}$ to $\boldsymbol{F}_{U(2)}$ and pseudo ID from $\boldsymbol{P}_{Rid(1)}$ to $\boldsymbol{P}_{Rid(2)}$ to check the next frame sent from the responder using equations (21), (22) and (23). Two rounds are required to update the access control filter:

First Round: the new responder's frame key generation

$$\boldsymbol{h}_{\left(K_{U(j)}\right)}[\,\boldsymbol{S}_{U(0)} \,||\, \boldsymbol{S}_{U(1)} \,||\, \boldsymbol{K}_{R} \,||\, \boldsymbol{K}_{U} \,||\, \boldsymbol{T}_{UF(1)}] \equiv \boldsymbol{MAC}_{UF(1)} \quad (21)$$

$\boldsymbol{MAC}_{UF(1)}$ is a new responder's secret which replaces $\boldsymbol{MAC}_{UF(0)}$. $\boldsymbol{MAC}_{UF(1)}$ is used by the responder to generate $\boldsymbol{MAC}_{U(2)}$ to update the access control filter $\boldsymbol{F}_{U(1)}$ to $\boldsymbol{F}_{U(2)}$. $\boldsymbol{MAC}_{UF(1)}$ will be truncated into $\boldsymbol{K}_{UF(1)}$ and $\boldsymbol{K'}_{UF(1)}$. $\boldsymbol{K}_{UF(1)}$ used as a new frame key for AES to encrypt the data when the frame is sent to the user; $\boldsymbol{K'}_{UF(1)}$ is a key used in HMAC for data integrity.

Second Round: Calculate the new MAC value and update the access filter

$$\boldsymbol{h}_{\left(MAC_{UF(1)}\right)}[\,\boldsymbol{K}_{R} \,||\, \boldsymbol{K}_{U} \,||\, \boldsymbol{N}_{i} \,||\, \boldsymbol{N}_{R} \,||\, \boldsymbol{MAC}_{U(1)} \,||\, \boldsymbol{T}_{UF(2)}] \equiv \boldsymbol{MAC}_{U(2)} \quad (22)$$

$\boldsymbol{f}_{U(2)}$ and $\boldsymbol{S}_{U(2)}$ will be truncated from $\boldsymbol{MAC}_{U(2)}$ using the method shown in Figure 3. $\boldsymbol{f}_{U(2)}$ is a new seed used to update the responder's access control filter from $\boldsymbol{F}_{U(1)}$ to $\boldsymbol{F}_{U(2)}$, as illustrated in equation (23).

$$\boldsymbol{h}_{\left(MAC_{UF(1)}\right)}[\,\boldsymbol{K}_{U} \,||\, \boldsymbol{N}_{i} \,||\, \boldsymbol{MAC}_{U(2)} \,||\, \boldsymbol{T}_{UF(2)} \,||\, \boldsymbol{f}_{U(2)}] \equiv \boldsymbol{F}_{U(2)} \quad (23)$$

$\boldsymbol{F}_{U(2)}$ is a new responder's access filter value and is stored in the responder's access filter table for filtering the next frame. $\boldsymbol{S}_{U(2)}$ is a new seed that is used to update the responder's secret from $\boldsymbol{MAC}_{UF(1)}$ to $\boldsymbol{MAC}_{UF(2)}$. When the user receives further frames from the responder, the user will repeat the same three steps outlined for ***Frame 4***.

4.5 The Summary of the Dynamic Data Communication Stage

During the DDCS, the frames sent from the user to the responder should appear as follows:

$$\boldsymbol{U} \rightarrow \boldsymbol{R} : \boldsymbol{P}_{Uid(t)}, \boldsymbol{F}_{R(t)}, \boldsymbol{E}_{\left(K_{UF(t-1)}\right)}[Data, \boldsymbol{T}_{UF(t)}, \boldsymbol{P}_{Rid(t)}]\ t = 2, 3\ldots$$

Also, the frames sent from the responder to the user should appear as follows:

$$\boldsymbol{R} \rightarrow \boldsymbol{U} : \boldsymbol{P}_{Rid(t)}, \boldsymbol{F}_{U(t)}, \boldsymbol{E}_{\left(K_{RF(t-1)}\right)}\ [Data, \boldsymbol{T}_{RF(t)}, \boldsymbol{P}_{Uid(t+1)}]\ t = 1, 2, 3\ldots$$

Generally, during the DDCS, as shown in Figure 7, there are three steps needed for both the user and the responder. The user and the responder check the filter values when they receive the frames. If the filter value matches, then the new filter value is calculated to address the other side's filter. Next, they update their access control filters for filtering the next frame. If an administrator determines that an adversary is unable to capture and replay the filter value at the frame speed, then the filter values may be changed for multiple frames instead of every frame. However, in a high security environment, the filter values should be changed for every single frame.

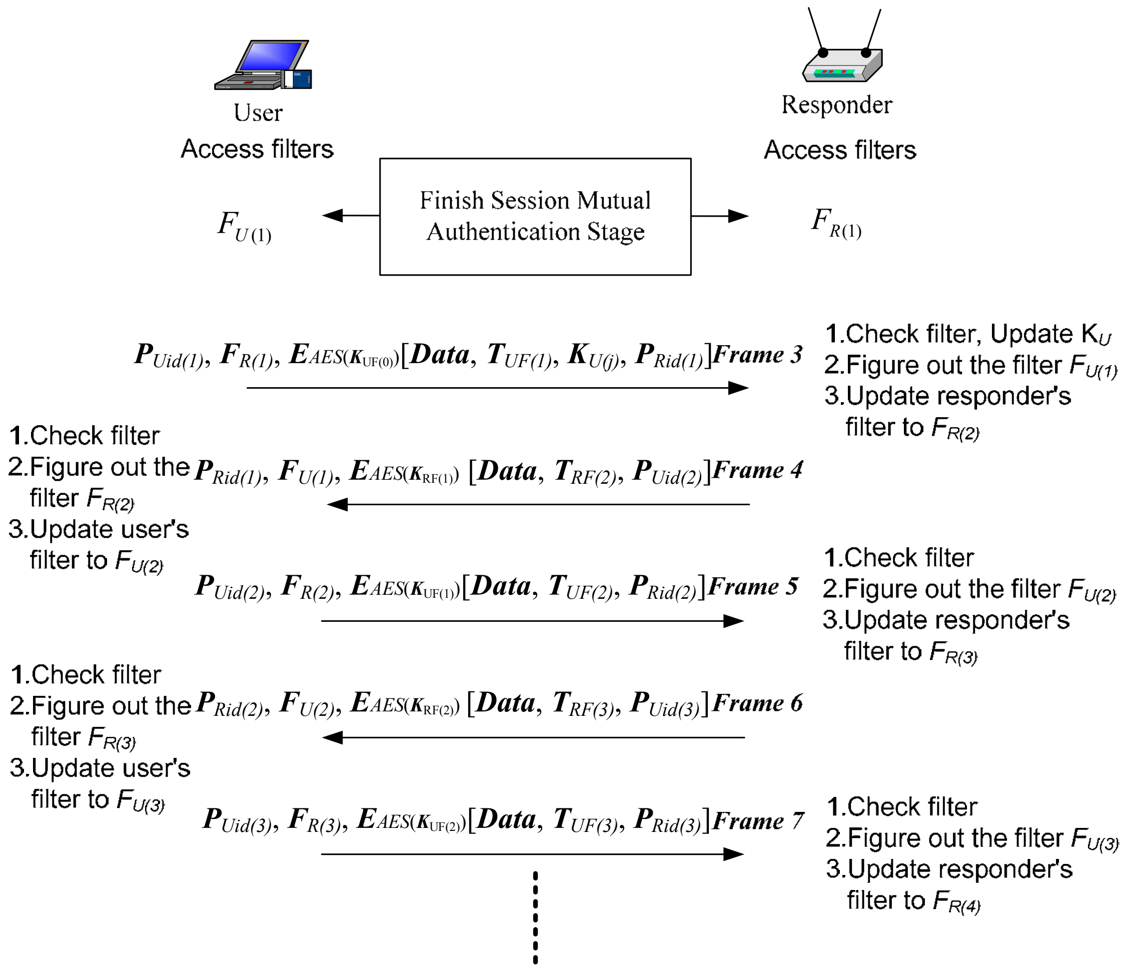

User
Access filters
$F_{U(1)}$
Finish Session Mutual Authentication Stage
Responder
Access filters
$F_{R(1)}$
$P_{Uid(1)}, F_{R(1)}, E_{AES(K_{UF(0)})}[Data, T_{UF(1)}, K_{U(j)}, P_{Rid(1)}]$Frame 3
1.Check filter, Update K_U
2.Figure out the filter $F_{U(1)}$
3.Update responder's filter to $F_{R(2)}$
1.Check filter
2.Figure out the filter $F_{R(2)}$
3.Update user's filter to $F_{U(2)}$
$P_{Rid(1)}, F_{U(1)}, E_{AES(K_{RF(1)})}[Data, T_{RF(2)}, P_{Uid(2)}]$Frame 4
$P_{Uid(2)}, F_{R(2)}, E_{AES(K_{UF(1)})}[Data, T_{UF(2)}, P_{Rid(2)}]$Frame 5
1.Check filter
2.Figure out the filter $F_{U(2)}$
3.Update responder's filter to $F_{R(3)}$
1.Check filter
2.Figure out the filter $F_{R(3)}$
3.Update user's filter to $F_{U(3)}$
$P_{Rid(2)}, F_{U(2)}, E_{AES(K_{RF(2)})}[Data, T_{RF(3)}, P_{Uid(3)}]$Frame 6
$P_{Uid(3)}, F_{R(3)}, E_{AES(K_{UF(2)})}[Data, T_{UF(3)}, P_{Rid(3)}]$Frame 7
1.Check filter
2.Figure out the filter $F_{U(3)}$
3.Update responder's filter to $F_{R(4)}$

