

Using an Identity-Based Dynamic Access Control Filter (IDF) to Defend Against DoS Attacks

Chien-Cheng Wang

Dept. of Electrical and Computer
Engineering
Auburn University
Auburn, Alabama 36849, USA
wangchi@auburn.edu

Chwan-Hwa “John” Wu

Dept. of Electrical and Computer
Engineering
Auburn University
Auburn, Alabama 36849, USA
wu@eng.auburn.edu

J.David Irwin

Dept. of Electrical and Computer
Engineering
Auburn University
Auburn, Alabama 36849, USA
jdirwin@eng.auburn.edu

Abstract—A denial of service (DoS)¹ attack is a well-known problem for networks. Although many research papers have addressed the problem of DoS attacks, some drawbacks still exist. The major problem is this: when a responder receives a request or key message, and system resources are required to verify that the request has been sent from a legitimate user², the additional computing time and memory storage involved provides an adversary the opportunity to launch a DoS attack. In order to prevent this attack, we present a new protocol called the Identity-Based dynamic access control filter (IDF), which checks every filter value on every received frame to determine if they have permission to enter the system and use system resources. In this paper we focus on wireless network protection, but the same concept can be applied to a wired network as well. A performance analysis of the critical step used in preventing a DoS attack is conducted and the result is compared with other Public Key-based schemes.

Keywords—Wireless network, Information Security; AAA; Denial of Service Attacks

I. INTRODUCTION

The current IEEE 802.1x or ongoing 802.11i standard still lack the ability to prevent DoS attacks on access points (APs) or authentication, authorization and accounting (AAA) servers. Placing more arms, e.g. cryptography, around these standards will simply make it easier for attackers to launch DoS attacks as a result of the added computational requirement. Attackers can launch DoS attacks by sending numerous faked request messages and secret keys to the responder³. This process will keep the responder busy verifying them. If a victim performs the required computation and stores information about the state of each illegitimate request, the victim will be swamped in a short period of time.

We launched DoS attacks by using the method mentioned above on IEEE 802.11 and 802.1x standards, including EAP-MD5, EAP-TLS and LEAP as well as the Temporal Key and Integrity Protocol (TKIP) with the Message Integrity Code

(MIC). These latter items are major issues within the ongoing IEEE 802.11i standard. Those experiments are based on Cisco’s current firmware and equipment, i.e. Cisco AP 1200 and 350. Different types of frames are modified, including authentication or association request frames, EAP frames, or incorrect keys to drag down the system’s performance. The results show that one laptop, acting as an adversary, has sufficient capability to launch a DoS attack against several APs because of the latter’s limited resources. For an AAA server, at least 10 or more adversaries are required to drag down the performance of the server. Although the 802.11i is considered to be more secure than the other standards, it is vulnerable to DoS attacks. In reality, the time between launching and obtaining a successful DoS attack on 802.11i is made shorter than other standards by taking advantage of the cost of computation. When a DoS attack is launched from one laptop, it takes, on average, 5 seconds to bring down an AP using TKIP with MIC, 8 to 10 seconds with EAP-TLS and LEAP, respectively, and about 15 seconds for WEP.

The use of ingress and egress filters is recommended in order to reduce the risk of DoS attacks, but it may fail to stop the DoS attacks that are launched using a spoofed source IP address from a legitimate user, or from a valid range of the source IP address. The drawback of an IP-based filter is the fact that an IP address is a fixed set and cannot change dynamically permitting the user to protect the IP address from being spoofed by an adversary. Our proposed IDF protocol is an ID-based filter, which can change dynamically on every frame and cannot be reused, thus preventing filter values from being spoofed. The faked frames will be blocked outside the filter in order to stop a junk message DoS attack. The remainder of this paper is organized as follows. Section 2 reviews the related works in defending against DoS or DDoS attacks. Section 3 proposes the concept for the IDF protocol. Section 4 explains in detail how the IDF protocol defends against DoS attacks. Section 5 describes DDoS user detection provided by an IDF and outlines a performance comparison. Section 6 contains the conclusions and suggestions for future work.

II. RELATED WORKS

To prevent a DoS attack, authentication is an important issue in helping the users and responders identify each other prior to accepting one another’s frames. There are several PKI signature authentication schemes that have been proposed to

¹ We do not distinguish between DoS and distributed denial of service (DDoS) attacks in this paper since a DDoS attack is a sophisticated attack method of DoS using many computers in a distributed manner.

² In this paper, the masculine or feminine term is used for a legitimate user or adversary, respectively.

³ The responder could be an access point, firewall, or an AAA or database server that stores user information.

