TOWARDS ACHIEVING SECURE CONTENT DISCOVERY IN
CENTRALIZED AND DECENTRALIZED SYSTEMS

A Dissertation in
Information Sciences and Technology
by
Sushama Karumanchi

© 2016 Sushama Karumanchi

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2016
The dissertation of Sushama Karumanchi was reviewed and approved* by the following:

Anna C. Squicciarini  
Associate Professor, College of Information Sciences and Technology  
Dissertation Advisor, Chair of Committee

Dinghao Wu  
Assistant Professor, College of Information Sciences and Technology

Zhenhui Li  
Assistant Professor, College of Information Sciences and Technology

Scarlett Miller  
Assistant Professor, Engineering Design and Industrial Engineering

Andrea Tapia  
Associate Professor, College of Information Sciences and Technology  
Director of Graduate Programs

*Signatures are on file in the Graduate School.
Abstract

A content or service selection process involves a user searching for a content or service of his choice over the Internet. Content is data which is in various forms such as video, text, audio, etc. Content is served to users via multiple formal channels such as Web services, content hosting networks such as peer-to-peer networks, Cloud, etc. Web Services are software technologies that offer software services to clients over the Web, by publishing the functions related to their services in public registries which can be invoked by the clients over the Web. In specific, Web services are unique in that they allow interoperability among Web applications. That is, a Web service is able to invoke or use other Web services to provide a required output to the user. The growing success of Web services has resulted in a large number of providers, which implement services of varying degree of sophistication and complexity. While on the one hand, the availability of a wide array of services has created a competitive and flexible market that suits well the needs of different types of users, on the other hand, it requires them to select among possibly hundreds of similar services. Web service selection plays a crucial role in Web service lifecycle. Here, several application-dependent requirements will need the users and the service providers to exchange a lot of information which is possibly sensitive, during the selection phase. Also, Web services are prone to attacks and are found to have vulnerabilities. Content hosting networks are prone to having malicious nodes that breach security and privacy of the network in various ways.

In this work, the security and privacy implications caused by the exchange of possibly large amount of potentially sensitive data required by optimized strategies for service and content selection, are studied. We begin with an in-depth empirical analysis of Web service vulnerabilities that hinder users’ and service providers’ privacy and security. Next, we propose two comprehensive frameworks for centralized and decentralized service and content hosting systems which focus on addressing security and privacy issues of the information of users and service providers, at the time of service selection. We are specifically interested in the following aspects
during the selection of services: 1. Protection of the search query of the user, 2. Protection of the business rules of the service providers, 3. Selection of services according to the user’s privacy and security requirements, and 4. Achieving privacy, security, and access control of the search query of the user when services are hosted on complex networks such as peer-to-peer networks. We propose innovative methods and mechanisms to achieve these objectives. Our contributions are as follows: 1. We propose a unique service recommendation and ranking model which performs partial matching of user criteria against service attributes, 2. We propose a solution which enables the users and the service providers to exchange information in a minimalistic manner, that is, an efficient approach that does not unnecessarily reveal sensitive information to parties participating in service discovery and provisioning, and finally, 3. With respect to decentralized systems, we propose a policy-based search query forwarding scheme in peer-to-peer networks and also tackle the challenging problem of verifying and validating the honesty of nodes in peer-to-peer networks with respect to search query forwarding. This dissertation also includes multiple experiments to test the complexity, the computational overhead and the effectiveness of our approaches.
Table of Contents

List of Figures viii
List of Tables x
Acknowledgments xi

Chapter 1
Introduction 1
  1.0.1 Service Selection in Centralized Architecture . . . . . . . . . 4
  1.0.2 Content Discovery in Decentralized Architecture . . . . . . . 7
  1.0.3 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . 8
  1.0.4 Layout of this Dissertation . . . . . . . . . . . . . . . . . . . 10

Chapter 2
Related Work 11
  2.0.1 Work related to Web Service Vulnerabilities . . . . . . . . . . 11
  2.0.2 Work related to Web service Recommendation . . . . . . . . 12
  2.0.3 Work related to Web service Selection . . . . . . . . . . . . . 13
  2.0.4 Work related to services and data hosted on P2P architectures 15
  2.0.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

Chapter 3
Web Service Vulnerabilities 19
  3.1 Web Services Standards and Security . . . . . . . . . . . . . . . 19
  3.2 A taxonomy of Web-Service Vulnerabilities . . . . . . . . . . . . 21
    3.2.1 A Client-target Taxonomy . . . . . . . . . . . . . . . . . 21
    3.2.2 Vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . . 23
      3.2.2.1 Static Vulnerabilities . . . . . . . . . . . . . . . . 24
      3.2.2.2 Dynamic Vulnerabilities . . . . . . . . . . . . . . 30
  3.3 Empirical Evaluation . . . . . . . . . . . . . . . . . . . . . . . 32
3.3.1 Methodology .............................................. 33
3.3.2 Results .................................................. 34

Chapter 4
Web Service Ranking and Recommendation .................. 37
  4.0.1 Architecture ............................................ 37
  4.0.2 Service Recommendation and Ranking .................. 42
  4.1 User Search Criteria and Service Provisioning Rules ...... 43
    4.1.1 Criteria and Rule conditions ........................... 43
    4.1.2 Satisfiability of conditions ............................ 47
  4.2 Ranking Model for Secure WS Delivery .................... 50
    4.2.1 Graph Model for Service Composition .................. 50
    4.2.2 Composite Web service ranking Strategies ............ 52
      4.2.2.1 Input on all criteria: Weighted Average of Criteria 52
      4.2.2.2 Priority-Based ranking ............................... 53
      4.2.2.3 Bounded ranking .................................... 55

Chapter 5
Private Web Service Selection ............................... 57
  5.1 Private Matching ......................................... 59
  5.2 Private Service Selection ................................ 60
    5.2.1 Two-Party Coordinator ................................. 60
    5.2.2 Multi-Party Coordinator ............................... 62
    5.2.3 Clause Selection from a Non-Atomic Rule ............ 65

Chapter 6
Policy-compliant Search Query Routing for Web Service Discovery in Peer to Peer Networks ..... 69
  6.1 Overview of the approach ................................ 72
  6.2 Policy Definitions ....................................... 74
  6.3 Bloom Filter Representation of the Attributes .......... 77
  6.4 Policy enforcement ....................................... 80
    6.4.1 Node Criteria evaluation on Bloom filter ........... 80
    6.4.2 Policy Satisfaction for Forwarding and Filtering Policy .... 83
    6.4.3 Protocol Steps ....................................... 84

Chapter 7
Path Verification for Policy-routed Queries ................. 86
  7.1 Cryptographic Background ............................... 89
    7.1.1 Attribute-based Encryption ........................... 89
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1.2</td>
<td>Identity-based Aggregate Signature</td>
<td>90</td>
</tr>
<tr>
<td>7.2</td>
<td>Design Goals and Threat Model</td>
<td>90</td>
</tr>
<tr>
<td>7.3</td>
<td>Policy Compliant Search</td>
<td>91</td>
</tr>
<tr>
<td>7.4</td>
<td>Malicious Node Detection</td>
<td>94</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Authenticating Paths</td>
<td>94</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Path Filtering</td>
<td>97</td>
</tr>
<tr>
<td>7.4.2.1</td>
<td>Policy and Path Comparison</td>
<td>99</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Compliance Checking</td>
<td>101</td>
</tr>
<tr>
<td>7.4.3.1</td>
<td>Baseline Method</td>
<td>101</td>
</tr>
<tr>
<td>7.4.3.2</td>
<td>Offloading Partial Computation to Verified Nodes</td>
<td>102</td>
</tr>
<tr>
<td>7.4.3.3</td>
<td>Reusing ABE Ciphertexts</td>
<td>106</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Alternate Probabilistic model</td>
<td>107</td>
</tr>
<tr>
<td>7.4.5</td>
<td>Determining the Number of Nodes to verify</td>
<td>107</td>
</tr>
<tr>
<td>7.4.6</td>
<td>Attacks and Countermeasures</td>
<td>111</td>
</tr>
<tr>
<td>7.5</td>
<td>Malicious Node Detection</td>
<td>94</td>
</tr>
</tbody>
</table>

**Chapter 8**

**Experiments and Complexity**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Complexity</td>
<td>113</td>
</tr>
<tr>
<td>8.2</td>
<td>Experiments</td>
<td>116</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Service and Content Selection in Centralized Systems</td>
<td>116</td>
</tr>
<tr>
<td>8.2.1.1</td>
<td>Ranking Algorithms</td>
<td>116</td>
</tr>
<tr>
<td>8.2.1.2</td>
<td>Private Service Negotiation</td>
<td>120</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Service and Content Selection in Decentralized Systems</td>
<td>123</td>
</tr>
<tr>
<td>8.2.2.1</td>
<td>Policy-compliant Query Routing</td>
<td>123</td>
</tr>
<tr>
<td>8.2.2.2</td>
<td>Policy-compliance Verification</td>
<td>127</td>
</tr>
<tr>
<td>8.3</td>
<td>Summary</td>
<td>133</td>
</tr>
</tbody>
</table>

**Chapter 9**

**Conclusion and Future Work**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Conclusions</td>
<td>134</td>
</tr>
<tr>
<td>9.2</td>
<td>Future Work</td>
<td>137</td>
</tr>
</tbody>
</table>

**Bibliography**

vii
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Taxonomy of Web Services Vulnerabilities.</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Overall Workflow and Architecture of the Selection and Provisioning Framework. A User provides search terms to our ranker component which ranks and recommends services to the user. Our private negotiator (PN) component compares the user and the service provider requirements, and also the service provider provisioning rules in a private fashion, finally leading to the user choosing a service and executing it.</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>Example of a directed acyclic graph (DAG) for a Travel Request Service. The graph represents 4 composite services each consisting of a travel, hotel and payment service.</td>
<td>52</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparing provisioning rules and clause selection in a private fashion among service providers in case of non-atomic provisioning rules.</td>
<td>67</td>
</tr>
<tr>
<td>6.1</td>
<td>Architecture of Content and Service Selection on Decentralized Systems.</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>An example of a bloom filter encoding attribute values of a node.</td>
<td>78</td>
</tr>
<tr>
<td>6.3</td>
<td>Attenuated Bloom Filters, Creation and Usage.</td>
<td>80</td>
</tr>
<tr>
<td>6.4</td>
<td>Distance between indices.</td>
<td>82</td>
</tr>
<tr>
<td>7.1</td>
<td>An Example for Policy-Compliant Distributed Search.</td>
<td>92</td>
</tr>
<tr>
<td>7.2</td>
<td>Resource Discovery: Search query is forwarded throughout the network in search of a resource.</td>
<td>97</td>
</tr>
<tr>
<td>7.3</td>
<td>Example of path filtering based on policy and path history.</td>
<td>100</td>
</tr>
<tr>
<td>7.4</td>
<td>Protocol for Checking Policy Satisfaction of Node in Iterative Model (the group generator $g$ and big prime $p$ are predefined as system parameters).</td>
<td>101</td>
</tr>
<tr>
<td>7.5</td>
<td>Workflow of Offloading Partial Computations to Verified Nodes.</td>
<td>104</td>
</tr>
<tr>
<td>7.6</td>
<td>Offloaded Computations to the Verified Nodes.</td>
<td>105</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>7.7 Probability Estimation of Dishonest Detection in Non-Iterative and</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Iterative Models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.1 Comparison with existing classification: comparing our approach</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>with QWS classification, which classifies services into four classes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2 Performance test for our approaches: Weighted Average (WA) and</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Single Criteria Preference (H)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3 Provision rule comparison in private fashion: Finding common</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>attributes by performing two-party set intersection, with two and four</td>
<td></td>
<td></td>
</tr>
<tr>
<td>service providers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.4 Provision rule comparison in private fashion: Finding common</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>attribute values by performing multiparty set intersection where the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>service providers are the multiple parties participating</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.5 WSDL parser performance test</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>8.6 Satisfaction ratios in case of equal number of nodes traversed,</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>for networks of 1000, 5000, and 9000 nodes. Our approach produces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>higher satisfaction values compared to random walk approach. This</td>
<td></td>
<td></td>
</tr>
<tr>
<td>graph is a zoomed-in version showing clear differences between the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.7 Satisfaction ratios in case of equal number of nodes traversed,</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>for networks of 1000, 5000, and 9000 nodes. Our approach produces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>higher satisfaction values compared to random walk approach. This</td>
<td></td>
<td></td>
</tr>
<tr>
<td>graph is the version showing the scale 0 to 1 for satisfaction values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.8 Satisfaction ratios by varying the number of conditions in each</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>clause of a policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.9 Computing Aggregated Signatures</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>8.10 Verifying Aggregated Signatures</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>8.11 CPABE Encryption</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>8.12 CPABE Decryption/Baseline Policy Compliance Verification</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>8.13 Policy Comparison</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>8.14 Path Intersection</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>8.15 Path Compliance Verification</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.1</td>
<td>Authenticating and Non-Authenticating Services</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Static Vulnerabilities by Service Type</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Security criteria</td>
<td>48</td>
</tr>
</tbody>
</table>
Acknowledgments

First and foremost, I want to thank my advisor Dr. Anna Squicciarini. I appreciate her valuable contributions of time, ideas, and support to make my Ph.D. journey productive and enthusiastic. Her keen enthusiasm for research has motivated me throughout the pursuit of my Ph.D. She has provided me a great example as a successful woman in the field of computer science, especially, information security.

I would also like to thank my committee members, Dr. Dinghao Wu, Dr. Zhenhui Li, and Dr. Scarlett Miller. They have provided me brilliant suggestions and feedback on my dissertation. I would like to thank them for spending their precious time to serve on my committee.

My family has provided me strong moral support throughout my Ph.D. I would like to thank them for their patience and perseverance.
A content or service selection process involves a user searching for a content or service of his choice over the Internet. Content is data which is in various forms such as video, text, audio, etc. A content or service selection by a user is a process of discovery of the content or service, desired by the user, hosted on a set of servers. The discovery is done over the internet. There is a significant amount of study on search for content on distributed networks [1-5]. These works focus on problems related to searching of content or resources hosted on peer-to-peer (P2P) networks. Literature also offers works dealing with constrained Web service selection. The most widely investigated type of constraints are related to QoS (quality of service), where several solutions focusing on different aspects of finding the optimal service selection have been proposed (e.g., [6], [7], [8], [9], [10]). While all of these efforts confirm the relevance and need of a constrained service and content discovery approach, none of them takes into account the privacy and security of data related to the users and the service providers. In particular, existing works fail to recognize that even optimized strategies for service-selection involve the exchange of large amount of potentially sensitive data, causing potentially serious privacy leaks. The increasing regulatory and legal requirements for personal privacy and the proliferation of Web services have strengthened the need to protect the personal privacy of Web service users [11]. Several researchers have acknowledged the importance of privacy in Web services [11, 12, 29-31]. Yee and Korba [12] propose a privacy policy negotiation approach for a service for securing the user’s privacy from the service providers. The same group of researchers [11] also propose an architecture for a privacy policy compliance system to verify whether the service providers abide by the privacy policies issued by the service users. While all of these works
focus on the data privacy of the service users, none of them take into account the problem of minimization of information exposure for the information that is needed to be disclosed at the time of service selection. For instance, the service providers might require user information before providing the service, to verify if the user is above 18 years. In the traditional approaches for service selection, all of the sensitive information of the user is provided to the service providers without any information exposure minimization techniques implemented. One of our goals in this dissertation is to minimize information exposure between the users and the service providers, taking into consideration the privacy of not just the users, which has been the focus of the previous works, but also the privacy of the service providers.

Although a Web service security standard [13] exists, it does not provide a comprehensive solution to security issues faced by Web Services. WS-Trust [14], WS-SecurityPolicy [15], WS-SecureConversation [16], and WS-Federation [17] are additional security standards that augment the WS-Security specification. For instance, XML-signatures can be used to protect the integrity of messages exchanged between the client and the service, but if the message sender itself is malicious, it can insert malicious content within the messages using its own genuine XML-signature [18]. Also, WS-Security is not effective to prevent many attacks. For instance, despite the WS-Security standards, an attacker can launch XML injection, parameter tampering attacks [19], Denial-of-Service [18], and information disclosure attacks. All of these observations motivate us to investigate the security and privacy issues during Web service selection.

The following are some security and privacy threats concerned with content and service discovery over the Internet:

- During Web service or content discovery, a large amount of sensitive information is provided by the users to the service providers which host the content [20]. The sensitive information includes but is not limited to:
  - Search queries issued by the users looking for a specific content or a service
  - Personal information such as full name, phone numbers, social security numbers, credit card information, etc
  - Geographical information such as location
– Medical data
– Social circle data

• During Web service or content discovery, a service provider’s business information is leaked such as location of the business, business policies, trade secrets, business requirements, etc, [20]. This information might be leaked to the users that access the service as well as to the business competitors who coordinate with the service provider in case of composite services.

• Web services are prone to attacks and have vulnerabilities such as injection attacks, session replay, parser-based attacks and so forth [21–23].

• Distributed systems that host content or services such as peer-to-peer networks are prone to having malicious nodes that breach security and privacy of the systems in various ways [24–28].

Privacy of users using services on the web is at risk as noted by some works [29–31]. Minkus and Ross [29] note that on Ebay website, when seller leaves a feedback for the buyers, the buyers’ purchase history is disclosed to the public even when there is no action by the buyer. Also, they mention that the seller leaving a feedback for the buyers happens for 60-70% of the transactions. This implies that more than half the time, the privacy of the buyers’ information is at breach. The ability to collect user search information and associate it to real people is of use to several actors. For instance, user purchasing behavior is collected by advertisers and content publishers in order to present targeted ads [29]. Also, marketers could analyze the purchases that are bought together in order to target their products at specific segments [29]. Background check companies and insurance companies could include purchasing behavior in their classification methods [29]. Malicious parties could build detailed profiles on users in order to enable sophisticated spear phishing attempts [29]. In the study conducted by Minkus and Ross [29], they crawled 130,991 usernames on Ebay website. They were able to match around 17% of usernames with a potential actual profile on Facebook. Also, they were able to find very sensitive information about users in regards to their purchase history which is related to gun accessories, pregnancy tests, and H.I.V. tests, and were able to match with the users on Facebook with a match rate of 15.4%, 17.2%, and 16.7% respectively. Also, Minkus and Ross conducted a survey to obtain users’ privacy
expectations, and the results show that a large portion (38%) of users want their purchase information not to be viewed by anyone except themselves. Only a 9% of users are fine with exposing their information to the public. This is evidence that users are very concerned about the privacy of their information revealed over the web. Yet, they are unable to control their information disclosure when interacting with Web services. Service providers also get affected, specially financially, when there are privacy breaches with respect to the users’ data stored at their end. Privacy breaches have a significant and negative impact on the service providers’ market value [31]. After involuntarily allowing criminals to access over 163,000 consumer credit reports, Choicepoint was forced to pay a $15 million in penalties [31]. Also, companies have lost a lot with respect to stock market: ChoicePoint’s stock has declined in a few weeks after the incident from $46.01 to $37.64, and the Internet advertising company DoubleClick lost 20% of its market value after privacy breaches that are associated with its acquisition of Abacus Direct, an offline consumer data company which is specialized in collecting purchasing habits [31]. From the above observations, we find that it is very critical to address security and privacy issues of user’s information while accessing web services.

In this dissertation, we focus on security and privacy issues related to content and service discovery or selection with respect to two types of architectures: centralized and decentralized. Search for content or services on centralized architecture deals with indexed content/services, whereas search on decentralized architecture involves query-flooding throughout a content-hosting network.

1.0.1 Service Selection in Centralized Architecture

In the context of centralized architectures, we focus on Web services. Web services are majorly searched for through a central registry server that maintains information about services. Web services are software entities which provide a means of communication for web applications over the World Wide Web [32]. In specific, Web services offer software services to the clients which are Web services themselves or are software applications. For example, a Web service could be offering weather services by taking the location as input and producing temperature as the output. The output is consumed by a software application or a Web service. Being built on established XML-based standards and robust technologies [33,34],
the adoption of Web services offers potential for lower integration costs and greater flexibility and interoperability, and it gives the possibility to combine single Web services together into a more complex one [35]. That is, multiple services can merge together to provide a single service to a client [36]. For example, a Shopping Web service is formed by merging an Inventory Web service, a Payment Web service and a Shipping Web service. The growing success of Web services as the preferred standards-based approach to implement distributed and heterogeneous systems has resulted in a large number of providers offering services of varying degree of sophistication and complexity, ranging from weather forecast to travel management [37]. On one hand, the availability of a wide array of services has created a competitive and flexible market that meets the needs of different type of users (e.g. individuals, organizations, law enforcement etc.). On the other hand, this large availability of heterogeneous services makes the service selection process a non-trivial task, making Web service selection and provisioning play a crucial role in Web service life-cycle.

A byproduct of the pervasiveness of Web services, in addition to the selection problem mentioned above, are the unique security concerns that Web Services face. For instance, the Simple Object Access Protocol (SOAP) protocol used for communication in Web Services can bypass a firewall and can get processed by the Web Service directly [19]. SOAP messages can be easily exploited [19], and be overturned to the attacker’s gain. More generally, Web service vulnerabilities pose a threat to the privacy and security of the user’s as well as the Web service provider’s sensitive information. Web services have vulnerabilities such as password in clear, and confidentiality and integrity vulnerabilities which expose sensitive information of the user and the service providers to the attackers.

Users might be interested in obtaining Web services which meet important criteria such as privacy compliance and appropriate security guarantees. For example, users might be very interested in filtering Web services with inaccurate vendor implementations, or coding mistakes that could lead to exploitable security vulnerabilities. This points out the need of a recommendation system for Web services which is able to accommodate multiple users’ criteria on dimensions which are highly disparate. More relevant challenges come from the user’s perspective. In general, it is not straightforward for him/her to express preferences on the ideal services. To overcome this possible limitation, we believe that the WS
recommender system should support users in the specification of preferences/requirements at different granularity/level. As such, users have to be able to pose constraints on a given dimension, e.g., the degree of security level, as well as, to express specific conditions on capabilities to be met to ensure recommendations of reliable Web services, e.g., the desired encryption algorithm. Another important requirement for service recommendation arises from the fact that given the possibly large number and heterogeneous capabilities a user may consider, one may expect that for many user searches, only a subset of the user conditions will be fully satisfied. This is particularly the case with composite Web services, where each service in the composition needs to meet the criteria, rendering exact matching unlikely. To this end, the recommendation system should compute recommendations according to partially matched conditions. This would allow to recommend a Web service even if the required condition is not fully satisfied, but if a minimum level of satisfaction is reached.

Also, during the Web service selection phase, there is a huge amount of sensitive information exchange between the users and the service providers, and among the service providers themselves. This information exchange leads to privacy issues both at the users’ and the service providers’ end. Literature offers a great amount of work dealing with issues of constrained Web services selection. The most widely investigated type of constraints are related to QoS, where several solutions focusing on different aspects of finding the optimal service selection have been proposed (e.g., see [6–10] as examples). While all these efforts confirm the relevance and need of a constrained service selection approach, none of them takes into account the privacy of users and service providers.

In this dissertation, we study the implications related to such information disclosure. In addition to considering the privacy requirements associated with the disclosure of the user’s personal data needed to locate Web services according to his/her search criteria, we also take into account the fact that service providers might have to verify a user’s profile information against some provisioning rules, prior to service invocation. For example, a supplier has to verify if the buyer is not underage before selling alcoholic beverages to him. To this end, we propose a comprehensive framework to uniformly protect the privacy needs of the users and the service providers, at the time of service selection.
1.0.2 Content Discovery in Decentralized Architecture

In the context of decentralized architecture, we focus on content hosting networks or P2P networks that host content that can be searched by the users through search queries. We focus on content hosting networks because they are highly popular and also, they host large amounts of user data that could be possibly sensitive. Hence, addressing security issues in these networks is very essential. With the proliferation of distributed systems, web services and content are increasingly hosted on peer to peer networks (P2P) [38,39]. Peer-to-peer (P2P) networks are distributed systems in which nodes of equal roles and capabilities exchange information and services directly with each other [4]. Hosting Web services or content in P2P networks introduces a number of advantages, the most relevant being the greater availability of services and content as compared to centralized models. Indeed, the same service or content may be hosted on multiple peers, therefore also avoiding the risk of a single point of failure and performance bottleneck typical of the traditional Web services system [39].

