COPYRIGHT © 2011 IJCIT, ISSN 2078-5828 (PRINT), ISSN 2218-5224 (ONLINE), VOLUME 01, ISSUE 02, MANUSCRIPT CODE: 110137

An Efficient, Simple and Scalable Automatic Generation of VPN Security Policies

Kazi Chandrima Rahman

Abstract—IPSec (Suite of protocols for IP layer Security) policies are widely deployed in firewalls or security gateways to restrict access or protect information , which is one of major mechanisms for Virtual Private Network (VPN). It is critical for policies to be specified and configured correctly because the security treatment (e.g. denies, allow or encrypt etc.) of all inbound or outbound traffic will be determined by the security policies. IPSec policies are generally manually configured to individual security gateway, which could be very inefficient and error-prone. An erroneous policy could lead to communication blockade or serious security breach. An automatic generation of policies is therefore demanded to systematically manage and verify various IPSec policies in order to ensure an end-to-end security service. Automation process requires that it must be efficient and scalable, as well as simple. Previous automatic generation of IPsec policies does not meet these requirements. This paper contributes to the development of an IPSec policy management system, which is simple, efficient and scalable according to requirement update and development of an algorithm which is complete, because which finds solution as long as there is one according to ideal policy management system. This system provides the facilities of the users to specify their requirements at a high level without concerning specific low level parameters, and then correct low level policies will be automatically generated efficiently.

Index Terms—VPN, Secutiy Policies, IPSec policies, Intra-domain.

1 INTRODUCTION

IT is a commonly accepted fact that Internet technologies have changed the way that companies disseminate information to their customers, partners, employees, and suppliers. Initially, companies were conservative with the information they published on the Internet – product information, product availability and other less business critical items. More recently, using the Internet as a means of providing more cost effective access to business critical information such as order status, inventory levels, or even financial information has gained wider acceptance through Virtual Private Networks or VPNs.

A Virtual Private Network is a business solution that provides secure, private connections to network applications using a public or "unsecured" medium such as the Internet. With a VPN deployed across the Internet, virtual private connections can be established from almost anywhere in the world [27]. VPN can be implemented using various tunneling and security protocols. Amongst them IPsec is the most prominent, rich and flexible protocol which can be used to create a wide range of protection schemes.

IPsec is a layer three, suite of protocols**,** providing security services and policy –enabled networking service and so IPsec can be deployed correctly only if policies are correctly specified and configured [21]. While manipulating traffic, with separate inbound and outbound security policy database (SPD) will consult for inbound and outbound traffic. This can result in complex management tasks that become especially difficult as networks scale up. Because, although manually configuring the IPSec policy database may be fine for a small network, it is inefficient and error-prone for large distributed networking systems.

Because of the growing number of secure Internet applications, IPSec policy deployment will be more and more complex in the near future. An erroneous policy could lead to communication blockade or serious security breach. In addition, even if policies are specified correctly in each domain, the diversified regional security policy enforcement can create significant problems for end-to-end communication because of interaction or conflicts among policies in different domains. Therefore, policy management systems is clearly demanded to automatically and systematically configure and manage various IPSec policies for VPN [11].

What makes correct policy specification very difficult is two folds: First, tunnel operations cause complications in selector choices. Second, Lack of high-level view of overall objectives although each individual policy may appear to satisfy its individual goal. Therefore, it is important to specify policy at a higher level that maps to low level policies efficiently and unambiguously.

General IPSec specification can be so specific that it is hard to understand what it is really wanted. It is realized in IPSec security policy working group of IETF that a higher-level policy language is needed with well-defined and clear semantics [14]. In [11], a higher level policy, which is called security requirements, whose functions can be clearly understood and rigorously proven. The low-level policies are correct only when they satisfy all requirements. In a policy management system, people can specify their security requirements in a higher level to a central policy database and correct low-level policies will be automatically generated and distributed to appropriate nodes to enforce. The auto-

• ***Kazi Chandrima Rahman*** *is serving as an Assistant Professor of the Department of Computer Science and Engineering, University of Asia Pacific (www.uap-bd.edu), Dhanmondi, Dhaka-1209, Bangladesh. E-mail: chandrima@uap-bd.edu.*

mation can not only save administrators tremendous labor but also guarantee the correctness of low-level policies generated by the system.

Everyday newer systems are created. In this paper, the previous automatic VPN security policy generation system is considered. But that is not efficient one. For a large distributed system with complex hierarchy, where there is many firewalls or gateways, and large number of requirements, policies have to generate efficiently. On the other hand, new requirements need new policies to generate. So, the system must be scalable as well as requires few policy updates. Based on the above requirements, a new efficient, simple and scalable automatic generation of VPN security policies for IPsec is proposed in this paper.

From the above discussion, it is very much clear that an automatic generation of VPN security policy management system for IPSec protocol is inherently necessary. This implements IPsec policies according to the requirements and identifies whether the requirements are satisfiable or whether there is any conflict.

The rest of the paper is organized in the following ways: Section 2 describes IPsec policy management problem and related work. In section 3, the disadvantages of previous system and criteria of an ideal system is specified in chapter 5 and based on these, a new system is proposed. The whole system is depicted with figures, algorithms and examples. In section 4, both systems are compared. The simulation technique, analyzation and performance comparison of both systems are discussed in this chapter with table and graph. Section 5, the final section of this paper carries the concluding remarks, and discussions. Some of the directions of the work are suggested that can be perpetrated in the future.

2 IPSEC POLICY MANAGEMENTT PROBLEMS AND RELATED WORKS

2.1 Policy Problems for Intra-Domain Environment

First, a typical scenario of intra-domain communications for a large distributed organization is illustrated below: -
In fig 1, a large organization is composed of multiple distributed sites, which can communicate with each other through VPN tunnels. Each site can have firewalls with specific security policies to protect their property. Furthermore, some sub-domains, like financial department etc., can have their own firewall with their specific policies to protect their sensitive data.

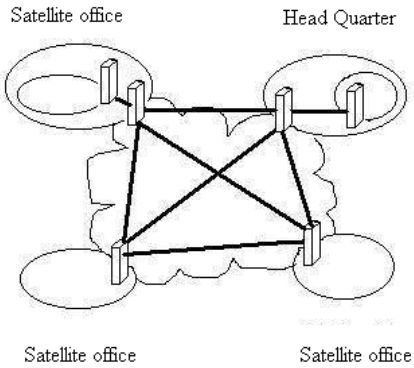

Fig. 1. Scenario for intra-domain site-to-site communications.

Although various policy issues [4-7] attracted a lot of attention, two important problems for policies are:-

1) There is no way to ensure correctness of security policies, and
2) There are no ways to systematically specify and distribute correct policies instead of manually configuring each node.