This work was partially supported by a Nokia grant.

prevent DoS attacks [1][2][3]; however, they are vulnerable to DoS attacks on the first and third message. For example, on the first frame of W. Aiello, et al [1] and J. Leiwo's [2] schemes, the request message including the user's ID or nonce is sent in unprotected clear text to the responder. If the responder requires a user identity check, as is done with W. Aiello's scheme, an adversary can launch a DoS attack by spoofing the legitimate user's identity in order to pass the responder's identity check. If the responder need not check the user's identity using the first message, as is done in J. Leiwo's scheme, the adversary can randomly generate a nonce on the request message to the responder in order to launch a DoS attack. In these two schemes, once the responder has received the first request message, the responder must generate a nonce and then compute a hash function for each received faked request message. This process causes DoS attacks, since the adversary has forced the responder to keep busy processing spoofed request traffic and thus has limited ability remaining for servicing legitimate users.

In the third flow, they must either verify the received signature sent from a legitimate user, or not. This step requires more computation and is vulnerable to a DoS attack, because an adversary can send a plethora of faked signatures to the responder for verification. Furthermore, if an adversary launches DoS attacks by continuing to send a pair, consisting of the first and third messages, to the responder, the burden on the responder will increase until the system buffer overflows. Just imagine, that if millions of pairs of frames are sent from an adversary to a responder, how much computation is required by the responder to determine if the messages are forged. The IDF can assist those schemes in blocking the faked message and thus preventing a DoS attack.

The client puzzle is another DDoS defending scheme that requires a client to do more work than a server in order to be served. Originally, cryptographic puzzle was used for key agreement not access control [4]. The idea that a client must commit resources first and do a considerable amount of computation was proposed by Dwork and Naor [5], and used to address the problem of junk e-mail. Juels and Brainard [6] introduced a cryptographic client puzzle, which is sent to a client when a server appears to be under TCP SYN flood attacks. A client must solve its puzzle correctly in order to get service. Later, Aura and Nikander proposed the DoS-resistant authentication scheme [7], which combined the stateless authentication protocols [8][9] with a client puzzle to address a DoS attack. Wang and Reiter [10] proposed a puzzle auction to address DoS attacks. This scheme permits the client to determine the difficulty, i.e. bid, of the puzzle. A higher bidder pays more, in terms of complex computation, to obtain a higher priority for receiving services. Although the puzzle scheme reduces the risk of DoS attacks, it sacrifices the legitimate users who must accept mandatory expensive operations on their machines. It is neither fair for legitimate users nor efficient.

Photuris [11] uses a cookie (hash value) based on PKI-based algorithm for DoS resistance. The user must return the cookie generated by the responder for verification. Although Photuris uses a cookie to guard against flooding attacks sent with bogus IP Sources or UDP Ports, it has vulnerability for DoS attacks that was pointed out by [12].

III. PROTOCOL DESIGN

In IDF, each user has their unique filter generated from their master pre-shared secret key, the details of which are discussed in section 4.2. The master key is protected by two-factor feature keys. 1. The user's password is memorized by the user and never stored on the device or sent in traffic. 2. A nonce and timestamp are generated by the user's system and unknown to the user. Only legitimate users and responders can update their filters and determine each other's next filter value. Furthermore, the Advanced Encryption Standard (AES)-Counter mode (CTR) with Cipher Block Chaining (CBC) and Message Authentication Code (MAC), i.e. AES-CTR+CBC-MAC, or what is now simply called the AES-CCM mode [13][14], is used to guarantee the confidentiality and integrity of data during communication. The system performs computations only after the frames have passed the access control filter, $F(t)$, which is a time-dependent function that changes every frame, as shown in Fig. 1. If a frame sent from a user or responder does not match the filter value, it will be blocked by the filter before the system resources are consumed. Similar access control techniques are being used in packet filters, routers and firewalls and are well-known for their processing speed.

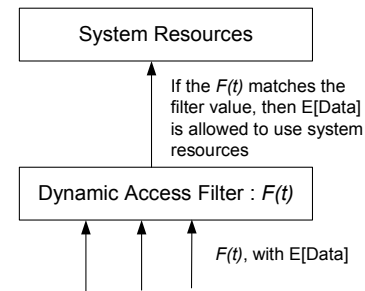


Figure 1. The concept of IDF: if the $F(t)$ matches the filter value, then $E[Data]$ is allowed to use system resources.

IV. THE IDF PROTOCOL

The IDF protocol consists of three stages. The first stage is the initial configuration stage. In this stage, the master secret key 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 every legitimate user will be generated and stored on a responder filter table before a responder starts to broadcast the beacon information. The second stage is the mutual authentication stage used by users and responders to identify one another. The third stage is the fast dynamic access control filter stage. Since the responder must process a large volume of traffic and the filter value is changed with every frame, the computational cost for the third stage should be less than other stages. In order to keep the notations simple and easy to understand, the discussions of the IDF protocol are based on one user with a responder.