In these environments, each peer hosts a set of services or content which can be invoked by other peers of the network using standard web service invocation protocols, or through a search query which is flooded throughout the network. In the case of pure P2P networks, the search query is transmitted over a number of random nodes, in search of content. However, a peer, due to security, privacy, company data policy, or functional compatibility reasons, might want its query to be forwarded only to nodes which conform to a set of constraints. Previous research [40,41] has noted the importance of policy-complaint forwarding in networks. Hoschek [41] mentions in his work that, for forwarding search queries to neighbors, neighbor selection queries or policies can be designed to select required neighbors to forward the query. However, this work does not deeply investigate the problem of policy-driven query routing. In this work, we explore this problem deeply. As an example, a company might want the data packets it receives to pass through multiple services, such as an accounting service and a packet-cleaning service [40]. Also, a company might want fine-grained control of traffic between different branch offices, respecting the business relationships of each branch office [40]. Or some providers might want to carry traffic only from friendly nations [40]. Also, P2P networks are prone to multiple privacy threats. Using computer networks al-
ways reveals some information to other nodes in the network.

We focus on search query routing issues in the context of peer-based Web services and content hosting networks, by enabling query traversal through policy-compliant nodes. In specific, we focus on search query traversal in unstructured P2P networks. Unstructured P2P networks are popular because they are robust and scalable [42]. Query forwarding techniques such as flooding and interest-based shortcuts cause large communication overhead and long delays in responses [42]. An unstructured P2P network has no global information available about the network, hence, an efficient and a low-cost query-routing approach is required for such networks [42]. Multiple P2P routing techniques have been proposed (e.g., [42–44]). For selecting the best neighboring nodes to forward the search query to, the approaches of all of these works just focus on the attributes or properties of the neighboring nodes, and do not focus on attributes or properties of other nodes in the network. Our work is unique in that we propose an approach which helps the search query forwarder to look into the properties of not just the neighboring nodes but also those of all other nodes in the neighboring branches up to a specified level. This helps in satisfying the policy associated with the search query to a higher degree, compared to existing techniques. Complementing the policy-driven approach, we also focus on policy-compliance verification of the nodes in the network, and on obtaining the authentic paths or routes taken by the search queries.

1.0.3 Contributions

We contribute a novel framework and approaches for secure Web service and content discovery in centralized and decentralized systems. Our overarching goal is to protect the data privacy of the users and the service providers, while looking into various security-related issues that could breach the privacy of data during service and content discovery. In specific, the following are our contributions:

- Firstly, our framework provides ranking and recommendation algorithms that allow the users to select the services according to their security specifications. The ranking and recommendation algorithms aid the user or the client to select the Web services which are most suitable to the user specified requirements. Recommendation of services, which is a filtering technique according
to the user specified requirements, is a good solution to the problem of efficient and effective selection of Web services. Some interesting solutions for Web services recommendations have been proposed so far [45–49]. However, several important problems have not been addressed yet. For example, although there have been works on composite service selection [50–52], these works are not suitable for composite service ranking as they consume large computation time for ranking. In addition, the few works that take into account user preferences, recommend atomic services and not composite services [47–49].

Second, our framework provides a private negotiator scheme that takes the ranked composite services as inputs and matches the search criteria of the user and the service descriptions of the ranked services in a private fashion. Also, the private negotiator matches the service provisioning rules against the provisioning rules of other service providers, and against the user profile, in a private fashion. The private negotiator helps in preserving the data privacy of the users and the service providers, by keeping the information exposure between the parties to a minimum, that is, only the information that is required to process the service is exposed. Information that is not required for service provisioning is not exposed, unlike the traditional service selection approaches where possibly a large amount of information of the parties is exposed amongst each other. We leverage existing private set intersection protocols and cryptographic protocols to design and implement efficient algorithms to secure search for services.

Third, our framework introduces a fully decentralized and interoperable content or service discovery approach that enables search query traversal through policy-compliant nodes in the context of content hosting networks or P2P networks. Policy-compliant search query routing will give the query owners control over their search query, by traversing only those nodes in the network that satisfy certain properties specified by the query owner, and not through random nodes which could possibly result in privacy breach of the query and the information that can be deduced from the query. Furthermore, we develop a solution to detect nodes that tamper with policy-compliant query routing protocols by forwarding the query to policy non-satisfying nodes. It
is a non-trivial challenge to address this problem due to a lack of trusted centralized server for search query monitoring. Our proposed scheme works by identifying the exact paths of search query propagation, and then by checking the policy satisfiability of all the nodes in the paths. Our scheme is secure, in terms of verifiability and non-repudiable search compliance. We also consider practical methods to further improve the efficiency of our approach, by proposing alternate methods to reduce the computational overhead with respect to policy-compliance verification.

1.0.4 Layout of this Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we present the related work. In Chapter 3, we discuss Web service vulnerabilities and present the results of our study. In Chapter 4 and Chapter 5, we present our framework for centralized systems, that is, Web service ranking and recommendation, and Web service private selection models respectively. In Chapter 6 and Chapter 7, we present our framework for decentralized systems, that is, policy-compliant routing of search queries and compliance verification of nodes in the network respectively. We present and discuss our experiments, complexity of our approach, and the results in Chapter 8. We conclude and discuss our future work in Chapter 9.
Chapter 2  |  Related Work

There has been a good amount of work in the areas of Web service recommendation, Web service selection, software vulnerabilities, and services and data hosted on P2P networks. In what follows, we discuss some of the works related to these research areas.

2.0.1 Work related to Web Service Vulnerabilities

Web service security standards [13] deal with implementing an end-to-end security solution between the sender and the receiver in a Service Oriented Architecture (SOA) system. The Web service security standards ensure confidentiality, non-repudiation and integrity of the messages exchanged between the services and the users by using digital signatures and message encryptions. The following are a set of additional security standards: WS-Trust, WS-SecurityPolicy, WS-SecureConversation, WS-Federation and WS-Authorization. Sidharth and Liu [19] present a set of SOAP-based attacks and explore the security vulnerabilities of SOAP messages, and also present a framework that can combat security threats to Web services. Some recent works [18, 53–55] have also proposed extensions to improve Web service security. Also, [19, 53, 56–58] have shown that Web Services are subjected to many other attacks, irrespective of the information stored in a WSDL document. However, the extent to which these works strengthen the security guarantees is not very clear.

Microsoft presents a set of vulnerability categories in [56, 57]. These vulnerabilities are based on the nature of the software error originating the vulnerability or the security breach. These classifications are general purpose and are not specifi-
cally related to Web services. Also, in [59,60], the authors propose taxonomies for generic software vulnerabilities. In spite of these taxonomies, we are not aware of any in-depth study of Web Services vulnerabilities.

2.0.2 Work related to Web service Recommendation

The literature offers a good amount of research work related to recommendation of Web services. We discuss some of them in this section. Sellami et al [48] propose an architecture that recommends service registries that have the highest probability to satisfy the user’s query. In recommending registries to the user, they take into consideration the user’s domains of interest, invocation history, and non-functional requirements. Similar to Sellami et al’s work, Liu et al [61] propose a Web service discovery approach by grouping similar services together and then dispatching the consumer requests to suitable services in terms of QoS requirements. Kalepu et al [49] use QoS criteria to predict the trustworthiness of the service providers by computing the difference between the QoS values portrayed by the services and the actual QoS values. These works, including [47–49], focus on selection of a single service and do not support recommendations for composite services. Composition based on QoS criteria has been studied by several authors, e.g. [46,52,62,63]. This body of work on QoS-aware selection of composite services inspired the design of WS-Rank. The goal of WS-Rank is complementary to the above works, in that it emphasizes the role of user criteria for recommendation purposes and it tackles the challenges of including criteria beyond QoS. Claro et al [64] propose a recommendation system for Web service compositions that utilizes the Gaussian Bayes Classifier to predict the combinations of services suitable to the user profile. Their approach is mainly based on user ratings, and requires the user to enter ratings for each type of service. Therefore recommendations are computed using different input than ours.

Zeng et al [52] present a middleware aimed at selecting Web services for composition such that user satisfaction is maximized when it is expressed as a utility over QoS functions. In contrast to Zeng’s work, we specifically consider how each of the user criteria affect the ranking of the service combinations. Also, in [50–52] the authors build a suitable composition by taking into consideration all possible candidate services for each activity in a work flow. In this dissertation, we assume
that all possible service combinations in the form of work flows are provided to our framework, and we identify and rank the best service combinations which when executed will lead to a composition. Finally, approaches for service selection based on fuzzy-logic have been recently proposed such as [65,66]. Almulla et al [66] recently developed a selection model based on fuzzy-logic. Similar to the work presented in this dissertation, the authors suggest non-exact matching techniques to rank available Web services. We differ from this body of work in several ways. First and foremost, we aim at providing a customized ranking, which strictly adheres to the user preferences. Therefore, we support user criteria of different granularity and on various types of criteria. Second, our model supports an expressive language for users to define user-criteria, which can be matching conditions, or high-level criteria. Third, we offer various strategies to accommodate users’ input. Finally, we deal with complex scenarios of service compositions, rather than single services.

2.0.3 Work related to Web service Selection

Work related to Web service security is associated with the following main topics: access control, privacy for Web services, Web service composition, and recommendation of services. We review relevant efforts in these areas in what follows. Access control in Web services focuses on authentication and authorization of the users or the service providers requesting a Web service, or processing a Web service in case of Web service composition. To date, a number of works providing solutions for access control in Web services have been proposed [67–71]. In what follows, we discuss few of the most representative approaches. Few works focus on attribute based access control in Web services [67,70,71] where the access control to resources is achieved based on the attributes of the subjects, resources and the environment under which the access takes place, and it is therefore well suited for Web services. For example, in [70], the authors posit that ABAC is a natural convergence of the following existing access control models: (1) discretionary access control (DAC) whose implementations include identity based access control, role based access control, (2) mandatory access control (MAC). They claim that ABAC is more fine-grained and the policy representation is richer and more expressive than the other access control models. Also, authors in [68] propose to extend WS-BPEL, so as to allow specification of user tasks and their enforcement
of access control of those tasks.

Parallel to work on access control, researchers have investigated how to address the client’s privacy concerns [69, 72, 73]. In [69], a framework focusing on clients’ privacy is proposed. The key component of their framework is the mobile agent which ensures accurate data privacy policy of the service provider to be implemented at the client. Similarly, authors in [72] describe an approach to prioritize the Web services for a client during selection depending on the privacy preferences specified by the client. Work to date has taken into consideration the privacy of the client’s information during the service provisioning phase and later, but do not consider the client’s information privacy during the service selection phase. Users’ search criteria information is as vulnerable to secondary usage as other data of the client stored at the service provider’s side. During the selection of Web services, the client needs to provide its search criteria to the UDDI registry entity or a search engine. To business critical clients this search information is very confidential. In this dissertation, we provide a solution to protect the search criteria of the client from disclosure to the service provider.

Web service composition is a related important and evolving paradigm, as it provides better services to the clients. Web service composition involves composition of different Web services on the fly, upon request from a client. Composition brings with it a new set of privacy problems to the service providers. During service composition, business related and sensitive information of the service providers needs to be exchanged among themselves in order to provide the composite service. On a first step towards this problem direction, authors in [74], consider the privacy problem that arises among the service providers during Web service composition. They analyze the problem of information flow during Web service composition, whereby service providers might derive some sensitive information during the composition process. Ustun and colleagues [75] provide an approach to manage the information flow between composed services in a decentralized manner using peer-to-peer processes. As in [74], they make use of the information flow policies and build algorithms to achieve the goal. Both these solutions fail to consider the privacy of these information flow policies from the other services participating in the Web service composition. In [76], the authors focus on controlling access to data resources from a client by making the composite Web service combine the access control policies of different Web services and construct its own set of access
control policies for the client requesting the composite service. However, the access control policies of each Web service are disclosed to the other services. While similar to the extent that we also deal with composing policies from multiple providers, we focus on privacy of access control policies of each service provider during Web service composition. In [77], the authors present a framework for Web service composition that is decentralized supporting and enforcing some security requirements like availability, confidentiality, authenticity and integrity of the execution order of the Web services. Carminati and colleagues [78] mainly provide an idea about how to compose Web services taking into consideration their security constraints and capabilities, and posit that the service composition should be made only after checking the compatibility of different Web service provider’s security constraints and capabilities. In this dissertation, we propose not only to consider the compatibility of security constraints and capabilities among the service providers, but we also consider various business policies and their confidentiality needs.

2.0.4 Work related to services and data hosted on P2P architectures

Hosting Web services on P2P architectures is a well established concept in the literature. Sanjeewa and Ranasinghe [79] develop a peer to peer Web service architecture based on mobile channels, using MoCha, an implementation of mobile channel concepts. Their architecture enables service discovery and invocation by allowing anonymous point to point communication and also mobility of the channel ends. Li et al., [39] propose a Web service discovery architecture based on P2P overlay network. Kashani et al., [80] also propose a decentralized Web services P2P discovery service with semantic-level matching capability. Amoretti et al., [81] propose an extension called JXTA-SOAP to the peer to peer JXTA technology to support peer-to-peer sharing of Web services which includes deploying, publishing, discovering, and invoking Web services in a network of JXTA peers. Papazoglou et al., [38] present an architectural approach and a formal framework to unify service oriented architectures and peer-to-peer-computing technologies.

In addition to static Web services, mobile Web services are also hosted on peer-to-peer architectures [82–85]. This body of work concentrates on the architecture level of the P2P networks, but do not consider the user requirements on the search
query itself when routing the query through the peer-to-peer network. In this work, we take into consideration the user requirements on the search query, which we define as a policy, and develop an efficient protocol to match the policy associated with a search query with the properties of a node forwarding the search query through the network. Also, we propose our solution for pure peer-to-peer networks where no centralized directory system is present.

Lin et al., [86] and Xilu et al., [87] propose a QoS-aware service discovery method in P2P networks. Unlike these works, we do not deal with QoS aspects of the services but present an efficient solution for search query traversal through policy complaint P2P nodes only. Zhou et al., [88] and Elenius and Ingmarsson [89] present interesting approaches for semantic-based service discovery in P2P networks. Our work is unique in that, in addition to discovering the needed service, we stress the importance of a policy-compliant routing of the search query in case of non-directory based P2P networks.

An interesting Web service discovery architecture and a peer-to-peer database framework is presented in Hoschek’s work [41] in which he abstractly mentions that for forwarding search queries to neighbors, neighbor selection queries or policies can be designed to select required neighbors to forward the query. However, this work does not deeply investigate the problem of policy driven query routing as it is done in our work. Finally, many P2P routing techniques have been proposed (e.g. [43], [42], [44]), with fewer introducing the idea of routing policies for Internet protocols [90, 91]. Although interesting, this body of work is different from ours, in scope, applications and techniques.

Trust establishment is a well-known challenge in distributed networks. Malicious nodes can abuse the data or the established search query forwarding protocols in a number of ways. To address these issues, a large body of work exists on secure distributed networks, tackling sybil attacks, denial of service, free riders and cheat detection [92–96]. In the context of content dissemination networks, researchers have focused primarily on the issues of denial of service attacks [93, 94, 97], privacy and security of the content propagated within the content centric networks [98], and sybil attacks [92]. Some recent work has also explored issues related to access control in ad-hoc networks [99]. In addition, issues related to secure search query propagation are very crucial to content dissemination networks as searching is the main purpose of content dissemination networks, and hence the protection
of search query propagation through the network is very important. In this work, we aim to tackle the security issues of search query propagation in distributed networks.

In this space, recent work has focused on efficient query processing. For instance, Durr et al., [100] analyzed different query forwarding strategies in privacy preserving social networks. Also, many have investigated intelligent query processing methods [101–104]. However, unlike in our work, intelligent processing methods do not consider the security aspects of the query forwarding process itself.

Zhang et al., [98] propose a mechanism to protect the confidentiality of data by encrypting them with identity based cryptography in content-centric networks. While it is sensible to utilize identity based cryptography to protect the confidentiality of the data propagated and selectively disseminate data, it is also important to detect malicious nodes in the network which propagate the data to false nodes. In this work, we employ efficient identity based signature schemes and attribute based encryption schemes to establish the integrity of the path taken by the search query and detect the malicious nodes that abuse the query forwarding algorithms. On a similar note, Padmanabhan and Simon [95], propose a mechanism to identify offending routers in a network and securely trace the path of the traffic. Their approach requires each node in a path to respond to the requester with an OK response that it received a packet. In contrast, we propose an efficient approach in which we use aggregated signatures to ensure the integrity of the path taken by a search query. Our protocol does not require a message from every node that the traffic or the query passes through. Mirzak et al., [96] also propose an approach to detect malicious routers, based only on the traffic information that each node has. Our approach is different from theirs in that it detects the malicious nodes mainly based on the attributes or properties of the nodes in the network, by making use of policies.

In summary, while several interesting works exist on policy compliance routing (e.g. [105,106]), we are not aware of any work on detection of malicious nodes that do not comply with the query forwarding protocol established for the network. Rather, previous works focus on policy specification, and assume that the nodes are honest. In this work, we detect malicious nodes that do not comply with such query routing protocols. Since search query forwarding is an important phase of the dissemination of content in content dissemination and peer to peer networks,
we aim to provide a solution to efficiently and effectively support verifiable query forwarding in these networks.

2.0.5 Summary

With respect to centralized systems, our contributions include a framework for privacy-aware service selection and recommendation. Compared to existing works related to service recommendations, we are unique in that we recommend services based on partial matching of attributes. As mentioned in Section 5.2, we consider not only the search information and attributes of the users, which has been the main focus of prior work, but also protect the privacy of service providers during service selection. Our contribution in this context, is to minimize the information exposed among the parties participating in service discovery, and to only expose the information which is actually required for service provisioning. With respect to decentralized system, we propose an efficient search query forwarding mechanism which unlike previous works, not only takes into consideration the attributes of the neighboring nodes while query forwarding, but also considers the attributes of other nodes in the network which are in the levels further down the neighboring nodes. In addition to policy-complaint routing in P2P networks, we also propose approaches to verify the policy-compliance of the nodes in the network, and hence detect malicious nodes.
Chapter 3  |  Web Service Vulnerabilities

In an effort to improve our understanding of Web-Services vulnerabilities, in this chapter, we introduce a novel client-based classification of Web-Service vulnerabilities. Our classification enables the client to know which vulnerabilities can be quickly prevented by the client as well as which ones can be prevented without revealing any information to the service provider or the attacker. Starting from our classification criteria, we discuss several Web Services vulnerabilities. We consider not only known Web-based vulnerabilities such as SQL injection, session replay etc, but we also analyze vulnerabilities born specifically as a result of poor Web-Service construction and service maintenance, such as lack of encryption, invalid XML, parser attacks, and log file attacks. In our analysis, we discuss remedies and impact for each of the identified vulnerabilities, and propose methods of detection based on real-time analysis of the WSDL document describing the exposed Web Service.

We also provide the results of a large scale study involving over 2,000 real-world Web Services. Our evaluation, carried out using a vulnerability detector prototyped by us, indicates that many of the least studied vulnerabilities are present in the wild such as Password in Clear and Invalid XML. We provide a discussion on possible solution mechanisms and countermeasures.

3.1 Web Services Standards and Security

We briefly overview some of the Web Services standards that are relevant for our study. Web Services Description Language (WSDL) is an XML format for describing Web Services. WSDL describes the structure of a specific service using
XML-formatted data. The information provided acts as an interface to the service, that is, the information includes service name and location, method names, argument types, return values, and types. Through a WSDL description, a client application can determine explicit instructions on how to communicate with previously private applications, as well as determine the operations that are available to the consumer that he or she can invoke. WSDL files are typically stored in Web registries that can be searched by potential clients to locate Web Service implementations of desired capabilities. Due to exposed detailed methods and location information, several attacks can be crafted based upon vulnerabilities leveraging information in WSDL files. Essentially, the attacker, by analyzing the WSDL, is provided with critical information about various methods and parameters needed for the attack. These set of vulnerabilities are sometimes referred to as WSDL vulnerabilities. We note that, although some recent work has already acknowledged that Web Services undergo WSDL threats [53,107], we are not aware of any study of the impact of such vulnerabilities. Further, previous work has shown that Web Services are subjected to many other attacks, irrespective of the information stored in WSDL [19,53,56–58]. For instance, in Web applications, a client communicates with the application through a Web browser whereas in Web Services, the client directly interacts with the service. Hence, the Web Services are more vulnerable to attacks than traditional Web applications due to the absence of a browser in the middle of communication.

In order to cope with these issues, Web Service security standards [13] are nowadays widely used to implement an end-to-end security solution between the sender and receiver in a SOA system. Digital signatures and message encryption are used within the WS-Security standard to ensure the confidentiality, non-repudiation, and integrity of the messages. WS-Trust, WS-SecurityPolicy, WS-SecureConversation, WS-Federation, and WS-Authorization protocols are additional security standards that augment the WS-Security specification. WS-Security standards, however, cannot provide a comprehensive security solution to Web Services. For instance, XML-signatures can be used to protect the integrity of messages exchanged between the client and the service, but if the message sender itself is malicious, it can insert malicious content within the messages using its own genuine XML-signature [18]. Also, WS-Security is not effective to prevent many attacks. For instance, despite the WS-Security standards, an attacker can launch XML injection,
parameter tampering attacks [19], Denial-of-Service [18], and information disclosure attacks. In light of these observations, we argue that security measures such as input validation and careful coding of Web Services are very important and complement WS-Security standards. These observations motivate us to undertake this study to find out how widely the Web Service vulnerabilities are spread out in the huge set of openly available Web Services, which would harm the service providers and the users, while provisioning Web services and selecting services respectively.

### 3.2 A taxonomy of Web-Service Vulnerabilities

We introduce a new taxonomy for Web Services vulnerabilities, shown in Figure 3.1, and discuss few representative vulnerabilities for each introduced element in the taxonomy.

#### 3.2.1 A Client-target Taxonomy

The design of a well-formed, non-ambiguous software vulnerability taxonomy is a non-trivial task, which has been attempted by various researchers [59,60]. In order to develop a sound taxonomy, a main challenge is to identify unambiguous, orthogonal classification criteria for a set of objects [108]. Defining unambiguous
classification criteria for software vulnerabilities is especially known to be non-trivial [109]. In my thesis, our main classification criteria is based on the method of vulnerability detection and prevention. Specifically, vulnerabilities can be discovered in two fundamentally different ways, that is, (1) by the detection method and (2) by checking if the client can itself prevent the attack related to the vulnerability. We define simple and well-defined requirements for vulnerabilities which result in a practical categorization of various well-known and relatively new security issues specific to Web-Services, so as to better drive the design of protection mechanisms from possible exploits. Accordingly, we define two mutually exclusive categories of vulnerabilities: Static and Dynamic vulnerabilities. Static vulnerabilities can be detected without the execution of a service whereas dynamic vulnerabilities can be detected upon execution. In order to detect dynamic vulnerabilities, the client needs to receive a response from the service, whereas, for the detection of static vulnerabilities, there is no need of any feedback or response from the service. To detect a static vulnerability, the client can utilize the available resources belonging to the service (e.g., WSDL) and the client does not need feedback or response from the service. Each of these two classes of vulnerabilities are further classified into Changeable and Unchangeable. Changeable vulnerabilities can be prevented by the client and the service on the fly when a client makes a call to the service. That is, the client modifies its original input or adds to its original input in order to prevent vulnerabilities. For example, a client protects its password by encrypting it with a key when the client finds that the service does not provide any mechanism for protecting the password. Unchangeable vulnerabilities can only be prevented if the service undergoes some architectural and structural change. That is, the service provider needs to modify the service in order to prevent vulnerabilities which cannot be done dynamically when a client calls the service. For example, if the service is prone to generate an error on the interface on execution, it cannot be prevented either by the client or the provider during the time of service execution by the client. Since the client does not have any control over preventing such vulnerabilities, these are named Unchangeable vulnerabilities.

In Figure 3.1, we report examples of vulnerabilities classified under our taxonomy. We note that our taxonomy is meant to be orthogonal to existing classifications. With the help of our classification, a client is able to know if he has any control to prevent attacks due to some of the vulnerabilities (i.e., changeable)
even before making a call to the service. By classifying vulnerabilities into static and dynamic, the client knows which vulnerabilities can be detected without revealing any of its information (i.e., static), which is the case to detect dynamic vulnerabilities.