Figure 7: The Overview of Dynamic Data Communication Stage

Three steps for a user:

First step: When a legitimate user receives the frame from a responder, the user compares it with the access control filter $\boldsymbol{F}_{U(t)}$ to see if there is a match using pseudo ID $\boldsymbol{P}_{Rid(t)}$. If the filter value matches the access filter value, the user will pass the encrypted data to the system for decryption, and then the system can derive the data and timestamp $\boldsymbol{T}_{RF(t+1)}$. Second step: Calculate the responder's access control filter using equations (27), (28) and (29). $\boldsymbol{f}_{R(t+1)}$ and $\boldsymbol{S}_{R(t+1)}$ will be truncated from $\boldsymbol{MAC}_{R(t+1)}$ using the same technique shown in Figure 3. $\boldsymbol{f}_{R(t+1)}$ is a new seed used for the user to calculate the responder's access control filter $\boldsymbol{F}_{R(t+1)}$ using equation (29). $\boldsymbol{F}_{R(t+1)}$ is a filter value sent with the user's encrypted data frame to the responder for checking so the encrypted data will be allowed in the responder's system. $\boldsymbol{S}_{R(t+1)}$ is a seed that must be stored in the user side for calculating the responder's secret $\boldsymbol{MAC}_{RF(t+1)}$. The same method employed for the responder with equation (27) needs to be used. This secret must also be stored on the user side for decrypting the next encrypted frame sent from the responder. $\boldsymbol{K}_{RF(t+1)}$ and $\boldsymbol{K'}_{RF(t+1)}$ will be truncated from $\boldsymbol{MAC}_{RF(t+1)}$. $\boldsymbol{K}_{RF(t+1)}$ is the frame key used by the user to decrypt the data and $\boldsymbol{K'}_{RF(t+1)}$ is the key for the data integrity. Third step: Generate a timestamp $\boldsymbol{T}_{UF(t+1)}$, and update the user's access filter from $\boldsymbol{F}_{U(t)}$ to $\boldsymbol{F}_{U(t+1)}$ and pseudo ID $\boldsymbol{P}_{Rid(t)}$ to $\boldsymbol{P}_{Rid(t+1)}$ for checking the next frame. Two rounds are required to update the access filter:

First Round: Frame key Generation Function

$$\boldsymbol{h}_{(K_{U(j)})}[\,\boldsymbol{S}_{U(t-1)} \,||\, \boldsymbol{S}_{U(t)} \,||\, \boldsymbol{N}_{i} \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{T}_{UF(t)}] \equiv \boldsymbol{MAC}_{UF(t)}\,,\ t = 1, 2, 3\ldots \quad (24)$$

$\boldsymbol{MAC}_{UF(t)}$ is the generation function for the user's frame key. It is an input key used by the HMAC to generate the new MAC value $\boldsymbol{MAC}_{U(t+1)}$ in equation (25) and to update the dynamic access control filter in equation (26). $\boldsymbol{K}_{UF(t)}$ and $\boldsymbol{K'}_{UF(t)}$ will be truncated from $\boldsymbol{MAC}_{UF(t)}$. $\boldsymbol{K}_{UF(t)}$ is the frame key used by the user to encrypt the data and $\boldsymbol{K'}_{UF(t)}$ is the key for the data integrity. Every frame has a unique frame key.

Second Round: Calculate the New MAC value and Update the User's Filter

$$\boldsymbol{h}_{(MAC_{UF(t)})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{MAC}_{U(t)} \,||\, \boldsymbol{T}_{UF(t+1)}] \equiv \boldsymbol{MAC}_{U(t+1)}\,,\ t = 1, 2, 3\ldots \quad (25)$$

$\boldsymbol{f}_{U(t+1)}$ and $\boldsymbol{S}_{U(t+1)}$ will be truncated from $\boldsymbol{MAC}_{U(t+1)}$ using the same technique as shown in Figure 4. $\boldsymbol{f}_{U(t+1)}$ is a new seed used to update the user's access control filter from $\boldsymbol{F}_{U(t)}$ to $\boldsymbol{F}_{U(t+1)}$ as illustrated in equation (26). $\boldsymbol{S}_{U(t+1)}$ is a seed used for the frame key generation function to update the secret from $\boldsymbol{MAC}_{UF(t)}$ to $\boldsymbol{MAC}_{UF(t+1)}$.

$$\boldsymbol{h}_{(MAC_{UF(t)})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{MAC}_{U(t+1)} \,||\, \boldsymbol{T}_{UF(t+1)} \,||\, \boldsymbol{f}_{U(t+1)}] \equiv \boldsymbol{F}_{U(t+1)}\,,\ t = 1, 2, 3\ldots \quad (26)$$

$\boldsymbol{F}_{U(t+1)}$ is used to replace the previous filter value $\boldsymbol{F}_{U(t)}$ and is stored in the user's filter table for checking the next frame sent from the responder.

Three steps for a responder:

First step: When a responder receives the frames from a user, the responder checks the access control filter $\boldsymbol{F}_{R(t)}$ to see if there is a match using pseudo ID $\boldsymbol{P}_{Uid(t)}$. If the received frame matches the access control filter value, the responder will pass the encrypted data to the system for decryption, and then the system can derive the data and timestamp $\boldsymbol{T}_{UF(t+1)}$. Second step: Calculate the user's access filter using equations (24), (25) and (26). $\boldsymbol{f}_{U(t+1)}$ and $\boldsymbol{S}_{U(t+1)}$ will be truncated from $\boldsymbol{MAC}_{U(t+1)}$ using the same technique shown in Figure 4. $\boldsymbol{f}_{U(t+1)}$ is a new seed used by the responder to calculate the user's access control filter $\boldsymbol{F}_{U(t+1)}$ using equation (26). $\boldsymbol{F}_{U(t+1)}$ is a filter value sent with the responder's encrypted data frame to the user for checking. $\boldsymbol{S}_{U(t+1)}$ is a seed that must be stored in the responder side for calculating the user's secret $\boldsymbol{MAC}_{UF(t)}$ via the same method employed for the user, i.e. equation (24). This secret must also be stored in the responder side for decrypting the next encrypted frame sent from user. $\boldsymbol{K}_{UF(t)}$ and $\boldsymbol{K'}_{UF(t)}$ will be truncated from $\boldsymbol{MAC}_{UF(t)}$. $\boldsymbol{K}_{UF(t)}$ is the frame key used by the responder to decrypt the data and $\boldsymbol{K'}_{UF(t)}$ is the key for the data integrity. Third step: Generate a timestamp $\boldsymbol{T}_{RF(t+1)}$, and update the responder's access control filter from $\boldsymbol{F}_{R(t)}$ to $\boldsymbol{F}_{R(t+1)}$ and pseudo ID $\boldsymbol{P}_{Uid(t)}$ to $\boldsymbol{P}_{Uid(t+1)}$for checking the next frame. Two rounds are needed to update the access control filter:

First Round: Frame key Generation Function

$$\boldsymbol{h}_{(K_{R(j)})}[\, \boldsymbol{S}_{R(t-1)} \,||\, \boldsymbol{S}_{R(t)} \,||\, \boldsymbol{K}_R \,||\, \boldsymbol{K}_U \,||\, \boldsymbol{T}_{RF(t)}] \equiv \boldsymbol{MAC}_{RF(t)}\,,\ t = 1, 2, 3\ldots \quad (27)$$

$\boldsymbol{MAC}_{RF(t)}$ is the generation function for the responder's frame key. $\boldsymbol{K}_{RF(t)}$ and $\boldsymbol{K'}_{RF(t)}$ will be truncated from $\boldsymbol{MAC}_{RF(t)}$. $\boldsymbol{K}_{RF(t)}$ is the frame key used by the responder to encrypt the data and $\boldsymbol{K'}_{RF(t)}$ is the key for the data integrity. Every frame has a unique frame key. $\boldsymbol{MAC}_{RF(t)}$ is an input key for the HMAC to generate the new MAC value $\boldsymbol{MAC}_{R(t+1)}$ in equation (28) and to update the dynamic access filter in equation (29).