A. Notations for IDF Protocol

- $R_{id} P_R$ A responder's identifier and password assigned by an administrator
- $T_R N_{Ri}$ Timestamp and initial nonce generated by responder's system when P_R is assigned

K_R	Master secret key of a responder generated by a keyed hash function during the initial setup stage
$U_{id} P_u$	User's identifier and password, typed in for each session and not stored in the device
$T_u N_{ui}$	Timestamp and initial nonce generated by a user's system when P_u is assigned
K_U	Master secret key of a user generated by a keyed hash function during the initial stage
N_i	A random number that is chosen by a user. This number remains the same during the user's login session. It will be updated automatically after a new K_U is generated, and used for the next login session
T_{KU}	Timestamp when K_U is generated
N_R	A nonce generated by a responder. It is encrypted using the AES-CCM mode when sent to challenge a user with beacon or broadcast frames, and updated after a configurable valid time interval
T_{NR}	Timestamp when a responder's nonce N_R is generated
$f_{R(t)}$	A 128-bit seed from a responder's truncated function used to update filter $F_{R(t)}$
$f_{U(t)}$	A 128-bit seed from a user's truncated function used to update filter $F_{U(t)}$
$F_{R(t)}$	The access filter is stored in a responder's filter table to filter the frames from users
$F_{U(t)}$	The access filter is stored in a user's filter table to filter the frames from a responder
$h_{(key)}$	A collision free keyed hash function. HMAC-SHA-512 is recommended in this paper
$MAC_{R(t)}$	The output of HMAC-SHA-512, which is used to generate the filter of a responder
$MAC_{U(t)}$	The output of HMAC-SHA-512, which is used to generate the filter of a user
S_{Ri}	A responder's initial seed during the initial setup stage
S_{Ui}	A user's initial seed during the initial setup stage
$S_{R(t)}$	A responder's seed used to update the session key $K_{RS(t)}$
$S_{U(t)}$	A user's seed used to update the session key $K_{US(t)}$
N_U	A nonce generated by a user. It is encrypted using the AES-CCM mode when sent to challenge a server during the mutual authentication stage
T_{NU}	Timestamp when a user's nonce N_U is generated
$K_{RS(t)}$	A responder's session key used to update the dynamic access control filter. It could also be an input key for the AES-CCM to encrypt data during communication
$K_{US(t)}$	A user's session key used to update the dynamic access control filter and could also be an input key for the AES-CCM to encrypt data during communication
$T_{RF(t)}$	A timestamp generated by a responder when updating its filter
$T_{UF(t)}$	A timestamp generated by a user when updating its filter

B. Initial Configuration Stage

Generate Master Keys and Store as Pre-Shared Secrets

K_U , K_R , and N_i are the pre-shared secrets of a user and a responder and are stored on both systems⁴. K_U and K_R are the master secret keys of a user and responder, respectively, and generated by the following equations:

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

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

N_{ui} and T_u are generated automatically by the user's system when a user types in the U_{id} and P_u during the initial configuration stage. Every time a user logs in, the system generates a different nonce N_{ui} and timestamp T_u to ensure a K_U is varying. N_{ui} is only known by the user's system and will be the input key for the HMAC function in (1). Like the user's system, a responder's initial nonce N_{Ri} and timestamp T_R are only known by the responder's system. They are generated automatically by a system when an administrator assigns the P_R and R_{id} to a responder. N_{Ri} is an input key for the HMAC function and used to generate the responder's master key K_R , as shown in (2). Both the user's password P_u and the responder's password P_R are dictionary attack resistant because they are protected by the nonce and timestamp that are unknown to everyone. The user or responder only knows half (U_{id} , P_u or R_{id} , P_R) of the master secret while their systems know the other half (N_{ui} , T_u or N_{Ri} , T_R). This mechanism is called two-factor feature and widely used to protect the real user's password from dictionary attacks.

N_i is a random number chosen by a user during the initial configuration stage and stored in both systems as a pre-shared secret. While this number remains constant during a single login session, it is different for other various login sessions. The new N_i can be updated locally, as illustrated in (3), while the user's K_U is updated and then stored for the next new login session (see the key refresh session for details).

$$N_i \leftarrow h_{(K_U \oplus N_i)}[U_{id} \parallel K_U \parallel T_{KU} \parallel N_i] \quad (3)$$

Initial Filter Setup in a Responder as a Filter Table

After a user and responder have saved the pre-shared secrets, e.g. N_i , K_U , K_R , in their systems, the responder must generate a filter table for all of its legitimate users. To create the initial filter table, a responder must generate a random nonce N_R , and timestamp T_{NR} with the two calculation steps outlined in (4) and (5). Then, the responder begins sending out the beacon message periodically. The user's access control filter will be created once he receives the beacon message and derives the N_R , and T_{NR} using the pre-shared key K_R .

$$h_{(N_i \oplus K_R)}[N_i \parallel N_R \parallel T_{NR}] \equiv S_{Ri} \quad (4)$$

The input key for the HMAC function is formed by K_R XOR N_i , where the latter is chosen by a particular user. Different users have different N_i and thus different S_{Ri} . S_{Ri} is a 512-bit responder's initial seed and used as an input key in (5). It will be changed as soon as a new responder's nonce N_R and timestamp T_{NR} are generated for updating the beacon message or after different N_i are generated during the users various logins. Because the input key, S_{Ri} , is never repeated, the output of the HMAC function in (5) is made very secure by avoiding dictionary attacks [15][16][17].

