On Building the Minimum Number of Tunnels – An Ordered-Split approach to manage IPSec/VPN policies

Yanyan Yang, Charles U. Martel, S. Felix Wu
Department of Computer Science
University of California, Davis
{yyyang, cumartel, sfwu}@ucdavis.edu

Abstract — Most of the current work in policy management for IPSec/VPN focuses on how to configure a single IPSec box or a pair of IPSec boxes. However, it has been shown in [3] that the local correctness of IPSec policies in every box individually does not necessarily guarantee global correctness. Therefore, it is critical to have a systematic way to analyze the security requirements globally and to generate, automatically and correctly, a set of IPSec policies to ensure the security for all the end-to-end connections. Previously, in [1, 2], two different algorithms (i.e., ***bundle*** and ***direct***) were introduced to solve the policy generation problem in an "offline" fashion. While these two algorithms are efficient in producing globally correct policy rules, the number of output policy rules (i.e., the results themselves) is much greater than necessary. In other words, while, the existing approaches can produce a solution quickly, the quality of the solution is far from optimal. In practice, this is undesirable for several reasons. For instance, "more IPSec policy rules" implies "more complicated virtual network topology." Therefore, in this paper, we focus on "how to produce a minimum set of IPSec/VPN tunnels". We formulate this target problem as a special type of task-scheduling problem, and develop a new method, the ordered-split approach, to produce a provably minimum set of globally correct policy rules. We have also compared the new approach with existing methods (such as the bundle approach) in simulation, and our results clearly demonstrate that the ordered-split approach performs significantly better.

Keyword — IPSec policy, security requirement, policy management

I. Introduction

IPSec (IP layer security protocol) [6, 7] is designed to provide interoperable and cryptographically based security for end-to-end traffic flow, while IPSec policies [5] are set up at security gateways and firewalls for VPNs to provide anticipated security protection. Even in an intra-domain environment as shown in the following figure, it is critical to enforce the security requirements of all the firewalls/gateways and insure they are satisfied and conflict-free.

As shown in Figure 1, the end-to-end communication goes through an AH (Authentication Header [7]) tunnel, an ESP (Encapsulation Security Payload [6]) tunnel and another AH tunnel between two sub-domains A and B. Needless to say, all the tunnels need to satisfy the security intentions and no information and packets should be leaked on the way

from the source to destination. A single subtle mistake in tunnel setup can cause information leakage (i.e. plain text being transmitted) or packets being dropped. Especially when all the IPSec/VPN tunnels are tunnel mode, it is very important to make sure that the intermediate routes and gateways don't drop encrypted traffic.

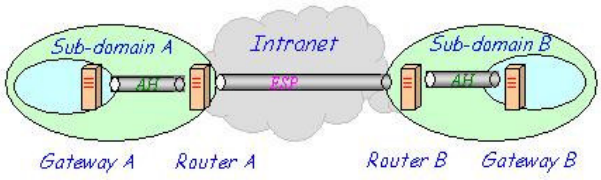


Figure 1 An IPSec/VPN tunnel setup in an intra-domain environment

- Security requirements and policies:

  [1, 2, 3] address the need to separate the definitions of security requirements and security policies. Security requirements are defined as high-level security policies to specify users' security intentions about some end-to-end communication across the same or different administrative domains. Security policies, however, are defined using a low-level security implementation, with which the network administrator can instruct a network device to perform certain low-level IPSec/VPN actions. Once the security requirements are specified, we should be able to determine what kind of security policies need to be enforced. Basically, in our research context, we define four categories of security requirements with *flow id*[1] that cover all types of requirements[2], as shown in Figure 2.

Requirement Categories	Definitions	Example
ACR (Access Control Requirement)	The requirements that specify access control.	*flow id -> deny | allow*
SCR (Security Coverage Requirement)	The requirements that define security coverage over an area (including the network links and nodes).	*flow id -> protect (sec_function, strength, from, to, trusted_nodes)*
CAR (Content Access Requirement)	The requirements that specify the need of accessing the content.	*flow id -> deny_sec | allow_sec (sec_function, access_nodes)*
SAR (Security Association Requirement)	The requirements that indicate the capability of performing IPSec security association.	*flow id -> deny_SA | allow_SA (SA_peer1, SA_peer2)*

Figure 2 Security requirements definitions and categories

- Satisfaction of security requirements by policies:

  With the definitions of security requirements/policies and the four types of requirements, we can easily tell if the security policies satisfy the requirements or if there are security violations as addressed in [3]. For instance,

[1] Flow is generally presented as a 5-tuple: source address, destination address, source port, destination port, and protocol.
[2] Because access control policies can be easily determined after the tunnel configuration completes, we omit ACR in the overall algorithm.

suppose that we have one security coverage requirement (e.g. strong encryption needed as a security coverage requirement from node 1 to node 4 as in the following figure with the policies (IPSec/VPN tunnels) built. The following SCR (Security Coverage Requirement) requires *strong* encryption from network node 1 to node 4 and the trusted (intermediate) nodes are 2 and 3, for network flow between source node 1 and destination node 4.

```
[srcIP=1 dstIP=4 prot=TCP srcPort=ANY dstPort=ANY] →
SCR  Encryption Strength=STRONG
     From=1 To=4
     Trusted={2,3}
```

Figure 3 A sample SCR

As illustrated in Figure 5, there are four ways to satisfy the requirement by building one tunnel, or two tunnels or three tunnels. The formal representation of a policy, for example the tunnel built in Figure 5 (a), is defined as follows. It builds an IPSec ESP tunnel from node 1 to node 4 with *strong* encryption and tunnel mode.

```
[srcIP=1 dstIP=4 prot=TCP srcPort=ANY dstPort=ANY] →
IPSec     Prot=ESP Mode=Tunnel
     Strength=STRONG
     from=1 to=4
```

Figure 4 A sample policy representation

Theoretically, the network administrator decides how many tunnels s/he would build for a single requirement, however, in practice performance is an important issue when data is being transmitted along the tunnels.