Here, we provide a novel client-centered perspective on the vulnerabilities Web Services currently face. Precisely, our approach is to take the viewpoint of a client to help it quickly detect and cope with the vulnerability being observed. The client-targeted classification has the following advantages: 1. The client knows which vulnerabilities can be prevented by the client (i.e., changeable), 2. The client knows which vulnerabilities can be *quickly* detected (i.e., static), and 3. The client knows which vulnerabilities can be prevented without revealing any information to the service provider or the attacker (i.e., static). This viewpoint addresses a known shortcoming of previously proposed taxonomies, and of similar classification approaches based on software vulnerabilities [59,60] which do not make their intended usage explicit.

For instance, a well known approach to classification of vulnerabilities [56,57], is based on the nature of the software error originating the vulnerability or the security breach. These classifications [56,57] are interesting but general purpose. Furthermore, although very extensive, using the software error as a criteria may result in some degree of ambiguity, in that some vulnerabilities may arguably fall into two different categories. For example, if the criterion for classification is security breach, the well-known *SQL injection* vulnerability could fall into both *Tampering* and *Information Disclosure* vulnerability categories, since SQL exploits may result in tampering of the Web Service as well as lead to unwanted information leakage. Also, we note that works on predict the occurrence of vulnerabilities exists [109]. However, we here do not aim at predictive methods, but rather at designing strong taxonomies.

### 3.2.2 Vulnerabilities

Next, we discuss some vulnerabilities for each of the classes we have identified. We select vulnerabilities covering a broad and diverse set of representative software and architectural problems. Our discussion focuses on Web Service-specific vulnerabilities, whose corresponding attacks mainly exploit information stored in WSDL files.
Our empirical evaluation demonstrates that most of the WSDL vulnerabilities are often under looked, and yet exist in the wild and are poorly protected.

We note that our discussion focuses on specific instances of selected vulnerabilities. Multiple variations for every discussed vulnerability may exist, which however have the same inherent nature and similar exploit methods. For instance, for the parsing vulnerability (V2 in the description below), there exist different variations such as XML Bomb, huge file size, SOAP array attack vulnerabilities, etc. In all of these instances, the basic idea is the same: the XML file is modified to make the processing time of the file huge.

### 3.2.2.1 Static Vulnerabilities

We now present examples of static vulnerabilities. We discuss several vulnerabilities that are WSDL-related (V1-V4), and provide an example of a non-WSDL-related vulnerability (V5).

**V1. Password in Clear**

**Purpose**: A service requires a password from its client to authenticate the client. A Web Service requests the client for a user name and password by creating an authentication method among its other methods related to the service. Just like any other method of the service, the interface of an authentication method is published in the WSDL. The client makes a call to this method by providing his user name and password, after which the service authenticates the user. The following is a code snippet showing a login method in terms of an operation.

```xml
<message name="LoginInput">
  <part name="body" element="xsd1:LoginRequest"/>
</message>
<message name="LoginOutput">
  <part name="body" element="xsd1:LoginResponse"/>
</message>
<portType name="LoginPortType">
  <operation name="Login">
    <input message="tns:LoginInput"/>
    <output message="tns:LoginOutput"/>
  </operation>
</portType>
```
Vulnerability: A service is said to have a *Password in Clear* vulnerability when it does not use password encryption methods to protect the password at the message level. Even though the client and the service employ a transport level security protocol such as TLS or SSL, the password is still at a message-level threat. Message-level security is very important in cases where there exist intermediate nodes receiving the client’s message or request. For instance, consider a service requesting the client for its password. The client sends the password in clear as an argument to the authentication method. However, the client method call might pass through multiple intermediate nodes or servers which are not necessarily authorized to access the client’s password. In this case, even though the client method call is encrypted using a transport layer protocol, after receiving the client call, the intermediate nodes can access the client’s password.

Remedy: The *Password in Clear* vulnerability can be resolved by defining policies in the WSDL of a service. WS-SecurityPolicy standard along with the WS-Policy standard can be used to define security policies within the WSDL to request the client to encrypt its password. This vulnerability is classified as *Changeable* as the client can encrypt its password with the service’s public key if it exists even when there is no policy defined on the password. The following is an example of an encryption policy defined in a WSDL file asking the client to encrypt its input message, in which case, the client encrypts its password.

```xml
<wsp:Policy wsu:Id="InputIdentifier">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:EncryptedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

V2. Invalid Parser
**Purpose:** A parser is required at the service end in order to parse the service requests of the client.

**Vulnerability:** A service is said to have a *Parser* vulnerability when proper validation techniques are not in place at the service end during parsing of the service requests. Two types of parsers are commonly used in the context of Web Services: DOM-based parser and SAX-based parser [104]. DOM-based parser is prone to denial of service attacks. This is mainly because DOM-based parser places whole of the XML request data in the memory for parsing. One such dangerous attack is the *XML Bomb* attack, where an attacker writes an XML file with huge number of nested elements or entities. Due to this attack, the parser allocates large memory and it is stuck indefinitely parsing the huge number of elements, leading to denial of service to other clients requesting the service. Also, the following attacks can take place: inputting large number of files for parsing, malformed XML (e.g., unclosed tags), malicious attachment, soap array attack (huge number of XML elements), and large XML document size. SAX-based parser is more prone to XML injection attacks, where an attacker inputs data to the service which can query data in unauthorized mode. Though both DOM and SAX based parsers are vulnerable to denial of service and XML injection attacks, the above attacks are more critical to each of the parsers. This vulnerability can be detected at the service provider end. Though *Parser* vulnerability is not explicitly dependent on the WSDL, it is considered to be partially dependent on the WSDL as the attacker can carefully frame the XML input which is logically correct according to the schema defined or referred to in the WSDL, but which is a malicious one.

**Remedy:** The *Denial of Service* attack in terms of *XML Bomb* attack can be resolved by validating the size of input stream when an XML request arrives at the service end. The *XML Injection* attack can be resolved by properly validating the input from the client by defining a proper XML Schema. This vulnerability is classified as *Unchangeable* as the code and the type of parsers cannot be changed during the client’s call to the service.

**V3. Invalid XML**

**Purpose:** Validation of an XML file is needed in order to prevent attacks that submit an XML file with malicious content or XML file with wrong data types.

**Vulnerability:** A service is said to have an *Invalid XML* vulnerability when the
schema related to the service is defined within the WSDL file of the service. The following is an example of a schema defined within the WSDL (the other parts of the file such as messages, port types, etc, are not shown):

```xml
<wsdl:types>
  <s:schema elementFormDefault="qualified"
    targetNamespace="http://test.org/">
    <s:element name="WeatherRequest">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
            name="City" type="s:string"/>
          <s:element minOccurs="0" maxOccurs="1"
            name="Zipcode" type="s:int"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="WeatherResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
            name="Resp" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
```

Exploiting this vulnerability, an attacker can modify the actual schema with a different schema and replace it in the WSDL file. This attack is called *Invalid XML or Schema Poisoning* attack, wherein when the client accesses the WSDL file, it is prompted with an *Invalid XML* error due to which the client would not be able to use the WSDL file to make function calls to the service.

For instance, in the above schema, the attacker might modify the above inline schema by replacing the `string` type with an `int` type. Hence, the client is forced
to provide an integer instead of a string due to which the client would get an Invalid XML error every time it calls the service.

**Remedy:** The Invalid XML vulnerability can be resolved by defining the schema outside of the WSDL file. The attacker will not have access to the schema for modification. This vulnerability is classified as Unchangeable as the service provider defines the schema beforehand in the WSDL and cannot be changed when the client makes a call to the service.

### V4. Confidentiality and Integrity

**Purpose:** A client needs to protect its data during transit from unauthorized sniffing and modification from attackers if the data is important and sensitive.

**Vulnerability:** A service is said to have a Confidentiality and Integrity vulnerability when it does not employ any kind of encryption or integrity techniques over its input and output data. When the input data sent to the methods by the client and when the output data from the service are not encrypted, there is a breach to the confidentiality of the client’s data in transit similar to the Password in Clear vulnerability. However, Password in Clear vulnerability is present only in the operations element of the WSDL file and deals only with the password, whereas Confidentiality and Integrity vulnerability is concerned to any data present within any element within the WSDL file. Also, the client’s data in transit to the service might be altered by attackers in which case the integrity of the client’s data is lost.

This vulnerability can be detected by analyzing the WSDL file of the service by checking if there are any security policies defined in the WSDL file regarding the encryption and integrity of the messages or data.

**Remedy:** The Confidentiality vulnerability for input data can be resolved by requiring the client to encrypt its data during transit. This requirement can be achieved by defining XML encryption policies using WS-SecurityPolicy standard along with the WS-Policy standard within the WSDL. For confidentiality of output data, the service needs to employ the encryption methods too. The Integrity vulnerability can be resolved by requiring the client to use digital signatures over its input data. The service can achieve this by defining XML digital signature policies using WS-SecurityPolicy standard along with the WS-Policy standard within the WSDL. This vulnerability is classified as Changeable as the client, before making a call to the service, can encrypt the message using the public key of the service.
to ensure confidentiality, and the client can sign the message using its private key to ensure integrity.

V5. Logging

Several static vulnerabilities that are not WSDL-related exist. For instance, vulnerabilities such as the Logging vulnerability and Services in Public Business Registries vulnerability arise irrespective of WSDL data.

**Purpose:** A service provider needs to log or store critical activities that take place with respect to the service for security reasons. For instance, a service logs the login time and login username. A service also needs to audit the stored log files for security reasons. When a failed login occurs, by auditing the log file, the service provider can investigate about a probable attack.

**Vulnerability:** A service is said to have a logging vulnerability when it is prone to log injection. That is, the service logging mechanism does not properly validate the input of the service. In a log injection attack, an attacker injects his own phrase into the service provider’s log file. The attacker injects a targeted phrase in the input he provides to the service. For instance, consider a service taking a user name of the user as one of its inputs. The following is a normal log file generated after a genuine user User1 invokes the service.

**Successful login attempt for User1.**


Failed login attempt for User2.
Failed login attempt for User3.
Failed login attempt for User3.
Failed login attempt for User3.
Failed login attempt for User3.
<table>
<thead>
<tr>
<th>Type of Service</th>
<th>Authenticating Services</th>
<th>Non-Authenticating Services</th>
<th>% of Services Authenticating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No of Services</td>
<td>Avg. no of Operations</td>
<td>No of Services</td>
</tr>
<tr>
<td>Business</td>
<td>38</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>Location</td>
<td>41</td>
<td>9</td>
<td>71</td>
</tr>
<tr>
<td>Comm/Etmnt</td>
<td>29</td>
<td>15</td>
<td>36</td>
</tr>
<tr>
<td>Sci/Sec</td>
<td>15</td>
<td>3</td>
<td>695</td>
</tr>
<tr>
<td>Search</td>
<td>5</td>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>Other</td>
<td>224</td>
<td>15</td>
<td>1115</td>
</tr>
<tr>
<td>Total</td>
<td>352</td>
<td>66</td>
<td>1994</td>
</tr>
</tbody>
</table>

Table 3.1: Authenticating and Non-Authenticating Services

Since the service logging mechanism does not properly validate the input of the service, especially, the linebreak in the example above, the attacker is able to inject into the log file. User3 is mistakenly suspected for his login behavior due to this logging vulnerability.

**Remedy**: The *Logging* vulnerability can be resolved by properly validating the inputs provided by the user. The logging validation mechanism by avoiding meta-characters such as linebreak, separators, etc. This vulnerability is classified as *Unchangeable* as the log files are at the service provider’s end.

### 3.2.2.2 Dynamic Vulnerabilities

*Dynamic Vulnerabilities* (i.e. can only be detected upon the execution of a service) are widely present in generic Web applications and have been studied [21,110] more in depth than the *Static Vulnerabilities*. We now provide a high level review of the most seen ones in Web services.

**V6. Error on Interface**

**Purpose**: Errors are used to let the developer know to alter the code if there is a bug in the code, and to let the user of the service know to alter his or her input
to the service.

**Vulnerability:** A Web Service is said to have an *Error on Interface* vulnerability when it throws an error on the client’s user interface or browser when the client makes a call to the service which might reveal to an attacker the internal details of the service such as their secret directory information, database dumps, and stack traces. For example, the following code of a service reveals the path information to the service invoker:

```c
char* path = getenv("PATH");
...
sprintf(stderr, " No file found on path %s n", path);
```

**Remedy:** The *Error on Interface* vulnerability can be avoided by handling error messages in the service code. This vulnerability is classified as *Unchangeable* as the service provider cannot change the service code while the client makes a call to the service.

**V7. SQL and XPath Injection**

**Purpose:** The inputs of the clients are transformed into SQL or XPath queries. The queries are used to query the SQL and XPath databases of the services.

**Vulnerability:** A Web Service is said to have an *SQL or XPath Injection* vulnerability when an attacker can input hidden queries in his or her XML requests to retrieve data from the database of the service. Similar to SQL injection attack, XPath injection attack takes place when the service is using XML documents to store user data instead of an SQL database.

**Remedy:** The *SQL and XPath Injection* can be avoided by defining an XML schema that carefully validates all possible types of inputs from the user. This vulnerability is *Unchangeable*.

**V8. Session Replay**

**Purpose:** Session is maintained between a client and a service, so that the client need not repeat providing the same data to the service for consecutive method calls. For instance, in order to authenticate a client, the service requires the user name and password from the client every time it calls the service. Hence, by maintaining a session with the client, the service need not authenticate the client multiple times.
<table>
<thead>
<tr>
<th>Type of Service</th>
<th>Total Services</th>
<th>Pwd in Clear</th>
<th>Conf and Int</th>
<th>Invalid XML</th>
<th>Total Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business</td>
<td>65</td>
<td>37</td>
<td>64</td>
<td>63 (96.92%)</td>
<td>164</td>
</tr>
<tr>
<td>Location</td>
<td>112</td>
<td>41</td>
<td>112</td>
<td>110 (98.21%)</td>
<td>263</td>
</tr>
<tr>
<td>Comm/Etmnt</td>
<td>65</td>
<td>29</td>
<td>65</td>
<td>61 (93.8%)</td>
<td>131</td>
</tr>
<tr>
<td>Sci/Sec</td>
<td>710</td>
<td>15</td>
<td>710</td>
<td>686 (96.62%)</td>
<td>1411</td>
</tr>
<tr>
<td>Search</td>
<td>55</td>
<td>5</td>
<td>55</td>
<td>52 (94.54%)</td>
<td>112</td>
</tr>
<tr>
<td>Other</td>
<td>1339</td>
<td>224</td>
<td>1339</td>
<td>1065 (79.53%)</td>
<td>2748</td>
</tr>
<tr>
<td>Total</td>
<td>2346</td>
<td>351</td>
<td>2345</td>
<td>2157 (91.94%)</td>
<td>4853</td>
</tr>
</tbody>
</table>

Table 3.2: Static Vulnerabilities by Service Type

**Vulnerability:** A service is said to have a *Session Replay* vulnerability when it maintains sessions through session IDs. An attacker can get hold of the session ID and reuse it to gain unauthorized control of the session of an authorized client by sending a request to the service using the session ID.

**Remedy:** *Session Replay* vulnerability can be resolved by the service and the client using a nonce during communication, with the client, that involves session ID. This vulnerability is classified as *Changeable* as the client can send a nonce along with the session ID when it makes a call to the service.

### 3.3 Empirical Evaluation

To assess the extent to which Web Services vulnerabilities are an actual problem in today’s Internet, we tested 2346 real Web Services. These services are taken from the Web Service collection of Al-Masri and Mahmoud [111], who obtained them from UDDI Business Registries and the World Wide Web. In total, the authors [111] collected 2507 services. Out of this dataset, we disregard 160 services which generated parser errors. The parser was unable to find the schema files located in different locations, that is, outside of the WSDL file.
3.3.1 Methodology

We developed a WSDL parser in order to efficiently detect static vulnerabilities. Each WSDL file belonging to the 2507 services is associated with a service. The parser reads all the WSDL files in a loop and processes each WSDL. It specifically detects static vulnerabilities by looking into certain elements and information within the WSDL. It first looks for the service name by parsing the file utilizing a `Definition` object. Then, based on the service keywords (extracted from the WSDL and compared with Wordnet dictionary to ensure semantic relevance with the category), the parser classifies the service into a type. The parser compares the service name with a set of words which are commonly used for a category. For instance, to detect if the service falls into business category, the parser compares the service name with the following words: `order`, `business`, `price`, `purchase`, `rate`, `quote`, `accounting`, `stock`, `tax`, `market`, `finance`, etc. More details on the type-based organization of the dataset used for our tests are provided in the next section.

Next, it checks for the elements defining the vulnerabilities. For instance, for the `Password in Clear` vulnerability, our parser looks for the term `Password` in the WSDL document within the `operations` elements. The parser also looks if there are any policies defined by looking if any of the `wsp` or `w sap` elements (see Section 3.2.2.1, V1) exist in the WSDL file. If both these cases are true, then the parser warns of a `Password in Clear` vulnerability. Similarly, the parser looks for the presence of `wsp` or `w sap` elements in the WSDL for detecting the `Confidentiality and Integrity` vulnerability. However, for detecting `Password in Clear` vulnerability, the parser, in addition to checking for the presence of policies in the WSDL file, looks for authentication methods, and within them looks for the argument "Password", whereas, for detecting `Confidentiality and Integrity` vulnerability, the parser just checks if any policies are defined within the WSDL file. For both of these vulnerabilities, checking for a `wsp` or `w sap` element enables us to know whether usage of encryption or integrity technologies are mandated by the service.

In order to detect whether `Invalid XML` vulnerability exists, the parser looks for the presence of a schema definition element within the `types` element in the WSDL file. If the schema is defined in the WSDL file, the parser concludes the presence of the `Invalid XML` vulnerability. The `Parser and Logging and Auditing` vulnerabilities cannot be detected by us as the parser type information is available.
at the service end, and the log and audit information is also available at the service provider. However, if the client is able to negotiate about this information with the service, then these vulnerabilities can be identified.

To test the dynamic vulnerabilities, we utilized a commercial Web Service vulnerability scanner called Acunetix Web vulnerability scanner [112]. We chose this scanner as it is well known for Web Services vulnerability detection [113] and can detect the most popular Web-based vulnerabilities (including Error on Interface, and SQL and XPath Injection) in a reliable fashion.

3.3.2 Results

As Web services can be of disparate types and can have varying degree of complexity, we organized the WSDL files of the dataset in different groups, based on the Web Service’s types. We anticipate that Web services belonging to the same type will have a very similar set of functionality and corresponding architecture, and therefore may be prone to a similar set of vulnerabilities.

We classified the services into 7 types based on their provisioned service: Business (eg., quote retrieval), Location (eg., weather), Communication and Entertainment (eg., email, travel, holiday), Scientific/Security (eg., gene variations, encryption), Search (eg., search for university data), and Others. These categories reflect the common types of Web services exposed in public registries.

In Table 3.1, we show the average number of operations for each service type. As reported, Scientific/Security services expose a large number of operations, followed by Communication and Other service types. We note that services in the Business category have higher percent of authenticating services, followed by Communication services. In the table, we further distinguish among authenticating (e.g. request a password to access) and non-authenticating services. We observe that the authenticating services have an average of 9.4 operations or functions in their services whereas the non-authenticating services have an average of 8.8 operations. Intuitively, this is because authenticating services are more complex than the non-authenticating services and usually offer complex and possibly sensitive operations.

Table 3.2 shows the breakdown of number of vulnerabilities detected by service type. As can be quickly observed, there is an extremely large number of vulnera-
bilities identified per service type. Any vulnerability present in each service occurs at least once.

First, we note that non-authenticating services have more static vulnerabilities (96.4%) compared to authenticating services (94.5%). Non-authenticating services are typically services with simpler communication protocols, that require limited interaction and storage of sensitive data with end users. Hence, they probably do not have a secure architecture compared to authenticating services.

Further, as shown in Table 3.2, almost all (96.6%) of the tested services do not specify any policies and hence they suffer from Confidentiality and Integrity vulnerabilities. A similar result is obtained for the Password in Clear vulnerability which is widely present among the services that request a password from the client. We find that all the authenticating services, irrespective of the service type have a Password in Clear vulnerability. We reason about the absence of message level confidentiality and integrity protection by the services as follows: 1. Confidentiality and Integrity of data can be achieved at the transport layer level with the use of HTTPS, and 2. The tested services are publicly available and are not related to internal processes of a business and hence, might not have maintained message-level confidentiality and integrity of services. Message-level security is very important when the client message path includes multiple other applications or services which might be connected with different, possibly non-secure transport protocols. In such a case, transport level security is not sufficient to protect the messages. Hence, we consider addressing the Confidentiality and Integrity and Password in Clear vulnerabilities very critical in the Web Service life-cycle.

Also, we note that the large majority of the services define their schema within the WSDL itself. As discussed in Section 3.2.2.1, exposing the schema leads to Invalid XML vulnerability. We note that this vulnerability is mostly observed in Location Services and Search Web-services. The services from the Other category are least vulnerable (79.5 %) to Invalid XML attacks when compared to the remaining service types. This is quite unexpected, and probably identifies that there is a great variety of Web-services which are more carefully architected than others.

In summary, from the above, we see that static vulnerabilities are alarmingly common in publicly exposed Web Services, with Confidentiality and Integrity vulnerability being the most common. Confidentiality and integrity is considered a most important vulnerability, as previous research [114] has shown that in spite
of the presence of cryptographic protocols such as SSL/TLS, breaches of confidentiality and integrity of data are likely.

In regard to dynamic vulnerabilities, we randomly selected 300 services and checked whether they had any dynamic vulnerabilities. We selected proportionally equal number of services from all service types. We specifically tested for Error on Interface and SQL Injection vulnerabilities as these are critical vulnerabilities that could dramatically affect Web Services, as discussed in Section 3.2.2.2. We found very few vulnerabilities. Interestingly, the tested Web Services have either 0 or multiple vulnerabilities. Precisely, 7% of the services have at least one vulnerability (i.e. 6 services), but all of such services have multiple occurrences of the same dynamic vulnerabilities (14 each, on average). We speculate that these are instances of poorly maintained services, and therefore, when the dynamic vulnerabilities exist in a service, they affect multiple service operations leading to major security holes.
Chapter 4  
Web Service Ranking and Recommendation

In this chapter, we present our approaches for recommending and ranking Web services according to user-specified criteria. We recall that we focus on two types of service and content systems: centralized and decentralized. Before we describe our ranking and recommendation strategy in depth, we present our overall architecture with respect to centralized systems. In what follows, we explain our centralized architecture.

4.0.1 Architecture

With respect to centralized systems, the goal of the proposed framework is to assist users to privately search for services able to carry out the required business, as well as to privately enforce possible provisioning rules of the selected service providers. More precisely, according to the user input, the framework offers a recommendation of list of Web services, which is ranked based on how much the selected services satisfy the user’s criteria. It also enables the user to privately process keywords search in the ranked list, and to privately evaluate on users profile the provisioning rules of providers. For this purpose, the proposed architecture has two main components: user criteria-oriented service Ranker (Ranker, in short) and Private Negotiator (PN, in short). The Ranker recommends ranked services to the user according to his or her criteria. The Private Negotiator instead enables private search in the ranked list, and also provides a private compatibility verification method that takes place among service providers, and between the service provider
Figure 4.1: Overall Workflow and Architecture of the Selection and Provisioning Framework. A User provides search terms to our ranker component which ranks and recommends services to the user. Our private negotiator (PN) component compares the user and the service provider requirements, and also the service provider provisioning rules in a private fashion, finally leading to the user choosing a service and executing it.

and the user. As such, the user search for the services is carried out in two distinct steps. The first step involves the user providing the Ranker non-private information on the service he is looking for (i.e., public search), whereas the second step involves the user providing the Private Negotiator, the private information, such as location or implementation requirements, for the private search and provisioning rule enforcement. We discuss these two steps in more detail in what follows.

**Ranker:** As the first step, the user enters information regarding the required service, that is, the description, the business type, its input and output (Figure 4.1, step 1). For example, the user provides “Travel Service” as description of the service, “airline, hotel, and payment services’ as the types of services, “input:date’ and “output:cost” as the input and output of the service. The user also enters a set of user criteria, that the resulting service has to satisfy. As described in Section 4.1, the proposed framework supports the specification of criteria of sev-
eral kinds. For example, the user could specify “time of execution < 2 hours” and “maximize overall QoS”. For each criteria, the user can express one or multiple fine-grained conditions. The user can also indicate how user criteria have to be taken into account during the multi-criteria recommendation process. More precisely, he can choose to prioritize a specific criteria, and also indicate how these criteria should be combined together during recommendation, that is, the strategies for criteria combination. We support three distinct strategies. The first strategy is preferred when the user enters priorities for a number of criteria of various kinds (e.g., QoS, privacy and security requirements, etc). The second strategy attempts to maximize the satisfiability of one criteria over the others, and expresses recommendations based on decreasing satisfaction of the criteria of lower priorities. The third strategy is preferred when a specific minimal set of requirements is indicated on global criteria, that is, criteria applying to the resulting composite Web service, on top of a set of constraints over local criteria, which apply to each of the Web services involved in the composition. According to the user request, the Ranker is given a set of all possible execution workflows by a planner module [115]. The planner module performs a search on the UDDI registry for the service requested by the user (step 2). After the search, the planner module prepares a list of composite Web service combinations that suit the user request, and sends these to the Ranker.\footnote{Each service is assumed to be annotated with capabilities (introduced in Sect. 4.1).} The Ranker ranks the service combinations, and sends the ranked list to the PN (step 3). Based on the workflows provided by the planner module, the Ranker constructs a directed acyclic graph (DAG) representing all possible service combinations for the required composition. After constructing a DAG, the Ranker executes the multi-criteria recommendation algorithms. The graph is defined to have satisfaction levels as weights which represent the amount of satisfaction of the user criteria by each candidate service. By using these weights, and a strategy selected by the user, the recommender algorithm is executed so as to find the best service compositions, that is, the compositions among those represented in DAG that maximize the satisfaction of the user’s preference. Precisely, the algorithm returns a ranked list of suitable service compositions (step 5), where the ones that better suit the user’s need are ranked higher.

**Private Negotiator:** The Private Negotiator (PN) enables the user to pri-
vately search for services in the ranked list, and the service provider and the user
to privately evaluate their provisioning rules and profiles in a private manner. It
also enables the same among the service providers in case of a composite service.
A protocol has been introduced by Malkhi in [116], to allow two parties to learn
in a private fashion the set of common attributes. Each of the two parties holds a
database, and wishes to jointly calculate the intersection of their inputs, without
leaking any additional information. The exchange is asymmetric, so that only the
party initiating the PM algorithm learns the result of the matching.

The PM protocol is activated when the user submits to PN a private search
request to be evaluated on the ranked list of services returned by the Ranker.
The user typically provides private information to PN as the search criteria. For
example, the user could search for "services processing Visa credit cards" and
"services that take input only from US citizens".

Upon receiving such type of request, the PN coordinates a PM protocol between
the user and the service provider on the ranked list of services, so as to evaluate
private search criteria conditions against the profiles of the services in a private
fashion. As further elaborated in Section 5.2.1, executing the two-party protocol
allows users to identify a suitable service combination without having to reveal in
clear any information related to their private search criteria. After evaluating the
private search criteria, a second ranked list of service compositions is provided to
the user, and the user selects one composition among these.
The service combination selected by the user is then further processed by the PN
so as to evaluate whether the associated provisioning rules of the services of the
composition can be satisfied, according to the user profile. In the case of a service
selection offered by a single provider, the provisioning rules are enforced using a
similar approach as the one used for private search criteria. That is, a two-party
private intersection protocol between the service provider provisioning rules and
user profile data is carried out (rather than service descriptions, as in the case of
user search criteria). However, in case the user requires a composite Web service,
the private enforcement of provisioning rules could be a complex task since the
number of rules to be satisfied could be high, and so would be the number of
required interactions of the two-party private intersection protocol for the end
user. To address this problem, as described in [117], our approach minimizes the
number of provisioning rules to be evaluated by the user by executing a multi-
party private intersection protocol on the service providers’ rules. To handle both
the private matching protocols (step 6 of Figure 4.1), PN is equipped with two
modules: the two-party coordinator (PM-2P) that evaluates private search criteria
and provisioning rules for simple service selection (see Section 5.2.1) and the multi-
party coordinator (PM-MP) that evaluates provisioning rules in case of multi-party
service (see Section 5.2.2). To facilitate and coordinate the usage of PN, clients
and servers should communicate with each other through the Private Negotiator
at the client side, and the Private Negotiator at the server side, as discussed in
Section 5.2.

Example 1 As an example, consider a user requesting a travel service which in-
cludes airline reservation (SP1), hotel reservation (SP2) and payment services
(SP3). As such, the travel service is a composite service with three types of ser-
vices. The user provides the description of the type of service he or she desires,
that is, a travel service, along with the dates of travel, that should be between May
20th of 2013 and June 15th of 2013. Additionally, he indicates that he is looking
for a flight which cost is below 500$ and he wants the transaction to occur over a
secure and encrypted channel. Ideally, the user would like to be informed about the
possible Web services options available to him. Once a suitable service or combi-
nation of services is provided, the user is likely to have to disclose service-specific
information to properly customize the service selection. In this case, this inform-
ation includes the user’s citizenship, form of payment, travel needs etc. Given
the sensitive nature of this information, the user is concerned about his privacy
and wishes his/her data to be handled in a confidential fashion. Hence, he allows
disclosing contact information, demographic data and financial information, but
he wants it to be retained from the travel company for at most 30 days. He is not
willing to release his record to third party companies.

The providers offering this complex service need to gather the users’ criteria and
profile, while on the one hand maximizing the chance of successful provisioning and
on the other hand, minimizing the amount of information disclosed with the other
service providers offering the Web service.
4.0.2 Service Recommendation and Ranking

Recommendation of Web services is the process of suggesting Web services to the user according to his or her requirements. These requirements are in the form of quality of service (QoS), and privacy and security needs. Recommendation of services aids the users by simplifying the Web service selection process. [45–49] offers some approaches towards an efficient and effective selection of Web services, where interesting solutions are proposed in the context of Web service recommendation. There exist works, [47–49], that take into account preferences of the users and recommend atomic services but not composite services. Some of the works including [50–52] offer approaches for composite service selection but these solutions are not suitable for composite service ranking. Ranking enables the user to select the best services possible according to his or her requirements.

Motivated by the above considerations, the goal of this the recommendation and ranker module is to achieve reliable and user-centric recommendations and ranking of single and composite Web services, where the recommended services are customized. Our proposed module offers some unique features: 1. the user may choose to specify his preferences by ranking various criteria or groups of criteria over disparate dimensions, and 2. various strategies are supported for computing recommendations as a single approach for computing recommendations may not be adequate for all contexts and users’ inputs. Our strategies not only work in cases where users only enter some criteria, but also can operate if a specific ranking of criteria is given, or if a minimum acceptable level for any simple or aggregated criteria is provided. Our strategies are modeled through classic multi-criteria optimization problems, thus ensuring low complexity. The ranking and recommendation model by itself might not solve the vulnerability problems existing in the Web services, but this model is one of the steps towards solving the service vulnerability problem during service selection by a user.

We conduct extensive experimental evaluations on the key algorithms underlying our system, using real-world datasets. Our results show that the ranking and recommendation module is at least as effective as other well-known approaches for recommendation systems, while achieving higher degrees of expressiveness and flexibility.
4.1 User Search Criteria and Service Provisioning Rules

Both the user private/public search criteria and service provisioning rules can be represented as boolean expression of atomic conditions, which are matched against Web service descriptions and user’s profiles’ attributes, respectively. In the following we first formally introduce the atomic condition, criteria and provisioning rules. We then introduce the concept of “satisfaction interval” of a user criteria, which is used for the enforcement of provisioning rule in private matching protocols (see Section 5.2) as well as in the model for the ranking of the recommended service.

4.1.1 Criteria and Rule conditions

We assume each Web service has a set of capabilities. These capabilities describe the non-functional properties of a Web service, such as pricing, estimated time of execution, adopted algorithm for data encryption, retention time of personal data, etc. Furthermore, we assume the presence of taxonomies according to which these capabilities can be grouped based on the common domain. By means of these taxonomies, it is possible to group all capabilities referring to privacy (e.g., retention time, third-party usage, purpose), security (e.g., encryption algorithm, signature algorithm), QoS (e.g., response time, availability, throughput), and other meaningful dimensions. We also assume that user profile consists of a set of attribute containing user’s personal information such as age, city of birth, and so forth. Under these assumptions, user search criteria and provisioning rules can be defined as boolean expressions combining atomic conditions on profile attributes and Web service capabilities, respectively. Moreover, with respect to user private/public search criteria, users should be able to specify atomic conditions in two non-exclusive forms: low-level conditions, i.e., conditions matched against Web service capabilities, and high-level conditions which enable the user to state the taxonomy whose service capabilities he or she wishes to maximize or minimize, without stating specific conditions on low level capabilities. For instance, a user may indicate that he wishes his privacy to be maximized, without providing conditions on the capabilities representing typical components of services’ privacy policies (e.g., retention, purpose, etc.). For this purpose, we define atomic conditions as follows.
Definition 1 (Atomic condition) Let \( C \) be the set of capabilities, \( T \) be the set of taxonomies defined on \( C \), and \( A \) be the set of user profile attributes. An atomic condition \( \text{cond} \) can be in four forms: (1) \( \text{Att} \, \text{OP}_{a} \, \text{value}_{a} \), \( \text{Att} \in A \); (2) \( \text{CAP} \, \text{OP}_{c} \, \text{value}_{c} \); (3) \( \text{Max}(\text{CAP}) \), and \( \text{Min}(\text{CAP}) \), \( \text{CAP} \in C \); (4) \( \text{Max}(t) \), and \( \text{Min}(t) \), \( t \in T \). Where \( \text{OP}_{a} \in \{\neq, \leq, >, \geq\} \) and \( \text{OP}_{c} \in \{\text{OP}_{a}\} \cup\{=, \text{inf}, \text{sup}\} \) are the matching operators and \( \text{value}_{a}, \text{value}_{c} \) are the user preferred value for \( \text{Att} \) and \( \text{CAP} \), respectively.

For search criteria, the user can specify conditions that can be satisfied even if only partially matched. To capture both partially and exactly matched conditions on capabilities, the above definition supports two versions for a same operator (e.g., exact equality \( =_{e} \) and partial equality \( =_{p} \)). As such, the user can simply require a partial or exact matching for a capability condition by using the proper operators (i.e., \( \text{operator}_{p} \) and \( \text{operator}_{e} \) for partial and exact, respectively). Note also that for the equality operator we give two different versions on the partial matching, denoted as \( =_{\text{inf}} \) and \( =_{\text{sup}} \), as this operator has possibly two different semantics. Indeed, a user might require that a specific capability should have a given value/set of values and that it is not admissible having a superset or a subset of the required set (denoted with the \( =_{\text{inf}} \) and \( =_{\text{sup}} \) operator, respectively).

A user can specify a prioritization on search criteria. As such, a user search criteria can be defined as follows.

Definition 2 (User Search Criteria) A user private/public search criteria \( uc \) is represented as a pair \((\text{exp}, \text{priority})\) where \( \text{exp} \) is a boolean expression of atomic conditions (cfr. Definition 8), whereas \( \text{priority} \in [0, 1] \) represents the user-specified priority level, denoted as \( \text{pr}(uc) \), associated with the user criteria.

Example 2 In the travel reservation scenario, two possible user criteria are:
\[ 'uc=\text{FlightPrice}<_{p}500 \, AND \, \text{CryptoAlgo}=_{\text{sup}}\{TDES,AES\}' \], where, for clarity, the priority levels are omitted.

Definition 3 (Service Provisioning Rule) A service providing rule \( ru \) is a boolean expression of atomic conditions only in the form (1) (cfr. Definition 8).

To simplify the computation of private matching protocols, hereafter we represent the service provisioning rule as a Disjunctive Normal Form (DNF) consisting of multiple clauses, where each clause consists of a conjunction of atomic conditions.