Second Round: Calculate the New MAC value and Update the Responder's Filter

$$\boldsymbol{h}_{(MAC_{RF(t)})}[\, \boldsymbol{K}_R \,||\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{N}_{INI} \,||\, \boldsymbol{MAC}_{R(t)} \,||\, \boldsymbol{T}_{RF(t+1)}] \equiv \boldsymbol{MAC}_{R(t)}\,,\ t = 1, 2, 3\ldots \quad (28)$$

$\boldsymbol{f}_{R(t+1)}$ and $\boldsymbol{S}_{R(t+1)}$ will be truncated from $\boldsymbol{MAC}_{R(t+1)}$ using the same method illustrated in Figure 3. $\boldsymbol{f}_{R(t+1)}$ is a new seed used to update the responder's access control filter from $\boldsymbol{F}_{R(t)}$ to $\boldsymbol{F}_{R(t+1)}$ using equation (29). $\boldsymbol{S}_{R(t+1)}$ is a seed used by the frame key generation function in updating the secret from $\boldsymbol{MAC}_{RF(t)}$ to $\boldsymbol{MAC}_{RF(t+1)}$.

$$\boldsymbol{h}_{(K_{RF(t)})}[\,\boldsymbol{K}_U \,||\, \boldsymbol{N}_i \,||\, \boldsymbol{MAC}_{R(t+1)} \,||\, \boldsymbol{T}_{RF(t+1)} \,||\, \boldsymbol{f}_{R(t+1)}] \equiv \boldsymbol{F}_{R(t+1)},\ t = 1, 2, 3\ldots \quad (29)$$

$\boldsymbol{F}_{R(t+1)}$ is used to replace the previous filter value $\boldsymbol{F}_{R(t)}$ and is stored in the responder's filter table for filtering the next frame sent from the user.

4.6 Privacy Protected Filter Exchange

After the ICS, the responder will be able to generate $\boldsymbol{F}_{R(0)}$, and the $\boldsymbol{F}_{R(0)}$ for each user (initiator) will be stored in the responder's system. When the user wants to initiate the SMAS, the user will send over the corresponding $\boldsymbol{F}_{R(0)}$. The responder will have to find out which user is trying to perform SMAS. The responder will compare the $\boldsymbol{F}_{R(0)}$ (as a function of $\boldsymbol{P}_{Uid}$) with the filter table to see if there is a match. If there is a match, the responder will be able to know the user's identity and correspond with $\boldsymbol{K}_U$, $\boldsymbol{N}_i$, $\boldsymbol{N}_{INI}$, and $\boldsymbol{T}_{INI}$. What was described above introduces the privacy-protected feature of the IPACF protocol. With the user identity hidden in the filter value and pseudo ID, which sent in plaintext for the stateless entry in SMAS, no one will be able to know which user initiated SMAS except the responder. After the user finishes SMAS, the responder will need to update the $\boldsymbol{F}_{R(t)}$ (t=1, 2, 3…etc.) for the DDCS and the $\boldsymbol{F}_{R(0)}$ for the next session. The user will need to update the $\boldsymbol{F}_{U(t)}$ (t=1, 2, 3…etc.) for the DDCS and generate $\boldsymbol{F}_{R(0)}$ for the next session.

The pseudo code algorithm to store and update for both the pseudo ID and filter value table on the responder side in ICS, SMAS, and DDCS is as follows:

```
m;                          // maximum number of users
n;                          // number of users
$P_{Uid}$ [m];              // pseudo ID table
$F_R$ [m];                  // filter value table
// initialization
for( i = 0; i < m; i++){
    $P_{Uid}$ [i] = i;
}
j = m                                       // j is for available positions
// When a user registers and Do Loop will run for each user
For(k = 0; k < n-1; k++) {
    random # = rand() ;                     // generate random #
    New $P_{id}$ = random # mod j;          // new pseudo ID index
    j = j – 1;                              // decrease the user size
    // shift the Pseudo ID table
    for( i = New $P_{id}$;  i < j-1 ; i++){
        $P_{Uid}$ [i] = $P_{Uid}$ [i+1];
    }
    $F_R$ [New $P_{id}$] = initial filter value;     // store initial filter value
}
```

```
Do while(a user sends in a message)
// When a user sends in a message
Index$\boldsymbol{P}_{Uid}$;                           // the pseudo ID for the current user
random # = rand();                        // generate random #
New $\boldsymbol{P}_{id}$ = random # mod $\mathbf{j}$   ;             // new pseudo ID index
// shift the Pseudo ID table
for( i = New $\boldsymbol{P}_{id}$;  i < $\mathbf{j}$-1 ; i++){
        $\boldsymbol{P}_{Uid}$ [i] = $\boldsymbol{P}_{Uid}$ [i+1];
}
$\boldsymbol{F}_R$ [New $\boldsymbol{P}_{id}$] = updated filter value;        // store updated filter value
$\boldsymbol{P}_{Uid}$ [$\mathbf{j}$-1] = Index$\boldsymbol{P}_{Uid}$;                 // add the used pseudo ID back to table
```

Two tables are created for pseudo IDs and the filter values using arrays. The filter value table uses the pseudo ID as an index to store each filter value for a specific user. The pseudo ID table will be initialized as the index of the array with **m** maximum number of users. When **n** new user registers, the responder generates a random number and uses the remainder of the random number divided by **j** (mod **j)** as the new pseudo ID for each new user. The pseudo ID that is used is removed from the table by the responder and the **j** available positions for the next new user are updated by shifting up the pseudo ID table.

When a legitimate user sends in a message, the responder updates the registered user's pseudo ID and filter value for the next frame in SMAS and DDCS as follows. The responder generates the new pseudo ID for the user by using the same technique in ICS, removes the new pseudo ID from the available pseudo ID array by shifting up the pseudo

ID table, updates the filter value table for the new pseudo ID, and attaches the previous pseudo ID to the end of the pseudo ID array to ensure that the previous pseudo ID is available for other users. The same algorithm can be used on a user side.

Figure 8 shows the flowchart of updating filter value by both responder and user where $\boldsymbol{F(t, P_{id(t)})}$ is either $\boldsymbol{F_R(t, P_{Uid(t)})}$ or $\boldsymbol{F_U(t, P_{Rid(t)})}$. After the responder receives the filter value $\boldsymbol{F(t, P_{id(t)})}$ from a user, the responder will compare the filter value by comparing the filter value table by using the pseudo ID. If the filter value $\boldsymbol{F(t, P_{id(t)})}$ is found, the responder will find the new pseudo ID for the filter value $\boldsymbol{F(t+1, P_{id(t+1)})}$ and insert the filter value $\boldsymbol{F(t+1, P_{id(t+1)})}$, and then the filter value $\boldsymbol{F(t, P_{id(t)})}$ will be deleted from the filter value table. The user conducts similar operations as the responder.

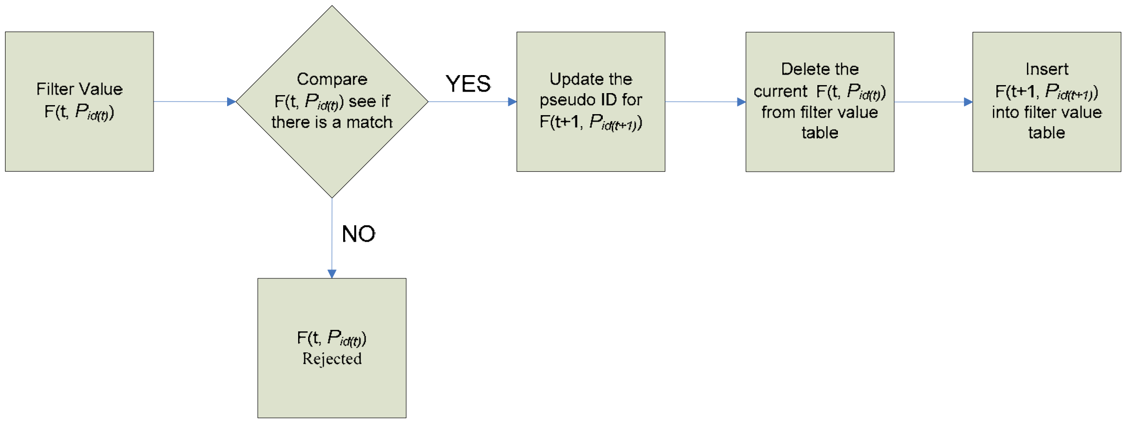

Filter Value F(t, P_id(t))
Compare F(t, P_id(t)) see if there is a match
YES
Update the pseudo ID for F(t+1, P_id(t+1))
Delete the current F(t, P_id(t)) from filter value table
Insert F(t+1, P_id(t+1)) into filter value table
NO
F(t, P_id(t)) Rejected