$$h_{(S_{Ri})}^{[N_i \oplus N_R]}[U_{id} \parallel K_U \parallel N_i \parallel N_R \parallel T_{NR}] \equiv MAC_{R(0)} \quad (5)$$

$[N_i \oplus N_R]$ is the number of rounds used to conduct the hash function. From a performance standpoint, it is recommended

⁴ The pre-shared secrets K_U , K_R , and N_i could be exchanged via a secure channel or a trusted third party to secure delivery in the initial stage.

that the number be truncated to 10 bits, i.e. 0 to 1023 rounds. $MAC_{R(t)}$ is a 512-bit keyed hash value and is truncated in three parts, which include the 128 most significant bits $f_{R(t)}$ and the 128 least significant bits $S_{R(t)}$, as shown in Fig. 2. t is the index of a time-dependent function, and $t=0$ is the first seed used to generate the first access control filter and session key.

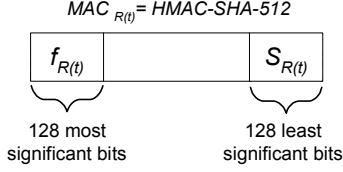


Figure 2. The responder's truncated function

$f_{R(0)}$ is the seed used to generate a 160-bit responder's filter $F_{R(0)}$ ⁵ using (6). $F_{R(0)}$ is the initial filter value stored in a responder's filter table, as shown in Table 1. The frame sent from the user must match this value in order that the remaining portion of this frame is admitted into the responder's system. $S_{R(0)}$ is a seed used to update the session key $K_{RS(t)}$ and generate the new filter.

$$h_{(S_{Ri})}[K_U \parallel N_i \parallel MAC_{R(0)} \parallel T_{NR} \parallel f_{R(0)}] \equiv F_{R(0)} \quad (6)$$

TABLE I. RESPONDER'S DYNAMIC ACCESS FILTER TABLE

Flag	U _{id}	Access Filter	Hash Value	Session Key	...	Packet counter	Idle Time
	A	$F_{R(0)}$	$MAC_{R(0)}$	$S_{R(0)}$...		

Each row specifies the information for a particular user. The packet counter will accumulate all the received frames passed by the control filter for the user, and begin recounting on the user's next login. This feature aids the responder in detecting 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 increment while the user remains connected. The responder should ask the user to re-authenticate if the idle time is too long.

User Login Session

When a user wants to establish a connection, he must type in the user ID and password in order to login. The user's system has stored N_{ui} and T_u from the initial setup, so it can generate a K_U based upon the password and ID that the user entered and computed using (1). If this K_U matches the pre-stored K_U , then the system identifies this login user as a legitimate one. If this K_U does not match the pre-stored K_U , after a reasonable number of trials, the user's system should be locked.

Key Refresh Session

The K_U is different for every login session. During each login session, when a pre-stored K_U is matched, the system will generate a new nonce N_{ui} and timestamp T_u . Given the new nonce and timestamp, the system performs a calculation using (1) to obtain the new K_U . The new K_U and N_i will be encrypted

using the AES-CCM and sent to the responder after the mutual authentication stage. The new K_U and N_i have to be stored as pre-shared secrets on both sides for the next login session.

C. Mutual Authentication Stage

Following the initial configuration stage, the responder will broadcast the beacon message periodically, as shown in Fig. 3. The NR and TNR are encrypted using the AES-CCM mode, and the master key of the responder, K_R , is used as an initial input key for this mode. This encrypted data will be sent out and bound with the beacon frame. The user's initial filter will be generated after he successfully decrypts the encrypted data. In the mutual authentication stage, the first three frames are used by the user and responder to identify each other.

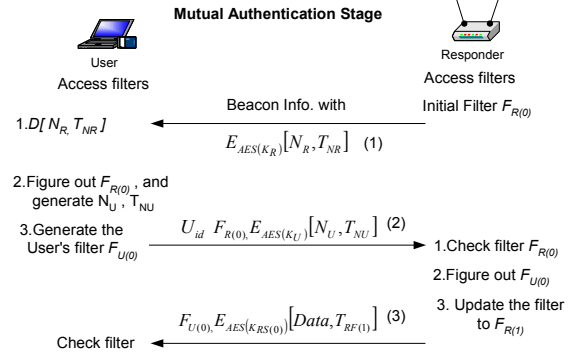


Figure 3. Mutual authentication stage overview

Frame 1 $R \rightarrow U$: Beacon with $E_{AES(K_R)}[N_R, T_{NR}]$

A Responder sends out a Beacon message periodically. N_R and T_{NR} will be changed in accordance with the beacon update time interval. The N_R and T_{NR} are used by the responder to authenticate a user. When a legitimate user receives the beacon message, he will execute three steps: First, use the pre-share K_R to decrypt the beacon message in order to obtain N_R and T_{NR} . Second, perform the same calculation, as his responder, using (4), (5) and (6) to obtain S_{Ri} , $MAC_{R(0)}$, $f_{R(0)}$, $S_{R(0)}$ and $F_{R(0)}$. $F_{R(0)}$ is a 160-bit filter value and returned to the responder for checking. Third, generate a user's nonce N_U , and a timestamp T_{NU} to authenticate the responder and, in addition, create the user's initial dynamic access filter using (7) and (8).