We now show that our notions of user criteria are expressive enough to implement privacy compliance. We also show examples of service provisioning rules. We defer to Appendix A for examples on security criteria and generic service criteria taxonomies.

**Example 3** According to a general privacy policy model, relevant actors are the customers (WS users), the collectors (service providers) and possible third parties to which collectors may pass customers’ personal data. A customer may express a privacy preference stating what portion of his ‘information’ can be given out, to which other parties, for what purposes, and for how long the data retention is allowed. The user privacy preferences can then be matched against the collector’s privacy policies. Here, we assume the presence of a set of capabilities modeling relevant information of both privacy preference and policy. As an example, the *AllowedInfo* capability’s domain is ‘contact information’, ‘demographic information’, ‘financial information’, ‘medical information’, and ‘internet information’ (e.g., IP address, passwords, and cookies). Other relevant capabilities are the retention time (*RetTime*), allowed information (*AllowedInfo*), and allowed third parties (*ATP*). By using these capabilities and criteria conditions, users are able to express their privacy preference in terms of criteria conditions. These criteria are then matched against WS capabilities, to compute their satisfaction. As discussed in Example 1, the traveler user of our running example has the following privacy preference: “contact and demographic information has to be retained only for 30 days, and information revealing to third parties is NOT allowed”. This preference can be modeled as a search criteria in the forms of:

\[
\text{AllowedInfo}=\text{inf}\{\text{contactInfo}, \text{demographicInfo}\} \quad \text{AND} \quad \text{RetTime}=30 \quad \text{AND} \quad \text{ATP}=\text{e}\{\text{none}\}.
\]

Let us assume three providers are available to provide the travel service of Example 1, dealing with airline reservation, hotel reservation and payment services, respectively. One of the potential airline providers sells flights at a competitive price only to US citizens and allows up to three baggages for free. The offer is valid also for French citizens, in which case 4 bags are included in the price. The hotel provider offers rooms for customers with a minimum stay of one day who can pay through VISA credit card and have a credit score higher than 1000. Alternatively, other forms of payment are accepted, but the minimum credit score threshold is higher. These provisioning policies described are translated as follows: The airline reservation provider will have: \(((\text{Citizenship} = \text{UnitedStates}) \quad \text{AND} \)
(Baggage = 3)) OR ((Citizenship = France) AND (Baggage = 4)).

The hotel reservation provider maintains the following rule: ((Citizenship = Any) AND (MinimumStay = 1 day)), whereas, the payment service requirements are abstracted by the rule: ((Method = VISA) AND (CreditScore > 1000)) OR ((Method = Any) AND (CreditScore > 3000)).

We now provide some examples of user security criteria in what follows.

**Examples of User Search Security Criteria**

To ensure secure and user-desirable compositions, our ranking system directly accounts for several security-related criteria. Specifically, the user can express more or less detailed security centered conditions. The taxonomy to specify user conditions, in this case, includes a set of known vulnerabilities of most common security technologies, as well as conventional security mechanisms (i.e., type of encryption supported and security standard mechanisms). The satisfaction level is then computed according to the mitigation techniques which are in use by the provider, and therefore whether certain security mechanisms are in place.

From an architectural standpoint, Web service security capabilities are expressed through SAML (Security Assertion Markup Language) assertions and matched for assessing their satisfaction level with respect to the user’s preferences. The SAML architecture relies on the presence of trusted authorities, issuing signed assertions on subjects (e.g., users, services, organizations), that is, a set of statements about the subject. In our approach, we assume the existence of a Secure Capability Authority (SCA) in charge of evaluating Web service security capabilities, and of issuing signed SAML assertions certifying such capabilities. In particular, we use the attribute statement of SAML assertions to express security capabilities of a Web service. According to the SAML specification, the attribute statement consists of an attribute name and an attribute value. We use the attribute name to denote the security vulnerability, whereas the attribute value gives information on how the security feature is mitigated by the corresponding Web service.

We consider the set of known security vulnerabilities or attacks and the corresponding known mitigation techniques (shown in Table 4.1) to define the capabilities. These mitigation techniques are to be checked by the SCA to assess whether a vulnerability is addressed or not.

Similar to the case of privacy requirements, the consumer expresses user-criteria conditions toward preventing a certain attack or to check the support of certain
security mechanisms. For example, consider a consumer who needs secure communications and wants the Web service to use the latest version of Secure Socket Layer (SSL). The satisfaction level measures the degree to which the security mechanisms of the service satisfy this condition. Intuitively, the satisfaction level would be 1 if the SSL is the latest version, and 0.8 if it is the last but one recent version. Similarly, if SSL is not supported at all, the satisfaction level would be set to 0.

Finally, if the user is unable to estimate the security guarantees offered by a composite Web service using fine-grained criteria, he may simply ask to maximize the composition’s security. The user can use the capability $\text{Max}(\text{Security})$, which implies the request to maximize all the security capabilities in the $\text{Security}$ taxonomy. In all of the above cases, the system matches the security requirements of the consumer with the security capabilities of the service by examining the SAML assertions issued by the SCA.

### 4.1.2 Satisfiability of conditions

In order to support the private matching protocol on provisioning rules as well as the partial satisfaction of user search criteria, we need to introduce some additional notions. First, we define the satisfaction interval of an atomic condition. As formalized next, the interval denotes the range of values satisfying a given condition, and is used for the private matching of provisioning rule.

**Definition 4 (Range of Condition Satisfiability)** Let $\text{Cond}$ be an atomic condition defined over $c$, where $c$ is an attribute in $A$ or a capability in $C$, $\text{Dom}(c)$ be the domain of $c$, and $\text{LDom}(c)$, $\text{UDom}(c)$ be the lower and upper bound in $\text{Dom}(c)$, respectively. The range of satisfiability of $\text{Cond}$ is defined as an interval $R_{S_{\text{Cond}}} \subseteq \text{Dom}(c)$, where:

1. $R_{S_{\text{Cond}}} = \text{Cond}.value$\(^2\) if $\text{Cond.}\text{OP} = '='$;
2. $R_{S_{\text{Cond}}} = [\text{LDom}(c), l]$ (i.e., $\text{LDom}(c), l$), if $\text{Cond.}\text{OP} = '<$ (i.e., $\leq$);
3. $R_{S_{\text{Cond}}} = (l, \text{UDom}(c)]$ (i.e., $l, \text{UDom}(c)$], if $\text{Cond.}\text{OP} = '>$ (i.e., $\geq$);
4. $R_{S_{\text{Cond}}} = [\text{LDom}(c), l) \cup (l, \text{DDom}(c))$, if $\text{Cond.}\text{OP} = ' \neq$.'

\(^2\)Hereafter, we use the dot notation to refer to elements of atomic condition (e.g., $\text{Cond}.\text{value}$ denotes the value in the condition $\text{Cond}$.)
### Table 4.1: Security criteria

<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CDATA Field Attack</strong></td>
<td>Validate the XML file using an XML schema</td>
</tr>
<tr>
<td>XML Injection Attack</td>
<td></td>
</tr>
<tr>
<td>Cross Site Scripting</td>
<td></td>
</tr>
<tr>
<td>SQL Injection</td>
<td></td>
</tr>
<tr>
<td>XPath Injection</td>
<td></td>
</tr>
<tr>
<td><strong>XML Denial of Service</strong></td>
<td>When processing, use: Filters, XML gateways, XML parser options to prevent them from processing malicious messages</td>
</tr>
<tr>
<td>Crafting XML messages with large payloads, recursive content, excessive nesting, malicious external entities, malicious DTDs</td>
<td></td>
</tr>
<tr>
<td><strong>Insecure Communications</strong></td>
<td>Use latest version of SSL or TLS</td>
</tr>
<tr>
<td></td>
<td>Use end to end security mechanisms like XML-encryption, XML-signature, and SAML assertions</td>
</tr>
<tr>
<td><strong>Attacks on Integrity</strong></td>
<td>Have Web service designs and configurations reviewed by a third party and practice secure software development techniques</td>
</tr>
<tr>
<td>Parameter Tampering</td>
<td></td>
</tr>
<tr>
<td>Schema Poisoning</td>
<td></td>
</tr>
<tr>
<td>Spoofing of UDDI Messages</td>
<td></td>
</tr>
<tr>
<td>Checksum Spoofing</td>
<td></td>
</tr>
<tr>
<td>Principal Spoofing</td>
<td>Configure perimeter security correctly</td>
</tr>
<tr>
<td>Routing Detours</td>
<td></td>
</tr>
<tr>
<td><strong>Attacks on Confidentiality</strong></td>
<td>Have Web service designs and configurations reviewed by a third party and practice secure software development techniques</td>
</tr>
<tr>
<td>Sniffing</td>
<td></td>
</tr>
<tr>
<td><strong>Privilege Escalation Attacks</strong></td>
<td>Difficult passwords, password strengthening rules</td>
</tr>
<tr>
<td>Dictionary Attack</td>
<td></td>
</tr>
<tr>
<td>Format String Attacks</td>
<td></td>
</tr>
<tr>
<td>Buffer Overflow Exploits</td>
<td></td>
</tr>
<tr>
<td>Race Conditions</td>
<td></td>
</tr>
</tbody>
</table>

In order to support the partial matching, we introduce the notion of *satisfaction level for atomic condition* defined in user search criteria. This aims at quantifying the distance between the capability requested and capability guaranteed by the service. This value ranges in the interval $[0,1]$, where $1$ represents exact matching and $0$ represents no matching at all. Next, we show how these satisfaction levels can be used to estimate the *satisfaction level of the search user criteria*.

In general, given a Web service WS, the satisfaction level of an atomic condition has to be computed according to the specific semantics of the operator used in the
condition, as the following definition clarifies.

**Definition 5 (Satisfaction of Atomic Condition)** Let Cond be an atomic condition, and \( v \) be the value of capability \( \text{CAP} \) for Web service \( \text{WS} \). The satisfaction level of \( \text{WS} \) w.r.t atomic condition \( \text{Cond} \), denoted as \( \text{sat}_{\text{WS}}(\text{Cond}) \), is defined as follows:\(^3\)

- \( \text{OP} \in \{=, \leq, >, \geq, <, \in, \notin, \subset, \supset\} \), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 \) if \( \text{Cond} \) is satisfied, 0 otherwise;
- \( \text{OP} = "=\_p" \) and value, \( v \in \text{Dom}(\text{CAP}) \) (value, \( v \subset \text{Dom}(\text{CAP}) \)), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 - \frac{|\text{value} \cap v|}{|\text{Dom}(\text{CAP})|} \); (\( = \frac{|\text{value} \cap v|}{|\text{value}|} \));
- \( \text{OP} = "=\inf\" \("=\sup\") \) and value, \( v \subset \text{Dom}(\text{CAP}) \), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 0 \) if value \( \subset v \) (value \( \subset v \)); \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 - \frac{|\text{value}| \cap v}{|\text{value}|} \), otherwise;
- \( \text{OP} = "\oplus_p\" \("\oplus_p\") \) and value, \( v \subset \text{Dom}(\text{CAP}) \), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 0 \) if value \( \oplus v \) (value \( \oplus v \)); \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 - \frac{|\text{value}| \cap v}{|\text{value}|} \), otherwise;
- \( \text{OP} = "\oplus\" \(">\_p\") \) and value, \( v \in \text{Dom}(\text{CAP}) \), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 \) if value \( \oplus v \) (value \( \oplus v \)); \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 - \frac{|\text{value}| \cap v}{|\text{Dom}(\text{CAP})|} \), otherwise;
- \( \text{OP} = "=\_p\" \("<\_p\") \) and value, \( v \in \text{Dom}(\text{CAP}) \), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 \) if value \( \leq v \) (value \( \leq v \)); \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 - \frac{|\text{value}| \cap v}{|\text{Dom}(\text{CAP})|} \), otherwise;
- \( \text{OP} = "\oplus\" \("<\_p\") \) and value, \( v \in \text{Dom}(\text{CAP}) \), then \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 \) if value \( \leq v \) (value \( \leq v \)); \( \text{sat}_{\text{WS}}(\text{Cond}) = 1 - \frac{|\text{value}| \cap v}{|\text{Dom}(\text{CAP})|} \), otherwise;
- \( \text{cond}=\text{Max}(\text{CAP}) \text{ (Min}(\text{CAP})) \), where \( \text{CAP} \in C \), then \( \text{sat}_{\text{WS}}(\text{cond}) = 1 - \frac{|\text{MaxDom}(\text{CAP}) - v|}{|\text{Dom}(\text{CAP})|} \) (\( = 1 - \frac{|\text{MinDom}(\text{CAP}) - v|}{|\text{Dom}(\text{CAP})|} \));
- \( \text{cond}=\text{Max}(\text{CAP}) \text{ (Min}(\text{CAP})) \), where \( \text{CAP} \in T \), such that \( \text{CAP} = \{\text{CAP}_1, \ldots, \text{CAP}_n\} \) then \( \text{sat}_{\text{WS}}(\text{cond}) = \sum_{\forall \text{CAP}\in C} \left( 1 - \frac{|\text{MaxDom}(\text{CAP}) - v|}{|\text{Dom}(\text{CAP})|} \right) \) (\( = \sum_{\forall \text{CAP}\in C} \left( 1 - \frac{|\text{MinDom}(\text{CAP}) - v|}{|\text{Dom}(\text{CAP})|} \right) \));

**Example 4** Consider the user criteria from Example 3 and that the flight reservation Web service \( (\text{WS}_1) \) has capabilities: \( \text{AllowedInfo} = \{\text{contactInfo}, \text{demographicInfo}, \text{financialInfo}\} \) AND \( \text{RetTime}=40 \) AND \( \text{ATP} = \{\text{WS}_2\} \). The corresponding levels of satisfaction are: \( \text{sat}_{\text{WS}_1}(\text{AllowedInfo}=\inf \{\text{contactInfo}, \text{demographicInfo}\})=0; \)

---

\(^3\)Hereafter, given a capability \( \text{CAP} \in C \), we assume that that \( \text{Dom}(\text{CAP}) \) is an ordered set denoting all possible input values (e.g., \( \text{Dom}(\text{CAP}) \subset \mathbb{N} \)), and \( \text{MinDom()}/\text{MaxDom()} \) to return the minimum/maximum value in this domain set.
sat_{WS_1}(\text{RetTime}=30) = 1 - \frac{|40-30|}{365} = 0.973, \text{ where } \text{Dom(RetTime)}=365 \text{ days};

sat_{WS_1}(\text{uc}=e\{\text{none}\}) = 0. \text{ As such, } sat_{WS_1}(uc) = 0.

As a further example, with respect to the criteria of Ex. 2, consider the following capabilities for the three services, WS_1, WS_2, and WS_3 (the priority levels are omitted, for simplicity): \{\text{FlightPrice}=450, \text{CryptoAlgo}=\{\text{TDES}\}\}, \{\text{FlightPrice}=510, \\
\text{CryptoAlgo}=\{\text{TDES,DES,AES}\}\}, \text{ and } \{\text{FlightPrice}=600, \text{CryptoAlgo}=\{\text{TDES,DES,AES}\}\} \text{ respectively, where } \text{Dom(Price)}=[0,10000]. \text{ The satisfaction level of atomic conditions for } WS_1 \text{ are } sat_{WS_1}(\text{FlightPrice}<p_{500})=1, sat_{WS_1}(\text{CryptoAlgo}=\text{sup}\{\text{TDES,AES}\})=0, \text{ for } WS_2 \text{ are } sat_{WS_2}(\text{FlightPrice}<p_{500})=1-\frac{10}{10000} = 0.999, sat_{WS_2}(\text{CryptoAlgo}=\text{sup}\{\text{TDES,AES}\})=1-\frac{1}{3} = 0.667, \text{ and for } WS_3 \text{ are } sat_{WS_3}(\text{FlightPrice}<p_{500})=1-\frac{100}{10000} = 0.99, sat_{WS_3}(\text{CryptoAlgo}=\text{sup}\{\text{TDES,AES}\})=1-\frac{1}{3} = 0.667.

4.2 Ranking Model for Secure WS Delivery

In this section, we introduce the data model over which the Ranker computes the ranking of Web service compositions to provide users with customized recommendations, as well as the strategies to combine together for criteria combination.

4.2.1 Graph Model for Service Composition

In regards to a composite service, a workflow (WF) is a set of activities \{a_1, \ldots, a_n\}, where each activity is a task that can be performed by a single Web service. A workflow is a set of activities that can be performed by a set of Web services, which we call as a composite service. Given a workflow WF, defined as a sequence of activities \{a_1, \ldots, a_n\}, and the set of candidate Web services for each of them, we make use of a graph abstraction to account for all compositions resulting from all possible combinations of candidate Web services that satisfy the workflow WF. In general, a level \(j\) of the graph is related to the \(j\)-th activity in the WF, where each node at level \(j\) represents a candidate Web service able to perform activity \(j\). Therefore, each path in the graph, referred to as composition graph, denotes a potential composition. Moreover, as the recommendations are based on user criteria, that is, on whether and how much candidate Web services satisfy user public criteria, the composition graph is defined as a weighted graph. More precisely,
given the set of user criteria $UC=\{uc_1, \ldots, uc_i\}$, the composition graph assigns to each edge, a vector of $i$ elements corresponding to satisfaction levels of each user criteria in $UC$. For clarity of presentation, we assume that one graph is created per workflow.

**Definition 6 (WS-Graph)** Let $WF=\{a_1, \ldots, a_n\}$ be a sequentialized workflow of activities, $WS_{cand}$ be the set of candidate Web services associated with each $a_j \in WF$, and $UC = \{uc_1, \ldots, uc_i\}$ be the set of user criteria. The Composition Graph $G = (V, E, \Phi)$ is a weighted directed acyclic graph, where $V$ is the set of vertices representing Web services in $WS_{cand}$, $E$ is the set of edges, where $e=(v,v') \in E$ iff $v$ and $v'$ are candidate Web services for activities $a_i$ and $a_{i+1}$ in $WF$, respectively. The labeling function $\Phi$ is defined as $\Phi: E \to < R^+ \times \ldots \times R^+ >$, where for each $e=(v,v') \in E$, $\Phi(e)=< (sat_v(uc_1), pr(uc_1)), \ldots, (sat_v(uc_i), pr(uc_i)) >$, that is, $\Phi$ returns the array whose components report for each user criteria in $uc_j \in UC$ the satisfaction levels of $v$ w.r.t. $uc_j$ and user priority weights for each criteria $pr(uc_i)$, where $\sum_{j=1}^{i} pr(uc_j) = 1$.

The composition graph is a powerful representation for analysis and recommendation purposes. The labeled edges model the level of satisfaction of user’s criteria $uc$ along with the corresponding priority $pr(uc)$. By traversing the graph’s paths, it is possible to find the most suitable Web service compositions meeting the criteria entered by the user.

**Example 5** With respect to Example 1, the Ranker finds a set of 4 Web service combinations. These combinations are generated according to the generated workflow, that consists of the following three activities: airline reservation, hotel reservation, and payment. The Ranker builds the composition graph represented in Figure 4.2 out of the 4 combinations, computes the satisfaction levels and assigns the priority weights provided by the user for each type of criteria. Accordingly, it assigns the pair of satisfaction value and user priority weight for each criteria to the edges. The graph has three levels, where each level consists of services with the same function. In addition, the graph includes a source and destination node to the graph which are just dummy nodes acting as starting and ending nodes in the graph to enable easier path evaluation.
4.2.2 Composite Web service ranking Strategies

The Ranker supports three strategies to compute the recommendations: **Average Criteria**, **Priority-based**, and **Bounded ranking**.

The strategies heavily rely on the WS-Graph of Definition 6, and are elegantly reduced to the well-known problem of finding the set of shortest paths in the composition trees, with multi-criteria edges. This class of problems, referred to as multi-objective shortest path (MOSP) problems, can be solved in an optimal fashion with reasonable complexity [118]. In order to apply this class of solutions, we modify the original multi-objective shortest path problem methods to solve our need for multi-objective optimal path problem.

4.2.2.1 Input on all criteria: Weighted Average of Criteria

In this strategy, paths are ranked according to criteria priority weights assigned by users for each criteria. To enforce this strategy we define a *path evaluating*
function, PEF, defined as the weighted average of the criteria values [118].

Let $G$ be a composition graph computed over a set of user criteria $UC$ according to Definition 5, where $Set_P$ is the set of paths in $G$. Moreover, let $P\text{.nodes} = \{P.v_1, P.v_2, \ldots, P.v_{n-1}, P.v_n\}$ be the set of nodes (i.e., Web services) belonging to path $P$, for each $P \in Set_P$. We denote by $P^{uc_i}_{max}$ the path in $Set_P$ with the highest sum of satisfaction levels for criteria $uc_i$. Given a $uc_i \in UC$, the function for calculating the average of satisfaction levels of $uc_i$ along path $P$ is called $AF_{uc_i}(P)$ and computed as follows:

$$AF_{uc_i}(P) = \frac{\sum_{v_j \in P\text{.nodes}} sat_{v_j}(uc_i)}{\sum_{v_j \in P^{uc_i}_{max}\text{.nodes}} sat_{v_j}(uc_i)}$$  (4.1)

The path evaluation function is $PEF(P) = \sum_{i=1}^{\text{|UC|}} pr(uc_i) \times AF_{uc_i}(P)$. Finally, we rank the paths according to the assigned PEF values, obtaining $R\_Path_{set}$. The higher the value assigned, the higher is the rank of the path or a set of Web services in the path.

**Example 6** With respect to the graph of Figure 4.2, first, we compute the sum of satisfaction values for each criteria. For the first criteria $uc_1$ and first path $P_1$ in Figure 4.2, the sum of satisfaction levels is computed as, $\sum_{j=1}^{\text{|P_1\text{.nodes}|}} sat_{v_j}(uc_1) = 0.7$. Similarly, for the next set of paths, $\sum_{j=1}^{\text{|P_2\text{.nodes}|}} sat_{v_j}(uc_1) = 1.4$, $\sum_{j=1}^{\text{|P_3\text{.nodes}|}} sat_{v_j}(uc_1) = 1.5$, and $\sum_{j=1}^{\text{|P_4\text{.nodes}|}} sat_{v_j}(uc_1) = 1$. The maximum sum belongs to path $P_3$. The average of satisfaction values for path $P_1$ and criteria $uc_1$ is computed as $AF_{uc_1}(P_1) = \frac{\sum_{v_j \in P\text{.nodes}} sat_{v_j}(uc_1)}{\sum_{v_j \in P^{uc_1}_{max}\text{.nodes}} sat_{v_j}(uc_1)} = 0.47$. Similarly, for the next set of paths, $AF_{uc_1}(P_2) = 0.93$, $AF_{uc_1}(P_3) = 1$, and $AF_{uc_1}(P_4) = 0.67$. Similar computations are performed on the remaining criteria for all paths. Finally, we compute the PEF value for each path whose values for each of the four paths are as follows: $PEF(P_1) = \sum_{i=1}^{\text{|UC|}} pr(uc_i) \times AF_{uc_i}(P_1) = 0.710$, $PEF(P_2) = 0.765$, $PEF(P_3) = 0.8575$, and $PEF(P_4) = 0.685$. The PEF value of path 3 is the highest among the other paths. Hence we rank the paths in the following order from highest to lowest: $\{3, 2, 1, 4\}$.

### 4.2.2.2 Priority-Based ranking

This strategy is implemented when the user provides a single high priority criteria. The strategy attempts to maximize satisfaction level for that criteria, while
checking for satisfaction of any other criteria the user may have entered. From an algorithmic standpoint, the high priority criteria is considered as the input of an objective function, according to the original formulation of the Hierarchization of Objective Functions algorithm [118]. Precisely, we adapt the original algorithm as follows. Let \( u_{c_1} \) be the user criteria with highest priority, and \( u_{c_{i+1}}, \ldots u_{c_{i+k}} \) be lower priority criteria, sorted by importance. For each path \( P \) in \( \text{Set}_P \), we consider the sum of satisfaction levels with respect to criteria \( u_{c_1} \), obtaining a new PEF function for path \( P \):

\[
\text{PEF}(P) = \sum_{j=1}^{P_{\text{nodes}}} \text{sat}_{v_j}(u_{c_1})
\]

The paths in \( \text{Set}_P \) are then sorted according to the PEF function value, and \( \text{R\_Path}_{u_{c_1}}^{\text{Set}} \) is obtained. Next, we compute \( \text{R\_Path}_{u_{c_1}}^{\text{Set}} = \text{Max}_k(\text{R\_Path}_{u_{c_1}}^{\text{Set}}) \), which is an iterative function reducing the input set to the \( k \) paths with highest sum of satisfaction values for \( u_{c_1} \). If the user has specified any other criteria, say \( u_{c_j} \), then, in the subsequent iterations of the algorithm, we filter out paths from \( \text{R\_Path}_{u_{c_1}}^{\text{Set}} \) whose nodes do not satisfy the additional criteria (i.e., \( \text{sat}_{u_{c_j}} = 0 \)). Paths are filtered taking into account a single criterion per each iteration, in the order of priority. The final set of ranked service combinations is sent as recommendations to the user.

**Example 7** Let the user give priority to the first criteria, \( u_{c_1} \), of Figure 4.2. Let \( k \) be set to 3. First, we compute the sum of satisfaction values of \( u_{c_1} \) for all four paths and we select the first 3 paths with maximum sum of satisfaction values and sort them, which are as follows: \( \text{R\_Path}_{u_{c_1}}^{\text{Set}} = \{P_2, P_3, P_4\} \). The input to the next iteration for finding the set of paths which satisfy the criteria \( u_{c_2} \) is \( \text{R\_Path}_{u_{c_1}}^{\text{Set}} \). The final set of paths are ranked according to the sum of satisfaction values of \( u_{c_1} \). The iterations for the remaining criteria ensure to select the paths whose services’ satisfaction value is not equal to 0. Hence, for \( u_{c_2} \), we verify that the nodes of paths in \( \text{R\_Path}_{u_{c_1}}^{\text{Set}} \) satisfy \( u_{c_2} \). In this example, all the paths satisfy \( u_{c_2} \) and \( u_{c_3} \). The final ranking of the paths is \( \{P_3, P_2, P_4\} \).
4.2.2.3 Bounded ranking

This strategy accounts for users requests of bounded capability values expressed through one or more global user criteria, each denoted as $\pi_c$. For instance, a traveler may request to book a trip that does not cost more than $500, or that it does not take more than 20 minutes of service processing time, including all services in the composition.

In proposing the strategy, we should also consider that the user might still associate with each global user criteria $\pi_c \in UC$ a priority level $pr(\pi_c)$. The strategy consists of integrating to the above two strategies additional controls. The key steps to be executed are sketched as follows ($Set_P$ is a non empty set of paths in the composition graph $G$). We provide the example for a lower bound on a specific criteria.

1. Compute $Set'_P$, using Weighted Average of Criteria or Hierarchization of Objective Functions algorithm.
2. Starting from the first path in $Set'_P$, check for global thresholds satisfaction. That is, let $\overline{\pi_c}$ be the threshold value for a capability $c$ to satisfy. Let us assume, for simplicity, that the capability value is expressed through an ordinal number, and that each Web service in the composition has an associated value $v$ for $c$. The inequality $\sum_{j=1}^{\vert P\text{.nodes} \vert} v_j \leq \overline{\pi_c}$ is computed, where $v_j$ is the capability value for the Web service $j$ in the path $P$, and $\vert P\text{.nodes} \vert$ is the number of nodes in $P$.
3. If more than one inequality is imposed, check for all the global criteria according to step (2), and sort them according to the priority value, if specified.
4. If the inequalities are satisfied, keep $P$ in $Set'_P$. Otherwise, $Set'_P=Set'_P\setminus P$
5. Check a different $P' \in Set'_P$

Notice that step (2) can be replaced with other types of global controls, from a request to minimal satisfiability level, to a request of maximizing or minimizing a given criteria only. For example, a user may ask to obtain a high level of security, in addition to specifying detailed capability conditions.

Example 8 Consider again the example from Figure 4.2, and additionally consider the global constraint the user specifies that $uc_1$, the total response time of all services in the composition to be less than 5 ms. The paths determined according
to the weighted average method are: \( \{P_3, P_2, P_1, P_4\} \). Let \( P_3, P_2, \) and \( P_4 \) satisfy the global constraint of the user. Hence, the set of ranked paths according to the local and global criteria of the user are \( \{P_3, P_2, P_4\} \).
Chapter 5 | Private Web Service Selection

Web service selection is a crucial step in the Web service life cycle as this is the step where users get to select a service that is most suitable to their requirements. However, there is a huge amount of private information exchange during the selection phase for both the users and the service providers.

We illustrate the requirements that are in place at the service providers’ and users’ side that may constrain the Web service selection, and discuss the privacy concerns that these requirements might bring or cause to the users and the service providers during the service selection phase. Users’ requirements mainly apply to the search criteria used to qualify their business needs and preferences so as to choose the best Web service among the available ones. For example, to identify a shipping company, the user might choose the Web service of the shipping company that guarantees a fast delivery or a low price, or the company that minimizes the overall cost. In general, these search criteria are defined according to the user’s business strategy or personal preference (i.e., of minimizing the cost, and of ensuring a fast delivery) and are matched against Web service descriptions. To enforce these search criteria the user has to interact with the company (or one of its Web services) so as to verify, for example, the time and location of delivery, and/or the overall cost and, therefore, disclose some potentially sensitive information (i.e., users location/address). Further, to verify the search criteria, the company might have to disclose confidential information to the user (for e.g., the minimum price it is offering the service for or the locations served by the shipping company). Service providers’ requirements, on the other hand, are related to provisioning rules, and dictate the conditions under which the service can be invoked. Providers might require to match their conditions against the users’ profile in order to release
the service. These conditions could be imposed by law, or by internal business rules. For example, in the context of e-commerce transactions, laws protecting against underage drinking might have to be applied. Again, in the case of a rental car service, the provider might prefer to lend luxury cars only to clients with a given income, or it might choose not to lend them to young drivers. All these conditions, which we refer to as service provisioning rules, have to be retrieved and matched against the user’s profile prior to service provisioning. Similar to the case of search criteria, these rules might have to be verified against private/sensitive user’s attributes (i.e., age and income), therefore giving rise to privacy problems.