Figure 8: Filter Value Table Update Flowchart

CHAPTER FIVE

IMPLEMENTATION AND PERFORMANCE EVALUATION

5.1 Implementation

IPACF protocol has been implemented in Linux systems. The software is implemented, compiled, and run in an IBM ThinkPad T42 with a Pentium M processor running at 1.7 GHz, with a L2 cache of 2MB, 512 MB of main memory and connecting with CAT 5e cable to 100BASET Ethernet switch. IPACF protocol can also be split into a dual servers design as shown in Figure 9. With a dual server design, the server 1 can filter out the packets for legitimate users and route the legitimate requests to Server 2. Server 1 listens for connections from the user on specific port number, which allows only the legitimate packets to go through to Server 2 by comparing the received filter value based on pseudo ID. Server 1 rejects the packet right away if the filter value does not match; otherwise, it routes the packets to Server 2. The Server 2 provides the service that generates the new filter value, then updates the filter value and sends it back to legitimate users through Server 1.

When Server 1 receives a packet, Server 1 compares the filter value with the filter value table alone with the pseudo ID. If there is a match, Server 1 can obtain the hidden user ID from the filter table and routes the packet with the user ID to Server 2; otherwise, the packet is dropped. If the received packet is legitimate, Server 1 sends the packet to

Server 2 that performs the SMAS stage. Server 2 updates the filter value for the specific user and updates the pseudo ID to ensures that the user real identity is hidden with four steps: first, generate the new pseudo ID, and the filter value is updated from the filter value table; the computation costs $\boldsymbol{O}(1)$ to update. Second, Server 2 removes the original filter value from the filter value table; the computation costs $\boldsymbol{O}(1)$ to remove. After Server 2 completes filter value update, it sends the updated filter value with the new pseudo ID to Server 1. Server 1 then updates the filter value table in order to maintain its current status.

In a DoS/DDoS attack, the attacker attempts to make network services unavailable by flooding the authentication server in the network with numerous requests. The CPU usage eventually reaches its maximum and the server service becomes unavailable. A dual server design can improve request response time using Server 2 to calculate filter values and frame key so that Server 1 only performs the comparison of filter values. In this paper, we will demonstrate the IPACF protocol with both single server and dual server and conduct the comparison for the performance evaluation.

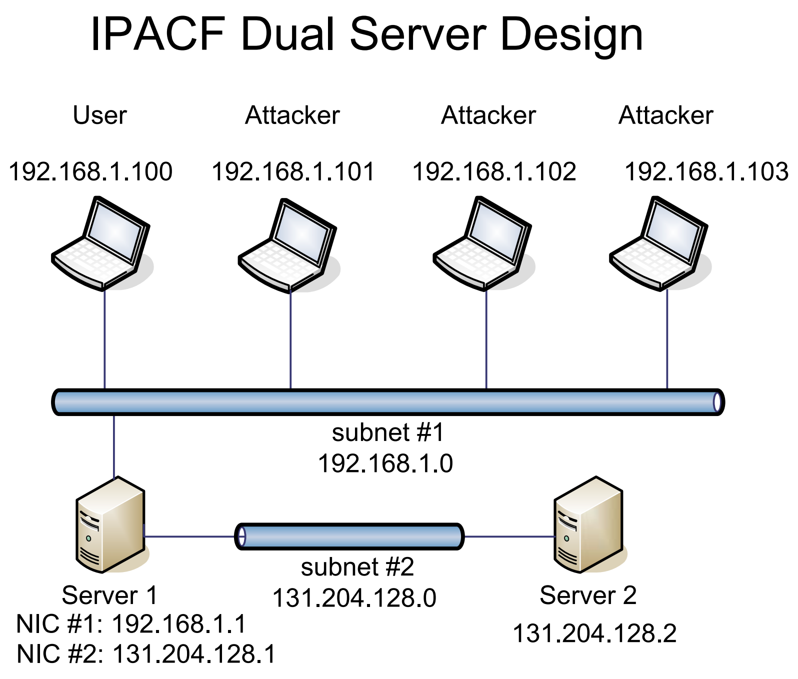


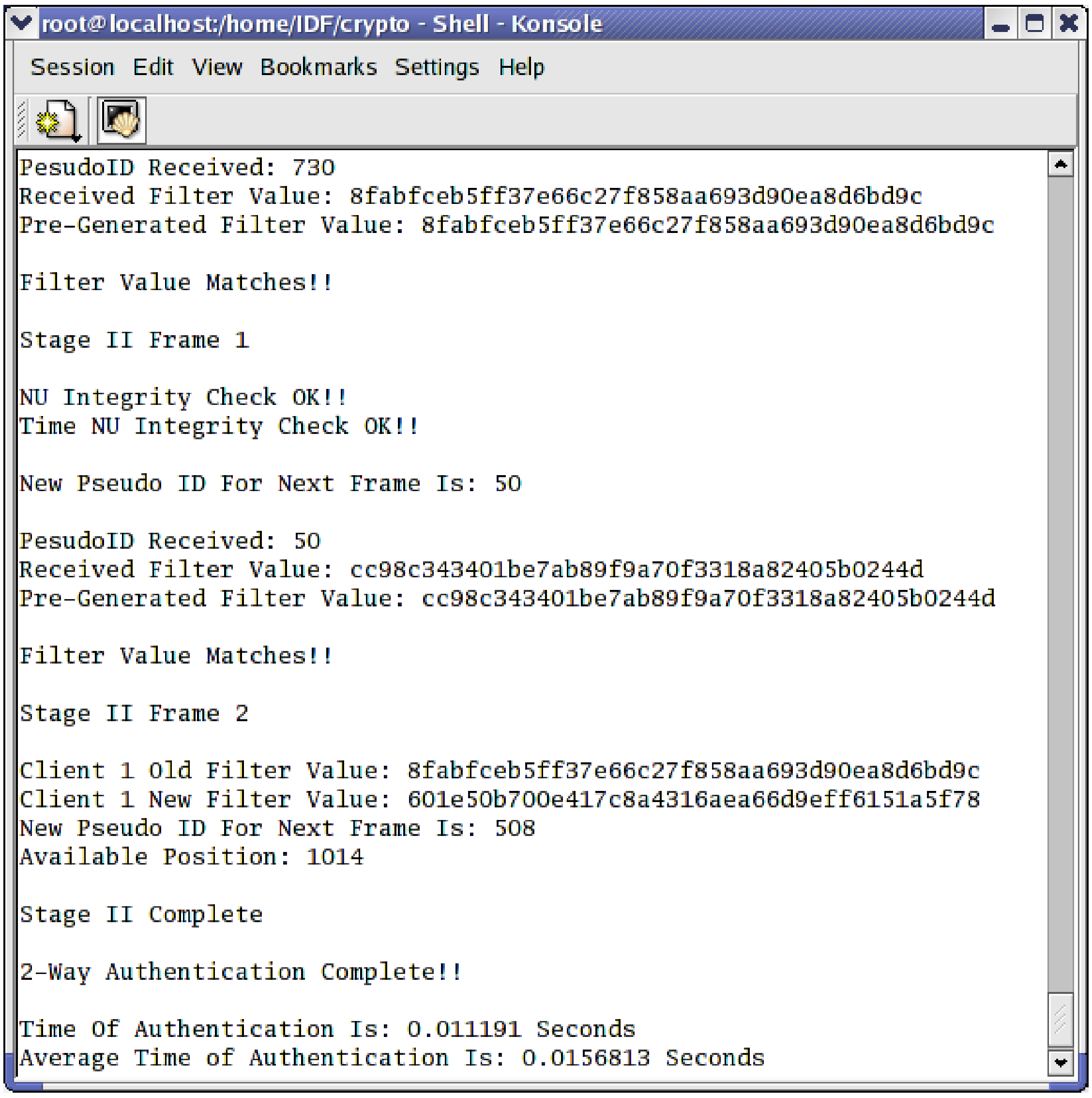
Figure 9: Diagram with dual servers design

IPACF protocol is implemented in RedHat Linux 9.0, using gcc and g++ compiler with Crypto++ Library. Crypto++ Library is a free C++ class library of cryptographic schemes. Integrity uses hash function HMAC with SHA1 keyed-hash and confidentiality uses AES are tested. In this experiment, we use 128 bits for both key size and block size in AES.

