Automatic Generation of Conflict-Free IPsec Policies*

Chi-Lan Chang, Yun-Peng Chiu, and Chin-Laung Lei

Department of Electrical Engineering, National Taiwan University,
No.1, Sec. 4, Roosevelt Road, Taipei, Taiwan 106
{qlchang, frank}@fractal.ee.ntu.edu.tw, lei@cc.ee.ntu.edu.tw

Abstract. IPsec (IP security) will function correctly only if its security policies satisfy all the requirements. If the security policies cannot meet a set of consistent requirements, we said there are policy conflicts. In this paper, we analyze all situations which could possibly lead to a policy conflict and try to resolve all of them. We induce only two situations which could cause conflicts and also propose an algorithm to automatically generate conflict-free policies which satisfy all requirements. We also implement our algorithm and compare the results of simulation with the other approaches and show that it outperforms existing approaches in the literature.

1 Introduction

IPsec (IP security) provides authentication and confidentiality and is widely used in building a VPN (Virtual Private Network) of an organization or a business. Because that IPsec implements security services at IP layer and directly protects all the packets to achieve its security goal, the applications at upper layer do not need to make any change. According to the security requirements of an organization, security policies are configured on every security gateway or router in the VPN to enforce the required protections. Security requirements are goals we want to reach, and security policies which are configured in all involved computers are the detailed methods to reach those requirements.

Fig. 1. An encryption requirement example

Basically, security policies are generated according to the security requirements. Correctly producing security policies from requirements is an important issue, because if there is an error setting on one of these gateways or routers, packets will be

* This work was supported in part by Taiwan Information Security Center (TWISC), National Science Council under the grants No. NSC 94-3114-P-001-001Y

dropped and the whole network will be blockaded, or the security will be breached. For example, in **Fig. 1**, we have a simple network topology of four nodes, A, B, C, and D, and we also have one encryption requirement which is to protect the traffic from node A to node D, including the network links A to B, B to C, and C to D. Node A is the gateway of domain 10.1.2.1/24, node B is the gateway of domain 10.1.1.1/16, node C is the gateway of domain 10.2.1.1/16, and node D is the gateway of domain 10.2.2.1/24. The form of requirement we used in this paper is defined in [1]. For more explanations, the readers can refer to Section 3.

Fig. 2. Four different configurations of security policies that satisfy our requirement

There are many possible ways to generate security policies to satisfy our requirements. For example, in **Fig. 2**, we show four different configurations of security policies to satisfy the requirement of **Fig. 1**. Security policies decide how the IPsec tunnels will be build. In (a) of **Fig. 2**, the policy builds an encryption tunnel directly from node A to node D. Therefore, all the packets which are subject to our requirement will be protected from this tunnel, and we can say this policy **satisfies** our requirement. In (b) of **Fig. 2**, two security policies were generated. One is to protect the traffic from node A to node B, and the other is to protect the traffic from node B to node D. Obviously, the packets subject to our requirement will go through the first tunnel from A to B, and then go through the second tunnel from B to D. Assume the node B is trusted and data decryption is permitted on this node, we can see these two security policies satisfy our requirement. In (c) and (d) of **Fig. 2**, the security policies also satisfy our requirement.

However, there may be some conflicts between policies, and not every possible configuration of security policies will satisfy the requirements. This issue was first addressed in [1]. Therefore, we bring up a policy-generation algorithm to produce security policies to satisfy all the requirements, and try to minimize not only the number of total tunnels but also the number of tunnels passed by all possible traffic. The reason to minimize the number of total tunnels is that if there are more tunnels, the network topology will be more complicated, and the network management and analysis will be more difficult, too. On the other hand, if there are fewer tunnels which are passed by all possible traffic, we can save more computational power and improve the efficiency of the entire network. In this paper, we present our algorithm for automatic generation of conflict free policies, and compare the results with other three algorithms which are proposed by the predecessors.

The rest of the paper is organized as follows. The policy conflict problem is described in Section 2. Related works are addressed in Section 3. In Section 4, we present the proposed algorithm for generating conflict-free IPsec policies. Some simulation results are given in Section 5, and finally we make a conclusion of this paper in Section 6.