We notice that, while literature has extensively investigated the problem of service discovery and selection based on users needs (e.g., QoS requirement - see [16], [7], [9], [2], [8] as example), the problem of privacy and information leakage implications associated with service selection has not yet been investigated. This problem is further compounded by the fact that many services are actually the result of a workflow, whereby multiple Web service providers are involved, and a dynamic composition process is to be carried out on the fly. In the presence of multiple providers that coordinate together to provide a composite Web service, privacy issues are amplified, due to the larger number of service providers involved (each of which may have individual provisioning rules). Further, not only end users may have privacy and confidentiality needs, but also Web service providers may need to maintain provisioning rules, as confidential as possible. A privacy-preserving strategy is therefore desirable, so as to verify search criteria against service description and to verify provisioning rules against the user’s attributes in a private fashion. In particular, for provisioning rules, this strategy should determine the set of combined service provisioning rules to be satisfied by the end-user, while minimizing the information leakage both at the client’s and at the various providers’ side. In light of this analysis, our goal in this chapter is twofold. First, we aim to define a solution where the matching of the search criteria against the Web services attributes is performed in a private way such that both criteria and attributes are kept private during the matching, and revealed only to the extent necessary to actually complete the transaction (i.e. after verifying that the service can be invoked). Second, we aim to address privacy issues due to the need of enforcing service provisioning rules at the time of Web service deployment and possible composition. By providing a privacy-preserving selection process of
suitable composite services, we also aim to lower the high computational costs for the end user, which arise due to the fact that for each Web service a different set of rules have to be verified against user’s profile. As such, the privacy-preserving solution has to be designed so as to reduce, as much as possible, the overhead added for the users by these additional controls.

5.1 Private Matching

The Private Matching Protocol (PM) has been proposed by Malkhi in [10], to allow two parties to learn in a private fashion the set of common attributes. Each of the two parties holds a database, and wishes to jointly calculate the intersection of their inputs, without leaking any additional information. The exchange is asymmetric, so that only the party initiating the PM algorithm learns the result of the matching.

The basic tool underlying the original formulation of the private matching is a semantically secure public key encryption scheme, with homomorphic properties. Examples of such schemes are El Gamal, Paillier, DJ [11]. The protocol consists of the following main steps. Let us assume that the two parties are C and S, and C is the protocol initiator. Moreover, assume that the values of C and S, namely X and Y, belong to the same domain $\delta$.

- C defines a polynomial of degree n whose roots are $x_1,...,x_n: P(y) = (x_1 - y)(x_2 - y)...(x_n - y) = a_ny^n + ... + a_1y + a_0$
- C sends to S the homomorphic encryptions of coefficients $Enc(a_n), ..., Enc(a_0)$
- S uses homomorphic properties to compute $\forall y \in Y$, $Enc(rP(y) + y)$ (r is random)
- If $y \in X \cap Y$, the result is $Enc(r0 + y) = Enc(y)$, otherwise result is $Enc(random)$
- S sends (permuted) results to C
- C decrypts and compares the results to its list.

In this chapter, we denote the outcome of the private matching protocol as $PM(C, S; \delta)$, where C and S are the two parties running the protocol, $\delta$ is the domain over which the protocol is carried out. C is always the party initiating
the process, and, therefore obtaining the output. Two-party private matching is a special case of the more general problem of secure multi-party computation. In a secure m-party computation, the parties wish to compute a function $F$ on their $n$ inputs. Using cryptographic techniques such as [11], [3], the protocol returns to the initiating party the results of the computation of the $F$ function. In our work, the function $F$ performs intersection among attributes belonging to the same domain.

5.2 Private Service Selection

We now discuss the role of the private negotiator and its main tasks. The private negotiator (PN) enables the process of private service selection by providing private search and private provision rule compatibility verification among the service providers, and between the service providers and the user. PN consists of two main components: two-party coordinator and multi-party coordinator, which coordinate the private search, private provision rule compatibility verification and final rule formation. The private negotiator is also installed at the client side called the PN_CLIENT plug-in, and at the service provider side called the PN_SP plug-in. The PN_CLIENT enables the user to specify the search terms and the priority, and helps in coordinating the two party protocol. The PN_SP plug-in helps the service providers in enforcing the two-party and multi-party protocols. Both PN_CLIENT and PN_SP are equipped with encryption functionalities. In our discussion of the two-party and of the multi-party coordinator, for the sake of clarity, we present the algorithms for the matching of individual conditions included in provisioning rules or in search conditions. Next, we discuss how the individual conditions are extracted to facilitate matching and reduce the computation overhead.

5.2.1 Two-Party Coordinator

We show how the two-party coordinator can aid the enforcement of the private search criteria, and, in case of single services, of service provisioning rules. Let us start by introducing the process for a single generic constraint evaluation, irrespective of what entity is applying the conditions to evaluate (i.e., user for search
criteria or service provider for provisioning rule), and over which/whose profiles the conditions have to be matched (i.e. user/provider). In general, users or providers’ conditions are defined by constraints evaluated against assertions by means of a two-party private intersection protocol. We recall that according to this protocol, given a condition $\textit{Cond}$ and its range of satisfiability $\textit{RS}_{\textit{Cond}}$, the constraint owner has to (1) generate a polynomial of degree $n$, where $n$ is the cardinality of $\textit{RS}_{\textit{Cond}}$; and (2) compute the homomorphic encryption of its coefficients obtaining the private range of satisfiability (i.e., $\textit{PRS}_{\textit{Cond}}$). The private range of satisfiability together with the attribute name over which the condition is posed (i.e., $\textit{Cond.name}_A$) are passed to the two-party coordinator. The Two-party coordinator asks the profile owner for the private encryption of $\textit{Cond.name}_A$ value. More precisely, the profile owner retrieves the value, say $\textit{value}_A$, and sends back the encryption $\textit{Enc}(rP(\textit{v}_a) + \textit{v}_a)$. The two-party coordinator sends this encrypted value to the rule owner, that is able to decrypt and obtain $\textit{value}_A$, if $\textit{value}_A \in \textit{RS}_{\textit{Cond}}$, a random number otherwise. Notice that as a result of this process, only the entity enforcing the conditions is able to see the outcome of private matching protocol, while the profile owner does not learn anything about the outcome. These steps are executed by both PN_CLIENT and PN_SP module. More precisely, in case of search criteria PN_CLIENT and PN_SP play the constraint and profile owner, respectively, and vice-versa in case of provisioning rule. In any case, the constraint owner is the initiator of the two-party process, and is the one who learns the outcome of the process, while the profile owner is the one that just provides the input to the two-party process. The same property applies in case of multi-party processes. These features are key to increase the secrecy of information among the participants, and will allow only the necessary participants to know the outcome of the processes.

After selecting a clause from the provisioning rule of an SP which is in the DNF, two party and multi-party private set intersection protocols (Sections 5.2.1 and 5.2.2) are performed over the attributes in the clauses. That is, each service provider uses its final clause during the provisioning rule compatibility check coordinated by the two-party and multi-party coordinators.
5.2.2 Multi-Party Coordinator

We now focus on the case where a user selects a composite service. Here, provisioning rules of multiple service providers have to be combined together for actual service provisioning. Our approach is to combine rules applied by the providers in a private fashion, before presenting them to the user. This strategy has two main benefits: first, it minimizes the number of provisioning rules that the user is eventually asked to comply to, and, second, it minimizes the amount of information disclosed by service providers (in terms of provisioning rules) to clients and competing providers. Each provider enforces one provisioning rule expressed as a DNF, from which a clause is selected as explained in Section 5.2.3. We also assume that the workflow is modeled as a sequence of activities. The following example is used throughout the section to clarify the main steps.

**Example 9** For the travel service from Example 1, consisting of the WF of three activities for which there are three services from distinct service providers, namely $SP_1, \ldots, SP_3$. Assume that each service provider has a clause whose conditions are defined over a set of attributes or capabilities, denoted as $SP_j$-ATT, where $SP_1$-ATT: \{Age, Cost, Bags, Weight\} $SP_2$-ATT: \{Age, Cost, Salary, Hours\}, $SP_3$-ATT: \{Age, Cost, Salary, Bags\}.

Let us consider in particular clauses specified over Age and let $RS_{\text{Cond\_Age\_}SP_j}$ be the range of satisfiability of condition imposed by rule at $SP_j$ side. The following conditions hold. $RS_{\text{Cond\_Age\_}SP_1} = \{1, 8, 9\}$; $RS_{\text{Cond\_Age\_}SP_2} = \{1, 9, 7, 6\}$; $RS_{\text{Cond\_Age\_}SP_3} = \{5, 1, 9\}$.

The overall process is presented in Algorithm 1. The algorithm has two main procedures: the first one allows individual service providers to identify the common set of attributes used by other service providers in their provisioning rules (lines 1-20). The second part instead consists of determining whether or not the conditions where the common attributes appear have overlapping ranges of satisfiability (lines 21-31). If no overlap among the common attributes exists, i.e. a common satisfiability range cannot be determined, the conditions of provisioning rules are incompatible with each other, and hence the rules of the service providers are not compatible. For example, with reference to Example 1, attribute Age is shared by SP1, SP2, SP3 and the common values for Age are 1,9. Specifically, the provisioning rule to be eventually evaluated against the user’s profile, is composed
Algorithm 1: Private Provisioning Rules Composition: Matching provisioning rules of each service provider in a private fashion

1: Let $n$ be the number of service providers, $a$ the maximum number of attributes in a clause
2: for $i = 0; i < n; i++$ do
3:     for $j = i + 1; j < n; j++$ do
4:         k=0
5:         Invite $SP_i$ and $SP_j$ to perform $PM(SP_i, SP_j)$
6:         $TPresult[i][j][k++] = recv(SP_i)$
7:     end for
8: end for
9: % PN has the pair-wise intersections
10: for $i = 0; i < n; i++$ do
11:     for $j = i + 1; j < n; j++$ do
12:         for $t = 0; t < a; t++$ do
13:             % identifying the attributes
14:                 for $k = j + 1; k < n; k++$ do
15:                     % set of SPs all having a common attribute with the initiator
16:                         if $TPresult[i][j][t] \in TPresult[i][k]$ then
17:                             mpcandidate[$m++$] = k;
18:                         end if
19:                 end for
20:             end for
21:         % MPM over $TPresult[i][j][t]$ is not completed then
22:             Invite $SP_i$ to perform $MultiPartyP(SP_1; SP_2, \ldots, SP_k, TPresult[i][j][t])$
23:             Invite $SP_j$ to perform $MultiParty(TPresult[i][j][t])$;
24:         end if
25:     end for
26:     Invite mpcandidate[$l$] to perform $MultiParty(TPresult[i][j][t])$;
27: end for
28: end for
29: $PMresult[x++][y++] = recv(SP_i)$
30: end for
31: end for
32: end for

generating according to the following set of steps. The workflow is generated, and a list $SPL = \{SP_1, \ldots, SP_n\}$ of service providers for each activity is identified. Each service provider, $SP_i$, uses its local component $PN_{SP_i}$ to initiate a two-party private set
intersection process with every $SP_j$ in SPL one by one, where $i = j$ and $i < j$, lines 2-9, (a service provider who has already participated in two-party private set intersection process with a particular service provider will not initiate the same process with it again). Each individual two-party protocol takes the same form of the protocol described in Section 5.2.1 where instead of profile attributes and range of conditions, the private set intersection protocol is computed between the attributes defining each service provisioning rule. That is, the private set intersection protocol is computed between $SP_i\_ATT$ and $SP_j\_ATT$. Upon completion of all the two-party protocols, each $SP_i$ learns all the attribute names shared with the remaining $SP_{i+1}, ..., SP_n$. Notice that this intersected set is known only to $SP_i$ (i.e. the SP that initiated the two-party private set intersection process). Then, each $SP_i$ returns this set to the PN.

**Example 10** In the above example, $SP1$ initiates two-party private set intersection process with $SP2$, and $SP3$ one after the other. Upon completion, $SP1$ learns that it has the following common attributes with the other SPs:

$SP_1\_ATT \cap SP_2\_ATT : \{Age,Cost\}$

$SP_1\_ATT \cap SP_3\_ATT : \{Age,Cost,Bags\}$

Next, the PN, based on the intersection sets obtained at the end of the previous steps from the SPs, coordinates multi-party intersection processes among the service party private set intersection is computed to determine the intersection among the ranges of satisfiability of the conditions defined over the common attributes (identified in the previous step). Given these intersected ranges, PN is able to generate a unique provisioning rule to be satisfied by the end user for actual service provisioning. For each attribute, a service provider, $SP_i$, is designated as the initiator of the protocol. The $SP_i$, mediated by its local PN\_SP component, initiates multi-party private set intersection protocols with those service providers having common attributes with it and identified at the prior step (known through PN). The multi-party set intersection aims at comparing the range of satisfiability of conditions defined on the attribute. The providers essentially compute the intersection among $RS_{CondX\_SP_i}$ and $RS_{CondX\_SP_j}$ with $j \in [1, n]$, where $X$ is the common attribute. The actual multi-party set intersection protocol builds on the Beaver-Micali-Rogaway cryptographic protocol, that operates on a Boolean circuit representation of the computed function (see [3] for details). As in the case of
two-party intersection, the outcome of this protocol, that is, the acceptable satisfiability range, is known only to the service provider who initiated the multi-party private set intersection process.

**Example 11** To compute the acceptable satisfiability range on conditions with domain Age, SP1 initiates a multi-party private set intersection process with SP2, and SP3. In this process, each service provider $SP_j$, $j \in \{1, 2, 3\}$ participates by passing encrypted version of its $RS_{\text{CondAge}_{SP_j}}$. Thus, the outcome of the multi-party private set intersection process initiated by SP1 on domain Age is the set \{1, 9\}. Similarly, multi-party private set intersection process is conducted on every common attribute with other service providers.

At the end of each and every computation, the multi-party coordinator (PM-MP from Figure 6.1) obtains a message with the resulting outcome. When all the individual conditions are evaluated for a given SP, the SP also indicates whether or not the provisioning rule of the service provider is satisfiable. Further, it receives attribute names and corresponding conditions not in common with others. Upon receiving input from all the providers, the multi-party coordinator generates a unique provisioning rule composed by the set of conditions (i.e., a unique set of range of satisfiability) identified for each common attribute whose range of satisfiability is computed as intersection, plus those conditions defined over non-common attributes. The resulting provisioning rule is then evaluated against user’s profile mediated by the two-party coordinator (PM-2P from Figure 6.1). If the set intersection protocols do not produce a final rule over the selected clauses (that is, the clauses have not complied with each other meaning there are no common values for at least one attribute of two clauses), we re-execute the clause selection algorithm to select one clause from each SP trying with a different clause than last time.

### 5.2.3 Clause Selection from a Non-Atomic Rule

We now discuss how to relax the assumption made in the previous section, that is, each service provider has a single provisioning rule defined by a single atomic clause. In reality, service providers (and users as well) are likely to maintain non-atomic rules wherein conditions are combined using disjunctions and conjunctions.
operators. In this case, each actor (e.g. user or provider) at each round needs to identify the atomic condition to match first. It is highly likely that if clauses are randomly selected for matching by each service provider, the compatibility match may fail, even if other compatible clauses exist. For example, if a given provider SP’ has a rule \((Age > 40 \text{ AND } Salary > 4000) \text{ OR } (Cost < 100)\), and SP” has a rule \((Age > 10 \text{ OR } Cost > 300)\), by matching \(Cost < 100\) and \(Cost > 300\) the providers will find an empty intersection of acceptable values, leading to a halt or additional compatibility checks despite the existence of two other compatible clauses, e.g., \((Age > 40 \text{ AND } Salary > 4000)\) and \(Age > 10\), respectively. To avoid these pitfalls, the proposed approach attempts to maximize the chance of matching compatible clauses.

We provide a step-by-step informal discussion of the overall algorithm through an example and a figure (Figure 5.1). We consider the scenario from Example 1, where \(SP_1\), \(SP_2\), and \(SP_3\) are the airline reservation, hotel reservation, and payment services respectively. We consider that the services have the following provisioning rules.

\[SP_1: (Age < 60 \text{ AND } Cost < 15) \text{ OR } (Cost < 15 \text{ AND } Weight > 35 \text{ AND } Salary < 5000);\]
\[SP_2: (Age < 30 \text{ AND } Cost < 25) \text{ OR } (Cost < 20 \text{ AND } Weight < 65) \text{ OR } (Days > 5 \text{ AND } CreditScore > 60);\]
\[SP_3: (Age < 55 \text{ AND } Salary > 2000 \text{ AND } Hours > 40) \text{ OR } (Cost > 50 \text{ AND } Bags < 5)\]

1. The first service provider, \(SP_1\), selects a clause from its provisioning rule which is the shortest among all other clauses of its rule, that is, a clause with lesser number of atomic conditions (e.g. \(Age < 60 \text{ AND } Cost < 15\)).

2. \(SP_1\) adds the attributes to an array, \(V\). Array \(V\) is used to add the attributes of the clauses from all the service providers which are likely to be in the final provisioning rule of the composite service. That is, \(V\) consists of \{\text{Age, Cost}\}, properly indexed.

3. Next, \(SP_1\) performs two-party private intersection with the remaining providers, one by one, over attributes in \(V\) and the attributes of all clauses of \(SP_i\). For example, the two-party private intersection takes place between \(V\) \{\text{Age, Cost}\} and \{\text{Age, Cost, Weight, Days, CreditScore}\} of \(SP_2\). \(SP_1\), as a result of the
Figure 5.1: Comparing provisioning rules and clause selection in a private fashion among service providers in case of non-atomic provisioning rules

computation, only sees the result of the private intersection but not the contents of $V$ in clear. $SP_1$ selects a clause from its provisioning rule that does not contain common attributes between $SP_1$ and itself, and adds the attributes to $V$. For example, $SP_2$ chooses the clause $(Days > 5 AND CreditScore > 60)$ and $V$ has $\{Age, Cost, Days, CreditScore\}$.

4. If all the clauses of $SP_i$ have different attributes from those in $V$, $SP_i$ selects a clause randomly, and adds the attributes of the selected clause to array $V$ if not in $V$ already.

5. If all the clauses of $SP_i$ have at least one common attribute with those in $V$, $SP_1$ selects a clause with the same operators as those attributes in array $V$, if known (the operators may be given by $SP_1$). There is a higher chance of compatibility of conditions if the conditions have same operators for the common attributes, than if the conditions have dissimilar operators. In our example, $SP_3$ requests $SP_1$ for the operators of Age and Cost which are < and < respectively. Hence, $SP_3$ chooses the clause $(Age < 55 AND Salary > 2000 AND Hours > 40)$ as the operator of 'Age' is the same, and it adds
these attributes to array $\mathcal{V}$: \{Age, Cost, Days, Credit Score, Salary, Hours\}. $SP_i$ selects a clause with higher satisfaction level of operators, that is, the clause with the higher number of operators similar to those in Age. Further, if two or more clauses have the same satisfaction level of operators, then the $SP_i$ selects a clause with common attributes having wider range of values. This way, attributes with wider range of values have higher chance to be compatible with any condition than narrow range of values. For example, suppose that the attribute Age of $SP_3$ has the operator $>$, different from that in array $\mathcal{V}$, $SP_3$ chooses clause $(Cost > 50 \text{ AND Bags } < 5)$ as $Cost > 50$ has a range of 50-infinity which is greater than 55-infinity, the range of $Age > 55$.

6. If there are no clauses with same operators, for the common attributes, as those in $\mathcal{V}$, $SP_i$ randomly picks a clause with an overall wider range of all of its attributes.

7. In any case, the attributes of the selected clause are added by each $SP_i$ to the array $\mathcal{V}$, and the algorithm is repeated for the remaining SPs (that is, executed from step 3).
Chapter 6  
Policy-compliant Search Query Routing for Web Service Discovery in Peer to Peer Networks

With the proliferation of distributed systems, Web services are increasingly hosted on peer to peer networks (P2P) [38, 39]. In these environments, each peer hosts a set of services which can be invoked by other peers of the network using standard Web service invocation protocols.

Hosting Web services in P2P networks introduces a number of advantages, the most relevant being the greater availability of services as compared to centralized models. Indeed, the same service may be hosted on multiple peers, therefore also avoiding the risk of a single point of failure and performance bottleneck typical of the traditional Web services system [39].

Previous work has shown that a single service directory system (centralized systems) for resource discovery in P2P networks, as typically seen in the context of centralized systems, would not scale well. In a P2P network, as the number of resources and peers increases, a directory server will become a bottleneck [119]. A distributed search for the available services is actually preferred for service discovery over the registry based discovery, since the Web services infrastructure is a self-organized federation of service providers for the purpose of service sharing and each service provider is autonomous [80]. In case of pure and decentralized P2P networks such as Gnutella, no centralized directory of resources exists, and the query for resources needs to be flooded throughout the network [120]. In these kind of decentralized P2P networks, the search query is transmitted to a number
of random nodes, in search for a service. However, a peer, due to security, privacy, company data policy, or functional compatibility reasons, might want its query to be forwarded only to nodes which conform to a set of constraints.

In this chapter, we introduce a fully decentralized and interoperable discovery approach that enables search query traversal through policy-compliant nodes. We focus on routing issues in the context of peer-based Web services. We design and develop a novel policy driven approach, that enables distributed search through service queries, while taking into account any security, routing or other functional criteria a node may have with respect to routing a query.

In our proposed approach, to ensure node-driven, policy compliant routing, we support two classes of policies, referred to as forwarding and filtering policies. As suggested by their names, these correspond to the rules for forwarding a query, and to any constraint that a peer may have with respect to filtering an incoming query, respectively. According to the proposed solution, the selection of neighbors for query forwarding involves identifying $k$ neighboring branches that might best satisfy the requester’s (query owner) search query policy, while fulfilling the possible filtering policies applied on the incoming query. Our policy matching is efficient and effective, in that it relies on efficient matching of data structures, e.g. attenuated Bloom filters, to estimate the best suitable sub-branches for node search. The filters keep track of compact information not only about a single node, but also about its sub network.

A policy driven search in distributed network is very useful in several application domains, especially for enterprise networks, where services may be requested by collaborating organizations, and open searches may undermine the organization’s interests and confidentiality needs. Any business or enterprise that can host Web services can become a member of the peer to peer network. Each server corresponding to a business hosts its services locally, and maintains a list of services it hosts. A server also maintains the list of services its neighbors are hosting. Any node in the network can request for a service by sending search queries and invoking requests through the network.

We have developed a prototype of the main search protocols and tested it on large scale networks. Our tests demonstrate improved performance as compared to state-of-the art search algorithms, and efficient execution times.

Our architecture is very simple. It consists of the following:
- Nodes or servers in the content hosting networks such as peer-to-peer networks.
- Resources hosted by the nodes in the network.
- A requester node that initiates the search for resources on the network by issuing a search query.

A search query traverses throughout the network via random nodes in search for the desired resource, as shown in Figure 6.1. We attach a policy to the search query so that the query traverses through nodes that satisfy the requirements specified in the policy. Each node consists of a policy comparison component that compares the requirements specified in the policy with the attributes of the neighboring nodes. Also, each node is embedded with the required cryptographic modules that our protocols use. Each node maintains an efficient data structure to store attribute information of its neighboring nodes.

**Example 12** Consider an example where a content distribution network (CDN) employs a peer-to-peer network (hybrid-CDN [30]) to host the contents of its client. For example, the CDN provider Akamai could host the contents of Netflix in its network, and to further increase the availability and bandwidth consumption, Akamai could use a P2P network of its users to host the contents of Netflix. Given this context, the users accessing the content and the content providers could have security and privacy issues related to data traversal in the content hosting network.
Again, the search query traverses through random nodes in the content hosting network, which might not comply with the business rules and laws associated with the query, or which might not comply with personal privacy preferences.

6.1 Overview of the approach

A peer network is formally defined as a non-directed graph $G = (N, E)$, where $N$ is a non-empty set of peers in the network hosting Web services as part of their resources. $E$ denotes the set of edges in the graph, such that $e = \{n_i, n_j\} \in E$ represents an edge between nodes $n_i$ and $n_j$.

We assume that services available in $G$ are organized according to standard classification/taxonomy ($T$) as well as described by means of attributes storing their main features. Examples of categories in $T$ are Standard Industrial Classification (SIC), or the North American Industry Classification System (NAICS). Moreover, we assume that although peers do not know where services are located, they are aware of the existing taxonomy/classification as well as attribute schemas, that is, the list of attribute names and corresponding data types. A search query for a service request is specified in terms of the requested service’s categories, and possible attribute conditions against the service attributes.

Given a service search query SSQ, its distributed search on a given network $G$ starts from the requester node that has generated it. The requester forwards the new SSQ to its direct neighbors, requesting them to evaluate it. If unable to satisfy the query, the neighbors will in turn propagate the query to their neighbors. More generally, each node receiving an SSQ evaluates whether one of its own services satisfies it, and if this is not the case, it propagates the SSQ to its direct neighbors.