$$h_{(K_U \oplus N_i)}[N_i \parallel N_R \parallel N_U \parallel T_{NU}] \equiv S_{Ui} \quad (7)$$

$$h_{(S_{Ui})}^{[N_i \oplus N_U]}[U_{id} \parallel K_U \parallel N_i \parallel N_R \parallel N_U \parallel T_{NU}] \equiv MAC_{U(0)} \quad (8)$$

S_{Ui} is a 512-bit user's initial seed that is only used in the initial configuration stage. This seed will be changed for every login session as a result of the varying properties of K_U , N_i , N_U , and T_{NU} . The primary aim of (7) is to generate different seeds for every login session in order 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. $MAC_{U(t)}$ is a 512-bit keyed hash value truncated in three parts, including the 128 most significant bits $f_{U(t)}$ and the 128 least significant bits $S_{U(t)}$, as shown in Fig. 4.

⁵ It is recommended that the 160-bit HMAC-SHA-1 is used to generate the IDF in this paper, but different applications may vary.

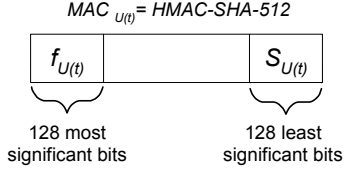


Figure 4. The user's truncated function

$f_{U(0)}$ is the initial seed used to generate a 160-bit user's access control filter $F_{U(0)}$, as illustrated in (9). $F_{U(0)}$ is the user's first access control filter value stored on the user's system for checking the frame sent from the responder. $S_{U(0)}$ is a user's first seed used to update the session key $K_{US(t)}$ and generate the new access filter.

$$h_{(S_{U(0)})}[K_U \parallel N_U \parallel MAC_{U(0)} \parallel T_{NU} \parallel f_{U(0)}] \equiv F_{U(0)} \quad (9)$$

Then, the user will create an access filter table, as shown in Table 2, to store the filter value and other key information.

TABLE II. USER'S DYNAMIC ACCESS FILTER TABLE

Flag	R _{id}	Access Filter	Hash Value	Session Key	Packet counter	Idle Time
	K	$F_{U(0)}$	$MAC_{U(0)}$	$S_{U(0)}$		

$$\mathbf{Frame 2} \quad U \rightarrow R: U_{id}, F_{R(0)}, E_{AES(K_U)}[N_U, T_{NU}]$$

The N_U and T_{NU} are encrypted by K_U and sent to the responder with the filter value $F_{R(0)}$. When the responder receives frame 2, it will perform three steps: First, use U_{id} to see if the received $F_{R(0)}$ matches the one on its filter table. If there is a match, the responder decides this frame was sent from a legitimate user, and passes the encrypted data to the system for decryption using this user's K_U to obtain N_U and T_{NU} . Second, use (7), (8) and (9) to derive this user's $S_{U(t)}$, $MAC_{U(0)}$, $f_{U(0)}$, $S_{U(0)}$ and $F_{U(0)}$. Third, generate a timestamp $T_{RF(t)}$ and update the filter value on the responder's access filter table for checking the next frame sent from this user using (10). $f_{R(t)}$ and $S_{R(t)}$ will be truncated from $MAC_{R(t)}$, as illustrated in Fig. 2. $f_{R(t)}$ is a seed used to update the responder's access control filter from $F_{R(0)}$ to $F_{R(t)}$, using (12). $S_{R(t)}$ is a seed used to update the responder's session key $K_{RS(t)}$ for the next frame.

$$h_{(K_{RS(0)})}[K_R \parallel K_U \parallel N_i \parallel N_R \parallel MAC_{R(0)} \parallel T_{RF(1)}] \equiv MAC_{R(1)} \quad (10)$$

$K_{RS(t)}$ is the responder's session key, as defined in (11). When $t=0$, $K_{RS(0)}$ is the first session key used by the responder to generate $MAC_{R(t)}$ in order to update the access filter from $F_{R(0)}$ to $F_{R(t)}$ using (10) and (12). This session key is also used as an input key by the AES-CCM to encrypt the data. This data, with filter value $F_{U(0)}$, will be sent to the user once the responder has updated its filter to $F_{R(t)}$.

$$h[S_{R(t)} \parallel S_{R(0)} \parallel T_{NR} \parallel K_U \parallel K_R] \equiv K_{RS(0)} \quad (11)$$

$$h_{(K_{RS(0)})}[K_U \parallel N_i \parallel MAC_{R(1)} \parallel T_{RF(1)} \parallel f_{R(1)}] \equiv F_{R(1)} \quad (12)$$

$$\mathbf{Frame 3} \quad R \rightarrow U: F_{U(0)}, E_{AES(K_{RS(0)})}[Data, T_{RF(1)}]$$