2 Policy Conflict Problem

The cause of a policy conflict is more than one policies have certain kind of intersection. More precisely, a conflict happens when the packets in one tunnel pass through a network node (a security gateway, a router, or any IPsec-enabled machine), but the packets are pulled down into another tunnel by the policy of that network node. Only in such situation, the policy conflict might have a chance to happen. Although not every intersection of tunnels will lead to a policy conflict, we can simplify the problem to consider only two tunnels and all kinds of intersection relationships between them. Since a policy conflict might happen only when packets in the first tunnel are pulled down into the second tunnel, we only need to consider the linear network area between the starting node and the end node of the second tunnel, and all the nodes of the first tunnel in this linear network area. Therefore, we only need to consider one-dimensional relationships between two tunnels. Here we refer to a tunnel as “first” or “second” depends on the order the packets encounter them. If two tunnels have no one dimensional intersection relationship, they will not conflict with each other.

For example, in **Fig. 3**, although the two tunnels are in two-dimensional, we could consider their one dimensional intersection part, which is the three points 2, 4, and 3. The point 2 and point 3 are the starting point and the end point of the second tunnel, so both of them are always on the same line. As for the point 4, it is the last node of the first tunnel in the line segment between point 2 and point 3.

Fig. 4 lists all intersection relationships between two tunnels in one-dimension, and we will analyze these relationships to find in what situations there will be policy conflicts. We omit other situations in which two tunnels do not intersect with each other. In the case (A) of **Fig. 4**, the upper tunnel and the lower tunnel have the same starting point and the same end point, and there will be no policy conflicts. In the case (B), the upper tunnel is the first tunnel and it starts at the same point where the second (lower) tunnel starts, and the traffic will be encapsulated twice at this starting point and decapsulated at the end point of the second tunnel. When the traffic is decapsulated at the end point of the second tunnel, it will be sent back to the end point of the first tunnel, and there might be a policy conflict because of the sending-back phenomenon.

In the case (C), the traffic is encapsulated at the starting point of the first (lower) tunnel. When the traffic travel the network to the starting point of the second (upper) tunnel, the traffic will be encapsulated again and sent to the end point of the second tunnel, which is also the end point of the first tunnel. So, the traffic will be decapsulated twice at the most right node, which is the end point of both of tunnels, and there will be no policy conflicts here. In the case (D), the traffic is encapsulated twice at the most left node, which is the starting point of both of tunnels, and then the traffic will be decapsulated and leave the second (lower) tunnel at the middle node, which is the end point of the second tunnel. Finally, the traffic is sent to the end point of the first (upper) tunnel and decapsulated there. In this case, there will be no policy conflicts, either. In the case (E), the traffic is encapsulated in the starting point of the first (upper) tunnel. At the middle node, which is the starting point of the second (lower) tunnel, the traffic will be encapsulated again at this middle node. Then, the traffic will be directly sent to the most right node, which is the end point of the second

tunnel, and the traffic will be decapsulated and sent back to the end point of the first tunnel. After arriving at the end point of the first tunnel, the traffic will leave the first tunnel and be sent to the most right node. Therefore, in the case (E), the policy conflict might happen. In the case (F), at first, the traffic is encapsulated by the longer (upper) tunnel, and then the traffic will encounter the shorter (lower) tunnel. The traffic will leave the shorter tunnel first and then leave the longer tunnel in the end. The length of a tunnel is defined as the hop count it passed. We make a summary of all the cases in which the policy conflict might happen in **Fig. 5**. There are only two situations which might cause the policy conflicts. The situation (1) has not been addressed yet in other papers before. The proposed algorithm could handle these two situations and generate conflict-free policies to meet the security requirements.

Fig. 3. An example: the intersection of two tunnels

Fig. 4. All intersection relationships between two tunnels

Fig. 5. Two situations which cause the policy conflicts

3 Related Works

In this section, we describe the definitions and syntax for security requirements and security policies we used in this paper. Security requirements are high-level goals we want to achieve, and security policies are actually configured at all related machines in a virtual private network. We also describe three existing algorithms which automatically generate security policies for the requirements. These algorithms were described in [2] and [3].