2.2 IPSec Policy Management: Problem Analysis

An erroneous policy could lead to communication blockade or serious security breach. Some problems might be caused by careless human error while some others might arise from interactions that can not be easily detected even with careful and experienced administrators.

2.2.1 Examples of Policy Problems

Illustration of three scenarios of policy problems are given below:-

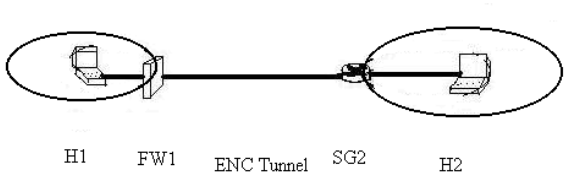

Fig. 2. Conflict between examination and encryption

Scenario 1: Conflicts Between Examination and Encryption

In the scenario 1, an encryption tunnel is built between H1 and H2 to protect their sensitive communication. In IPSec policy, policies can be specified to deny encrypted packet by denying all packets with ESP protocol like (src=* dst=* prot=esp $\rightarrow$ deny). Therefore, if Firewall FW1 has policy to

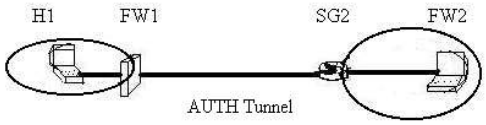

Fig. 5. Selector confusion

deny all encrypted traffic because of the need to examine the content of the traffic, all packets will be dropped in the middle of the transmission.

Scenario 2: Selector Confusion

In the scenario 2, H1 has policy to authenticate all packets from H1 to H2 through a tunnel from H1 to SG2. FW1 is a

firewall to perform access control or selective security enforcement. There may have the following three example cases:

- In the first case, if there is an assumption that FW1 has simple access control policy as (src=H1, dst=H2→ allow) and (others→ deny). However, because of the authentication tunnel that changes destination to be SG2 in outer header, the FW1 will mistakenly drop all traffic from H1 to H2 that should be allowed. Similarly, it can also mistakenly allow some traffic to pass due to confusion with selectors.
- In the second case, the FW1 also has security enforcement policy as (src=H1, dst=H2 → ipsec_action: Encrypt from FW1 to SG2) and (others → allow). For the same reason, the firewall might pass the authenticated traffic without encryption, which may cause serious information leakage during the transmission.
- In the third case, the FW1 may want to block encrypted packets in order to examine content for purpose of intrusion detection. The policy is simply set to be (prot=ESP→deny). However, this may not successfully block all encrypted traffic. If the packets are authenticated after encryption, then the protocol field in outer header becomes AH. Similarly, the packets can be encapsulated with more outer headers. Therefore, it may be difficult to specify correct policies because of selector changes.

The above three cases showed outer header changes for tunnel implementation could cause confusion in policy setup such that the results might deviate from what are originally intended to be. One solution for this is to change selectors accordingly in order to make policies right. However, this not only requires administrators are aware about other policies and specific attributes, but also make intention of policy more and more unclear as it grows larger. In other words, tunnels can make it hard to specify policies correctly.

Scenario 3: Tunnel Overlaps

In the scenario 3, there are financial department 1.1 in site 1 and financial department 2.1 in site 2(as in Fig 4). It is assumed each department has its security gateways and has authority to make security policies to protect their property. In this example, department 1.1 decides that all traffic from

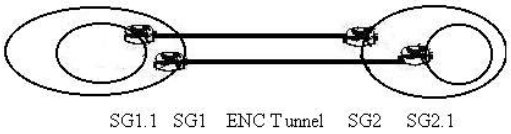

Fig. 4. Tunnel overlaps

1.1 to site 2 must be encrypted through a tunnel from SG1.1 to SG2. At the same time, the administrator for site 1 decides that the communication between this site and the financial department 2.1 is important and all traffic from site1 to department 2.1 must be encrypted through a tunnel from SG1 to SG2.1. Therefore, the traffic from SG1.1 to SG2.1 shall be governed by two policies and should be protected from SG1.1 all the way to SG2.1. However, the policies might not work as desired. First, if the administrator does not adjust the selector for the upper tunnel in specifying policies in SG1, then the traffic might skip lower tunnel completely such that it lacks the required protection from SG2 to SG2.1. On the other hand, if it is indeed adjusted to include the tunneled traffic, then the traffic will go through two tunnels. With this configuration, traffic is encapsulated with a new header by SG1.1 and then encapsulated with another new header by SG1 to send to SG2.1. When SG2.1 decapsulates and finds out the destination is SG2,SG2.1 will send traffic back to SG2. Finally SG2 will decapsulate and send traffic to its real destination. Although it is originally intended to encrypt traffic from SG2 to SG2.1, the traffic is eventually sent in clear from SG2 to SG2.1 because of tunnel interaction.

2.3 Previous Work of IPsec Policy Management System in an Intra-Domain environment

2.3.1 Security Requirements Analysis

One important task of IPSec policy management is to represent security requirements in a high level efficiently and unambiguously [14].

In security requirement, people can clearly specify their intention of security treatment on specific traffic without worrying how selectors might change in low-level policies. Therefore, the attributes of flow identities specified in requirements are all those of original flows.

Access Control Requirement (ACR)

One fundamental function of security is to conduct access control that is to restrict access only to trusted traffic. A simple way to specify an ACR is:

flow id → deny | allow

Flow is typically identified by 5-tuple (source address, destination address, source port, destination port, protocol).

Security Coverage Requirement (SCR)

Another important function is to apply security functions to prevent traffic from being compromised during transmission across certain area, which requires the security protection of the traffic to cover links and nodes within the area. A simple way to specify a SCR to protect traffic from "*from"* to "*to"* by a security function with certain strength could be:

flow *id.→ protect(sec_function, strength, from, to, trusted_nodes)*

Content Access Requirement (CAR)

Some nodes may need to access content of certain traffic, for example, a firewall with an intrusion detection system (IDS) may need to examine content to determine the characteristic of the traffic.

CAR is allowed to be explicitly specified to express the need for specific nodes to access content of certain traffic. CAR can be expressed as denying or allowing certain security function to prevent the nodes from accessing certain traffic as follows:

flow *id.→ deny_sec | allow_sec (sec_function, access_nodes)*

Security Association Requirement (SAR)

Security Associations (SA) [10] need to be formed to perform encryption/authentication function. There might be needs to specify some nodes to desire or not desire to set up SA with some other nodes because of public key availability, capability match or mismatch etc. A simple way to specify a SAR could be:

flow *id.→ deny_SA | allow_SA (SA_peer1, SA_peer2)*

2.3.2 Determining Policies to Satisfy Security Requirements

The *Requirement Satisfiability Problem* is, generally speaking, "Given a set of security requirements, whether it is satisfiable or not" .The input of the problem is a set of security requirements and relevant topology. The output of the problem is either a satisfying policy set or a "failure" message indicating the requirement set is unsatisfiable [11].