The responder sends the encrypted timestamp $T_{RF(t)}$ and the data with the filter value $F_{U(0)}$ to the user. When the user receives frame 3, the following three steps are performed: First, $F_{U(0)}$ is checked to see if it matches the filter value. If there is a match, it is believed that this frame was sent from his responder. The encrypted data is then passed into the system for decryption. Since the user knows the same secrets $S_{R(t)}$, $S_{R(0)}$, T_{NR} , K_U , and K_R as the responder, he can perform the same calculation, i.e. (11). This calculation will derive the same session key $K_{RS(0)}$, used by the responder, to decrypt the data and obtain the timestamp $T_{RF(t)}$. Second, $K_{RS(0)}$ and $T_{RF(t)}$ are used to perform the same calculations, i.e. (10) and (12), as the responder to derive the responder's $MAC_{R(t)}$, $f_{R(t)}$, $S_{R(t)}$ and $F_{R(t)}$. $F_{R(t)}$ is a filter value that will be returned to his responder for checking. $S_{R(t)}$ is a seed used by the user to compute the responder's next session key $K_{RS(t)}$. Third, generate a timestamp $T_{UF(t)}$ and then update the access filter from $F_{U(0)}$ to $F_{U(t)}$ for checking the next frame from the responder.

$$h[S_{U(t)} \parallel S_{U(0)} \parallel N_i \parallel N_R \parallel T_{NU}] \equiv K_{US(0)} \quad (13)$$

$K_{US(t)}$ is the user's session key, as defined in (13). When $t=0$, $K_{US(0)}$ is the first session key employed by the user to generate $MAC_{U(t)}$ in order to update his filter from $F_{U(0)}$ to $F_{U(t)}$ using (14) and (15). This session key is also used as an input key for the AES-CCM to encrypt the data when replying to the responder's frame.

$$h_{(K_{US(0)})}[K_U \parallel N_i \parallel N_R \parallel MAC_{U(0)} \parallel T_{UF(1)}] \equiv MAC_{U(1)} \quad (14)$$

$$h_{(K_{US(0)})}[K_U \parallel MAC_{U(1)} \parallel T_{UF(1)} \parallel f_{U(1)}] \equiv F_{U(1)} \quad (15)$$

$f_{U(t)}$ and $S_{U(t)}$ will be truncated from $MAC_{U(t)}$ as illustrated in Fig. 4. $f_{U(t)}$ is a seed used to generate a new user's access control filter $F_{U(t)}$, using (15). $S_{U(t)}$ is used to update the user's session key to $K_{US(t)}$. Once the user identifies the responder on the third frame, the mutual authentication stage for the IDF protocol is completed. Next, the fast dynamic access filter stage is initiated.

D. Fast Dynamic Access Control Filter Stage

This stage is called the fast dynamic access control filter stage because the user and responder need not generate the nonce in order to identify each other. In addition, the amount of computation is less than that required for the other stages, since fewer rounds of the hash function are performed. A time-dependent scheme is used to refresh the session key and update the filter on both sides. In this stage the filters for both the responder and user keep changing in every frame. If a frame is lost in this stage, a re-transmission by TCP or LLC (Logical Link Control) still maintains connectivity. If a connectionless link is being used, then the mutual authentication must be performed again in order to maintain high security for the re-established link

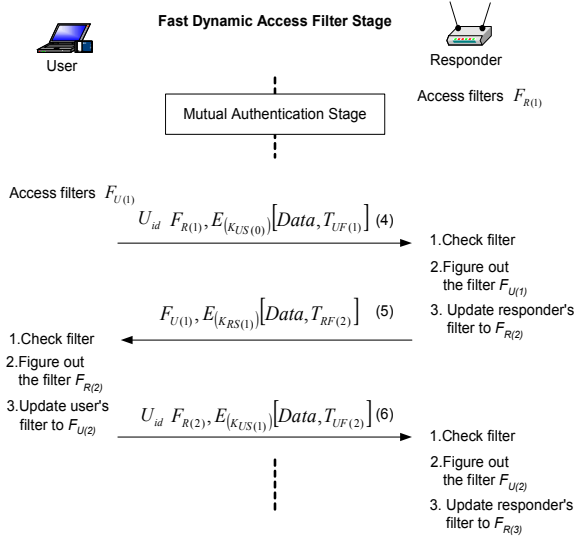


Figure 5. The fast dynamic access filter stage

In this stage, three steps must be performed on both sides. First, the access filter is checked on each side when the frames are received. Second, filter values are determined for the other side. Third, filter values are updated for the next frame. The following frames, i.e. 4 and 5, are the first and second frames for this Stage, as shown in Fig. 5.