3.1 Security Requirements

In [1], the authors clearly defined two levels of security policies, which are the requirement level and the implementation level. A security policy set is **correct** if and only if it satisfies all the requirements. A requirement R is a rule of the following form: If condition C then action A :

$$R = C \rightarrow A$$

There are four kinds of requirements in [1]:

- Access Control Requirement (ACR):

$$flow\ id \rightarrow deny \mid allow$$

- Security Coverage Requirement (SCR):

$$flow\ id \rightarrow enforce(sec\text{-}function, strength, from, to, [trusted\text{-}nodes])$$

- Content Access Requirement (CAR):

$$flow\ id, [sec\text{-}function, access\text{-}nodes] \rightarrow deny \mid allow$$

- Security Association Requirement (SAR):

$$flow\ id, [SA\text{-}peer1, SA\text{-}peer2] \rightarrow deny \mid allow$$

flow id is used to identify a traffic flow, and is composed of 5 to 6 sub-selectors including *src-addr*, *dst-addr*, *src-port*, *dst-port*, *protocol*, and optional *user-id*. A requirement is **satisfied** if and only if all packets selected by the condition part execute the action part of the requirement.

3.2 Security Policies

For different implementations of IPsec, there are different ways to specify these policies. For convenience, we choose the syntax specified in **setkey** of the IPsec-Tools project [5] to describe a security policy in this paper. The IPsec-Tools project is a port of KAME's IPsec utilities to Linux-2.6 IPsec implementation. It supports NetBSD and FreeBSD as well. The **setkey** is a tool of the IPsec-Tools project to manipulate IPsec policies. According to the manual of **setkey**, the syntax to describe a security policy is as the following:

$$src_range\ dst_range\ upperspec\ policy;$$

src_range and *dst_range* are selections of the secure communication specified as IPv4/v6 address or address range, and it may accompany TCP/UDP port specification. *upperspec* is the upper-layer protocol to be used, such as TCP, UDP, or ICMP, and it can be specified as **any** for "any protocol." *policy* uses the format:

$$-P\ direction\ [priority\ specification]\ discard/none/ipsec\ protocol/mode/src\text{-}dst/level\ [...]$$

For instance, the configuration (a) of **Fig. 2**, which builds a tunnel from node 10.1.2.1 to node 10.2.2.1, has the policies as follows.

At node 10.1.2.1 (the gateway of domain 10.1.2.1/24):

$$10.1.2.1/24\ 10.2.2.1/24\ any\ -P\ out\ ipsec\ esp/tunnel/10.1.2.1\text{-}10.2.2.1/require;$$

At node 10.2.2.1 (the gateway of domain 10.2.2.1/24):

$$10.1.2.1/24\ 10.2.2.1/24\ any\ -P\ in\ ipsec\ esp/tunnel/10.1.2.1\text{-}10.2.2.1/require;$$

3.3 The Bundle Approach

The bundle approach is the first algorithm for solving the IPsec policy-conflict problem, and it was proposed in [2]. The bundle approach has two phases. First, it

Fig. 6. The three requirements example

Table 1. The three requirements of Fig. 6

R_1	$(\{10.1.*.*\}, \{10.2.*.*\}) \rightarrow (\text{encryption, weak, } 10.1.*.* , 10.2.*.*)$
R_2	$(\{10.1.2.*\}, \{10.2.*.*\}) \rightarrow (\text{authentication, strong, } 10.1.2.* , 10.2.*.*)$
R_3	$(\{10.1.*.*\}, \{10.2.2.*\}) \rightarrow (\text{encryption, strong, } 10.1.*.* , 10.2.2.*)$

Fig. 7. The solution of bundle approach for the three requirements example

separates the entire traffic into several disjoint traffic flow sets, called bundles, each of which is subject to a unique set of security requirements. Second, given a set of requirement actions for each bundle, the bundle approach generates corresponding security policies.

For example, in **Fig. 6**, we use a simple network for explanation. The node 10.1.2.1 is the gateway of domain 10.1.2.1/24, the node 10.1.1.1 is the gateway of domain 10.1.1.1/16, the node 10.2.1.1 is the gateway of domain 10.2.1.1/16, and the node 10.2.2.1 is the gateway of domain 10.2.2.1/24. The specifications of the three requirements are listed in **Table 1**. In the first phase, the bundle approach separates the entire traffic into four traffic filters: $F_1 = (\{10.1.*.*\}, \{10.2.*.*\})$, $F_2 = (\{10.1.2.*\}, \{10.2.*.*\})$, $F_3 = (\{10.1.*.*\}, \{10.2.2.*\})$, $F_4 = (\{10.1.2.*\}, \{10.2.2.*\})$. Then, the approach generates a set of policies for each filter and the solution is shown in **Fig. 7**.