2.3.3 Security requirements examples

For example, there are a set of three requirements: ***Three_Reqs*** = {Req1 (src = 1.* dst = 2.* → Weak ENC 1.* 2.*), Req2 (src = 1.1.*, dst = 2.* → Strong ENC 1.1.* 2.*), Req3 (src = 1.*, dst = 2.1.* → Strong AUTH 1.* 2.1.*), }. In Figure 4-5, the white bar illustrates protection area for Req1. The black bar illustrates protection area for Req2 and the gray bar illustrates the protection area for Req3. The question is what policies to satisfy the three requirements are.

In the next section an existing automatic generation of VPN security policy, based on IPsec is described, which is called *bundle approach*.

2.3.4 Existing (Bundle approach) Policy Automation Process

The solution is to separate entire traffic into several disjoint traffic flow sets, called bundles, each of which is subject to a unique set of security requirements. In Three_Reqs example, there are four bundles that are governed by different set of requirements: (src = 1.1, dst = 2.1) subject to {Req1, Req2, Req3}, (src = 1.1, dst = 2 – 2.1) subject to {Req1, Req3}, (src = 1 – 1.1, dst = 2.1) subject to {Req1, Req2} and (src = 1 – 1.1, dst = 2 – 2.1) subject to {Req1}. In bundle approach, policies will be generated to satisfy all requirements for each bundle. For one particular bundle, the condition part contains bundle selectors and action part contains appropriate security actions to meet all requirements for the bundle.

Using bundle approach, the problem is resolved in two steps. First, from given requirements, entire traffic flows will be grouped into a number of disjoint bundles and find out the subset of the requirements that are applied to each bundle. Second, for each bundle, given a set of requirements for the bundle, policy_action algorithm to generate action part of policies for the bundle is used and traffic flows are grouped into bundles and calculate what applicable requirements for each bundle are.

In bundle approach, the following Policy Automation Process is performed--

- Obtaining security coverage requirements for each link and node
- Content Access requirement conflict check
- Finding zero security association link
- ***Initialization*** of the graph
- Checking Content Access Requirement violation for security association
- Distrusted node processing
- Finding eligible primary security associations
- Policy action generation algorithm

Finally, Dijkstra's single source shortest path algorithm [15] will be run to determine whether there is an eligible primary SA path from source to destination. If so, it can be found a solution with minimum number of SAs. Otherwise, the requirements are unsatisfiable.

3 A NEW APPROACH: AN EFFICIENT SIMPLE SCALABLE AUTONOMOUS SYSTEM

Although the bundle approach is complete and correct, it is still less ideal in its efficiency and scalability.
The two main problems of Bundle approach are described below: -

1. Inefficient policy generation
2. Inefficient requirement update

In requirement draft [8] in IPSP working group of IETF, IPSec policy has been described and this work has been followed. The algorithm to find solution is desired to be correct, complete and efficient [11].

3.1 The Proposal of an Efficient System: Directly Building Chained Tunnels For Each SCR

In this section, a new system, which is called ***Direct Approach***, is proposed to generate policies to correspond to each SCR without separating traffic into bundles, which is more efficient and requires less update. The reason is that

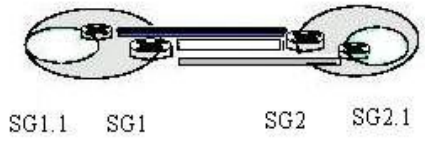

Fig. 6. Security Requirements Example

most tunnels can directly work together to provide necessary protection in real life. Only when there are overlaps [11] between interacted tunnels (i.e. selectors have non-nil intersection) can tunnels together cause requirement violation. In this section, algorithm will be developed to generate non-interacted-overlapping tunnel policies for each SCR.

3.1.1 Algorithms To Generate Non-overlapping Tunnel Policies For One SCR

The focus of the direct approach is then how to build chained tunnels for each SCR that do not overlap with any existing tunnels and satisfy the SCR and relevant CARs and SARs. So, the development of the algorithms is to generate non-overlapping tunnel policies for one SCR (as in Fig 5).

Building a Primary Graph

First a primary graph will be built to find eligible SAs. Before building tunnels, relevant CARs and SARs are needed to check out. CARs and SARs are just scanned one by one and one CAR or SAR is chosen if its selector has non-nil intersection with the selector of the SCR. With CAR and SAR, primary graph can be built.

In the following algorithm, the details of functions Link Node Coverage and Initialize Graph etc. from line 2 to 6 are omitted, since these are described in previous work-bindle approach. The array sec_node [N] and sec_link [N-1] are needed to store the result. There is no zero SA edges in the SA graph. Only primary graph is needed since there is only one SCR. The linear path will start from the SCR.fromB to SCR.toB

Algorithm Build_Direct_Primary_Graph (SCR)
1. Find All Relevant SARs and CARs
2. Find Link Node Coverage
3. Initialize the Primary Graph
4. Perform the CAR Preprocessing
5. Check whether there is CAR Conflict
6. Preprocess the Distrusted Nodes
End Of Algorithm

Complexity of the Algorithm-- Build_Direct_Primary_Graph

In the algorithm, the first line takes at most O (J+K) time, in which J is the number of CARs and K is the number of SARs. The second line takes time O (N). The third line takes time O ($K + N^2$). The line 4, 5, 6 takes O ($J \times N^2$). Therefore, the running time for the algorithm is at most O ($K + J \times N^2$).

Finding Existing Connecting Nodes From the Group of Tunnels

A set of tunnel groups are given, each of which has a set of connecting nodes. The existing connecting nodes will be organized and ordered, based on which Must_Be_Connecting, Common Connecting and Have_To_Be_Connecting nodes can be determined. Existing connecting nodes are sorted based on their relative position from left to right. Each connecting node has a set of attributes (group_id, pre, next) in which pre, next is the previous and next node that the connecting node is associated with in the group identified by group_id. One node might be connecting node in several groups. Connecting nodes can be represented using link list.

Algorithm Organize_Existing_Connecting_Nodes (Tnlgroups)
1. Sorted_connect_queue = {} // an ordered priority queue
2. for each $group_i$ in Tnlgroups
3. for each $Connect\_node_{ij}$ in $group_i$
4. insert $Connect\_node_{ij}$ (pre = $connect\_node_{ij-1}$, next=$connect\_node_{ij+1}$, group_id = i) to *Sorted_connect_queue*
End Of Algorithm

Complexity of the Algorithm Organize_Existing_Connecting_Nodes

The priority queue insertion takes at most O (N) time. The algorithm takes O ($L \times N^2$) time.