5.2 Performance

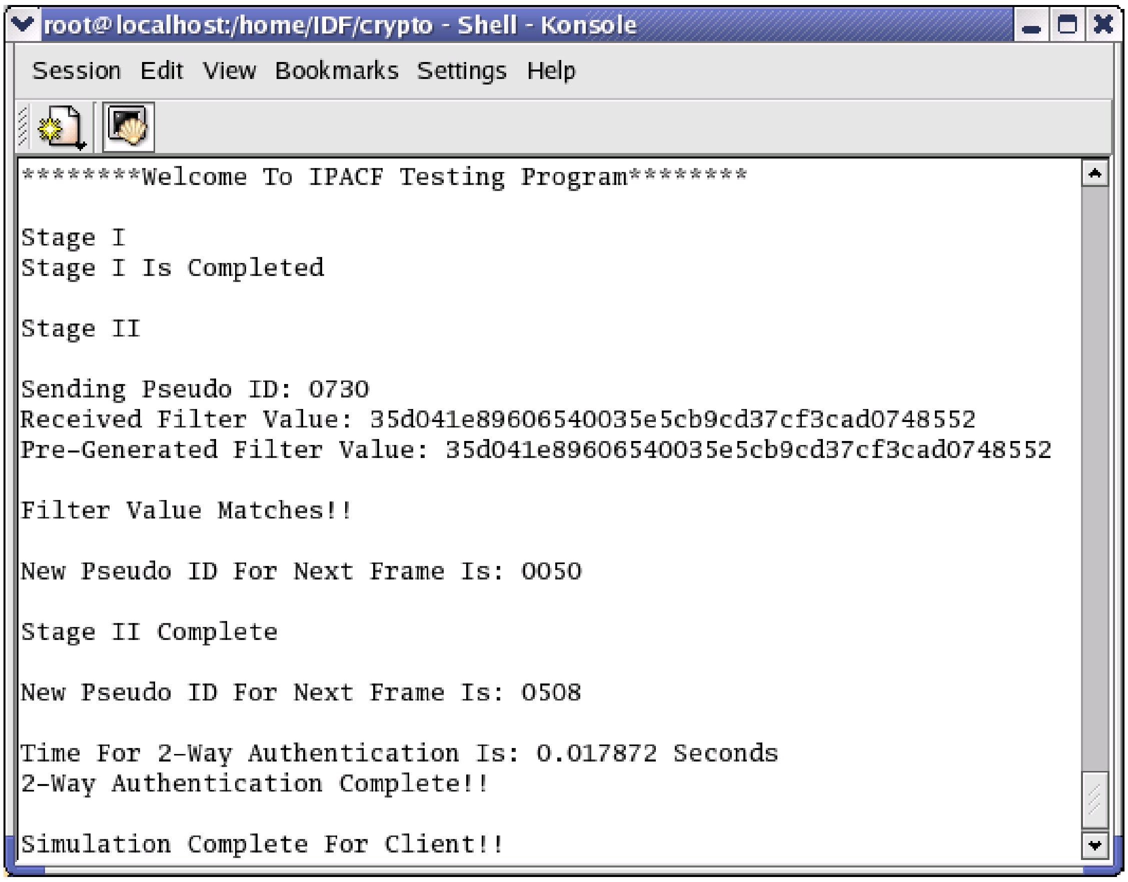
5.2.1 Performance in authentication

The server accepts the packet from a user and verifies the filter value. If the corresponding filter value of the user matches, the server will perform the SMAS. During the SMAS stage, after the server verifies the filter value, the server has authenticated the user. The server will decrypt the secrets, which are the time stamp and nonce, sent by the user to perform the same calculation for the user to authenticate the server. The server will send the filter value, time stamp, and the master secret key $\boldsymbol{K}_{R(j)}$ back to the user, and allow the user to verify the filter value and update the server secret master key from $\boldsymbol{K}_{R(j)}$ to $\boldsymbol{K}_{R(j+1)}$. After the user authenticates the server by comparing the filter value sent by the server, the SMAS is completed, and the filter value $\boldsymbol{F}_{R(0)}$ will be updated for the DDCS and for next session login. Figure 10 and Figure 11 show the session mutual authentication for the responder and the user, respectively. The averaged wired authentication time for the responder in SMAS is 4.2003 millisecond (ms) while IDF needs 181.55 ms, IPACF needs only 2.31% authentication time of IDF in wired links; the averaged wireless (using 802.11b instead of Ethernet) authentication time for the responder in SMAS is 9.12124 ms while IDF needs 189.37 ms, IPACF needs only 4.86% authentication time of IDF in wireless links.


root@localhost:/home/IDF/crypto - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
PesudoID Received: 730
Received Filter Value: 8fabfceb5ff37e66c27f858aa693d90ea8d6bd9c
Pre-Generated Filter Value: 8fabfceb5ff37e66c27f858aa693d90ea8d6bd9c

Filter Value Matches!!

Stage II Frame 1

NU Integrity Check OK!!
Time NU Integrity Check OK!!

New Pseudo ID For Next Frame Is: 50

PesudoID Received: 50
Received Filter Value: cc98c343401be7ab89f9a70f3318a82405b0244d
Pre-Generated Filter Value: cc98c343401be7ab89f9a70f3318a82405b0244d

Filter Value Matches!!

Stage II Frame 2

Client 1 Old Filter Value: 8fabfceb5ff37e66c27f858aa693d90ea8d6bd9c
Client 1 New Filter Value: 601e50b700e417c8a4316aea66d9eff6151a5f78
New Pseudo ID For Next Frame Is: 508
Available Position: 1014

Stage II Complete

2-Way Authentication Complete!!

Time Of Authentication Is: 0.011191 Seconds
Average Time of Authentication Is: 0.0156813 Seconds
```

Figure 10: The SMAS for the responder

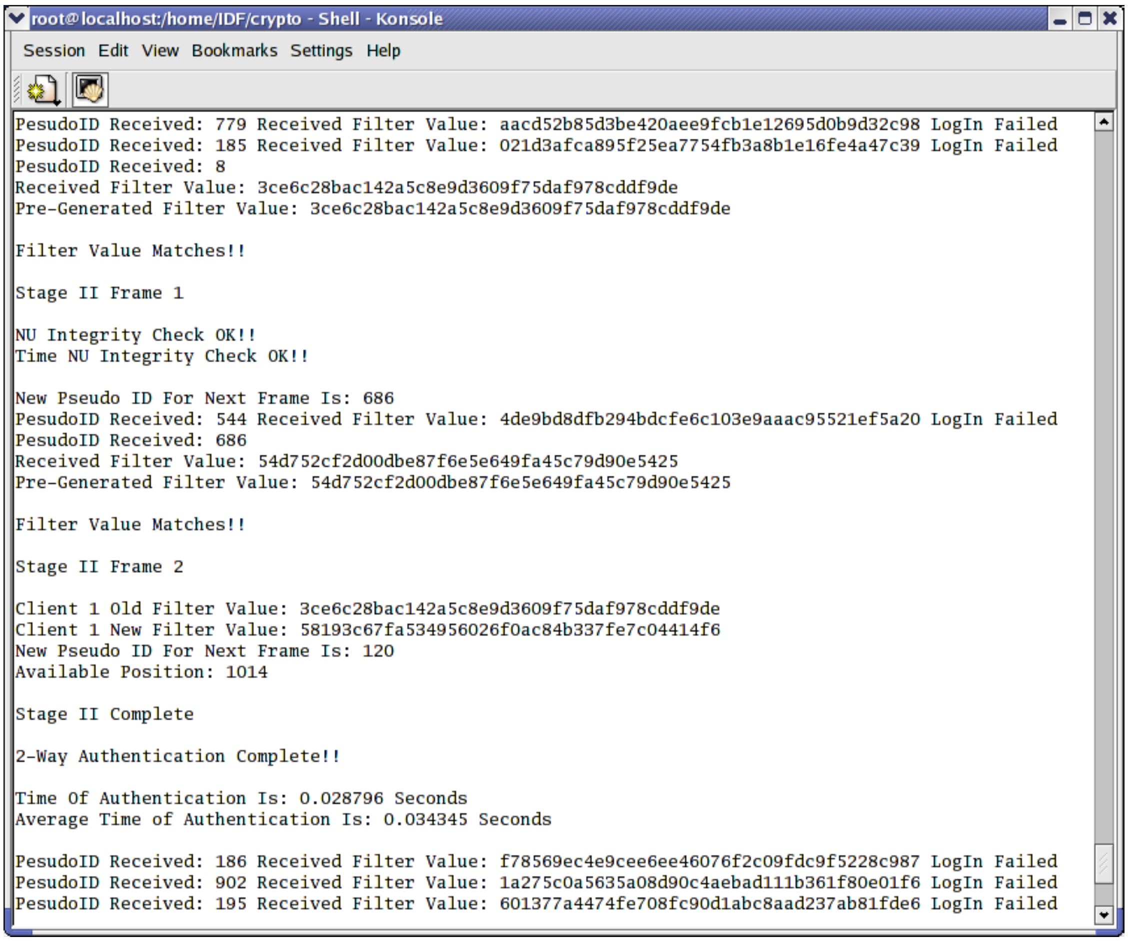

root@localhost:/home/IDF/crypto - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
********Welcome To IPACF Testing Program********

Stage I
Stage I Is Completed

Stage II

Sending Pseudo ID: 0730
Received Filter Value: 35d041e89606540035e5cb9cd37cf3cad0748552
Pre-Generated Filter Value: 35d041e89606540035e5cb9cd37cf3cad0748552

Filter Value Matches!!

New Pseudo ID For Next Frame Is: 0050

Stage II Complete

New Pseudo ID For Next Frame Is: 0508

Time For 2-Way Authentication Is: 0.017872 Seconds
2-Way Authentication Complete!!

Simulation Complete For Client!!
```