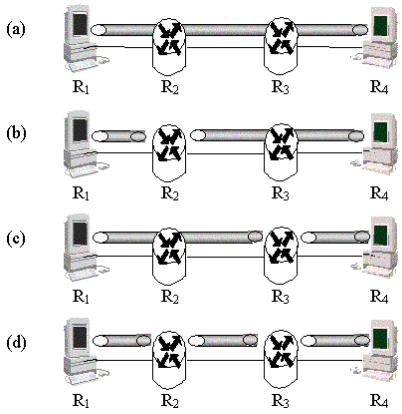


Figure 5 One requirement and Four possible solutions for it

- Conflicts among security requirements/policies:

  Nevertheless, different security gateways and firewalls usually have different security requirements, even within an intra-domain environment. [1, 2, 3, 4] address scenarios where requirement/policy conflicts occur and security

violations happen if the IPSec/VPN policies are not set up correctly. In the following example, we see that changes in network environment or a carelessness network administrator could easily create a bad tunnel setup.

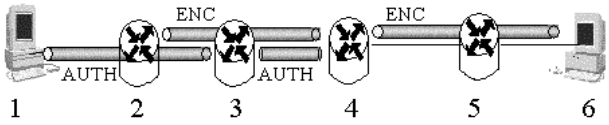


Figure 6 Security Violation/Conflict

As shown in the figure, it seems that each policy satisfies its corresponding requirement. However putting all the policies together may cause conflicts. In this example, the flow is tunneled to node 3 with authentication. It is also tunneled from node 2 to node 4 with encryption before the authentication tunnel exits. It's easy to see that the authentication function applies at node 1 and will not be de-applied at the tunnel node 2 to node 4. Thus, the traffic will be encapsulated by node 1 for authentication and be encapsulated again by node 2 for encryption. When node 4 decapsulates and finds out that the destination is node 3, the flow will be sent back to node 3. Eventually, node 3 will decapsulate the flow and send it to its destination. As a result, the flow is sent in plaintext from node 3 to node 4 because of the tunnel interaction, which violates the original security intentions (i.e. security requirements). This is one possible mistake that often occurs. Also, changes in network environment (e.g. network topology) may also cause trouble.

- Performance considerations with many tunnels in the network:

  From the definitions of security requirements and policies, one set of requirements may be satisfied with multiple policy solutions. As long as the policies themselves don't conflict with each other and don't violate security requirements, the policy solution is one of the ways of building the tunnels.

  Yet with the significant increase in network nodes and network traffic, we need to consider many other factors that may impact our end-to-end communication. For IPSec/VPN communication, in order to achieve privacy and integrity, cryptographic mechanisms are performed to encrypt and decrypt. It's known that strong cryptosystems involve a lot of *expensive* mathematical calculations (e.g. exponentiation in RSA algorithm). The cost of performing such calculations has greatly impacted the communication performance. For instance, in our earlier example, it is obvious that the solution with fewer tunnels gives us a better way to perform IPSec/VPN operation, that is, in Figure 5, solution (a) gives us the optimal solution that has only one tunnel to provide the desired

security coverage for the area. Because it only needs to carry out the encryption-decryption calculations once at node R_1 and R_4, respectively, while solution (b) and (c) need to do the calculations twice and (d) needs to do them three times.

Therefore, we seek an algorithm to automatically generate policies to satisfy the requirements and also provide us an optimal solution with the minimum number of policies, such that the network flow travels to the minimum number of tunnels and thus has the least performance cost.

Previously, in [1, 2], two different algorithms (i.e., *Bundle* and *Direct*) were introduced to solve the policy generation problem in an "offline" fashion. While these two algorithms are efficient in producing globally correct policy rules, the number of output policy rules (i.e., the results themselves) is much greater than necessary. In other words, while, these approaches produce a solution quickly, the quality of the solution is far from optimal. In practice, this is undesirable for several reasons. For instance, "more IPSec policy rules" implies "more complicated virtual network topology." Therefore, in this paper, we develop a new method, the Ordered-Split approach, to produce a provably minimum size set of globally correct policy rules. The Ordered-Split algorithm is based on traditional "task scheduling" to automatically generate security policies to satisfy all the requirements. We also prove that our algorithm produces an optimal solution with the minimum number of the policies. In this paper, our algorithm will only handle the SCR requirements. In practice, other requirements, such as SAR or CAR, can be validated after the SCR results are produced. Meanwhile, based on the specification of CAR and SCR requirements, we can decide which intermediate network nodes along the route path are trusted. In order to give an intuitive sense of the algorithm compared with the previous one, we present a simple example to show that the Ordered-Split algorithm produces less expensive solutions.

```
SCR#1: ENC      2-5   trusted   {3,4}
SCR#2: AUTH     2-7   trusted   {4,5}
SCR#3: ENC      1-4   trusted   {2,3}
SCR#4: ENC      3-7   trusted   {4,5,6}
CAR#1: (ENC, AUTH)    by        2
SAR#1: non-ENC                  3-5
```

Figure 7 An Example: 6 security requirements (4 SCRs, 1 CAR and 1 SAR)

Suppose that we have 7 requirements for the security routes and security gateways in an intra-domain as shown in Figure 7. Both the previous algorithm and the Ordered-Split algorithm take requirements as input and output the policies as a solution. With the bundle/direct approach, the solution produced in the time order is shown in Figure 8.

On the other hand, if we run Ordered-Split algorithm, we would end up with fewer policies (shown in Figure 9) to satisfy the same set of security requirements.

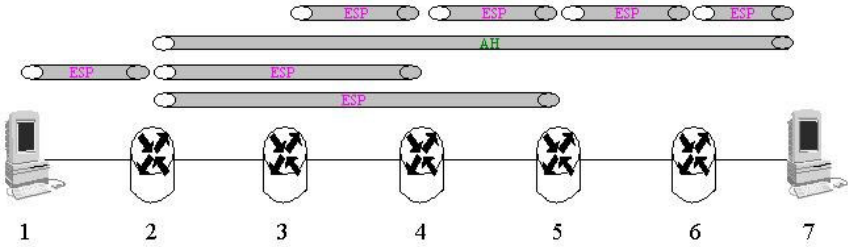


Figure 8 The solution with the previous algorithm

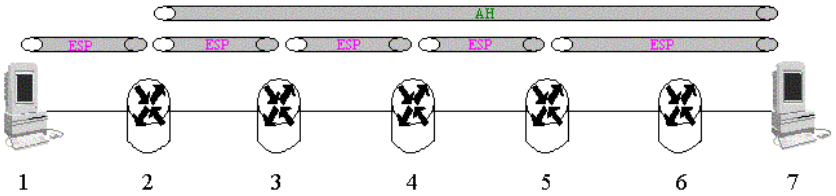


Figure 9 The solution with the Ordered-Split algorithm

The rest of the paper is structured as follows. Section II gives brief overview of the related work. We introduce the Ordered-Split algorithm in detail with our formal analysis and theoretical proofs towards the correctness, the completeness and the optimality of the algorithm in Section III. Then Section IV presents the simulation results and analysis. Finally, we conclude our work and outline some future work in Section V.