Given a set of tunnel groups' configuration, the algorithm outputs a set of ordered existing connecting nodes, each of which is associated with a set of group identities, a set of previous nodes and a set of next nodes. In later context, the ordered existing connecting nodes are represented as set *Sorted_connect_queue* ={$Sorted\_connect_0$, $Sorted\_connect_1$, …, $Sorted\_connect_K$}. For any $Sorted\_connect_i$, the set of group identities, previous nodes, next nodes are represented as $Sorted\_connect_i.group\_ids$, $Sorted\_connect_i.pres$ and $Sorted\_connect_i.nexts$ respectively.

Finding Must_Be_Connecting Nodes

Now, first Must_Be_Connecting nodes will be looked for in order to get non-overlapping solution.

Criteria of Must_Be_Connecting Nodes Between Two Nodes

A set *Sorted_connect_queue* = {$Sorted\_connect_0$, $Sorted\_connect_1$, …, $Sorted\_connect_K$} are given. Now a group of chained tunnels are to be built from *Start_node* to *End_node*, and the new group of tunnels must not overlap with any of existing tunnels. Any $Sorted\_Connect_i$ node is a *Must_Be_Connecting* node iff it resides between Start_node and End_node, and at least one of its previous nodes resides before Start_node or at least one of its next nodes resides after End_node, i.e. $Start\_node < Sorted\_Connect_i < End\_node \;\&\&\; (\exists\, j\ Sorted\_Connect_i.pre_j < Start\_node \;||\; \exists k\ Sorted\_Connect_i.next_k > End\_node)$.

Without the node as connecting node, it can be either built one non-stop tunnel or be built chained tunnels that do not connect at the node. First, if one non-stop tunnel is built from Start_node to
End_node, obviously the tunnel will be overlapping with either the tunnel ($Sorted\_Connect_i.prej$, $Sorted\_Connect_i$) or the tunnel ($Sorted\_Connect_i$, $Sorted\_Connecti.next_k$).

Second, chained tunnels can be built that do not connect at the node. Suppose the connecting nodes adjacent to $Sorted\_Connect_i$ are $Connect\_node_l$ and $Connect\_node_{l+1}$. Then the new tunnel ($Connect\_node_l$, $Connect\_node_{l+1}$) is

overlapping with either the tunnel (*Sorted_Connect$_i$.prej*, *Sorted_Connect$_i$*) or the tunnel (*Sorted_Connect$_i$*, *Sorted_Connect$_i$.next$_k$*).

So, the algorithm to recursively finds all Must_Be_Connecting nodes as follows.

Algorithm Finding_Must_Be_Connecting_Nodes (Sorted_connect_queue, Start_node, End_node)

1. Static must_connect_queue = {Start_node, End_node}
2. for each Start_node < Sorted_connect$_i$ < End_node in Sorted_connect_queue
   1. for each pre$_j$ in Sorted_connect$_i$.pres and each next$_k$ in Sorted_connect$_i$.nexts
   2. if (Sorted_connect$_i$.pre$_j$ < Start_node || Sorted_connect$_i$.next$_k$ > End_node)
   3. if (sec_node [Sorted_connect$_i$] == 0) / / it is allowed to be connecting node
   4. insert Sorted_connect$_i$ into must_connect_queue
   5. else return("Fail! The node is not trusted to be connecting node.");
   6. for every new pair (node$_i$, node$_{i+1}$) in must_connect_queue
   7. Finding_Must_Be_Connecting_Nodes (Sorted_connect_queue, node$_i$, node$_{i+1}$)

End Of Algorithm

Complexity of the Algorithm Finding_Must_Be_Connecting_Nodes

Between Start_node and End_node, there are at most N connecting nodes to check, and the time to check them is at most O (L x N). If there are P new pairs, the total time will be O (N) = L x N + O (N_1) +O (N_2) + … + O (N_P) in which $N_1 + N_2 + … + N_P <= N$, and O (1) = 1. Therefore, the total running time is at most O (L x N^2).

Property of Must_Be_Connecting Nodes

If one non-stop tunnel can be built between every pair of adjacent Must_Be_Connecting nodes, then the resultant tunnels are not overlapping with any of existing tunnels.

Therefore, if there is one non-stop tunnel solution between every pair of Must_Be_Connecting nodes, the resultant tunnels are correct. If non-stop tunnel solution is not possible, chained nonoverlapping tunnel solution will be found between each Must_Be_Connecting node pair. The overall solution will be correct if satisfying nonoverlapping solution can be found between every Must_Be_Connecting node pair. To come up with correct solution, the first thing to check is *common connecting nodes*.

Finding Common Connecting Nodes

Basically, common connecting nodes are those connecting nodes common to all existing tunnel groups that have at least one connecting node in between. It does not have to consider the tunnel groups that have no connecting node in between because there will be no overlapping with them no matter how connecting nodes is chosen in between. Formally, common connecting node is defined as follows.

Criteria of Common Connecting Nodes between Two Nodes

A set of tunnel groups are given, the ith of which has a set of connecting nodes *{Connect_nodei_0 =fromB_i, Connect_node$_{i1}$, …, Connect_node$_{iNi}$ = toB_i}*. Now chained tunnels are to be built between Start_node and End_node. Among existing tunnel groups, the tunnel groups that have at least one connecting node between Start_node and End_node are called *Connect_groups*, i.e. *group_id$_i$ ∈Connect_groups iff ∃ □j Start_node < Connect_node$_{ij}$ < End_node*. Between Start_node and End_node, any node Node$_i$ is a common connecting node iff 1) It is a connecting node common to all tunnel groups that are in *Connect_groups* and cross over the node, i.e. *∀group_id$_j$ (group_id$_j$∈ Connect_groups && Connect_node$_{j0}$ <= Node$_i$ <= Connect_node$_{jN j}$)*

∃ k Node$_i$ = Connect_node$_{jk}$. 2) None of tunnel groups in *Connect_groups* cross over it, i.e. *!∃j j ∈Connect_groups && Connect_node$_{j0}$ <= Node$_i$ <= Connect_node$_{jNj}$.*

In the above criteria, first the connecting nodes for Connect_groups are only considered because any chaining configuration will not cause overlap with tunnel groups that are not in Connect_groups. Second, all tunnel groups crossing over certain node are only considered because connecting at the node would not cause any overlapping with the tunnel groups that do not cross over the nodes. Based on the above criteria, the algorithm to find common connecting nodes as follows.

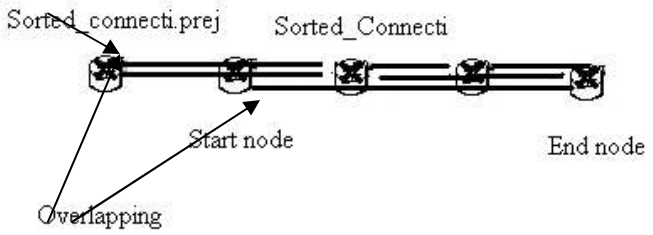

Fig. 7. Example of overlapping if *Sorted_Connect$_i$* is not a connecting node.