Frame 4: $U \rightarrow R: U_{id}, F_{R(1)}, E_{(K_{US(0)})}[Data, T_{UF(1)}]$

In the first frame of this Stage, the user sends to the responder $F_{R(1)}$ with timestamp $T_{UF(1)}$, which is protected by the AES-CCM. When the responder receives Frame 4, it performs three steps: First, U_{id} is used to see if the received $F_{R(1)}$ matches the one in its filter table. If there is a match, the responder identifies this frame as one that was sent from the legitimate user. The encrypted data is then passed to the system for decryption using this user's session key $K_{US(0)}$ to obtain the data and timestamp $T_{UF(1)}$. Since the responder knows the previous user's secrets $S_{U_i}, S_{U(0)}, N_i, N_R$ and T_{NU} , it can perform the same calculation, i.e. (13), as the user did to derive the session key $K_{US(0)}$. Second, $K_{US(0)}$ and $T_{UF(1)}$ are used to perform the same calculation, i.e. (14) and (15), as the user to derive the user's $MAC_{U(1)}, f_{U(1)}, S_{U(1)}$ and $F_{U(1)}$. $F_{U(1)}$ will be returned to the user for checking. $S_{U(1)}$ is a seed used by the responder to compute the user's next session key $K_{US(1)}$. Third, the responder generates a timestamp $T_{RF(1)}$ and updates its filter from $F_{R(1)}$ to $F_{R(2)}$ for checking the next frame sent from the user, using (16), (17) and (18). $K_{RS(1)}$ is the new responder's session key used to generate $MAC_{R(2)}$ in (17). This key is also used as a new input key by AES-CCM to encrypt the data when the next frame is sent to the user.

$$h[S_{R(0)} \parallel S_{R(1)} \parallel K_R \parallel K_U \parallel T_{RF(1)}] \equiv K_{RS(1)} \quad (16)$$

$$h_{(K_{RS(1)})}[K_R \parallel K_U \parallel N_i \parallel N_R \parallel MAC_{R(1)} \parallel T_{RF(2)}] \equiv MAC_{R(2)} \quad (17)$$

$f_{R(2)}$ and $S_{R(2)}$ will be truncated from $MAC_{R(2)}$ using the same method shown in Fig. 2. $f_{R(2)}$ is a new seed used to update the responder's filter from $F_{R(1)}$ to $F_{R(2)}$, as illustrated in (18). $F_{R(2)}$

is the new responder's access filter, is stored in the responder's access filter table and used for filtering the next received frame.

$$h_{(K_{RS(1)})}[K_U \parallel N_i \parallel MAC_{R(2)} \parallel T_{RF(2)} \parallel f_{R(2)}] \equiv F_{R(2)} \quad (18)$$

Frame 5: $R \rightarrow U: F_{U(1)}, E_{(K_{RS(1)})}[Data, T_{RF(2)}]$

When the user receives Frame 5, he will repeat the same three steps outlined for Frame 3. Both the user and responder repeat the same three steps on each side for the remaining frames until the user logs out. Note that if the responder or user does not receive the frame from each other during the connection established, the filter value should remain the same to allow the frame retransmission. The DoS attack could only happen when a legitimate user sends a filter value to the responder, which never receives that value due to communication error, so that the responder neither updates its filter value nor responds to the user. An adversary could spoof this filter value and sends it to the responder to launch a DoS attack. When the responder receives this spoofed filter value that matches the one on its filter table, the session key will be used to decrypt the encryption part of this frame. This action will fail due to the adversary does not have the correct session key to encrypt the frame. Then the responder will send an alert message to ask that user to re-login by using the K_U and N_i that were generated after the most recent login, and consequently the spoofed filter value is changed to a new value. The DoS attack could be successfully launched only on the first frame and will be stopped. All the repeated frames will be blocked since the filter value is changed. The legitimate user still can log in by simply typing in his new password to the system.

V. DDoS DETECTION AND PERFORMANCE ANALYSIS

A. DDoS User Detection and Prevention

The IDF protocol can help the responder detect an abnormal action from a user by checking the packet counter, shown in Table 1. The packet counter contains the number of successful packets that have been accepted by the responder. If the user's packet number increases by an abnormal amount in a short period of time or is much higher than all the others, the responder can check to see if a destination site is being attacked. If an attack is underway, the responder can send an alarm with a de-authentication message to the user and break the connection, thus stopping the DDoS attack.

B. Performance Comparison with Signature Schemes

The critical step in the IDF protocol for preventing DoS attacks is the processing in frame 2 where the responder verifies the user's authentication credentials. When a responder receives the frame, a comparison with the filter table is conducted using an Exclusive OR (XOR). The XOR operation produces a logic 1 output only if its inputs from two filter values are different. If the inputs are the same, the output is a logic 0. The verification time for the IDF protocol to do a XOR computation of 160 bits is about 2 nanoseconds for a Pentium 4 2.2GHz machine with 512 MB DDR SDRAM running at 266 MHz under Windows XP SP 1. A Comparison of the Frame 2 verification times of the IDF protocol with other authentication protocols, based on RSA or DSA signature schemes using the same PC, is shown in Table 3. Since every scheme must fetch

the authentication information from the database using the user's ID, the fetch time is not included in the table. If a numeric ID is used, then the fetch time is a memory read cycle when the authentication information is available in main memory. The source code for Crypto++ 5.1 is compiled by Microsoft Visual C++ 6.0. The public exponent is 65537 for RSA and 160 bits for DSA. Clearly, the IDF protocol is more capable of preventing a DoS attack than these other protocols.