II. Related work

There have been quite a few research and development projects conducted towards IPSec/VPN policy management. Under the current IETF (Internet Engineering Task Force) working groups, IPSec protocol WG focuses on the work to enhance the present IPSec protocol design [6, 7] and IPSP (IP Security Policy) WG carries out research on IPSec policy framework [5], configuration and requirement.

Based on the RFCs and Internet drafts defined at IEFT, [1] proposed an approach making use of Dijkstra's shortest path algorithm to generate the policies in an intra-domain environment. It provides the first algorithm to automatically and correctly generate the policies without human intervention. However, as we addressed earlier, the solution may use additional tunnels (as shown in the earlier example). The approach introduces two different algorithms handling the requirements and generating the policies, bundle approach and direct approach.

The Bundle approach basically separates traffic into disjoint traffic flows, which are called bundles, and builds a set of policies on each flow (i.e. each bundle). For instance, as shown in Figure 10, suppose that there are several requirements between two network domains, the Bundle approach builds policies for each network flow (bundle). The policies certainly satisfy all the requirements for each bundle, but as we can see, the solution may contain redundant tunnels for different bundles but for the same area.

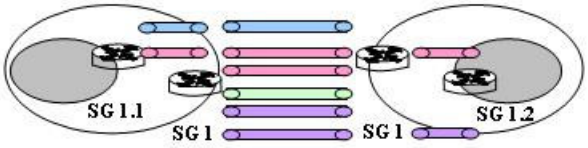


Figure 10 A sample solution for requirements of different bundles between two domains

The Direct approach builds policies for each different requirement, rather than grouping the traffic into bundles, in order to achieve higher efficiency. With the same network topology and security requirements in the example above, the Direct approach produces fewer policies to satisfy requirements with a more efficient policy generation.

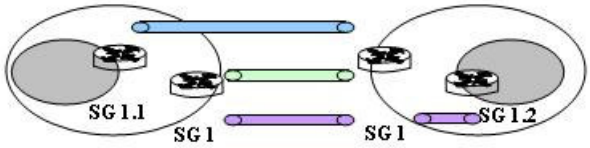


Figure 11 A sample solution that direct approach produces

If we define an optimal solution as a solution with the minimum number of tunnels built, we will measure all the approaches with respect to the number of tunnels. Neither the bundle approach nor the direct approach provides an optimal solution. As we addressed earlier, we need to provide an optimal solution with the minimum number of tunnels to satisfy the requirements, avoid tunnel overlaps and alleviate the expense of tunneling by building the minimum number of tunnels. In the next section, we will introduce the Ordered-Split algorithm that produces such a solution and prove that it produces correct solutions with the minimum number of tunnels.

III. Ordered-Split Algorithm

The Ordered-Split algorithm borrows some concepts from traditional “task scheduling”. Given a set of requirements *R*, we want to find a set of tunnels *T* to satisfy the requirements, avoid overlaps and use the minimum number of tunnels. We start with some notation and basic properties we use, then we describe the algorithm and sketch its proof. More details of the proof are in the Appendix.

A. Notation

- Requirements: We represent a requirement r as: $(f, t), f < t$.

  $f \quad < \quad t$

  |-------|

- Policies (tunnels): We represent a policy p as: $[f, t], f < t$.

  $f \quad < \quad t$

  O------O

- Tunnel chains to satisfy requirements

  To satisfy a requirement $r = (f, t)$, we should have k tunnels $s_0, s_1, ..., s_{k-1}$, $s_i = [f_i, t_i]$ such that:

  $f = f_0$

  $t = t_{k-1}$

  $t_i = f_{i+1}$, *for* $i = 0, 1, ..., k-2$

- Tunnel overlap

  We define the following scenario as an overlap. Suppose that we have two tunnels s_i $[f_i, t_i]$ and s_j $[f_j, t_j]$. Overlap occurs if and only if:

  $f_i < f_j < t_i < t_j$

  O-------O

  O--------O

  Tunnel overlaps are prohibited, because they may cause security violations or conflicts as we addressed in the previous section. We say that tunnel $[f_i, t_i]$ has a *left overlap* with $[f_j, t_j]$, and $[f_j, t_j]$ has a *right overlap* with $[f_i, t_i]$.

- Containing tunnels (no overlaps)

  Suppose that we have two tunnels s_i $[f_i, t_i]$ and s_j $[f_j, t_j]$ then we consider the following cases as containing tunnels, that is, the two tunnels share one end point or one is fully contained by the other, i.e. left contained as shown:

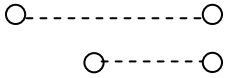
  $f_i = f_j \quad < \quad t_i \quad < \quad t_j$

  O----------O

  O-------O

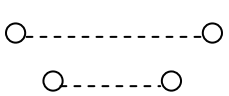
  OR right contained as shown:

$f_i < f_j < t_i = t_j$



OR fully contained as shown:

$f_i < f_j < t_j < t_i$



Though left and right contained tunnels are legal, we will show that we can always find minimum tunnel solutions that use no left or right contained tunnels.

B. Problem definition

With the notations and definitions of the IPSec tunnel concepts specified in the previous sections, we define the formal problem as:

Given a set R of n requirements (f_i, t_i) where $i = 1, 2, \ldots, n$, we want to find a minimum size set of tunnels T such that:

1) No two tunnels in T overlap, i.e. never have (f, t) and (f', t') in T with $f < f'$ and $f' < t < t'$.
2) Each requirement in R is met by a chain of tunnels in T.