Even though distributed service search in P2P has several benefits, it might not accommodate most of the requesters’ needs. For instance, a requester might have some concern on the fact that the search query is accessible to virtually any node in the network, even when SSQ contains sensitive information of the requester (or is to be read only by peers with certain capabilities). Henceforth, the requester might want to forward the query only to selected nodes in the network, according to the requester’s company data management policies. If the requester is concerned about the security and privacy of its data which may be revealed or inferred from the search query, then it might want to forward the query only to nodes which satisfy
security-related conditions. For instance, it may request for a specific algorithm for encrypting data in transmission, check the data retention time of the node, or require a certain protocol used to securely transmit the data (e.g. Transport Layer Protocol). Similarly, the requester might request a certain OS to be run by the intermediate nodes, for compatibility purposes. Furthermore, the requester might request the query to be forwarded only to nodes within a certain mask of IP addresses.

As the previous examples highlight, we focus on a scenario where nodes are willing to participate in the distributed search, but they might have concerns on the participation of other unknown peers. That is to say, the rationale underlying our protocol is to ensure that only policy-compliant nodes can carry search queries while empowering both requesting and forwarding nodes to apply own criteria for any routing provided or requested. We consider two types of criteria. The requester might have forwarding criteria, that is, constraints on the nodes that forward the query. Intermediate peers, i.e., nodes that do no have the service but participate in the evaluation, might wish to express filtering criteria to the incoming search queries, and accept only queries that meet specific conditions, such as the service category (e.g. a node may refuse to forward requests for services providing streaming video).

We assume that all these criteria can be modeled as a set of policies that are attached to the SSQ, in case of forwarding policy, or locally placed at the peer node. Moreover, each peer has a profile denoting its main features, and that the policies can be easily evaluated over these profile attributes.

Given a service search query SSQ, we aim to route it in the peer network through the best possible paths in the network. In this context, the best possible path is a path defined by nodes satisfying the locally-defined policies as well as policies defined by requester to the greatest extent possible within their scope of application (i.e. by both the immediate neighbors as well as the subnetwork connected to any given node). Such a policy-driven routing might however limit the number of services it is able to find. Indeed, it might be hard for all filtering/forwarding policies to be satisfied 100% by all traversed nodes, as each node has unique properties. Hence, if 100% of policy satisfaction is required to route a search query through the network, the requester would not be able to forward the query through most of the nodes in the network. Also, the protocol would miss
out on the nodes that do not satisfy the policy exactly, but have attribute values that are very close to satisfy the query.

As such, the goal of the algorithm is to find a path wherein *policies associated with SSQ can be satisfied partially if not fully*, upon traversing a search path $\mathcal{P}$, i.e., an ordered set of nodes $\{n_1, \ldots, n_k\}$ on which the search query is forwarded, with $n_1$ the node originating the request and $n_k$ denoting the terminal node able to invoke the service requested in the original SSQ. The existence of at least one $\mathcal{P}$ is always guaranteed, provided that (1) there exists a node in the network able to satisfy SSQ and (2) there is at least one path connecting the resource requester with the node holding the resource requested in SSQ.

Policy matching is based on the use of Bloom filters, which are efficient probabilistic data structures to store aggregated information about a node’s subnetwork [121] (see Section 6.3).

By matching policy conditions against Bloom filters, the search query holder is able to identify the directions in the network where a policy could be satisfied fully or partially. The directions depend on its neighboring node branches whose attributes fully satisfy the policy conditions or have lesser policy satisfactions of the policy. More precisely, each node chooses the best $k$ neighbors out of the comparison results. After passing SSQ to $k$ neighbors, each of the neighbors compares its own individual filtering policy. If the filtering policy is satisfied, the neighbor accepts to forward SSQ. The neighbor again performs the same procedure described above for forwarding SSQ until a resource node is found.

### 6.2 Policy Definitions

In this section, we present the core notions used in our policy-aware routing algorithms. We begin by formally defining the Search Service Query (SSQ), as follows.

**Definition 7 (Search Service Query)** Let $T$ be a service taxonomy, and $A$ be an attribute schema. A Service Search Query (SSQ) is an expression of the form: $SSQ = (\{c_1, \ldots, c_n\}; \{a_1, \ldots, a_m\})$, where $\{c_1, \ldots, c_n\} \in T$ are a set of service categories, and $\{a_1, \ldots, a_m\}$ are a set of service attributes in $A$, $\Theta_j \in \{<, >, =, \geq, \leq\}$, and $\{v_1, \ldots, v_m\}$ are the set of attribute values.
Example 13 Let the requester specify the following search service query, $SSQ= (\{\text{Weather}\}; \{\text{Fee}<100d, \text{ExecTime}<20s\})$. The requester is looking for a service belonging to the Weather category, and the service should charge the requester less than 100 dollar fee and should execute in less than 20 seconds.

Next, we focus on policy specification. Hereafter, we assume that the profile of each node is modeled by a set of attributes describing its capabilities, related to privacy and security, system requirements, and routing. As such, the conditions against a node’s profile can be represented as boolean expressions of attribute predicates.

Definition 8 (Node Criteria (NCriteria)) A node criteria is defined as a combination of clauses in disjunctive normal form $c_1 \lor \cdots \lor c_n$, where each $c_j$ is an atomic clause, denoting a single or a conjunction of conditions $c_j = \text{cond}_1 \land \cdots \land \text{cond}_j$, and each $\text{cond}_i, i \in [1,j]$ is of the form: $\text{ATT} \ \text{OP} \ \text{value}$ where: (1) $\text{ATT}$ is an attribute, (2) $\text{OP}$ (e.g., $=$, $\geq$, $\leq$) is a matching operator; (3) $\text{value}$ is the node preferred value for $\text{ATT}$. Value can be a constant, or a variable.

A forwarding policy (FP) is defined by a requester to specify its own constraints on intermediate nodes that should be involved in the query evaluation. Notice that these constraints may or may not be time-dependent. That is, at times, a requester might need to solve the query within a short time interval. This is typically the case for search for services that depend on (possibly external) temporary constraints, such as the current location of a user requesting for a service. For instance, if the requester is temporarily in a given location (i.e. a mall or a given corporation), it might need the query to be solved while in that location. If the search requires to traverse a long path for it to be satisfied, the results might arrive once the requester is no longer interested.

To deal with this requirement, the requester is able to define the time to live for its search service query, in terms of number of nodes along the search path to which the search can be forwarded. As such, by limiting the number of hops, the requester might get response from lesser number of resource nodes but in a shorter time.

Definition 9 (Forwarding Policy) Given a service search query $SSQ$, a forwarding policy $P_{fp}$ is defined as a couple $(\text{NCriteria}, \text{NHop})$, where: $\text{NCriteria}$...
is the node criteria specified according to Definition 8; \( NHop \) can take either a value \( n, n \geq 0 \), or it can be set to *\( \). \( NHop \) denotes the maximum number of intermediate nodes, that \( SSQ \) will be allowed to traverse per a possible path, whose profile satisfies \( NCriteria \). \( NHop = * \) denotes that no restrictions are placed on the hop count.

To track \( NHop \), each copy of the query is associated with a counter to count the number of hosts it travels.

**Example 14** Let the following be a forwarding policy: \( \{(Enc = \text{DES}) \land (Auth = \text{Yes})\} \lor \{(DataRet < 30dd) \land (OS = \text{Windows})\} \lor \{(\text{Domain} = 192.168.250.X), 1000\} \).

In the above example, the policy requests to forward its query only to nodes that use DES algorithm to encrypt the messages and implement authentication, or nodes that retain the requester’s data for no longer than 30 days and use a Windows operating system, or the nodes whose domain is 192.168.250.X. Also, the requester specifies the number of hops \( SSQ \) should traverse to be at most 1000 nodes.

As mentioned in Section 6.1, nodes in the network might have preferences on the service search queries to evaluate and to eventually forward to their neighbors. These preferences might pose conditions both on the type of services (i.e., categories) requested by a query, as well as on the profile of the nodes the incoming queries are coming from. To model and enforce these preferences, we introduce the *filtering policy* (\( Fil \)). These are locally defined by each node in the network and matched against each incoming query before its evaluation and eventual forwarding is performed.

**Definition 10 (Filtering Policy)** Let \( T \) be a service taxonomy. A filtering policy \( P_{fil} \) is defined as a couple \( \{c_1, \ldots, c_n\}, NCriteria \) where \( \{c_1, \ldots, c_n\} \in T \) are the categories of the requested service, and \( NCriteria \) is the node criteria specified according to Definition 8.

According to the above definition, we say that an \( SSQ \) forwarded by node \( N \) to node \( N' \) satisfies the filtering policy of \( N' \) if the categories of \( SSQ \) are contained into categories of \( P_{fil} \) and the profile of \( N \) satisfies conditions expressed by \( NCriteria \) of \( P_{fil} \). For simplicity of presentation, we assume policy conditions
are positive and not negative. It is straightforward to adapt the above definition and the corresponding semantic to include negative conditions.

**Example 15** Let the filtering policy of an intermediate node be \(\{\{\text{Weather, Gaming}\}, \{\text{Domain}=192.168.250.X\}\}\). Let the domain of the requesting node that has forwarded SSQ to this intermediate node be Domain = 192.168.250.3. For the SSQ in Example 13, the intermediate node verifies the categories of SSQ over the categories of its filtering policy, and also verifies its NCriteria against the profile of the requesting node. Since the category and the Domain attribute have been satisfied, the filtering policy is said to be satisfied and the intermediate node accepts to forward SSQ.

\(N'\) might also have its own forwarding policy, and only if the policy is satisfied by its neighbors, will \(N'\) forward SSQ to its neighbors. The policy comparison method and choosing of neighbors is discussed in the next sections.

### 6.3 Bloom Filter Representation of the Attributes

We assume that each node stores both profile attributes and resource names using a Bloom filter (BF). According to the proposed solution, the Bloom filter representing the resource names is concatenated with the Bloom filter of the attributes forming a single Bloom filter for a node.

To encode resource names, \(t\) commonly agreed hash functions by the network, \(\{h_1, h_2, ..., h_t\}\) are employed for representing the resource names. For instance, let a node provide two services whose names are \{LocalWeather, TimeinPST\}. The node creates a Bloom filter using the \(t\) hash functions. Let \(t = 2\). The hash values for the resource names of the node are as follows: \(h_1(\text{LocalWeather}) = 5, h_2(\text{LocalWeather}) = 10, h_1(\text{TimeinPST}) = 3, h_2(\text{TimeinPST}) = 8\). Hence, a 1 is placed at these positions \((5, 10, 3, 8)\) in the filter. A 0 is placed in the rest of the positions. Notice that we employ multiple hash functions for resource encoding, so that even if there are collisions among the index values belonging to different nodes, having more than one hash function enables a node (comparing its policy) to know with greater probability the existence of a resource.

Local attributes, on the other hand, are stored as follows in the filter. For every attribute (say \(fee\)) its corresponding value is encoded by means of a single
order preserving hash function (\cite{122}) $h$ agreed upon by the network. The hash function generates an index value for the attribute value, where a 1 is placed in the corresponding position in the filter. For instance, assuming $fee = 100$ and $h(100) = 7$, the corresponding attribute filter should have the 7th element set to 1, whereas the remaining elements are set to 0. Note that since the profile of a node consists of a set of attributes, a Bloom filter associated with a profile is generated as the concatenation of attribute filters encoding all attributes values, in a known order. We use a single order preserving hash function for attribute encoding, so as to preserve the order of the attribute values of the nodes. By using an order preserving hash function, there is no need of using multiple hash functions to represent the attributes as there would not be any collisions among the index values. Attributes which are in the form of categorical values (e.g. OS, encryption algorithm) are also encoded in the filter. The categorical values of an attribute are sorted according to a predefined taxonomy, within which the order can be defined based on the adopted tree traversal algorithm. We assume that according to the taxonomy, we can compute and quantify the difference between any two categorical attribute values. A simple example of a Bloom filter for a node $N$ is as follows (Figure 6.2).

As introduced in Section 6.1, before propagating the SSQ, a node having the SSQ evaluates forwarding policies against possible routing nodes, so as to find the best $k$ neighbors to propagate the query. Since this selection aims at finding the best subgraph to be explored, this evaluation has to be computed not only on the profile of the neighboring nodes, but also on the profiles of the neighbors’ neighbors, and so forth until a given depth $l$. Again, to make any intermediate node be able to perform this evaluation we make use of Bloom filters so as to encode node profile attributes of neighbors until depth $l$.

To this end, we adopt attenuated Bloom filters ($AttBF$), each of which consists
of multiple \((l)\) layers of basic Bloom filters. Each level \(i \leq l\) in an \(AttBF\) encodes in a single array the values of the OR operation against all attributes of the nodes at level \(i\) from \(N\). Hence, the first layer of the filter contains the information for the close neighbors, while the second layer contains the information about the nodes one hop away, and so on. Attenuated Bloom filters are generated in a recursive fashion.

For example, let a node \(N\) have two neighbors, and let the level \(i\) of their \(AttBFs\) encode the attribute fee in position 4 and 3 respectively. When building the \(AttBF\) to send to its other neighbors, node \(N\) performs an OR operation for level \(i\) of the two neighbors’ \(AttBFs\). The level \(i\) of the \(AttBF\) built by \(N\) is: \([001100000000]\). If the neighbors’ \(AttBFs\) have the attribute fee encoded in the same position, say 4, the level \(i\) of the \(AttBF\) built by \(N\) results as \([002000000000]\) reporting in position 4 the number of nodes of level \(i\) having the same value. The higher levels of an \(AttBF\) are also associated with the corresponding identifiers of the nodes, \(NIDs\), present in the levels.

On the filters representing resource names (i.e. the Web services being offered), we also perform the same OR operation, except that we keep the filters in a binary representation, with 1s denoting one or more resource. After receiving the \(AttBFs\) from its neighbors, a node counts the number of resources containing nodes by computing the \(t\) hash functions over the resource name it is looking for. For instance, if the node is looking for a resource named "Weather" (which is mentioned in the category section of \(SSQ\)) and two hash functions are used, the node computes \(h_1(Weather)\) and \(h_2(Weather)\) which represent the index values to compare with the \(AttBF\). It verifies if the two positions corresponding to the hash values have a 1 at each level. If the two positions have a 1 at a level \(i\), then there is a service named "Weather" at level \(i\) hosted by one or more nodes.

Figure 6.3 shows a requester node and three intermediate nodes with their Bloom filters representing their own attributes and their \(AttBFs\). The intermediate nodes \(Int1\) and \(Int2\) build an \(AttBF\) each, and send them to \(Req\). \(Req\) builds an \(AttBF\) by performing OR operations on the \(AttBFs\) received from \(Int1\) and \(Int2\) and sends it to \(Int3\).

If there are loops in the network, an \(AttBF\) received by a node might include the encoding of its own attributes at some level. We do not discuss here how to deal with these collisions, but we notice that counters and node ids can help avoid
such situations.

6.4 Policy enforcement

In this section we show how the $AttBF$ of a neighbor helps the $SSQ$ holder to obtain an estimate of the set of branches that best satisfy the associated policy.

6.4.1 Node Criteria evaluation on Bloom filter

We begin our discussion on policy evaluation by explaining how a clause $c_j$ in a given policy is matched against an $AttBF$. Next, we show how, building on condition matching results, the degree of satisfaction of a node for a node criteria on a policy $P$ is computed.

First, we consider each individual atomic condition $cond$ composing $c_j$ (if more than one condition is joined together). For each condition $cond$, we compute the satisfaction index, or $\delta.Index^i_{cond}$, for a level $i$ in the $AttBF$. The satisfaction index takes values in $[0,1]$ interval, and denotes the normalized shortest distance between the value (or range of values, in case of inequalities) requested by condition $cond$ and the actual attribute value(s) stored at the level $i$ of the Bloom filter. $\delta.Index^i_{cond}$ is computed as follows.
Let us consider an atomic condition $cond$ of the form $\text{Att OP value}$ (see Definition 8), and a generic level $i$ of the $AttBF$. Let $h$ be the hash function used for encoding the attributes in the filter and $index_{cond}$ be the index associated with $cond$.

1. $index_{cond}$ is computed by hashing $h(value)$. For example, for partly bounded conditions (e.g. $A < 10$), the upper or lower bound of the condition are hashed (e.g. $h(10)$).

2. $index_{cond}$ is compared with the positions of the non-null index values of level $i$ of the $AttBF$.

   (2.a) The condition is satisfied fully, and $\delta.Index^i_{cond}=1$, if the index values of a given level $i$ of an $AttBF$ have 1s or higher number:

   - (2.a.i) in the exact position equal to $index_{cond}$, the index or hash value of the condition, in case of equality conditions ($\text{Att} = \text{value}$),

   - (2.a.ii) in the positions anywhere before or after $index_{cond}$, in case of partly bounded conditions ($\text{Att} \geq / \leq \text{value}$),

(2.b) If the index values of a level $i$ do not satisfy either one of the above conditions, then none of the nodes $i$ hops away from the requester fully satisfy the requester’s condition. The non-null index of the closest index value to $index_{cond}$ is selected (name it $index_{att}$). $\delta.Index^i_{cond}$ is computed as

$$1 - \frac{|index_{cond} - index_{att}|}{Dom} \quad (6.1)$$

where $Dom$ is the size of the attribute domain (e.g. for an attribute of integer values ranging in the interval $[1,100]$, if 100 slots exist in the array, then $Dom=100$. If the domain of the same attribute is broken into sub intervals, say 0-5, 5-10, 10-15, etc., only 20 slots are allocated, and $Dom=20$).

Note that the difference of indexes is proportional to the difference of the corresponding actual attributes, therefore preserving the original 'distance' among the attribute values. This is guaranteed by the order preserving hash function used to encode the attribute values in a Bloom filter (see Sect. IV).

**Example 16** Consider a condition of the form $fee < 100$ of the requester and the level $i$ of an $AttBF$ of one of the requester’s neighbor, as shown in Figure 6.4. Let
index_{fee<100} = h(99) = 6 and the domain of fee be 100. Here, index_{att} = 3, as the 3rd position is the closest to position 6 in the level i of the AttBF. \delta.Index^i_{fee<100} = 1 - \frac{|9 - 3|}{10} = 0.97.

Following the above steps for every joint condition cond_1, \ldots, cond_k in the clause c_j, we compute k satisfaction indexes. The overall satisfaction index for the clause (\delta.Index^i_c) is obtained as the weighted average of the k satisfaction indexes. Weights denote the number of resources satisfying a condition. This information is by construction stored in the index itself. Intuitively, the more the nodes satisfying a condition, the higher the “value” of the sub branch.

Formally, let iv_j be the actual value of the index stored for condition j, k the number of joint conditions in the clause and NoNodes_i, the total number of nodes encoded in level i of the AttBF. The satisfaction value for clause c for level i is

\delta.Index^i_c = \frac{1}{k} \sum_{j=1}^{k} \frac{iv_j}{NoNodes_i} \AST \delta.Index^i_{cond_j} \tag{6.2}

Using \delta.Index^i_c, we can now compute the satisfaction for NCriteria.

**Definition 11 (Satisfaction of Node criteria (\delta.P^i))** Given a node criteria NCriteria for a policy P_{fp}, and a level i of an AttBF of m levels, we define the satisfaction of NCriteria for level i as the maximum among the satisfaction indexes across the clauses for NCriteria,

\delta.P^i = \text{Max}\{\delta.Index^i_{c_1}, \ldots, \delta.Index^i_{c_t}\}

where each \delta.Index^i_{c_j} is the satisfaction index for c_j and t is the number of clauses.
in \textit{NCriteria}.

**Example 17** Consider a node criteria of the form, \((\text{Fee} < 100 \lor \text{DataRet} < 30 \lor \text{ExecTime} < 20)\). Let \(\delta_{\text{Index}_{\text{Fee}<100}} = 0.9\), \(\delta_{\text{Index}_{\text{DataRet}<30}} = 0.4\), and \(\delta_{\text{Index}_{\text{ExecTime}<20}} = 0.6\). For level \(i\) of the \textit{AttBF}, \(\delta.P^i\) is computed as follows. \(\delta.P^i = \text{Max}\{0.9, 0.4, 0.6\} = 0.9\).

### 6.4.2 Policy Satisfaction for Forwarding and Filtering Policy

The Policy Satisfaction with respect to an \textit{AttBF} follows naturally from Definition 11, and is computed according to the Definition below.

**Definition 12 (Satisfaction of forwarding policy for an AttBF)** Let \(P_{fp}\) be a forwarding policy with \textit{NCriteria} \((c_1 \lor \ldots \lor c_k)\), and \textit{NHop}. Assuming the \textit{NHop} criterion is satisfied \(\text{i.e. the query is still active}\), the Policy Satisfaction of \(P_{fp}\) \text{w.r.t AttBF} of depth \(m\) belonging to a neighboring node \(N\), is computed as 
\[
\delta.P^N_{fp} = \frac{1}{m} \sum_{i=1}^{m} \frac{(m+1-i)\delta.P^i}{m},
\]
where each \(\delta.P^i\) is computed according to Definition 11.

Note that the policy satisfaction computed for every level of the \textit{AttBF} is weighted, and the weights are inversely proportional to the level of the \textit{AttBF} they apply to. The intuition here is that filters representing nodes several hops away from the requester are less likely to be visited, in that additional \(k\) options open up for investigation for every level being visited.

**Example 18** Let the satisfactions of node criteria for each level in an \textit{AttBF} of a neighboring node \(N\) having 3 levels be \(\{\delta.P^1 = 0.8, \delta.P^2 = 0.9, \delta.P^3 = 0.6\}\). \(\delta.P^N_{fp}\) is computed as: 
\[
\delta.P^N_{fp} = \frac{1}{3} \sum_{i=1}^{3} \frac{(3+1-i)\delta.P^i}{3} = 0.53.
\]

The evaluation of filtering policy follows a different approach, that requires redefining the satisfaction index of a clause, originally defined in Equation 6.2. Let \(C_{SSQ}\) and \(C_{fil}\) be the category sets of \textit{SSQ} and the filtering policy respectively.

**Definition 13 (Satisfaction of filtering policy for an AttBF)** Let \(P_{fil}\) be a filtering policy with \textit{NCriteria} \((c_1 \lor \ldots \lor c_k)\) of a node \(N\). Let \(C_{SSQ}\) and \(C_{fil}\) be the category sets of \textit{SSQ} and the filtering policy respectively. Two cases may arise.
1. If $CSSQ \subseteq C_{fil}$, then the satisfaction of a clause $c$ in the filtering policy w.r.t. a neighbor $n$ is computed as $\delta.Index_c = \frac{1}{k} \sum_{j=1}^{k} \delta.Index_{cond_j}$, where $\delta.Index_{cond_j}$ is computed according to the procedure describe in Section 6.4.1. Notice that since the attribute values of a neighboring node are stored in level 1 of its $AttBF$, only the level 1 of the $AttBF$ is considered. The Policy Satisfaction of $P_{fil}$ of a neighbor $n$ is computed as $\delta.P_{n_{fil}} = \text{Max}\{\delta.Index_{c_1}, \ldots, \delta.Index_{c_k}\}$.

2. If $CSSQ \notin C_{fil}$, then $P_{n_{fil}} = \emptyset$. In case (2), the node $n$ does not forward $SSQ$. Otherwise, $SSQ$ is accepted if $\delta.P_{f_{fil}}$ is above a locally defined threshold.

### 6.4.3 Protocol Steps

We now describe our distributed policy evaluation algorithm, from the perspective of a requester node whose search query maintains a forwarding policy $P_{neigh}^{fp} = (NCriteria, NHop)$ (see Definition 9).

(a) Upon receiving (or generating) the SSQ, the requester first verifies if the neighbors have any heavily overlapping sub-graphs among one another. That is, starting from level 2 up to $m$ levels, the requester checks whether the $NIDs$ obtained as part of the $AttBF$s from the various neighbors heavily overlap. It disregards the neighbors with multiple $NIDs$ in common (above a locally defined threshold $\text{max}_N\text{NIDs}$). This step can be executed periodically, and the results can be used for several searches (we assume the network is stable).

(b) Having filtered unsuitable branches, the requester counts the number of levels in $AttBF$ (referred to as $RLevels_{n_{neigh}}$) whose resources match the requested resource. The requester computes $RLevels_{n_{neigh}}$ for each suitable neighboring branch. The branches with $RLevels_{n_{neigh}} > 0$ are given priority for searching and policy comparison.

(c) The requester performs policy comparison of the policy $P_{fp}^{neigh}$ associated with SSQ, against the $AttBF$ of every potentially suitable neighbor. First, $NHop$ is checked. If $NHop$ is not satisfied, $SSQ$ has expired, and the search halts.

(d) If $NHop$ is satisfied, the search is still active. The requester computes $\delta.P_{f_{fp}}^{neigh}$ according to the Definition 12. $\delta.P_{f_{fp}}^{neigh}$ is computed for each of the re-
quester’s neighbors. Notice that, if a node has already been evaluated for SSQ or if the node is the original requester node, policy comparison is not performed for that node.

(e) Upon having computed $\delta.P_{fp}^{neigh}$ for all candidate neighbors, the algorithm selects the best $k$ neighbors with higher $\delta.P_{fp}^{neigh}$ values, to forward SSQ.

(f) Each of the selected $k$ neighbors compares its own individual filtering policy $P_{fil}^{neigh}$ with SSQ, as per Def. 13.

(g) If the filtering policy is satisfied, the neighbor accepts to forward SSQ. Also, every neighbor that has received SSQ is marked to be visited.

(h) If the filtering policy is not satisfied and the forwarding request rejected, a new candidate $k + 1$ is forwarded SSQ.

In this chapter, we have presented an approach that efficiently protects and routes a search query taking into consideration the policies attached to the query, on P2P networks. Our approach is unique in that while forwarding the query to the neighboring nodes, we enable the forwarding node to get an insight into the attributes and thus satisfaction of policy of not just the neighboring nodes but also of the nodes in the neighboring branches further levels down.
Chapter 7 | Path Verification for Policy-routed Queries

Content hosting or dissemination networks are those networks that store content in a distributed manner. Today, such networks are gaining popularity. Content providers such as Netflix and Youtube utilize content distribution networks to store their data [92]. Most of the current Internet activities are based on content retrieval than point-to-point communications [123]. Resources in content sharing and dissemination networks (CDN) are discovered through search queries, disseminated along the network using a routing protocol, raising potential security and privacy concerns against the query and the search route.

In these networks, user information privacy and security are considered important issues [92], as content providers, in addition to their own information, store their client’s information in the CDNs. In order to sustain their businesses, clients’ information should be handled very carefully.

Although relatively simple and easy to deploy, search query propagation throughout the network via random nodes raises security and privacy issues, for both the query owner and the search query itself [40]. The query may contain sensitive information related to the query owner, because of which the query owner would want to forward the query only through those nodes that satisfy certain properties. For instance, the search query can be combined with the requested resource and the requester’s IP address to produce a comprehensive database about the requester [124]. Also consider an example where a content distribution network (CDN) employs a peer-to-peer network (hybrid-CDN [125]) to host the contents of its client. For example, the CDN provider Akamai could host the contents
of Netflix in its network, and to further increase the availability and bandwidth consumption, Akamai could use a P2P network of its users to host the contents of Netflix. Given this context, the users accessing the content and the content providers could have security and privacy issues related to data traversal in the content hosting network. Again, the search query traverses through random nodes in the content hosting network, which might not comply with the business rules and laws associated with the query, or which might not comply with personal privacy preferences.