TABLE III. COMPUTATION TIME FOR VERIFICATION

Schemes	Verification Time / frame
IDF 160-Bit Filter Verification	2 Nanoseconds
RSA 1024 Signature Verification	0.28 Milliseconds
RSA 2048 Signature Verification	0.96 Milliseconds
DSA 1024 Signature Verification	3.74 Milliseconds

VI. CONCLUSION AND FUTURE WORKS

In this paper, we have introduced the IDF in order to prevent a DoS problem. The IDF protocol possesses many unique properties. It is an ID-based, not IP-based, filter and unique for every user. The filter value varies with every frame on both sides to prevent filter spoofing. The pre-shared secret K_U is not the user's real password and also not a permanent value stored on the device. It is different for every login session to protect user's privacy. In addition, it employs a very important underlying principle that is used to protect the responder from a DoS attack, and that principle is simply this: *Do not create any state or do any expensive computation before you can ensure that the received frame is legitimate.* Furthermore, the IDF protocol has two other features that aid in preventing DoS attacks. First, if an adversary has never been a legitimate user, she needs to discover the responder's master secret key K_R in order to decrypt the encryption beacon frame to obtain the nonce N_R and timestamp T_{NR} so as to be in a position to launch a DoS attack. Second, if she has been a legitimate user at one time, she may have known the responder's master key K_R for a short period before the responder updated it. In spite of the fact that the adversary knows the responder's K_R , she is still unable to fake the filter value to impersonate a legitimate user and launch a DoS attack, because she does not know the other legitimate user's secrets K_U , N_i , N_U and T_{UN} needed to calculate their current filter value. Each legitimate user has their unique filter value which is known only by their responder. Any faked frame will be blocked by the access control filter.

The IDF protocol has provides the capability for a user to move to a different computer/device. The user can store pre-shared secrets (N_i , K_U , K_R) and the session information N_{ui} and T_u in a media, such as a smart card. Then, the user can put the smart card on other devices and repeat the procedures outlined in section 4.2 for user login and a key refresh session. The IDF is also capable of combining with other protocols and standards, in order to make them more secure. Because all of

the sensitive information can be afforded extra protection, e.g. using the AES-CCM encryption, it is not sent out in clear text and thus resilient to an eavesdrop attack. When any other scheme is combined with the IDF, all the frames must pass a filter check before system resources are committed. As a result, this action provides a good defense against a DoS attack.

VII. REFERENCES

- [1] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis, and et al, "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
- [2] 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.
- [3] Shouichi Hirose and Kanta Matsuura. "Enhancing the resistance of a secure key agreement protocol to a denial-of-service attack", In Proceedings of the 1999 Symposium on Cryptography and Information Security (SCIS '99), pages 899-904, Kobe, Japan, January 1999.
- [4] R. C.Merkle, "Secure communications over insecure channels", Communications of the ACM, 21:294- 299, April 1978.
- [5] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail", In E. Brickell, editor, Proceedings of Advances in Cryptology - Proc. CRYPTO '92, volume 1323 of LNCS, pp. 139-147, Santa Barbara, CA USA , August 1992. Springer-Verlag.
- [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), pp. 151-165, February 1999.
- [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.
- [8] T. Aura and P. Nikander, "Stateless connections", In Proc. Of International Confereneec on Information and Communications Security (ICICS '97), Lecture Notes in Computer Science volume 1334, pp. 87-97. Springer, November 1997.
- [9] P. Janson, G. Tsudik, and M. Yung, "Scalability and flexibility in authentication services: The KryptoKnight approach", In IEEE INFOCOM'97, Tokyo, April 1997.
- [10] X.F. Wang and M.K. Reiter, "Defending Against Denial-of-Service Attacks with Puzzle Auctions", pp. 78-92, In IEEE Symposium on Security and Privacy, May 2003.
- [11] P. Karn and W. A. Simpson, "Photuris: Session-key management protocol", Request for Comments 2522, IETF Network Working Group, March 1999.
- [12] R. Perlman, " Understanding IKEv2: Tutorial, and rationale for decisions", draft-ietf-ipsec-ikev2-tutorial-01.txt, February 2003.
- [13] S. Whiting, R. Housley, and N. Ferguson, "AES Encryption and Authentication using CTR Mode and CBC-MAC", IEEE 802.11-02/001r2, May 28, 2002
- [14] O. Letanche and D. Stanley, "Proposed TG1 2.2 Clause 8 AES-CTR CBC-MAC (CCM) text", IEEE 802.11-02/144r4, July 11, 2002
- [15] M. Bellare, R. Canetti, and H. Krawczyk, "The HMAC Construction", CryptoBytes, Spring 1996
- [16] 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 Science, pp. 1-15, Springer-Verlag, 1996.
- [17] M. Bellare, and R. Canetti, "HMAC: keyed-hashing for message authentication", Request for Comments 2104, Internet Engineering Task Force, February 1997.