Note that a set of tunnels that satisfies 1) is LEGAL. A set of tunnels that satisfies 2) satisfies R.

C. Canonical solutions

Given any optimal set of tunnels T for R, we can convert to an equal size tunnel set where no two tunnels start or end at the same time. We can do this by first removing ties at start values: begin with the earliest start value with ties and remove these, then continue on the resulting tunnel set at the next largest value with a tie. Once we have removed all ties when tunnels start, then we remove all tunnels that end at the same value starting with the largest tie value.

Proof: Suppose f is the earliest tie for a start time, so we have e.g. in T, $[f, t_1]$, $[f, t_2]$, $[f, t_3]$ with $t_1 < t_2 < t_3$.

We replace these by tunnels $[f, t_1]$, $[t_1, t_2]$, and $[t_2, t_3]$.

First, we still cover R by chains. Any chain using $[f, t_2]$ in T now just uses $[f, t_1]$ and $[t_1, t_2]$ (and similarly for $[f, t_3]$). The new set is called T'.

Second, we create no new overlaps. Suppose some tunnel [f', t'] in both T and T' overlaps a new tunnel, say [t_1, t_2]. So $f' < t_1 < t' < t_2$ (if right overlap with t_2, then [f', t'] overlapped the original tunnel [f, t_2]). Now if $f' > f$, then [f', t'] overlaps [f, t_1], and if $f' < f$ then [f', t'] overlaps [f, t_2] (can't have $f = f'$ since we just got rid of ties with f). Either of these overlaps contradicts [f', t'] in T. Thus the new set T' is valid, covers R and has the same size as T. Essentially the same argument works for removing ties at *TO* values.

Thus we can always assume we are dealing with a *Canonical Solution* where all tunnels start/end at *FROM or TO* values in R and no two have the same start time and no two have the same end time (we may have two tunnels where one ends at a value v and another starts at v).

D. Tie-free requirement sets

If we have multiple requirements, which have the same *FROM* value, e.g. (f, t_1), (f, t_2), (f, t_3) we can replace these by three requirements (f, t_1), (t_1, t_2), (t_2, t_3). Clearly any solution to the new requirement set is valid for the original one (we can easily combine chain's from the new solution to get chain's for the old one), and it is a corollary to the above result on converting an optimal solution with two (or more) tunnels starting at the same time to one with no ties, that the new requirement set uses the same number of tunnels as the old one. Note however that we may reduce the number of requirements when we do this transformation. In the example above, if we already had a requirement (t_1, t_2) we would essentially drop the requirement (f, t_2) since we "replace" it by a requirement, which already exists (so we don't need to create the requirement (t_1, t_2)).

We call these TIE-FREE requirement sets. Since we can take any set of requirements and convert it to a new set where no two requirements share a *FROM* value and no two share a *TO* value (though the same value may be both a *FROM* and *TO*), we restrict our attention to solutions for TIE-FREE requirement sets.

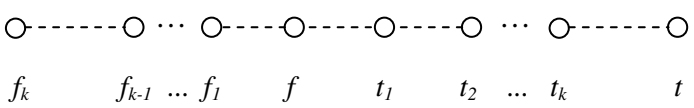
E. Ordered-Split Algorithm

We assume that we start with a TIE-free requirement set R and apply the Ordered-Split algorithm to the sorted set of requirements (by *FROM* value). We process each requirement in sorted order. We split existing tunnels that overlap the current requirement and then add any additional tunnels needed to complete the tunnel chain for the current requirement.

Suppose that we have applied our algorithm to the first *i-1* requirements to create a solution T_{i-1}. We next process requirement $r_i = (f, t)$ as follows:

1) If no tunnel in T_{i-1} overlaps (f, t), build [f, t] and let $T_i \leftarrow T_{i-1}$ + [f, t].

2) Some tunnels in T_{i-1} overlap (f, t). This will always include tunnels that overlap f (left overlap). Let $[f_1, t_1]$, ... $[f_k, t_k]$ be the left overlap tunnels with $t_1 < t_2 < ... < t_k$. Note that these k tunnels can't overlap each other and all have $f_j < f$, so are nested (i.e. $[f_j, t_j]$ contains $[f_{j-1}, t_{j-1}]$, $j = 2,... , k$).



In this case we remove these k nested tunnels from T_{i-1} and replace them by $[f_k, f_{k-1}]$, $[f_{k-1}, f_{k-2}]$, ..., $[f_1, f]$, $[f, t_1]$, $[t_1, t_2]$, ..., $[t_{k-1}, t_k]$. Clearly, new tunnel start values are at f (a *FROM* value) and t_1, ..., t_{k-1} each of which is a *TO* value (since greater than f) of a requirement which has a left overlap with (f, t).

A. If no tunnel has a right overlap with (f, t), we take the largest value $v < t$ such that there is a tunnel ending at v, and build a new tunnel $[v, t]$ and add that to T_{i-1} to get T_i.

B. There can be a single tunnel $[f', t']$, which has right overlap with (f, t) (only one since this is the result of an earlier tunnel overlap, so no nesting can occur). If so we remove $[f', t']$ and add $[f', t]$ and $[t, t']$ to T_{i-1} to get T_i (so f' plays the role of value v of case A).

Final cleanup: If T_i has any pair of tunnels which start at the same time or end at the same time remove these as described in section C to make the new solution Canonical.

Proof sketch (more details in the Appendix):

After each step i, the overall algorithm maintains the following properties:

- We have a legal and minimum size *Canonical Solution* for the first i requirements.
- All tunnels start and end at a *FROM*/TO value of one of the first i requirements.

We need to show that i) the solutions we construct are legal (no overlaps), ii) they meet all the requirements (a chain covers each requirement), and iii) the number of tunnels used is minimum.

For legality, when we process the i^{th} requirement (f, t) we remove all tunnels which overlap (f, t) and replace them by a set of non-overlapping tunnels. The only possible concern is that some new tunnels added might overlap an existing

tunnel. However, since we always make tunnels shorter, a simple case analysis (described in the appendix) shows that any tunnel that overlaps a newly created tunnel would have overlapped an existing tunnel.