Figure 11: The SMAS for the user

5.2.2 Performance in DoS attacks

When a hacker performs an attack to the server, we can see the screen capture in which packets get rejected by the server after the verification of the filter value is failed as shown in Figure 12. When a legitimate user tries to get authenticated by the server while attackers are trying to perform the attacks to the server, we can see the rejected messages from the server. When the attack is performed, the server shows the “Login Failed” message after comparing the filter value that does not match. When the SMAS is performed successfully for the legitimate user, we can see “Filter Value Matches” in Figure 12.


root@localhost:/home/IDF/crypto - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
PesudoID Received: 779 Received Filter Value: aacd52b85d3be420aee9fcb1e12695d0b9d32c98 LogIn Failed
PesudoID Received: 185 Received Filter Value: 021d3afca895f25ea7754fb3a8b1e16fe4a47c39 LogIn Failed
PesudoID Received: 8
Received Filter Value: 3ce6c28bac142a5c8e9d3609f75daf978cddf9de
Pre-Generated Filter Value: 3ce6c28bac142a5c8e9d3609f75daf978cddf9de

Filter Value Matches!!

Stage II Frame 1

NU Integrity Check OK!!
Time NU Integrity Check OK!!

New Pseudo ID For Next Frame Is: 686
PesudoID Received: 544 Received Filter Value: 4de9bd8dfb294bdcfe6c103e9aaac95521ef5a20 LogIn Failed
PesudoID Received: 686
Received Filter Value: 54d752cf2d00dbe87f6e5e649fa45c79d90e5425
Pre-Generated Filter Value: 54d752cf2d00dbe87f6e5e649fa45c79d90e5425

Filter Value Matches!!

Stage II Frame 2

Client 1 Old Filter Value: 3ce6c28bac142a5c8e9d3609f75daf978cddf9de
Client 1 New Filter Value: 58193c67fa534956026f0ac84b337fe7c04414f6
New Pseudo ID For Next Frame Is: 120
Available Position: 1014

Stage II Complete

2-Way Authentication Complete!!

Time Of Authentication Is: 0.028796 Seconds
Average Time of Authentication Is: 0.034345 Seconds

PesudoID Received: 186 Received Filter Value: f78569ec4e9cee6ee46076f2c09fdc9f5228c987 LogIn Failed
PesudoID Received: 902 Received Filter Value: 1a275c0a5635a08d90c4aebad111b361f80e01f6 LogIn Failed
PesudoID Received: 195 Received Filter Value: 601377a4474fe708fc90d1abc8aad237ab81fde6 LogIn Failed
```

Figure 12: User performs SMAS to the responder while attackers perform DoS attack

As shown in Figure 13, the averaged rejection time as a function of the number of attacking PCs is defined as the time between an attacker sends out ***Frame 1*** and receives the rejection from the server. Both IPACF and IDF have an averaged reject time 5.90 ns (nanoseconds) by the responder.

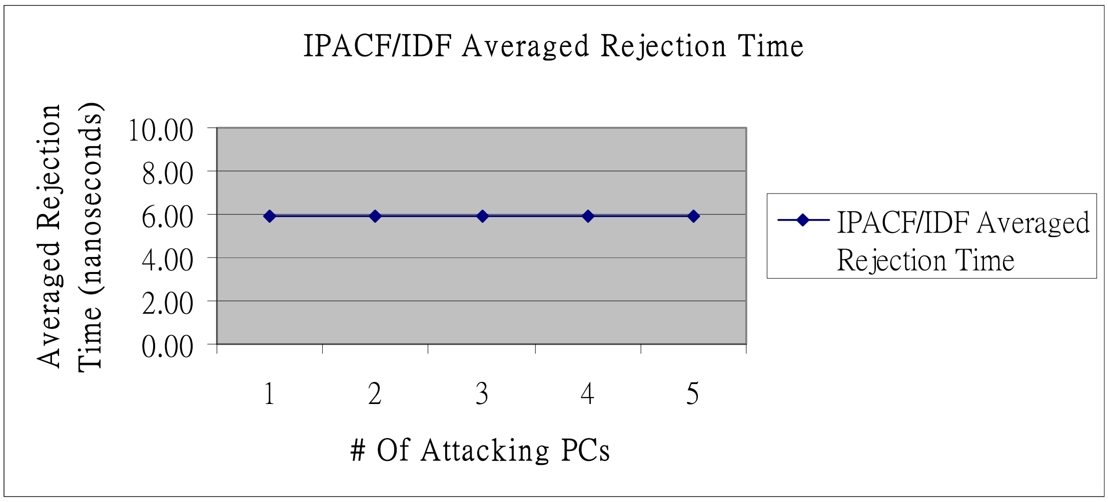


Figure 13: Comparison chart of IPACF and IDF in averaged rejection time

5.2.2 Performance in dual server over single server

In our experiments, the dual server needs only 5.4% authentication time of single server while a number of attackers are performing attacks as shown in Figure 14.

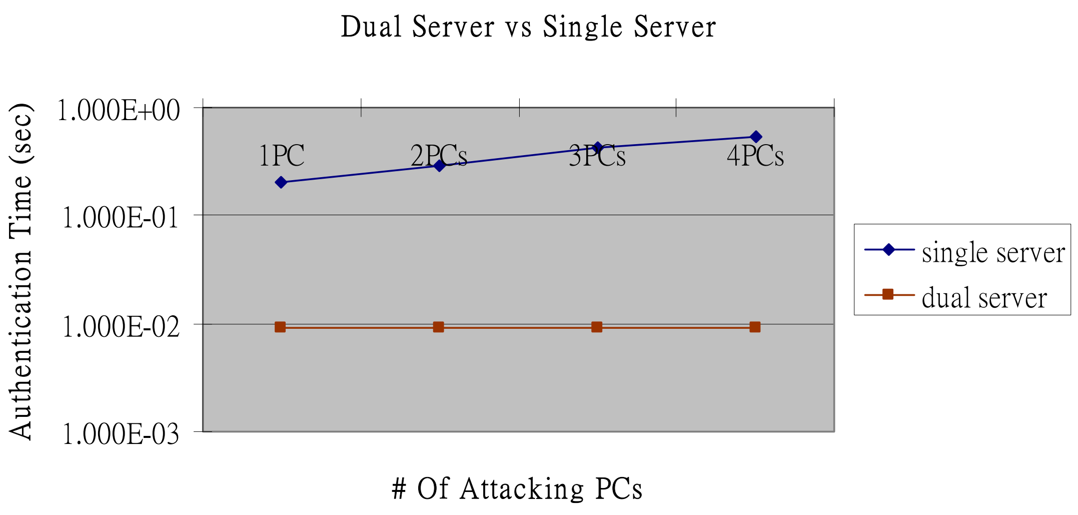
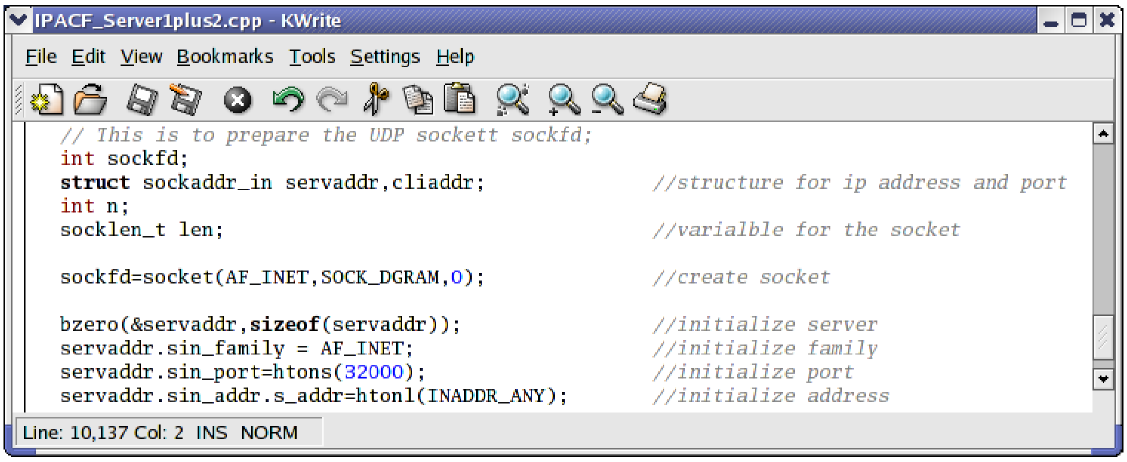


Figure 14: Comparison chart of dual server vs single server in average round trip time

5.3 Interoperability of IPv4 (user application) and IPv6 (responder)

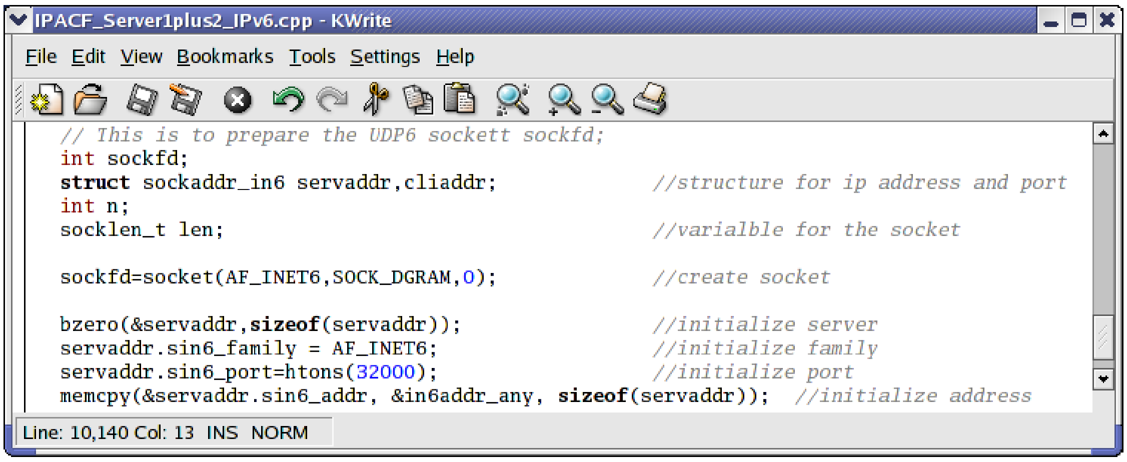
The IPACF implementation is designed to be compatible with IPv6 networks, a server code is bound to the IPv6 address and the server will be able to accept connections from IPv4 and IPv6 clients. When an IPv4 client is connecting to the IPv6 server, the dual stack kernel converts the client IPv4 address to the IPv4-mapped IPv6 address since the IPv6 server can only deal with IPv6 connections. When porting to IPv6, most of changes will be made in the transport module, which is User Datagram Protocol that is in charge of establishing communications to remote nodes. If the IPACF implementation is changed to IPv6, only the transport module should be modified. Figure 15 and Figure 16 show the difference between UDP and UDP6 socket. The **inet6** family is an IPv6 version of **inet4** family. While **inet4** implements Internet Protocol version 4, **inet6** implements Internet Protocol version 6. **inet6** is a collection of protocols layered atop the Internet Protocol version 6 (IPv6) transport layer, and utilizing the IPv6 address format. UDP/UDP6 is used to support the SOCK_DGRAM abstraction (UDP) in **inet6** family that provides access to the IPv6 protocol.

Each protocol-specific data structure is designed to carry the addresses for each protocol, so it can be cast into a protocol-independent data structure - the "sockaddr" structure. The sockaddr_in structure is the protocol-specific address data structure for IPv4; the sockaddr_in6 structure is the protocol-specific address data structure for IPv6. They both pass addresses between applications and the system in the socket programming functions. A new address family name, AF_INET6, distinguishes between the original AF_INET sockaddr_in address data structure and the new sockaddr_in6 data structure. The sin6_port field contains the 16-bit UDP port number. This field is used in the same way as the sin_port field of the sockaddr_in structure and the port number is stored in the network byte order. Applications use in6addr_any similarly to the way that they use INADDR_ANY in IPv4. In Figure 15, the server creates a sockaddr_in structure with AF_INET family, which contains its source IPv4 address to bind the socket to port number 32000. In Figure 16, the server creates a sockaddr_in6 structure with AF_INET6 family, which contains its source IPv6 address to bind the socket to port number 32000.


IPACF_Server1plus2.cpp - KWrite
File Edit View Bookmarks Tools Settings Help