3.4 The Direct Approach

The direct approach [2] builds tunnels directly and makes sure new tunnels do not overlap with any existing old tunnels. If a new tunnel overlaps with an existing tunnel, the approach builds two new connecting tunnels that do not overlap with others instead. Although the direct approach builds fewer tunnels than the bundle approach and has better efficiency, the paper [2] told us that there would be no solution using the direct approach if it could not find non-overlapped solution for one

requirement. However there might be solution using bundle approach. Therefore, the paper suggested a combined approach which uses the direct approach first and if it cannot find the solution, uses the bundle approach instead. The solution for the three requirements example is shown in Fig. 8.

Fig. 8. The solution of direct approach for the three requirements example

3.5 The Ordered-Split Algorithm

The ordered-split algorithm was proposed in [3]. It handles the requirements of authentication and the requirements of encryption separately. It first converts original requirements into tie-free requirement sets and then generates minimal size canonical solutions for the new requirements. A canonical solution is defined as a solution with no two tunnels start at the same place or end at the same place. A Tie-free requirement set is defined as a requirement set with no two requirements share the same *from* values and the same *to* values. The *from* and *to* values of one requirement determine the network area which needs to be protected. According to the simulation and analysis of [3], this algorithm generates fewer tunnels than the bundle/direct approach and the enhanced bundle/direct approach. The solution for the three requirements example is shown in Fig. 9. There are three tunnels in this solution, and the middle tunnel will provide both the encryption and authentication.

These algorithms are used to find correct security policy sets and minimize the total number of tunnels. As we said before, if the total number of tunnels is small, the network topology will be simpler and the policy management will be much easier. Generally speaking, if there are fewer tunnels, there would be fewer tunnels packets are needed to pass. The total computational cost will decrease, and all the security requirements are still satisfied. However, a question is, if two security policy sets have the same number of total tunnels, which policy set is a better solution? The answer is the policy set which has longer average tunnel length will be a better choice. It is because that we will need the tunnel length to be as long as possible in order to avoid traffic being encapsulated and decapsulated many times. In this paper, we present a policy-generation algorithm achieve these goals: (1) generates correct security policies, (2) minimizes total number of tunnels, and (3) makes every tunnel as long as possible.

Fig. 9. The solution of the ordered-split algorithm for the three requirements example

4 Policy-Generation Algorithm

Our algorithm will handle the two situations which cause policy conflicts, and will generate the correct security policies according to the security requirements. In **Fig. 5**, the situation (1) is a kind of nested tunnels and can be avoided if we always set the priority of the longer tunnel higher than the priority of the shorter tunnel. In other words, if there are more than two tunnels which start at the same point, we will set the policies at the starting point so that a longer tunnel will always be processed earlier than a shorter one. In most implementations of IPsec, the priorities of security policies at a network node are decided by the order with which we specify. If we always process a longer tunnel first, we will never encounter the situation (1) in **Fig. 5**. This is the reason for our algorithm to keep the policies in the descending order regarding the lengths.

Table 2 lists our algorithm. First, the algorithm takes the **Network**, and **Reqs** as its arguments. The **network** is a general graph structure and represents the entire network. We need to know the entire network topology to build each tunnel. And **Reqs** represents the list of security requirements. At first, according to the network range given by the requirements, we directly build a tunnel (policy) for each

Fig. 10. Two possibilities of situation (2) in **Fig. 5**

Table 2. The policy generation algorithm

```

1 PolicyList
2 Policy_Gen_Algorithm(Graph Network, ReqList Reqs) {
3   PolicyList policies;
4   policies = Directly_Build_A_Tunnel_For_Each_Requirement(Network, Reqs);
5   Sort_By_Length_Descending(policies);
6   Resolve_If_Tunnel_Overlapped(policies);
7   Delete_Redundant_Tunnels(policies);
8   return policies;
9 }
```

requirement. And then, all tunnels are sorted in the descending order with respect to their lengths.. The reason for doing this is to avoid the situation (1) in **Fig. 5**, which may incur a policy conflict. Then, we check if there are overlapped tunnels and resolve them. In **Fig. 10**, **cp** means the current policy we want to check and **ep** means the existing policy which is build. The resolving process is simple, because now we only need to worry about the situation (2) in **Fig. 5**. There are two possibilities: one is