Our solutions meet all requirements since the new tunnels created maintain all prior chains, and we always construct a new chain to meet the current requirement (for 2A and 2B it's easy to argue that there is always a chain from f to either v or f', so adding $[v, t]$ or $[f', t]$ creates a chain from f to t). Showing that the number of tunnels is minimum is the tricky part. To prove this we show that each tunnel in T_i starts at either:

a) A *FROM* value of a requirement in R, or

b) A *TO* value t for a requirement (f, t) among the first i in R such that either:

   i). There is an overlapping requirement $r = (f', t')$ among the first i with $f < f' < t < t'$ OR

   ii). *For* $f < f' < t < t'$ there is a required chain of tunnels from f' *to* t'.

The properties above show that the final set of tunnels is optimal since clearly we need one tunnel to start at each *FROM* time in R (if no tunnel started at a *FROM* time, can't have a valid chain for that requirement). In addition, we show in the Appendix that when we have a *TO* value t as in case b), we must have a tunnel starting at t in any *Canonical Solution*. Thus since we have exactly one tunnel for each required start point, the number of tunnels is optimal. Note that the new tunnels built in step 2 start at f (a *FROM* value) or a value t_j, which must be a *TO* value of a requirement (f', t_j) with $f' < f < t_j < t$ (since earlier requirements all have smaller *FROM* values). Thus (f, t) overlaps t_j and case b-i) applies. Similarly, for cases 2A and 2B the tunnels we build meet b-i) or b-ii) above.

We need the final cleanup step because when we create a tunnel such as $[t_1, t_2]$ in step 2) there could be an existing tunnel, which starts at t_1 (this was a tunnel contained by $[f_2, t_2]$). Thus we now have two tunnels starting at t_1.

The algorithm never creates more than $2n-1$ tunnels (though often much fewer as shown in the next section) for requirements. It is also easy to implement this algorithm to run in $O(n^2)$ time.

F. Example

To better understand the Ordered-Split algorithm, recalling the example (Figure 9) shown in the previous section, we present a detailed explanation of how the algorithm runs to produce an optimal solution.

Because the requirement set contains both ENC requirements and AUTH requirements, we take one kind of requirement at a time. For ENC requirements, we have (1, 4), (2, 5) and (3, 7) as FROM and TO value pairs (in the

sorted order). Clearly they overlap with each other. We first build the tunnel [1, 4]. Then we process the next requirement (2, 5) and it overlaps with the existing tunnel [1, 4], thus we split the existing tunnel to create [1, 2], [2, 4] and then apply case 2A to build [4, 5]. It's obvious that this tunnel chain meets both requirements. Then we process (3, 7), which has only a left overlap (from the existing tunnel [2, 4]). Thus we split [2, 4] to [2, 3] and [3, 4]. Again case 2A applies (with v =5), so we build [5, 7]. At this stage, we are done processing all the ENC requirements and move to AUTH requirements (in this example only one (2, 7)). We have two options here to build the tunnels for AUTH requirements. Either build AH tunnels as for ENC tunnels, but we need to avoid overlaps with existing ENC requirements; or we just add *AUTHENTICATION* onto the ESP tunnels already built. It depends on how users want to configure the whole tunnel set, because the first choice provides separate AH tunnels but more care needs to be taken to avoid overlaps; on the other hand, the second choice is overlap free but some traffic may experience unnecessary overhead of performing authentication that is not required. Note that the *AH* tunnels may be left or right contained with some of the existing ESP tunnels, because the requirements they meet belong to different kinds of requirements.

IV. Simulation and analysis

We have implemented the algorithm in C under the Linux platform and simulated for various scenarios. We use the BANDS architecture [4] to locate the route path first, then discover all the requirements, and then run the Ordered-Split algorithm to get the final solution of the tunnels. Since in practice the average number of routers that network traffic normally traverse usually does not exceeded 35, we assume that there are no more than 35 routers along the route path of the network flow in the network topology. In fact, we believe that, even in the future, we should see normally much less than 35 security gateways along the route path for an end-to-end connection. The X-axis represents the number of the requirements (that are generated randomly), while the Y-axis represents the number of policies (i.e. tunnels) that are either set up manually or generated by different approaches. Since we take the number of tunnels as the major measurement among different approaches, we focus on comparing the number of security policies each approach produces.

- *Compared with Manual policy configuration*

  Generally, the network administrator sets up IPSec/VPN policies manually by building one tunnel for each requirement. Thus, the number of tunnels is equal to the number of the requirements. As shown in Figure 12,

when the number of requirements increases, the number of policies keeps growing tremendously, compared to the results of Ordered-Split approach. Also, building a tunnel for each requirement increases the risk of security violations, because as we discussed earlier, tunnels overlapping each other causes security conflicts among the tunnels.

When we set up a policy configuration and build tunnels, the most critical issue is to avoid any security violations and conflicts. On the basis of this, in order to keep the expenses as minimum as possible, we try to seek optimal solutions to decrease the number of the tunnels.

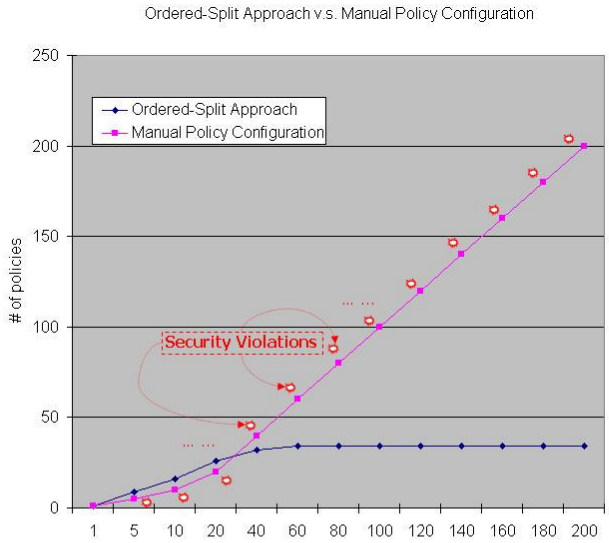


Figure 12 The Ordered-Split Approach versus Manual Policy Configuration

- *Compared with Bundle/Direct approach*

With both the Bundle/Direct approach and the Ordered-Split approach, the results show that they provide a correct solution, where policies satisfy all the requirements without causing any security violations and conflicts. In other words, the solutions from both approaches are overlap free.