```
// This is to prepare the UDP sockett sockfd;
int sockfd;
struct sockaddr_in servaddr,cliaddr;            //structure for ip address and port
int n;
socklen_t len;                                  //varialble for the socket

sockfd=socket(AF_INET,SOCK_DGRAM,0);            //create socket

bzero(&servaddr,sizeof(servaddr));              //initialize server
servaddr.sin_family = AF_INET;                  //initialize family
servaddr.sin_port=htons(32000);                 //initialize port
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);     //initialize address
```

Figure 15: UDP socket


IPACF_Server1plus2_IPv6.cpp - KWrite
File Edit View Bookmarks Tools Settings Help

```
// This is to prepare the UDP6 sockett sockfd;
int sockfd;
struct sockaddr_in6 servaddr,cliaddr;           //structure for ip address and port
int n;
socklen_t len;                                  //varialble for the socket

sockfd=socket(AF_INET6,SOCK_DGRAM,0);           //create socket

bzero(&servaddr,sizeof(servaddr));              //initialize server
servaddr.sin6_family = AF_INET6;                //initialize family
servaddr.sin6_port=htons(32000);                //initialize port
memcpy(&servaddr.sin6_addr, &in6addr_any, sizeof(servaddr));  //initialize address
```

Figure 16: UDP6 socket

CHAPTER SIX

CONCLUSIONS AND FUTURE WORKS

In this thesis, we introduced an identity-based privacy-protected access control filter (IPACF) to solve DoS/DDoS problems. The IPACF protocol provides the following unique properties.

- The IPACF filter is based on the legitimate users' identities, which is hidden in the filter values that are generated by the pre-shared secrets, nonce, timestamp, user ID and password.
- The IPACF filter value varies with every frame for both responder and user to prevent sniffing attacks.
- A filter value table is initialized for both users and responder during ICS. The identities of both users and responder are tabulated with pseudo ID in the filter value table.
- The privacy of both user and responder is guarded by the one-time filter value, which is the temporary equivalent identity that is accessible in the communication. Only the legitimate user and responder can figure out the identity from the filter value table.
- The IPACF protocol is stateless because the input filter value is checked against the filter table without creating a state unless the filter value is legitimate. When a

legitimate filter value is received, a new state is created by calculating the new filter value for the next frame. When a legitimate filter value comes in, a sorted filter value table is maintained by deleting the old filter value, searching the new index for the new filter value and inserting it into the filter table.

- The stateless property provides the capability to resist DoS/DDoS attacks.

The IPACF protocol ensures a secure update for both user and responder session master key $\boldsymbol{K}_{R(j)}$ and $\boldsymbol{K}_{U(j)}$. A pairwise key $\boldsymbol{K}_{\mathrm{R}}$ exists for each user so that a user cannot pretend to be a responder.

Future research direction is proposed as follows. In PKI ((Public Key Infrastructure) authentication protocols, a server must use the system resources to compute and store a hash, it makes the DoS/DDoS attack feasible [12]. When a web server integrates with the IPACF protocol, the web server immediately turns into an application server to offer safe, secure information exchange to registered/legitimate clients. The implementation of PKI that combined with IPACF to defend against DoS/DDoS attacks is via web server and client certificates.

There are three steps during the initial configuration stage in a secured channel such as a SSL (Secure Socket Layer) channel. First, a client and the web server exchange their certificates via a SSL channel. Second, a client generates the master secret key $\boldsymbol{K}_U$ using equation (30) without a user ID and a password, using its private key $\boldsymbol{K}_{PU}$ instead, encrypts $\boldsymbol{K}_U$ and $\boldsymbol{N}_i$ with the web server's public key, and then sends the encrypted information to the web server. Third, after the web server receives the encrypted information, it will decrypt the received information using its private key $\boldsymbol{K}_{PR}$. The web server will generate the master secret key $\boldsymbol{K}_R$ using equation (31) without a responder ID and a password, using its private key $\boldsymbol{K}_{PR}$ instead, encrypts $\boldsymbol{K}_R$, $\boldsymbol{T}_{INI}$ and $\boldsymbol{N}_{INI}$ with the client's public key, and then sends the encrypted information back to the client. Since most of users do not have a certificate, as an alternative, a user can use a user ID and a password instead of a private key as shown in equation (1).

$$\boldsymbol{K}_U = \boldsymbol{h}_{(N_{ui})}[\,\boldsymbol{K}_{PU} \,||\, \boldsymbol{T}_u \,||\, \boldsymbol{N}_{ui}\,] \quad (30)$$

$$\boldsymbol{K}_R = \boldsymbol{h}_{(N_{Ri})}[\,\boldsymbol{K}_{PR} \,||\, \boldsymbol{T}_R \,||\, \boldsymbol{N}_{Ri}\,] \quad (31)$$

After the initial configuration stage, the client will be able to perform authentication with the web server. The authentication scheme will be stateless for both client and web server to against DoS/DDoS attacks. The initial configuration stage will not need to be performed unless it is necessary for a new client to register. The session key will be generated for any further authentication.

The web server uses the authentication and encryption/decryption services of IPACF and can securely communicate with clients, which will be able to use the server-side resources. The IPACF protocol provides compatibility with the PKI schemes to prevent a DoS/DDoS attack.

BIBLIOGRAPHY

[1] IEEE Standard for Local and metropolitan area networks: Port-Based Network Access Control, IEEE Std 802.1X-2003.

[2] IEEE Std 802.11i/D4.1, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Security Enhancements," July 2003.

[3] A. Saxena and B. Soh, "Distributed Denial of Service Attacks and Anonymous Group Authentication on the Internet," Third International Conference on Information Technology and Applications, vol. 2, ICITA 2005, pp.460-464.

[4] C. Wang, C. Wu and J. D. Irwin, "Using an Identity-Based Dynamic Access Control Filter (IDF) to Defend Against DoS Attacks," In *IEEE Wireless Communications and Networking Conference*, vol. 1, March 2004, pp. 639-645.