Algorithm Finding_Common_Connecting_Nodes (Start_node, End_node)

1. common_queue = {}
2. connect_groups = union all group_ids of connecting nodes between Start_node and End_node
3. for every node$_i$ between start_node and end_node
4. cross_groups = {}
5. for every group_id$_j$ in connect_groups
6. if connect_node$_{j0}$ < node$_i$ < connect_node$_{jNj}$
7. add the group_id into cross_groups
8. if (cross_groups == NULL)

9. insert $node_i$ into common_queue
10. else
11. if $node_i$ is in connect_queue && $node_i$.group_ids == cross_groups&& connect_groups
12. insert $node_i$ into common_queue
End Of Algorithm

Complexity of the Algorithm Finding_Common_Connecting_Nodes

In the algorithm, line 2 takes O (N x L^2). In the loop from line 3 to line 12, the inner loop from line 5 to line 7 takes O (L), line 11-12 takes O (N + L), and time for the loop takes O (N x (N+L)). Therefore, the running time for entire algorithm is at most O (N x L^2 + N^2).

Property of Common Connecting Nodes

If there are to build chained tunnels between Start_node and End_node, and one non-stop tunnel is not possible. If there are K > 0 common connecting nodes Common_Set ={$Common_1$, $Common_2$, …,$Common_K$} between them as defined in criteria of common connecting node, and if no connecting node in new group is the same as any of node in Common_Set, there will be overlap.

Criteria of All Non-Overlapping Solutions with Common Connecting Nodes

Chained tunnels have to build between Start_node and End_node, i.e. one tunnel solution is not possible. If there are K > 0 common connecting nodes Common_Set ={$Common_0$ = Start_node, $Common_1$, $Common_2$, …, $Common_{K+1}$ = End_node} between them as defined in the above criteria, then any of the nonoverlapping solutions can be represented as {$Common\_connect_0$ = Start_node, $Connect_{01}$, …$Connect_{0N0}$,$Common\_connect_1$, $Connect_{11}$, …, $Connect_{1N1}$, $Common\_connect_2$, …,$Common\_connect_{M+1}$ = End_node}, in which $Common\_connect_i$ is one of common connecting nodes and $Connect_{ij}$ is not any common connecting node. The number of common nodes as connecting nodes 0 < M <=K. If $Common\_connect_i$ and $Common\_connect_{i+1}$ is not adjacent, then the N_i=0, i.e. the only solution is one non-stop tunnel to connect two common nodes. If $Common\_connect_i$ and $Common\_connect_{i+1}$ is adjacent, then the N_i > = 0, and the chained tunnels between $Common\_connect_i$ and $Common\_connect_{i+1}$ must be nonoverlapping with any of existing tunnels. In other words, the above represents all non-overlapping solutions.

Based on the above criteria, given a set of common nodes, the solution can be found as follows.

Algorithm Common_Nodes_Solution (Common_nodes, start_num, Path, num_tnls)
1. static solutions = (min = 100, path = NULL)
2. eligible_queue = {}
3. for every $common_i$ (start_num + 1, K) in Common_nodes
4. if i == start_num + 1
5. if ($length_i$ = Nonoverlapping_ Between_Nodes($common_0$, $common_1$) != 0)
6. insert ($common_i$, $length_i$) into eligible_queue
7. else
8. if there is one non-stop tunnel from $common_0$ to $common_i$
9. insert ($common_i$, $length_i$=1) into eligible_queue
10. for every ($common_i$, $length_i$) in eligible_queue
11. Path = Path + $node_i$
12. total = num_tnls + $length_i$
13. if $node_i$ == End_node
14. if total < solution.min
15. replace solution with (path, total)
16. else
17. common_nodes_solution (Common_nodes, i, Path, total)
18. return (solutions)
End of Algorithm

Initially, the start_num is 0, Path is with Start_node and num_tnls is 0. The above algorithm recursively check all possible solutions and eventually find the solution with minimum number of tunnels.

Complexity of the Algorithm Common_Nodes_Solution

In the algorithm, the loop from line 3 to line 9 takes at most O (N^2). The algorithm will at most check through all edges, which takes at most O (N^2).

To find non-overlapping solution between two nodes, if there are common connecting nodes, the above algorithm can find all possible solutions. If there are no common connecting nodes, it will need to find Have_To_Be_Connecting nodes as follows.

Finding Have_To_Be_Connecting Nodes

The following so-called Borders have to be connecting nodes in order not to overlap. First border will be found first.

Criteria of First Border

Between two nodes (Start_node, End_node), assume there is no non-stop tunnel solution, and there is no common connecting node. A set *Sorted_connect_queue* = *{$Sorted\_connect_0$, $Sorted\_connect_1$, …, $Sorted\_connect_K$}*, are given in which the set of group identities of any node i is stored in $Sorted\_connect_i.group\_ids$. The first border is the connecting node $Sorted\_connect_i$ whose group_ids supersets all previous connecting nodes' group_ids and none of any latter connecting node's group_ids supersets all previous ones, i.e. $\forall j\ (j<i)\ Sorted\_connect_j.group\_ids \subseteq Sorted\_connect_i.group\_id$ sand $!\exists k(k>i)\ \forall l\ (l<k)\ Sorted\_connect_l.group\_ids \subset Sorted\_connect_k.group\_ids$.

First Have_To_Be Connecting Node

The first border *Sorted_connect*$_i$ has to be a connecting node in order not to overlap with any existing tunnels.

Criteria of Subsequent Borders

After a border$_i$, the first connecting node whose group_ids is not a subset of border$_i$'s group_ids is a border border$_j$. Before border$_j$, any node whose group_ids is not subset of the border$_j$'s group_ids is also a border.

Subsequent Have_To_Be_Connecting Nodes

All subsequent borders have to be connecting nodes in order not to overlap with any of existing tunnels.

Algorithm Finding_Have_To_Be_Connecting_Nodes (Start_node, End_node)

1. have_to_connect_queue = {}
2. first_border = connect$_1$
3. for each Start_node < Sorted_connect$_i$ < End_node in Sorted_connect_queue
4. if Sorted_connect$_i$.group_ids=union(all group_ids of Sorted_connect$_1$ to Sorted_connect$_i$)
5. first_border = connect$_i$
6. add first_border in have_to_connect_queue
7. pre_border = first_border
8. while (pre_border < End_node)
9. for each node pre_border < Sorted_connect$_i$ < End_node
10. if Sorted_connect$_i$ .group_ids $\neq$ pre_border.group_ids
11. next_border = Sorted_connect$_i$
12. add next_border in have_to_connect_queue
13. for each node pre_border < Sorted_connect$_i$ < next_border
14. if Sorted_connect$_i$ .group_ids $\neq$ next_border.group_ids
15. add Sorted_connect$_i$ in have_to_connect_queue
16. pre_border = next_border

End of Algorithm

The algorithm takes at most O (L x N^2) time.