However, using the traditional Bundle/Direct approach, the simulation results in Figure 13 show that the number of policies in the solution increases sharply when the number of requirements increases, while the number of policies generated by the Ordered-Split approach keeps is essentially constant once the number of requirements gets to about 40. Because of the nature of Ordered-Split approach, when the amount of requirements increases to a certain level that each router needs to set up tunnels with another, the approach simply builds one tunnel between each consecutive pair of routers.

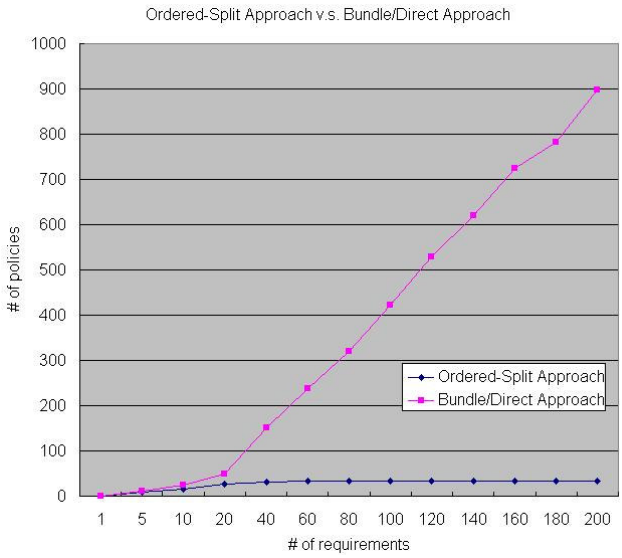


Figure 13 The Ordered-Split Approach versus Bundle/Direct Approach

- *Compared with enhanced Bundle/Direct approach*

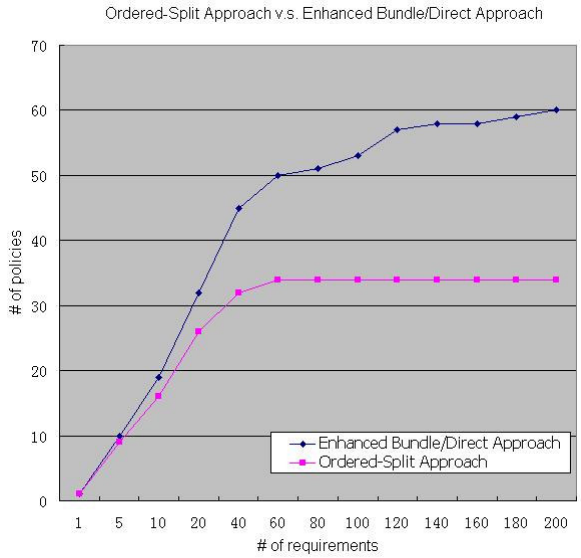


Figure 14 The Ordered-Split Approach versus Enhanced Bundle/Direct Approach

The original Bundle/Direct approach doesn't avoid duplicate tunnels. If one tunnel has already been set up between two routers earlier and the approach produces a new tunnel that is exactly the same as the existing one, the Bundle/Direct approach builds one more, instead of getting rid of the redundant one. Therefore, we enhance the approach by removing all the redundant tunnels, and the results are shown in Figure 14.

With the enhanced Bundle/Direct approach, the number of tunnels has been dramatically reduced, since all the redundant tunnels are eliminated. However, when the number of requirements grows, the difference between

enhanced Bundle/Direct and the Ordered-Split approach is obvious. The former uses almost twice as many tunnels as the latter.

V. Conclusions

"How to build the minimum number of IPSec/VPN tunnels to satisfy a set of security requirements" is the central focus of this paper. Our first contribution is to model this problem as a special type of task scheduling problems. To our best knowledge, our task-scheduling approach for solving IPSec/VPN has not been used before, and our specific solution method is different from those used in the operations research and scheduling community. As the second major contribution of this paper, we present a new approach, the Ordered-Split algorithm, to automatically generate IPSec/VPN policies to satisfy requirements, to avoid any security conflicts and also, importantly, to keep the expense low by maintaining the minimum number of policies. We have also formally proved that the Ordered-Split approach is both correct and complete. With the simulation based on different scenarios, we compared the Ordered-Split approach with manual policy configuration (as many network administrators do in reality) and our previous work — the Bundle/Direct approach and the enhanced Bundle/Direct approach. Our results show that the Ordered-Split performs significantly better, especially when the network topology is very complicated and has many requirements. With the Ordered-Split approach, people could easily manage policy configuration in a distributed environment by exploiting a distributed architecture (for instance, the BANDS architecture [4]) to discover all the requirements and compute the policy solutions. Furthermore, in the sense of the Ordered-Split approach being able to handle new requirements after an existing and optimal policy solution is settled, the Ordered-Split approach provides a more on-line mechanism (unless the new solution needs to remove tunnels and replace them with new tunnels) than the previous works to manage the solution to a dynamic problem.

While our current simulation focuses on comparing our Ordered-Split approach to other algorithms in terms of the number of tunnels we build to meet the requirements, future simulations will compare our approach to others with different performance and optimization goals in the future work. Also we will conduct further research on how to satisfy both existing requirements and the new requirement if we add an out-of-order requirement.

References

[1] Z. Fu and S. F. Wu, "Automatic Generation of IPSEC/VPN Policies in an Intra-Domain Environment", 12th International Workshop on Distributed Systems: Operations & Management (DSOM 2001), October 15-17, 2001, Nancy, France.

[2] Z. Fu, "Automatic Generation of Security Policies", Technical Report, http://shang.csc.ncsu.edu/secpolicy.pdf.

[3] 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.

[4] Y. Yang, Z. Fu, S.F. Wu, "BANDS: An Inter-Domain Internet Security Policy Management System for IPSec/VPN", 8th IFIP/IEEE International Symposium on Integrated Network Management 2003, Colorado Springs, Colorado, March 2003.

[5] M. Blaze, A. Keromytis, A. Keromytis, L. Sanchez, "IP Security Policy Requirements", draft-ietf-ipsp-requirements-02.txt, Internet Draft, IPSP Working Group, August 2002.