Fig. 11. An example of redundant tunnels

ep is “first” tunnel and **cp** is the “second”, and the other is **cp** is the “first” tunnel and **ep** is the “second” tunnel. A tunnel is “first” or “second” depends on the order with which the packets encounter. The expressions (A) and (B) in the bottom of **Fig. 10** represent the logical relationships between **cp** and **ep**. If a policy conflict happens, we will cut the overlapped tunnel into two pieces. We cut **cp** because the length of cp is always less than or equal to the length of each existing policy **ep**. We want to keep all tunnels as long as possible, so we cut a short tunnel. In practice, we can check the *trusted_nodes* attribute of the requirements to see if the cutting node is trusted. In the final phase of our algorithm, we delete all the redundant long tunnels. A tunnel is redundant if there exist another tunnels with the same beginning point and the end point. **Fig. 11** shows an example of redundant tunnels.

We take the example in **Fig. 6** and **Table 1** to illustrate our algorithm and compare it with other three algorithms. At first, we directly build three tunnels for the three requirements individually, and sort them regarding their length. The result is shown in **Fig. 12**. In the second phase, we check if there are any overlapped tunnels like the situation (2) of **Fig. 5** and resolve these tunnels. The result of this checking process is shown in **Fig. 13**. Finally, we delete all redundant tunnels and get the final solution shown in **Fig. 14**.

We compare the four solutions for the three requirements example in the **Table 4**. The “tunnels passed” is the accumulated number of tunnels which are passed by the six possibilities of the traffic with the same direction (from left to right) listed in **Table 3**.

Fig. 12. Directly build a tunnel for each requirement

Fig. 13. Check the overlapped tunnels and resolve them

Fig. 14. The final solution for the three requirements example by our algorithm

Table 3. The six possibilities of the traffic

From	To
10.1.2.1/24	10.1.1.1/16 - 10.1.2.1/24 ¹
10.1.1.1/16	10.2.1.1/16
10.2.1.1/16 - 10.2.2.1/24	10.2.2.1/24
10.1.2.1/24	10.2.1.1/16
10.1.1.1/16	10.2.2.1/24
10.1.2.1/24	10.2.2.1/24

Table 4. Comparison of four solutions

	Number of Tunnels	Tunnels Passed
Bundle	10	36
Direct	4	13
Ordered-Split	3	10
Ours	3	9

Table 5. An example of four requirements

Req1	(3, 7) → (E, 3, 7)
Req2	(1, 4) → (E, 1, 4)
Req3	(2, 7) → (A, 2, 7)
Req4	(2, 5) → (E, 2, 5)

We give the other example in **Table 5** and **Fig. 15**. There are four security requirements in this example. We use a concise form to describe a requirement. We use the node identifier number to replace the IP address, and we use the notations *E* and *A* to represent the encryption and authentication. Our solution is shown in **Fig. 16**. **Fig. 17** is the solution by using the direct approach. There is one problem may occur in this solution. At node 4, there are three tunnels and the processing order of these three tunnels is undefined in the direct approach. So it is possible to cause a policy conflict if the situation (1) in **Fig. 5** happened. In fact, [2] and [3] did not consider this kind of the policy conflict. When we compare our algorithm with the algorithms in [2] and [3], we assume the situation (1) in **Fig. 5** will not cause a conflict. In the end, **Fig. 18** shows the solution by using the ordered-split algorithm, and we eliminate the tunnel which is an authentication tunnel from node 2 to node 7, because the other short tunnels T2, T3, T4, and T5 already provide both the encryption and authentication.

Table 6 shows the results of these algorithms. By using the direct approach, if we send traffic from node 3 to node 7, the traffic will pass through the tunnels T6, T3,

¹ The minus operator means that the network domain 10.1.2.1/24 is excluded.

Fig. 15. An example of four requirements

Fig. 16. The solution for the example of Fig. 15 by using our algorithm

Fig. 17. The solution for Fig. 15 by using the direct approach

Fig. 18. The solution for Fig. 15 by using the ordered-split algorithm

Table 6. The compare of three algorithms

	Total Number of Tunnels	Accumulated Number of Tunnels Passed
Direct	7	52
Ordered-Split	5	45
Ours	5	40

T5, and T7 in Fig. 17. By using the ordered-split algorithm, the same traffic will pass through the tunnels T3, T4, and T5 in Fig. 18. And by using our algorithm, the same traffic will pass through the tunnels T2, T3, and T6 in Fig. 16. The rest may be deduced by analogy. We calculate the accumulated number of tunnels passed by all

possible traffic with the direction from left to right. The fewer the accumulated number of tunnels is, the less computation power is consumed. As we can see, our algorithm has fewer accumulated number of tunnels.