Overall Direct Non-overlapping Algorithm

In summary, to find nonoverlapping solution between two nodes, it will first check if there are common connecting nodes. If there are, then common nodes solutions will found. If there are not, then Have_To_Be_Connecting nodes will be checked. The algorithm to find nonoverlapping solution between two nodes can be written as follows.

Algorithm Nonoverlapping_Between_Nodes (start_node, end_node)

1. if there is direct tunnel between start_node and end_node
2. return(1)
3. Finding_Common_Connecting_Nodes (Tnlgroups, Sorted_connect_queue)
4. if (common_queue! = NULL)
5. path = Common_Nodes_Solution (common_nodes, 0, K, path=Start_node, num_tnls = 0)
6. else
7. Finding Have_To_Be_Connecting_Nodes
8. for each pair of Have_To_Be_connecting nodes
9. path = path + Nonoverlapping_Between_Nodes (node$_i$, node$_{i+1}$)
10. return (path)

End of Algorithm

Therefore, the overall algorithm can be written as follows.

Algorithm Direct_Nonoverlapping_Tunnel_Group (SCR, interacted_tunnels)

1. Build_Primary_Graph(SCR)
2. Organize_Existing_Connecting_nodes( )
3. Finding_Must_Be_Connecting_Nodes( )
4. num_tnls = 0
5. For each pair (node$_i$, node$_{i+1}$) in must_be_connect_queue
6. num_tnls + = Nonoverlapping_Between _Nodes (node$_i$, node$_{i+1}$)

End Of Algorithm

Running time of Direct_Nonoverlapping_Tunnel_Group

The algorithm takes at most O (L^2 x N^2), in which L is the number of existing requirements, N is the number of nodes on the path.

4 SIMULATION AND COMPARISON

The simulation is performed for proposed VPN policy management system and existing system (Bundle approach) to automate the VPN policy management in an Intra-domain environment. Then the performance of the two systems is compared.The simulation program is coded using widely used C programming language. Simulation is behind the practicality, so the result is always comparative.

4.1 SIMULATION

The simulation is organized in the following ways. First, the simulation environment that includes network topology is described. Then the parameters used in the simulation are described.The technique of the simulation for both existing and proposed system is depicted. Then the performance evaluation of the both systems is given.

4.1.1 Network Topology for Intra-Domain Environment

For the simulation, domains in are hierarchically organized, as illustrated n Fig 8.

For the simulation experiment, it is assumed that each domain has four level of hierarchy, in which each sub-domain can have as many as 10 sub-domains. Security gateways reside at borders of sub-domains or domains. For simplicity, the security gateways of a particular sub-domain are treated as the same, and are represented as the domain address. Therefore, the linear path of each communication can be automatically determined as the border routers along the transit. For instance, for communication from

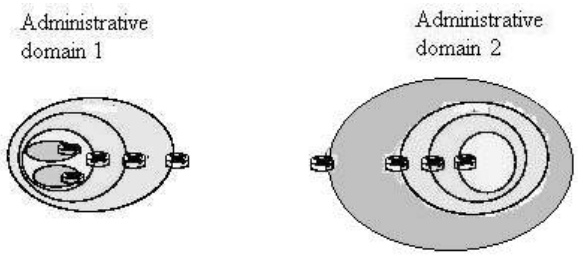

Fig. 8. Four level hierarchical domain with number of nodes at most on the path = 8.

1.1.*.* to 2.2.*.*, the enroute nodes are border routers of 1.1.*.*, 1.*.*.*, 2.*.*.* and 2.2.*.*. With four-level hierarchical domains (as in Fig 8), the number of nodes on the path is at most 4 * 2 = 8.

TABLE 1
TIME COST COMPARISON OF EXISTING AND PROPOSED SYSTEM

Number of Requirements	Existing system (Bundle approach) Time in μ seconds ($\times 10^6$)	Proposed system (Direct approach) Time in μ seconds ($\times 10^6$)
100	25000	12500
200	90000	45000
300	170000	150000
400	230000	100000
500	400000	180000
600	500000	250000
700	650000	240000
800	720000	242000
900	1000000	300000
1000	1250000	350000

4.1.2 Parameters taken for simulation program

Security Requirements

For the Security Coverage requirements: inputs are taken for each SCR as (Security function, strength, from_ node, to_node, trusted nodes). For the Content Access Requirements: inputs are taken for each CAR as (Security function, access_nodes). For the Security Association Requirements: inputs are taken for each SAR as (SA_node1, SA_node2, security function) that represents security association from node1 to node2 cannot be applied with specified security function.

Time

Times are calculated to process the requirement files one by one. The time it takes to generate all policies for different number of requirements. It is noted that the time recorded

The output is a policy file that contains automatically generated policies for both systems.

Simulation Technique for the Proposed Automatic Generation of IPsec Policies

For the proposed system, simulation is done in the following ways: -

1. Take the input for the existing group of tunnels with their connecting nodes.
2. Take inputs for requirements.
3. Generate policies for each SCR with their relevant CAR and SAR
4. Take the output as number of tunnels to cover for each security requirement and time to generate policies in microsecond for the number of requirements taken as input.

4.1.3 Performance Evaluation Process

In the experiment, 50, 100, 150, until 1000 requirements are randomly generated and use bundle/direct approach to process the requirement files one by one. The time it takes to generate all policies for different number of requirements is calculated. For the different number of requirements taken as input the number of generated policies is also calculated. Then the result is compared for both systems according to the taken readings.

Performance Comparison

The achieved results from the simulation program for the two policy management systems are given in tabular and graphical form in this subsection.

Time Cost Comparison

The time it takes to generate all policies for different number of requirements for both systems is calculated. The performance testing results are shown below in the table 1

The graphical representation of the policy number comparison from the table 1 is given in Fig 9.

It has been analyzed that the generation time could be at worst exponential to the number of requirements. Only when all selectors of the requirements intersect with each other can performance be at the worst case, the chance of

which is very small if not zero. At best, if there is no overlapping selector at all, the performance should be linear to the number of requirements. In the experiment, the selectors of each requirement are randomly generated and the occurrence of selector overlapping is also randomized. Two selectors are overlapping only when one source address is subset of the other source address but its destination address is superset of the other destination address or vice versa, the probability of which is small.

From the result, it can be clearly seen that both approaches' performances are close to linear and bundle approach takes longer generation time than direct approach does.