[6] S. Kent, "IP Encapsulating Security Payload (ESP)", draft-ietf-ipsec-esp-v3-06.txt, Internet Draft, IPsec Working Group, July 2003.

[7] S. Kent, "IP Authentication Header", draft-ietf-ipsec-rfc2402bis-04.txt, Internet Draft, IPsec Working Group, July 2003.

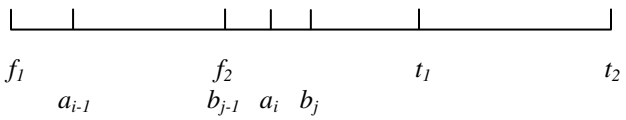
Appendix

Here we prove a key technical lemma and then fill in the details of the proof of the Ordered-Split algorithm.

A.1. Overlap Lemma

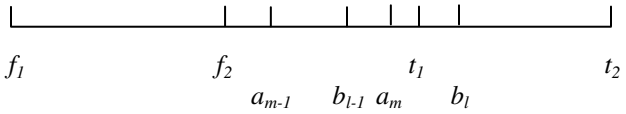
Suppose we have two overlapping requirements $r_1 = (f_1, t_1)$ and $r_2 = (f_2, t_2)$ with $f_1 < f_2 < t_1 < t_2$ in R, then any *Canonical Solution* for R will have chains from: :*i)* f_1 to f_2 ; *ii)* f_2 to t_1 and *iii)* t_1 to t_2.

Proof: We know a valid solution has tunnel chains $[a_1, a_2]$, $[a_2, a_3]$, ... , $[a_{k-1}, a_k]$ for r_1 (so $a_1 = f_1$, $a_k = t_1$) and a chain $[b_1, b_2]$, ..., $[b_{r-1}, b_r]$ for r_2. To show i), consider the first value $a_i \geq f_2$. If $a_i = f_2$, $[a_1, a_2]$, $[a_2, a_3]$, ... , $[a_{i-1}, a_i]$ is the desired chain. Otherwise $a_i > f_2$. Now consider the first value j such that $b_j > a_i$ (no $b_j = a_i$ or we would have two tunnels ending at the same time violating our *Canonical* assumption). We have $a_{i-1} < f_2 \leq b_{j-1} < a_i < b_j$ as below.



However, this means these two tunnels overlap, which is a contradiction to them being part of a *Canonical* (and thus valid) *Solution*. Thus the first case with $a_i = f_2$ must apply.

Similarly, to show ii) and iii), consider the first value $b_l \geq t_1$. If $b_l = t_1$, $[b_1, b_2]$, $[b_2, b_3]$, ... , $[b_{l-1}, b_l]$ is the desired chain for ii) and $[b_l, b_{l+1}]$, ..., $[b_{r-1}, b_r]$ for iii). Otherwise $b_l > t_1$. Now consider the first value m such that $a_m > b_{l-1}$ (no $a_m = b_{l-1}$ or we would have two tunnels ending at the same time violating our *Canonical* assumption). We have $a_{m-1} < b_{l-1} < a_m \leq t_1 < b_l$ as below.



However, this means these tunnels overlap, which is a contradiction to them being part of a *Canonical* (and thus valid) *Solution*. Thus $b_l = t_1$ must apply and we have the desired chains from f_2 to t_1 and t_1 to t_2.

Any chains of tunnels, which must appear in all canonical solutions, are called *required chains*. Thus we have:

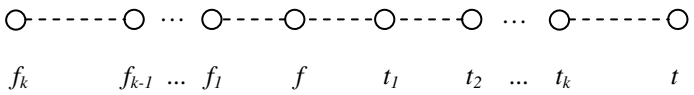
Overlap Corollary: If a requirement set R has required chains from f_1 to t_1 and from f_2 to t_2 with $f_1 < f_2 < t_1 < t_2$, then we also have required requirement chains from f_1 to f_2 , from f_2 to t_1 and from t_1 to t_2.

A.2. Optimization: The Ordered-Split algorithm gives us a legal solution with the minimum number of tunnels.

We repeat the algorithm for convenience: Suppose that we have applied our algorithm to the first *i-1* requirements to create a solution T_{i-1}. We next process requirement $r_i = (f, t)$ as follows:

1) If no tunnel in T_{i-1} overlaps (f, t), build $[f, t]$ and let $T_i \leftarrow T_{i-1} + [f, t]$.

2) Some tunnels in T_{i-1} overlap (f, t). This will always include tunnels that overlap f (left overlap). Let $[f_1, t_1]$, ... $[f_k, t_k]$ be the left overlap tunnels with $t_1 < t_2 < ... < t_k$. Note that these k tunnels can't overlap each other and all have $f_j < f$, so are nested (i.e. $[f_j, t_j]$ contains $[f_{j-1}, t_{j-1}]$, $j = 1, 2, ... , k-1$).



In this case we remove these k nested tunnels from T_{i-1} and replace them by $[f_k, f_{k-1}]$, $[f_{k-1}, f_{k-2}]$, ..., $[f_1, f]$, $[f, t_1]$, $[t_1, t_2]$, ..., $[t_{k-1}, t_k]$. Clearly, new tunnel start values are at f (a *FROM* value) and t_1, ..., t_{k-1} each of which is a *TO* value (since greater than f) of a requirement which has a left overlap with (f, t).

A. If no tunnel has a right overlap with (f, t), we take the largest value $v < t$ such that there is a tunnel ending at v, and build a new tunnel $[v, t]$ and add that to T_{i-1} to get T_i.

B. There can be a single tunnel $[f', t']$, which has right overlap with (f, t) (only one since this is the result of an earlier tunnel overlap, so no nesting can occur). If so we remove $[f', t']$ and add $[f', t]$ and $[t, t']$ to T_{i-1} to get T_i (so f' plays the role of value v of case A).

Final cleanup: If T_i has any pair of tunnels which start at the same time or end at the same time remove these as described in Section C to make the new solution Canonical.

Theorem: The Ordered-Split algorithm produces a legal tunnel set for R with the minimum number of tunnels.