5 Simulation

From the previous section, we observed that our algorithm has fewer tunnels and less waste of computational power when comparing to the ordered-split algorithm, which is better than the bundle/direct approach according to the analysis of [3]. Therefore, we implemented the ordered-split algorithm and our algorithm in C++, and compare the results of these two algorithms for various scenarios. We will focus on the total number of tunnels and the number of tunnels passed by all possible traffic. If there are fewer tunnels, the network topology will be simpler, and the network management and analysis will be easier, too. On the other hand, if there are fewer tunnels passed by all possible traffic, we can avoid unnecessary encryptions/decryptions. Therefore our algorithm consumes less computation power and improves the efficiency of the entire network.

Fig. 19. The average numbers of policies in the network of 20 routers

Fig. 19 compares the result of our algorithm and order-split algorithm (shown as OSA in the graph) in the network of 20 routers. The X-axis is the average number of policies, and the Y-axis is the number of requirements. Our simulation program randomly generates 10,000 cases and calculates the average values produced by both algorithms. We can see our algorithm generates fewer tunnels than the ordered-split algorithm. **Fig. 20** shows the average number of tunnels passed by all possible traffic in the network of 20 routers. The X-axis of **Fig. 20** is the number of requirements, and the Y-axis is the average number of tunnels passed by traffic. **Fig. 21** and **Fig. 22** show the comparison in the network 50 routers. In practice, a VPN of a business or an organization seldom has more than 50 routers. By observing every simulated case, we can see that our algorithm generates fewer tunnels and requires fewer computational costs than the ordered-split algorithm.

Fig. 20. The average number of tunnels passed by traffic in the network of 20 routers

Fig. 21. The average numbers of policies in the network of 50 routers

Fig. 22. The average number of tunnels passed by traffic in the network of 50 routers

6 Conclusions and Future Work

In this paper, we analyzed what situations would cause the policy conflicts. And we induced only two situations would cause conflicts. Then, we proposed an algorithm to automatically generate conflict-free policies and satisfy all the security requirements. Our algorithm generates fewer tunnels and save more computational power than other existing approaches in the literature. Our algorithm is “off-line,” and there might be more improvement to speed up this algorithm. In the future, we can try to find a solution to resolve the policy conflict problem online. Besides, the delivery of policies is still a problem. If we have generated correct and efficient policies, how do we deliver them to each network node and configure them? Every security gateway, router, or network device may have different implementation of IPsec, and we need different ways to configure them. Therefore, it is our future work to develop a uniform interface of delivering and configuring policies.

References

1. Z. Fu, S. F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine, and C. Xu, “IPsec/VPN Security Policy: Correctness, Conflict Detection, and Resolution,” IEEE Policy 2001 Workshop, 2001.
2. Z. Fu and S. F. Wu, “Automatic Generation of IPsec/VPN Security Policies in an Intra-Domain Environment,” 12th International Workshop on Distributed Systems: Operations & Management (DSOM 2001), 2001.
3. Y. Yang, C. U. Martel, and S. F. Wu, “On Building the Minimal Number of Tunnels - An Ordered-Split approach to manage IPsec/VPN policies,” 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), April 2004.
4. W. Stallings, “Cryptography and Network Security: Principles and Practices,” 3/e, Prentice Hall, 2002.
5. The IPsec-Tools Project, <http://ipsec-tools.sourceforge.net/>
6. S. Kent and R. Atkinson, “Security Architecture for the Internet Protocol,” RFC 2401, Internet Society, Network Working Group, November 1998.
7. S. Kent and R. Atkinson, “IP Authentication Header,” RFC 2402, Internet Society, Network Working Group, November 1998.
8. S. Kent and R. Atkinson, “IP Encapsulating Security Payload (ESP),” RFC 2406, Internet Society, Network Working Group, November. 1998.
9. D. Maughan, M. Schertler, M. Schneider, and J. Turner, “Internet Security Association and Key Management Protocol (ISAKMP),” RFC 2408, Internet Society, Network Working Group, November 1998.
10. J. Jason, L. Rafalow, E. Vyncke, “IPsec Configuration Policy Information Model,” RFC 3585, Internet Society, Network Working Group, August 2003.
11. J. D. Moffett, “Requirements and Policies,” Position paper for Policy Workshop 1999, November 1999.