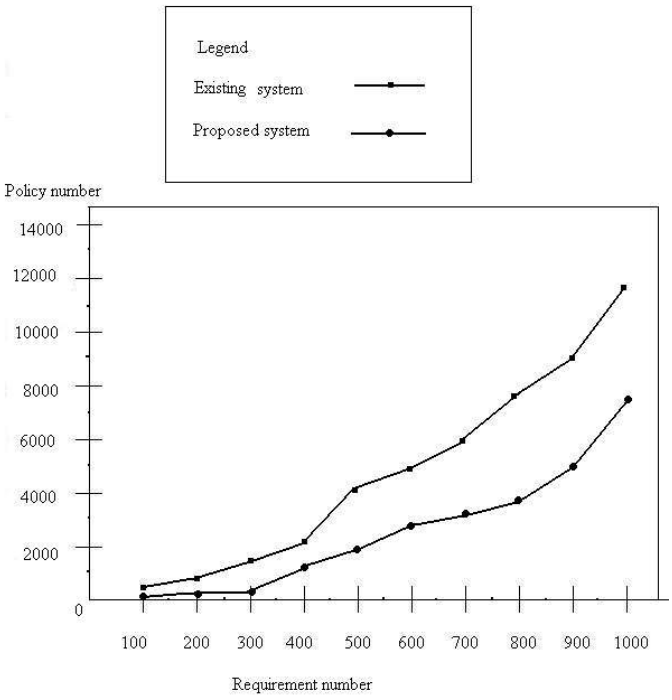

Fig. 9. Graphical representation of policy number comparison from table

Policy Number Comparison

Policy numbers generated using two approaches are tested for different number of requirements. The results are shown below in the table 2.

The graphical representation of the policy number comparison from the above table is given below in Fig 10.

The above Figure shows policy numbers generated using two approaches. It can be seen that from simulation experiment results that there may be many more policy numbers than requirement numbers. There are two reasons. First, multiple tunnels may correspond to one requirement. Some SARs and CARs as well as SCRs are randomly generated. To satisfy all requirements, chained tunnels may have to build. From the above table and graph it can also be seen that the policy number is especially large for bundle approach. The experimental results revealed drawback of policy inefficiency in bundle approach. On the other hand the number of generated policies for Direct Approach is small compared to Bundle approach.

TABLE 2
POLICY NUMBER COMPARISON OF EXISTING AND PROPOSED SYSTEM

Number of Requirements	Existing system (Bundle approach) Number of generated policies	Proposed system (Direct approach) Number of generated policies
100	200	90
200	900	500
300	1600	400
400	2300	1200
500	4100	2000
600	5000	2400
700	6000	3000
800	7600	3800
900	9200	4500
1000	11600	5500

4.2 Theoretical Comparative Study

For bundle approach, given R requirements including M SCRs, J CARs and K SARs, and a linear path with N nodes, the policy actions can be generated in O(R x N2 + N3), in which N is the number of nodes on the path. On the other hand, the algorithm of direct approach takes at most O(L2 x N2), in which L is the number of existing requirements, N is the number of nodes on the path. So, direct approach requires less time to generate policies.

Bundle approach is less scalable according to requirement update. In bundle approach, a new requirement may trigger series of policy change due to requirement list change for contained bundles Direct approach is more efficient in regard to requirement update. In direct approach, new tunnels for new requirement will be selecting all applicable traffic, and thus the new requirement will be automatically applied to all applicable traffic without need to change existing policies. To make sure the new tunnel applies to all applicable traffic, the only additional work to do is to adjust selectors to include the encapsulated traffic.

However, direct approach sometimes results in more "bumps" on the transit path of the packets. Although total

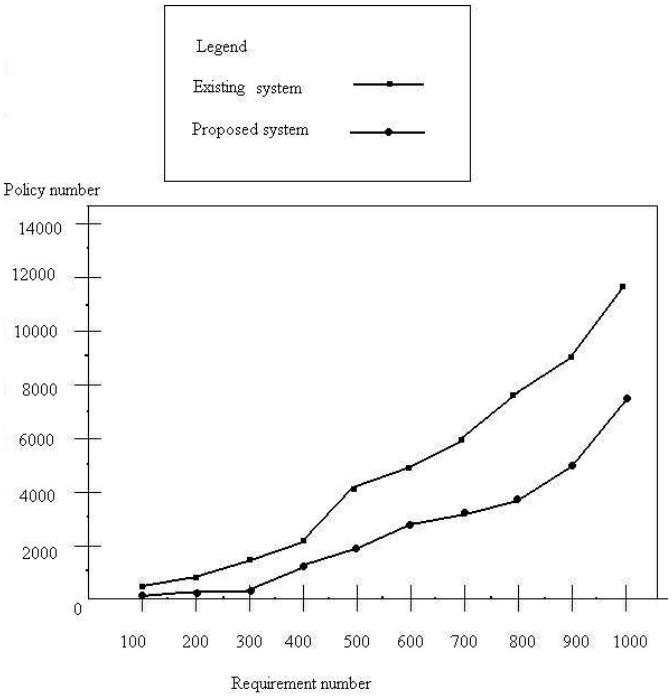

Fig. 10 Graphical representation of policy number comparison from table 2.

number of policies in direct approach is smaller since there might be much more bundles than SCRs, the number of policies for a particular packet might be bigger. To get non-overlapping solution, direct approach more often decomposes a transit path into multiple tunnels than it is with bundle approach. The packets will go through all of the tunnels with multiple encapsulation /encryption and decapsulation /decryption operations, which slow down the packet transmission. The number of tunnels may not affect the SA establishment performance at the beginning as many SAs can be established in parallel. Therefore, although the direct approach is advantageous in generation performance and policy management if there is solution with direct approach, it may be less advantageous when high transmission performance is required for some traffic.

According to the features of the ideal policy management stated in [11], the two systems can be compared as follows:

The policies generated using Bundle approach can satisfy all given requirements. So, bundle approach is correct. The bundle approach is complete, i.e. the algorithm can find one solution as long as there is one. The main point of bundle approach is to generate policies for different bundles separately, which can find correct solutions completely.

The Direct Approach is correct, i.e. the resulted tunnels satisfy SCR and relevant SARs, CARs, and do not overlap with any existing tunnels. The algorithm is complete, in the sense that the algorithm can find a solution as long as there is one and finds all possible solution. Again, it can be said that Direct Approach is incomplete. Direct approach is not always able to find a solution when there is one. Because, in Direct Approach, traffic is not separated. Interacted tunnels should not overlap with each other in order work together to provide security coverage, which imposes restrictions in the way to build tunnels. However, as proved, the Direct Approach only works conditionally and is not able to find solution all the time.