Proof: We now show that the solution each tunnel set T_i produces has no overlapping tunnels, has a chain for each of the requirements so far and uses the minimum number of tunnels. We assume inductively that this is true for T_{i-1} with respect to the first $(i-1)$ requirements. In addition, we will assume that whenever we have a tunnel $[f, t]$ where f is the *TO* value of a requirement, there is a required chain from f to t for R. Finally, we assume that each tunnel starts at either:

a) A *FROM* value of a requirement in R, or

b) A *TO* value t for a requirement (f, t) among the first i in R such that either:

i). There is an overlapping requirement $r = (f', t')$ among the first i with $f < f' < t < t'$ OR

ii). *For* $f < f' < t < t'$ there is a required chain of tunnels from f' *to* t'.

These properties are all trivially true initially when we build a single tunnel for the first requirement. We now show that they are all maintained when we process the next requirement.

No overlaps: If we build the single tunnel $[f, t]$, by definition it has no overlaps with the existing tunnels, so the new set has no overlaps. When there are left overlaps with (f, t) we remove all such tunnels and replace them with a set of tunnels, which has no pairwise overlaps. Thus the only possible problem with these tunnels is if one overlapped some existing tunnel $[f', t']$, which had no left overlap with (f, t) (so was not removed). For new tunnels of type $[f_{k-1}, f_{k-2}]$, if $[f', t']$ has a left overlap, then it would have overlapped the original tunnel $[f_{k-1}, t_{k-1}]$ and if it has a right overlap it would have (left) overlapped $[f_{k-2}, t_{k-2}]$ (since $t' < f$). Similarly for $[f_1, f]$, $[f, t_1]$, and $[t_{k-1}, t_k]$ tunnels, its easy to show that if $[f', t']$ overlaps one of these tunnels, it would have overlapped an original tunnel. For the new tunnel $[v, t]$ constructed in 2A, if $[f', t']$ has a left overlap that contradicts the definition of v (since t' would be a larger value less than t where a tunnel ends) and if it has a right overlap, it contradicts the no overlap condition of 2A.

For 2B, we need to justify the claim that there can only be one tunnel, $[f', t']$, which has right overlap with (f, t) (a second such tunnel is the only type which could overlap the newly created tunnels $[f', t]$ and $[t, t']$). To show this, first note that since $f' > f$, both f' and t' must be *TO* values of earlier requirements. Now, to prove a contradiction, suppose there are two nested tunnels $[f', t']$ and $[f'', t'']$ which overlap t, with $f < f' < f'' < t < t'' < t'$. Let (x, f'') be the original requirement which has f'' as its *TO* value. Thus there is a required chain of tunnels from x to f'' and this overlaps the required chain from f' to t' (by our induction assumption, since there is a tunnel $[f', t']$ there is also a required chain). Thus by our *Corollary* to the overlap lemma there is a required chain from f' to f'', so since there is only one tunnel starting at f' it can't be $[f', t']$. Thus no such overlapping pairs exist.

Chains for each requirement: Since all tunnels we remove are replaced by chains that cover the old tunnels, it's clear all the prior requirements are still covered. To make sure we cover the new requirement the only issue is to show that there is a chain from f to v if 2A applies and from f to f' if 2B. Let y be the largest *TO* value among the first i requirements, which is less than t (so $y = v$ for

2A and f' in 2B). If $y = t_k$, then by construction there is a chain from f to t_k . If $y > t_k$, let (x, y) be the requirement with this *TO* value. Since $x < f < y < t$, the two requirements overlap. Let the required tunnel chain from x to y be $[x, b_1]$, ..., $[b_{r-1}, b_r]$ $[b_r, y]$. The tunnel ending at y does not overlap f and cannot overlap $[f_k, t_k]$, thus $b_r \geq t_k$. If this holds with equality we have a chain from f to y (the chain from f to t_k and then the final tunnel $[b_r, y]$]). If $b_r > t_k$ then simply continue backwards in the (x, y) chain until we hit the first tunnel $[b_{j-1}, b_j]$ with $t_k \geq b_{j-1}$. There must be such a tunnel, and since this doesn't overlap f and cannot overlap $[f_k, t_k]$, $t_k = b_{j-1}$. Thus we have the desired chain from f to t_k and then the rest of the chain to y. So in all cases we get a chain to cover (f, t).

Minimum Number of Tunnels: Here we will show that each new tunnel created starts at a *FROM* value or a *TO* value meeting conditions b-i) or b-ii) above. In addition, we will show that all newly created tunnels $[a, b]$ reflect a required chain from a to b (actually this later condition is sufficient to show the number of tunnels is minimum). If we just build $[f, t]$ all conditions are trivially true. The new start times of tunnels built in step 2 start at f (a *FROM* value) or a value t_j, which must be a *TO* value of a requirement (f', t_j) with $f' < f < t_j < t$ (since earlier requirements all have smaller *FROM* values). Thus (f, t) overlaps t_j and case b-i) applies, by the *Overlap Lemma* there is a required chain from t_j to t, so a tunnel must start at t_j. Also, since the required chain from f' to t_j overlaps with the required chain from t_{j-1} to t, the tunnel from t_{j-1} to t_j is a required chain. Similarly, for cases 2A, v must be a *TO* value of some requirement (v', v) with $v' < f < v < t$ so this requirement overlaps with (f, t) and there is a required chain from v to t. In case 2B the tunnel $[f', t']$ has an associated requirement (f'', f') with $f'' < f < f' < t$ so there is a required chain from f' to t. Finally, for the tunnel $[t, t']$, since there was a tunnel $[f', t']$ there is a required chain from f' to t' (overlapping (f, t) and thus by the *Overlap Corollary*, there is a required chain from t to t'). Thus in all cases the new tunnels we build starting at a *TO* value x represent a required chain and thus a tunnel must start at x and we maintain our induction invariant.

The final cleanup step clearly preserves chains, creates no overlaps, and uses the same number of tunnels. In addition, its easy to see that if we have two existing tunnels $[f, t_1]$, $[f, t_2]$ which represent required chains, then any Canonical solution must have a chain from t_1 to t_2, and thus the new tunnels we create when converting to a Canonical solution, also represents a required chain. Thus, at the end we have a set of tunnels where one tunnel starts at each required start of a chain, so we have a minimum size set of tunnels.