[5] P. Owezarski, "On the impact of DoS attacks on Internet traffic characteristics and QoS," In *Proc. 14th International Conference on Computer Communications and Networks, ICCCN 2005*, pp. 268-274.

[6] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," In *Proc. of the Network and Distributed Systems Security Symposium (NDSS '99)*, February 1999, pp. 151–165.

[7] T. Aura, P. Nikander, and J. Leiwo, " DOS-resistant authentication with client puzzles," In *Proc. of the 8th International Workshop on Security Protocols*, April 2000, pp.170-177.

[8] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis, and O. Reingold, "Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols," In *Proceedings of the 9th ACM conference on Computer and communications security*, Washington D.C., 2002, pp. 48-58.

[9] K. Matsuura and H. Imai, "Modified aggressive mode of Internet key exchange resistant against denial-of-service attacks," In *IEICE Transactions on Information and Systems*, May 2000, pp. 972–979.

[10] J. Leiwo, P. Nikander, and T. Aura, "Towards network denial of service resistant protocols," In *Proc. of the 15th International Information Security Conference (IFIP/SEC)*, August 2000, pp. 301-310.

[11] C. Meadows, "A formal framework and evaluation method for network denial of service," In *Proc. of the 12th IEEE Computer Security Foundations Workshop*, June 1999, pp. 4–13.

[12] W. Zhiguo, Zhu Bo, R.H. Deng, Bao Feng, and A.L. Ananda, "DoS-resistant access control protocol with identity confidentiality for wireless networks," *IEEE Wireless Communications and Networking Conference*, vol. 3, March 2005, pp. 1521-1526.

[13] JAN, J.K., and TSENG, Y.M., "Two integrated schemes of user authentication and access control in a distributed computer network", *IEE Proc. Comput. Digit. Tech.*, 1998, 145, (6), pp. 419-424.

[14] W.H He, and T.C. Wu, "Security of the Jan-Tseng integrated schemes for user authentication and access control" *IEE Proc. Comput. Digit. Tech.,* 147, (5), 2002, pp. 365-368.

[15] G. Carl, G.Kesidis, R.R.Brooks, Rai Suresh, "Denial-of-service attack-detection techniques," *IEEE Internet Computing*, vol. 10, January 2006, pp.82-89.

[16] J. Mirkovic, J. Martin, and P. Reiher, "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms," *ACM Sigcomm Computer Comm. Rev.*, vol. 34, no. 2, 2004, pp. 39–53.

[17] A. Ferrante, V. Piuri, and J. Owen, "IPSec hardware resource requirements evaluation," *Next Generation Internet Networks*, April 2005, pp. 240-246.

[18] M. Bellare, and R. Canetti, "HMAC: keyed-hashing for message authentication," Request for Comments 2104, Internet Engineering Task Force, February 1997.

[19] T. Aura and P. Nikander, "Stateless connections," In *Proc. Of International Conferenec on Information and Communications Security (ICICS '97), Lecture Notes in Computer Science*, vol. 1334, Springer, November 1997, pp. 87–97.

[20] P. Janson, G. Tsudik, and M. Yung, "Scalability and flexibility in authentication services: The KryptoKnight approach," In *IEEE INFOCOM'97*, Tokyo, April 1997, pp. 725-736.

[21] R. C.Merkle, "Secure communications over insecure channels," *Communications of the ACM*, vol. 21, April 1978, pp. 294– 299.

[22] C. Dwork and M. Naor, "Pricing via processing or combating junk mail," In *E. Brickell, editor, Proceedings of Advances in Cryptology - Proc. CRYPTO '92*, vol. *1323*, Santa Barbara, CA USA , August 1992, Springer-Verlag, pp. 139–147.

[23] D. Dean and A. Stubblefield, "Using Client Puzzles to Protect TLS," In *Proceedings of the 10th USENIX Security Symposium*, Washington D.C., August 2001, pp. 1-8.

[24] X.F. Wang and M.K. Reiter, "Defending Against Denial-of-Service Attacks with Puzzle Auctions," In *IEEE Symposium on Security and Privacy*, May 2003, pp. 78-92.

[25] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten, "New client puzzle outsourcing techniques for DoS resistance," *The 11th ACM Conference on Computer and Communications Security (CCS 2004)* ACM Press, 2004, pp. 246–256.

[26] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," In *10th USENIX Security Symposium*, 2001, pp. 1–8.

[27] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," In *Proc. of the Network and Distributed Systems Security Symposium (NDSS '99)*, February 1999, pp. 151–165.

[28] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," Request for Comments 2409, Internet Engineering Task Force, November 1998.

[29] N. Ferguson and B. Schneier, "A Cryptographic Evaluation of IPSec," In *http://www.counterpane.com/ipsec.pdf*, January 2000.

[30] C. Kaufman and R. Perlman, "Analysis of IKE," In *IEEE Transactions on Network Computing*, vol. 4, November 2000, pp. 50-56.

[31] Matsuura and H. Imai, "Resolution of ISAKMP/Oakley key-agreement protocol resistant against denial-of-service attack," In *Proc. of Internet Workshop (IWS '99)*, February 1999, pp. 17–24.

[32] W. A. Simpson, "IKE/ISAKMP Considered Harmful," USENIX ;login:, December 1999, pp. 48-58.

[33] D. Harkins, C. Kaufman, S. Kent, T. Kivinen, and R. Perlman, "Proposal for the IKEv2 Protocol," In *draft-ietf-ipsec-ikev2-01.txt*, Internet Engineering Task Force, April 2002. Work in progress.

[34] K. Matsuura and H. Imai, "Resolution of ISAKMP/Oakley key-agreement protocol resistant against denial-of-service attack," In *Proc. of Internet Workshop* (IWS '99), February 1999, pp. 17-24.

[35] M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," Abridged version appears in *CRYPTO '96, vol. 1109 of Lecture Notes in Computer Scienc*, Springer-Verlag, 1996 *e,* pp. 1-15.

[36] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," In *Proc. of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security*, September 1999, pp. 258-272.

[37] R. Oppliger, "Protecting key exchange and management protocols against resource clogging attacks," In *Proc. of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99)*, September 1999, pp. 163–175.

[38] S. Hirose and K. Matsuura, "Enhancing the resistance of a provably secure key agreement protocol to a denial-of-service attack", In *Proceedings of the 2nd International Conference on Information and Communication Security (ICICS '99)*, Lecture Notes in Computer Science vol. 1726, Sydney, Australia, November 1999, Springer, pp. 169–182.

[39] M.T. Goodrich, "Leap-frog packet linking and diverse key distributions for improved integrity in network broadcasts," In *IEEE Symposium on Security and Privacy*, May 2005, pp. 196-207.