5 CONCLUSION

In this paper, algorithms are developed to automatically generate correct low level policies to meet all requirements. Therefore, people can just specify the desired requirements for protection then correct low level policies will be systematically specified and delivered to appropriate devices to enforce, which will greatly improve policy management. In the new and proposed technique, non-overlapping policies are built for each security coverage requirement respectively, and then the resultant policies can satisfy all requirements. The policies generated using direct approach will be less than that using bundle approach, which ultimately makes policy management easier.

In direct approach, new tunnels for new requirement will be selecting all applicable traffic, and thus the new requirement will be automatically applied to all applicable traffic without need to change existing policies. So, this novel system is scalable according to requirement update.
This approach is correct, because it can satisfy all requirements.

Future works

This research develops a system to automatically and systematically generate policies, which is simple, efficient and scalable. It should be possible to divert this research to many fields.

In the future, the research can be extended in several aspects:

- The research can be extended for distributed policy management in which distributed policy servers can communicate and make joint decision on correct policies
- The requirement conflict resolution techniques will demand further research.
- More levels of security policy may be specified until the whole hierarchy is clearly established.
- By combining the two approaches, it can be achieved both efficiency and completeness.
- In this research, policies are generated for Intra-domain environment; the research can be extended for Inter-domain environment.

REFERENCES

[1] Ruixi Yuan, W. Timothy Strayer, *Virtual Private Networks : Technologies and Solutions* Addition-wesley professional computing series, April, 2001.

[2] Andrew S. Tanenbaum, *Computer Networks*, Third Edition, Prentice-Hall of India,1998.

[3] Sharad Sangi, Managing Director, Netcore, *Virtual Private Data Networks*.

[4] M. Condell, C. Lynn, J. Zao, "*Security Policy Specification Language*",<draft-ietf-ipsp-spsl-00.txt>*, Internet Draft, March 2000.*

[5] J. Jason, "*IPsec Configuration Policy Model*", Internet Draft,<draft-ietf-ipsp-config-policy-model-00.txt>, March 2000.

[6] R. Pereira, P. Bhattacharya, "*IPSec Policy Data Model",* Internet Draft, <draft-ietf-ipsec-policy-model_00.txt>, Feb. 1998.

[7] J. D. Moffett and M. S. Sloman, *Policy Hierarchies for Distributed Systems management*, IEEE Journal on Selected Areas in Communication, vol. 11, pp. 1404-1414, 1993.

[8] M. Blaze, A. Keromytis, M. Richardson, L. Sanchez, "*IPSP Requirements*", <draft-ietf-ipsp-requirements_00.txt>, Internet Draft, July, 2000

[9] Darren Bolding, Network Security, Filters and Firewalls, ACM crossroads, Students magazine, 2.1 September 1995.

[10] S. Kent, R. Atkinson, "*Security Architecture for the Internet Protocol"*, RFC 2401, Internet Society, Network Working Group, Nov. 1998.

[11] Zhi (Judy) Fu and S. Felix Wu, Technical Report, Automatic Generation of IPSec/VPN Security Policies In an Intra-Domain Environment, *Networks and Infrastructure Research Lab, Motorola*, USA.

[12] Rivest, R., A. Shamir, and L. Adleman," *A Method for obtaining digital signatures and Public-key cryptosystems"*, Communications of the ACM, vol2, no.2, pp.120-126, February 1978.

[13] *Davis*, D., and W. Price, "The Application of Digital Signature Based on Public-key Cryptosystems", *Proceedings of the Fifth International Computer Communications Conference*, October 1980.

[14] Z. Fu, S. F. Wu, H. Huang, K. Loh, F. Gong, "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution", IEEE Policy 2001 Workshop, Jan. 2001.

[15] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press Inc., 1978.

[16] William Stallings*, Data and Computer Communications*, Fifth Edition, Prentice-Hall of India, 1998.

[17] Carlton R. Davis, *IPSec: Securing VPNs,* ISBN: 0-07-212757-0.

[18] Chris Brenton with Careron Hunt*, Active defense – a comprehensive guide to Network security*, BPB Publications.

[19] S. Kent, R. Atkinson, "*IP Authentication Header*", RFC 2402, Nov. 1998

[20] S. Kent, R. Atkinson, "*IP Encapsulating Security Payload*", RFC 2406, Nov. 1998

[21] W. Richard Stevens, "*TCP/IP Illustrated, Volume 1: the Protocols*", Addison Wesley, 1994.

[22] N. Doraswamy, D. Harkins, "*Ipsec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*", Prentice Hall, ISBN: 0130118982

[23] IPsec*, Cisco White paper*. Posted: Sat Jul 1 03:04:23 PDT 2000.

[24] Diffie, W, and M. Hellman, *"New Directions in Cryptography," IEEE Transactions on Information Theory,* vol. IT-22, no.6, pp. 644-654,November 1976.

[25] Stallings William, *Cryptography and Network Security: Principals and practice*, Second Edition, Prentice-Hall, Inc, New Jersey, 1999.

[26] Sanna Jaalinoja, *Aspects of cryptography and Internet Security*, The University of Oulu, Department of Mathematical Sciences, M.Sc thesis, November 1999.

[27] Check Point software technologies Ltd., ***Virtual Private Network Security Components*****,** A Technical White Paper, 2001.

[28] Kevin Fenzi, & Dave Wreski, Linux Security HOWTO v1.3.1, 11 February 2002.

[29] Robert Bova*., VPN: The time is now, White paper, Internet journal, 2001,INT media group.*

J.M.P. Martinez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, "Integrating Data Warehouses with Web Data: A Survey," IEEE Trans. Knowledge and Data Eng., preprint, 21 Dec. 2007, doi:10.1109/TKDE.2007.190746.(PrePrint)

Kazi Chandrima Rahman has been serving as an Assistant Professor at the Department of Computer Science and Engineering (CSE), The University of Asia Pacific. She joined UAP in October 2009 as an Assistant Professor and on October 2005 as a Lecturer. Prior to joining UAP, she was a Lecturer at BRAC University from September 2003 to September 2005. She also worked as a Software Developer in Grameen Solutions from May 2002 to January 2003. She Completed her B.Sc. (Hons) and M.Sc in Computer Science from University of Dhaka. She has research papers on Research and Educational Network, Sensor Network and TCP Congestion Control in ICCT, IJCIT, and IJENS-IJECS. Her research interests are on sensor network, network security, distributed and wireless network etc. She has been selected as a Potential Invitee of Emerging Research and Education Network for South Asia Region for her research work on Research and Educational Network. She is an advisor of Technical Journals Online: www.technicaljournalsonline.com. She is a member of International Association of Computer Science and Information Technology (IACSIT). Membership # 80339174. She is a member of Alumni Executive Committee of Dhaka University, CSE Department. Her website: kchandrima.wordpress.com