Approaches for policy-based search for resource discovery and routing have been recently proposed (e.g. [126,127]). However, policy-enforcement verification, that is, verifying that the network paths or routes taken by the data or the search query during resource discovery are in accordance with the given policy is still a challenging and open problem. Challenges arise due to the following main reasons: (i) the number of nodes whose policy-compliance is to be verified might explode even in small networks, (ii) there is no easy way to check the nodes’ behavior during a distributed search and, (iii) even if there exists a way to check nodes’ behavior, it is hard to verify node compliance without introducing large computational overhead on the verifier(s). In this work, we take a step toward addressing the above challenges by proposing an efficient approach to verify policy-compliant routing.

We assume the existence of a policy based routing protocol in place as presented in Chapter 7, wherein the routing preferences of a node requesting a resource through a distributed search are expressed by means of a set of policy conditions. A policy-compliant routing algorithm defines a policy which is a machine-readable expression of the query owner’s requirements that are imposed on the nodes which are part of the network. Only those nodes that satisfy the requirements specified in the policy are able to receive the search query. It is worth noting that our focus on policy-compliant distributed search is different from the problem of protecting the content of search query, which just aims at preventing other (policy-non-compliant) nodes from learning the content of search query, and can be easily achieved using one-to-many encryption [128–130]. Here, we consider a more challenging issue, i.e., guaranteeing the query is transmitted in correct path, which not only implies protecting the content of search query, but also limits unnecessary access of the query over the network. Our main focus is two-fold: (i) we aim to develop new
mechanisms to verify the policy-compliance of every node in the network that has received the search query, and (ii) we aim to leverage these mechanisms to efficiently detect the malicious behavior of the nodes in the network with respect to query forwarding. In other words, we aim to detect incorrect forwarding of query in the network.

Toward developing solutions for ensuring policy-compliant distributed search, we design a three-phased routing compliance verification mechanism in the context of content dissemination networks. Our proposed scheme works by firstly identifying the correct paths of search query propagation, then filtering these paths so as to remove nodes that are known to be non-malicious for sure, and then checking the policy satisfiability of all the nodes in the filtered paths. Our scheme is secure, in terms of verifiability and non-repudiable search compliance.

Our contributions are summarized as follows:

- We design an approach to authenticate the paths taken by the search query of a user, that is, an approach to obtain the exact paths taken by the search query without modification.

- To prevent the problem of explosion of the number of paths or nodes to verify for policy compliance, we design an approach that efficiently filters paths or nodes that are not required to be verified. Our experiments show that our approach saves a large amount of computational overhead compared to an approach that does not use path or node filtering.

- We propose three approaches for verifying the policy-compliance of the paths yielded by successful search queries. Our baseline approach relies on random message challenges. In particular, we challenge the target node (i.e. the node to be verified) with two random messages encrypted using CP-ABE [131]. Policy-compliance is verified by checking which of these two random messages is successfully decrypted. To reduce verification overhead, we propose an optimization approach, which is based on using “honesty-verified” nodes in the paths and offloading verification computations to them. Our approach guarantees that query owner’s local computation is minimized while neither affecting the correctness of verification nor imposing heavy computational burden on the nodes. Finally, our third approach is based on the idea of re-using ABE ciphertext, which maintains the same randomness for all the
ABE ciphertexts in verification. This approach is extremely efficient: the requester just needs to compute one pair of CP-ABE encryptions, whereas the computation of our previous two approaches is linear with the number of nodes to be verified.

We evaluate our approach extensively, and compare our efficient verification methods with the baseline method. Our results show that our approach is very efficient and effective.

7.1 Cryptographic Background

7.1.1 Attribute-based Encryption

Attribute-based encryption (ABE) has been widely applied to impose fine-grained access control on encrypted data [128]. Two kinds of ABE have been proposed so far: key-policy attribute-based encryption (KP-ABE) [129] and ciphertext-policy attribute-based encryption (CP-ABE) [130]. In KP-ABE, each ciphertext is labeled with a set of descriptive attributes, and each private key is associated with an access policy that specifies which type of ciphertexts the key can decrypt. In CP-ABE, the access policy is specified in ciphertext and the private key is associated with a set of attributes. In this paper, we will utilize CP-ABE for policy-compliance checking, and thus introduce its main primitives below.

- **Setup(λ)**: The setup algorithm takes as input a security parameter λ, and outputs \((pk, msk)\), where \(pk\) denotes the public key and \(msk\) denotes the master secret key of ABE system.

- **KeyGen(ω, msk)**: The key generation algorithm takes as input an attribute set \(ω\) and the master secret key \(msk\), and outputs the decryption key \(dk_ω\).

- **Enc(m, P)**: The encryption algorithm takes as input a message \(m\) and the policy \(P\), and outputs the ciphertext \([ct]_P\) with respect to access policy \(P\).

- **Dec([ct]_P, dk_ω)**: The decryption algorithm takes as input a ciphertext \([ct]_P\) which was assumed to be encrypted under a policy \(P\) and the decryption key \(dk_ω\) for attribute set \(ω\), and outputs the original message \(m\) if and only if \(ω\) satisfies \(P\).
7.1.2 Identity-based Aggregate Signature

An aggregate signature is a single short string that convinces a verifier that a set of $n$ messages are signed by $n$ distinct signers [132]. In this paper, we will utilize a special line of aggregate signature, namely identity-based aggregate signature, in which users’ identities (e.g., email address) are used as their public keys, and thus the verifier only needs a description of who signed what for verification. The algorithms of identity-based aggregated signature are described as follows.

- **Setup($\lambda$)**: The setup algorithm takes as input a security parameter $\lambda$, and outputs $(pk,msk)$, where $pk$ denotes the public key and $msk$ denotes the master secret key of identity-based aggregate signature.

- **KeyGen(id,msk)**: The key generation algorithm takes as input a descriptive identity $id$ and the master secret key $msk$, and outputs the signing key $sk_{id}$.

- **Sign(m,sk_{id})**: The signing algorithm takes as input a message $m$ and the signing key $sk_{id}$, and outputs the signature $[\sigma]_{id}$.

- **Agg([\sigma]_{S_1},S_1,[\sigma]_{S_2},S_2)**: The aggregate algorithm takes as input two sets of identity-message pairs $S_1$ and $S_2$, and two identity-based (aggregate) signatures $[\sigma]_{S_1}$ and $[\sigma]_{S_2}$ on the identity-message pairs contained in sets $S_1$ and $S_2$ respectively; if $\text{Ver}([\sigma]_{S_1},S_1) = 1$ and $\text{Ver}([\sigma]_{S_2},S_2) = 1$, this algorithm outputs the signature $[\sigma]_{S_1 \cup S_2}$ on the identity-message pairs in $S_1 \cup S_2$.

- **Ver([\sigma]_{S},S)**: The verification algorithm takes as input the (aggregate) signature $[\sigma]_{S}$ and a description of the identity-message pairs in $S$, and outputs 1 if and only if $[\sigma]_{S}$ could be a valid signature output from **Sign** or **Agg** for $S$.

7.2 Design Goals and Threat Model

Our overarching goal is to guarantee policy compliant search, where policies can be specified by means of a set of conditions against the relaying nodes. Our specific objectives to accomplish this goal are outlined as follows.
(1) **Verifiable Search Compliance** The main design goal of this work is to provide a mechanism to verify that a search query in a CDN is forwarded in compliance with the requester’s preferences. These routing preferences are defined over the nodes’ attributes by means of policies, similar to conventional policy-based routing. Note that we do not aim to define a new way of performing policy-routing. We assume the existence of a policy-compliant routing scheme such as [105]. We aim to provide an effective mechanism to verify that policy routing is carried out correctly.

(2) **Non-repudiable Search Compliance:** we would like to ensure that if a node is involved in a search query, it cannot deny having received the query. (3) **Cost-effective:** the modifications and overhead for providing verifiable policy routing should not represent a major additional cost to conventional routing, nor should they alter the way either routing or caching operate.

Our approach to meet these objectives is based on the following threat model and assumptions. Nodes have knowledge of their direct neighbors, but may not know any peers beyond their first degree neighbors. Each node in the network is globally identifiable, and initially assigned with its identity-based secret key $sk_{id}$ and attribute-based decryption key $dk_{prof}$. Nodes find resources by forwarding requests through distributed search protocols [133], wherein a resource request is evaluated by a receiving node and either satisfied or relayed to the neighbor node in search of a node able to provide the requested resource. Precisely, we assume that only nodes with certain properties, indicated in a policy by the node originating the request, are asked and allowed to forward the resource requests. We assume that the majority of the nodes are semi-honest. That is, the nodes keep their individual identifiable information (e.g., $sk_{id}$ and $dk_{prof}$) away from other nodes to avoid the leakage of private information. Malicious nodes may not adhere to the policy-compliant search protocol, and may send the search query to nodes which do not satisfy the requester’s policy.

### 7.3 Policy Compliant Search

Given a node $n$ and a search policy $SP = (NCriteria, NHop)$ (as defined in Chapter 7) specified by a requester $r$, a node is compliant if $NCriteria$ is satisfied by the profile $(prof_n)$ of $n$ and the number of hops transmitted from $r$ to $n$ is not
Figure 7.1: An Example for Policy-Compliant Distributed Search

A policy-compliant distributed search is simply defined as a list of connected nodes satisfying $SP$.

**Definition 14 (Policy-Compliant Distributed Search)** Let $G = \langle N, E \rangle$ be a network, and $(SP, SQ)$ be a pair of search policy and query specified by a requester node $r$. Suppose there is a (cycle-free) sequence of connected nodes $\overline{Path} = \{r, n_1, \ldots, n_k = d\}$ in $G$ connecting $r$ with a node $d$ able to resolve the query $SQ$. If every node $n_i \in \overline{Path}$ satisfies the search policy $SP$, then $\overline{Path}$ distributed search is policy-compliant with respect to $SP$.

Note that in the definition above we essentially request a sequence of nodes in the network graph where each node satisfies the policy and that leads to the successful resolution of query. We do not impose any condition against how this path is found or against any other properties of the path itself (if it is an optimal path or if it is minimal). Several path finding algorithms could be used, with no impact on our problem statement.

A weaker notion of the above definition, which will be useful for our verification algorithms is defined as $\alpha$ compliance.

**Definition 15 ($\alpha$-Policy-Compliant Distributed Search)** Let the pair $(SP, SQ)$ be a search policy and query specified by a requester node $r$. Let the set $\overline{Path}$ contain all the nodes transmitted during a time of distributed search $(SP, SQ)$. If, for

\[1\text{If } NHop = *, \text{ we consider it is infinitely large.}\]
any arbitrarily sampled node \( n \in \text{Path} \), the probability that \( n \) satisfies the search policy \( SP \) is not less than \( \alpha \) \((0 < \alpha \leq 1)\), then \( \text{Path} \) is \( \alpha \)-policy-compliant.

An example of policy-compliant distributed search is given below.

**Example 19** Assume a P2P network is organized as in Fig. 7.1. A requester node (denoted as \( r \) in Fig. 7.1) sends a query asking for academic files. \( r \) requests that the files do not cost more than \( \$25 \). Accordingly, the search query is formalized as \( SQ = (\{\text{Academic files}\}, \{\text{fee} \leq \$25\}) \). Moreover, \( r \) requests the search to be carried out only within its local area network and hence, defining the following node criteria or policy: \( N\text{Criteria} = \{(\text{Domain} = 192.168.250.X)\} \). This policy indicates that the search is required to be performed within a subnet, the IP address of which ranges from 191.168.250.1 to 192.168.250.255, and also restricts the search zone to be its direct neighbor nodes (\( NHop \) is set to be 1). The corresponding distributed search for this request is shown in Fig. 7.1. It is clear that this query should not be transmitted to node C or node D, because the former is in a different domain from the one specified in \( N\text{Criteria} \), while the latter violates the \( NHop \) restriction.

In this Chapter, we use the notations \( pol_{own} \) and \( SP \) interchangeably to denote the policy of the query generator or the owner of the query, to which the routing of the query needs to comply with. Also, we assume \( pol_{own} \) is a simple conjunction of conditions, instead of a set of clauses for simplicity reasons. We denote \( SetPol_{hist} \) as the set of policies that have been issued by other nodes in the network prior to \( pol_{own} \). \( pol_{hist} \) denotes a policy from the history \( SetPol_{hist} \).

The search query is to be forwarded according to the policy associated with the query throughout the network.

**Definition 16 (Network Path)** A network path \( NP \) is a sequence of connected nodes \( \{ID_{N_1}, \ldots, ID_{N_k}\} \) in a network, where \( ID_{N_i} \) is the ID of a node \( N_i \). We denote a path of nodes compliant with a policy \( pol \) as \( NP_{pol} = \{ID_{N_1}, \ldots, ID_{N_k}\} \), where \( pol \) is the policy to which the nodes in the path adhere to.

In addition to forwarding according to the search query, the nodes can employ any forwarding strategy such as flooding the query to all the neighbors or selectively forwarding to \( k \)-nodes, or so forth. Whenever a query is resolved by finding a
resource, the resource traverses along the same path that the query has taken to find the resource. Note that the required resource may be found, that is, there could be multiple nodes that host the resource originally requested by SQ. For any resource, the resource is obtained along with the path information. The path information includes the set of node IDs in the path through which the resource has been discovered\(^2\).

The history of policies $SetPol_{hist}$ is generated as follows. Every node in the network, given its respective query and associated policy, receives information about the network paths through which the query has found the required resource, where a network path is in the form $\{ID_{N1}, \ldots, ID_{Nk}\}$. A node receives as many network paths as the number of places the query is able to find the resource. Every node saves the set of paths as part of history along with the related policies associated with the search queries. The policy and path history is then periodically distributed throughout the entire network.

### 7.4 Malicious Node Detection

We now describe our routing compliance verification mechanism. Our solution includes three main phases or steps: authenticating paths, path filtering and compliance checking phase. During the authenticating phase, the requester propagates a pair $(SP, SQ)$ of search policy and query to discover the resources satisfying $SQ$, while restricting the query routing only through the nodes satisfying $SP$. Upon resolving the query, the discovered resource $R$, as well as a path proof $PF$, are returned back to requester. In the path filtering phase, the requester conducts filtering of paths or nodes in order to remove nodes that are known to be non-malicious for sure. In the compliance checking phase, the requester takes as input the filtered paths, and verifies the policy-compliance of these paths.

#### 7.4.1 Authenticating Paths

We now describe the path authentication phase of our policy verification mechanism.

\(^2\)Depending on the specific network setup and infrastructure, additional meta data and cryptographic guarantees may be added
Suppose a requester issues a search query \( SQ = (\{c_1, \ldots, c_n\}; \{a_1 \Theta_1 v_1, \ldots, a_m \Theta_m v_m\}) \) (see Def. 7), simultaneously restricting the search query propagation path to be controlled by a search policy \( SP = (NCriteria, NHop) \) (see Def. 9). The following steps are executed.

Assume an exhaustive search across the network is enforced, where the query is forwarded to all suitable nodes. The requester \( r \) firstly evaluates the node criteria on all of its neighbor nodes. Let us consider each individual atomic condition \( cond_i \) in search policy \( SP \). Recall that \( cond_i \) is in the form of \( \text{Att} \ OP \ \text{Value} \) (see Def. 8), and \( h \) is the hash function used for encoding the attribute \( \text{Att} \) in the bloom filter. The requester computes the index \( index_{cond_i} \) of this atomic condition by hashing \( h(\text{Value}) \). For example, for partly bounded conditions (e.g. \( A < 1 \)), the upper or lower bound of the condition are hashed (e.g. \( h(10) \)). The computed \( index_{cond_i} \) is then compared with the positions of the non-full index values of bloom filter in \( conBF_r \). This condition \( cond_i \) is fully satisfied by a bloom filter \( BF \) in \( conBF_r \), when 1) \( BF \) in the exact position equal to \( index_{cond_i} \) have 1, in case of equality conditions (\( \text{Att} = \text{value} \)); 2) or \( BF \) in the positions anywhere before or after \( index_{cond} \) have 1s, in case of partly bounded conditions (\( \text{Att} \geq / \leq \text{value} \)).

For every node (we say \( n_j \)) satisfying \( BFs \), the requester stores a hop item to collect three pieces of information: the identity of the previous node (\( \bot \) for requester), \( id_{prev} \), the identity of current hop \( id_{cur} \) and the identity of next hop \( id_{nxt} \). Each of the hop items is signed under the requester’s signing key \( sk_r \), and encapsulated into a hop list \( L \). The hop list is then to be passed along with the search query and updated and signed by each node, such that the requester can finally verify the authenticity of the propagation path.

During resource discovery, every node \( n_j \) receives \( (SQ, SP) \) as well as \( (L = \langle e_r, \ldots, e_{ij} \rangle, \sigma) \), where \( L \) is the hop list consisting of the hop items the search query has transmitted by and \( \sigma \) is the signature aggregated on these hop items. More precisely, suppose \( n_j \) receives this data from a previous node \( n_i \) (\( n_i \) could be the requester \( n_r \)). \( n_j \) firstly verifies the authenticity of \( L \). This is achieved by completing two operations.

1. \( n_j \) picks out the last item \( e_{ij} = (\cdot, id_i', id_j') \) of \( L \) and checks whether \( id_i' = id_i \) and \( id_j' = id_j \), to guarantee that the search query propagates in a authentic way at this hop from \( n_i \) to \( n_j \);
2. \( n_j \) checks the validity of signature \( \sigma \) on messages \( e_r, \ldots, e_{ij} \) and identities \( id_r, \ldots, id_i \) to guarantee this search query propagated correctly in all of the previous hops.

Operation 1) guarantees that \( n_i \) honestly sends query following the hop information recorded in \( L \), while 2) guarantees that none of the faked hop items exists in previous propagation path. If either of the checks fails, an error is reported and the search for resource is aborted.

Node \( n_j \) then checks the satisfaction of local resources and neighbor node criteria based on bloom filter, using the same approach described above for node criteria evaluation. One of the following two cases could arise:

- **Case 1.** If a query-satisfying resource is found locally by \( n_j \) or none of the neighbor nodes satisfies the search policy, the search is over. A new hop item \( e_{j\perp} = (id_i, id_j, \perp) \) is generated to indicate “end hop”, and signed using \( n_j \)'s signing key \( sk_j \). The signature (on \( e_{j\perp} \)) is then aggregated with the previous aggregated signature \( \sigma \) to generate a new version of \( \sigma \). Finally, after appending \( e_{j\perp} \) with \( L \), the authenticated path \( (L, \sigma) \) is sent back to the requester (either traversing backward through the whole path or directly, depending on the specific query resolution algorithm being adopted).

- **Case 2.** Otherwise, there must exist at least one neighboring node satisfying \( SP \). For every satisfying neighbor node (we say \( n_k \)), \( (L, \sigma) \) is replicated, and another hop item \( e_{jk} = (id_i, id_j, id_k) \) is generated and appended with the new copy of \( L \), to indicate that next hop is \( n_k \). Similar to the first case, after signing \( e_{jk} \) and aggregating the new signature into (the copy) \( \sigma \), the updated \( (L, \sigma) \) is then sent to node \( n_k \), along with the query-policy pair \( (SQ, SP) \).

Note that, although we present for the case of exhaustive search, our scheme can be easily adapted to support any routing protocol (e.g. random walk). As compared to existing protocols, in our scheme, the requester is able to restrict the query to be forwarded only through certain nodes by defining a policy over the query.

**Example 20** Fig. 7.2 shows a toy example for the process of resource discovery. Two neighbor nodes (\( n_1 \) and \( n_3 \)) are respectively sent the resource query from \( n_r \).
For the node $n_3$, a satisfying resource is found locally, and returned back to requester along with the authenticated path. For the node $n_1$, it forwards the query to a next policy satisfying node $n_2$, which does not have any policy satisfying neighbor nodes. So, another path of authenticated nodes is sent back to the requester following this path: $n_2 \rightarrow n_1 \rightarrow n_r$.

Upon receiving the set of authenticated paths, the requester checks the authenticity of the paths. The requester examines the hop list $L$, in specific, whether the concatenation of nodes is correct. For example, for any two continuous hop items (we say $e_{ij}$ and $e_{jk}$), the requester checks whether $id_{cur}$ in $e_{ij}$ equals $id_{prev}$ in $e_{jk}$ and whether $id_{nxt}$ in $e_{ij}$ equals $id_{cur}$. Then, it verifies the validity of the aggregated signature $\sigma$ using all the identities stored in the current node entry of hop items in $L$. This is achieved by examining whether $\sigma$ is a valid aggregated signature on a series of messages $e_r, \ldots, e_{jk}, e_{k\perp}$ by the public identities $id_r, \ldots, id_j, id_k$, where $id_i$ $(i = r, \ldots, j, k)$ is the identity of node generating and signing the hop item $e_i$. If either of the verification steps fails, an error is reported. The final result of this phase is a set of authenticated paths, $P_{\text{pol}_{\text{own}}}$, traversed by the search query $SQ$, where $\text{pol}_{\text{own}}$ is the policy of the owner of the query or the requester.

### 7.4.2 Path Filtering

The second step is to filter paths from the set of authenticated paths received from the previous step. A query owner with his policy $\text{pol}_{\text{own}}$ obtains a set of paths $P_{\text{pol}_{\text{own}}}$ traversed by his query $SQ$, from the previous step (Section 7.4.1).
After obtaining $P_{pol_{own}}$, the query owner tests for malicious paths or nodes among $P_{pol_{own}}$. An intuitive approach would be to verify policy compliance for every path and every node in the path. This is however a computationally expensive method, that could potentially require an exponential number of nodes to be checked. We overcome this problem by comparing the paths taken by the query in question to the recent network paths traversed by similar queries. Given a search query $SQ$, each of its returned paths will eventually be classified into one of three categories: non-malicious (NM), possibly malicious (PM), and malicious (M). We leverage the path history of previous policies to classify paths from $P_{pol_{own}}$. We can obtain reliable path history by employing methods that are used in reliable reputation systems [134].

To check for policy similarity, we adopt the following notion of policy subsumption, building on the well-known subsumption relation between two constraints [135]. The subsumption relation states that a computable subsumption $\Rightarrow$ relation on two constraints $cond_1, cond_2$ is true if all substitutions (i.e. values) that satisfy $cond_1$ also satisfy $cond_2$.

**Definition 17 (Policy Subsumption)** Given two policies $pol_1$ and $pol_2$, we say that $pol_1$ is subsumed by $pol_2$, denoted as $pol_1 \Rightarrow pol_2$ if there exists for every condition $cond_i$ in $pol_1$ there exists at least one condition $cond_j$ in $pol_2$ such that either $cond_i = cond_j$ or $cond_i \Rightarrow cond_j$.

Note that in the definition above, if $pol_2$ has extra conditions it does not matter, as the policy being subsumed ($pol_1$) is still fully verified upon verifying $pol_2$. We provide two examples for better understanding of policy subsumption.

**Example 21** The following two cases to demonstrate policy subsumption. As the first case, consider two simple policies $pol_1 = \{a > 20 \land b > 45 \land c = 97\}$, and $pol_2 = \{a > 10 \land c > 30\}$. $pol_1$ is a subsumption of $pol_2$. This is because $pol_1$ has a condition $\{a > 20\}$ and a condition $\{c = 97\}$, and a node which satisfies these conditions also satisfies the conditions $\{a > 10\}$ and $\{c > 30\}$ in $pol_2$. As the second case, consider two other simple policies $pol_1 = \{a = 50 \land b > 45 \land c = 97\}$, and $pol_2 = \{a = 50 \land c > 30\}$. $pol_1$ is a subsumption of $pol_2$. This is because both $pol_1$ and $pol_2$ have an identical condition $\{a = 50\}$, and a node that satisfies the condition $\{c = 97\}$ in $pol_1$ also satisfies the condition $\{c > 30\}$ in $pol_2$. 
Path filtering includes two steps. We first compare the policy of the query owner \(pol_{own}\), with all the existing policies in the history policy set \(SetPol_{hist}\). All the policies in \(SetPol_{hist}\) which have a subsumption relation with policy \(pol_{own}\) as per Def. 17 are extracted and stored in \(SetPol_{hist}'\). Next, we compare the paths generated under \(pol_{own}\) and the paths generated under the set of policies \(SetPol_{hist}'\), and filter non-malicious paths, as described next.

### 7.4.2.1 Policy and Path Comparison

We match paths of the history with the set of paths generated by \(SQ\) under the constraints imposed by policy \(pol_{own}\). The matching process results in placing nodes from \(SetPol_{hist}'\) into one of the three different classes namely, non-malicious, malicious, and maybe-malicious. Two cases are possible, depending on the subsumption relation between the owner’s policy and the policy being matched:

1. Let \(pol_{hist}_{i}\) be a policy from \(SetPol_{hist}'\) which is a subsumption of the owner’s policy \(pol_{own}\) per Def. 17, that is \(pol_{hist}_{i} \Rightarrow pol_{own}\). Let \(P_{pol_{own}}\) denote all the paths traversed according to \(pol_{own}\) and \(P_{pol_{hist}_{i}}\) be the paths traversed according to \(pol_{hist}_{i}\), respectively.

   - **Non-malicious nodes** are nodes belonging to any of the paths in \(P_{pol_{own}}\) that also appear in any path in the path set \(P_{pol_{hist}_{i}}\).
   - **Possibly-malicious nodes**, are nodes belonging to any of the paths in \(P_{pol_{own}}\) that do not appear in any path in the path set \(P_{pol_{hist}_{i}}\).

   Note that the union of the two malicious and possibly malicious sets include all nodes in \(P_{pol_{own}}\).

Let \(pol_{hist}_{i}\) be a policy from \(SetPol_{hist}'\) which subsumes the owner’s policy \(pol_{own}\) per Def. 4, that is \(pol_{own} \Rightarrow pol_{hist}_{i}\). Let \(P_{pol_{own}}\) denote all the paths traversed according to \(pol_{own}\) and \(P_{pol_{hist}_{i}}\) be the paths traversed according to \(pol_{hist}_{i}\), respectively.

   - **Malicious nodes** are nodes belonging to any of the paths in \(P_{pol_{own}}\) that do not appear in any path in the path set \(P_{pol_{hist}_{i}}\).
   - **Possibly-malicious nodes** are nodes belonging to any of the paths in \(P_{pol_{own}}\) that appear in any path in the path set \(P_{pol_{hist}_{i}}\).
Again, the two sets include all nodes in $P_{pol_{own}}$.

Note that, policy subsumption checks are carried out for all policies $pol_{hist_x}$ in $SetPol'_{hist}$. If there does not exist even a single subsumption relation between the policies in the history and the owner’s policy $P_{own}$ (for example if the node has only recently joined the network or the whole network is in itself new), all the paths taken by $SQ$ of the requester $P_{pol_{own}}$ need to be verified.

**Example 22** Figure 7.3 shows an example view of a network. Let $R$ be the requester node which sends out the query. Let the paths represented by red arrows be the paths traversed by the query associated with policy $pol_{own}$, and the paths represented by blue arrows be the paths traversed by a policy in the history $pol_{hist_i}$. Let the two policies hold the following subsumption relation $pol_{hist_i} \Rightarrow pol_{own}$. All the nodes in the path taken by $pol_{own}$ \{R,F,G,H,I\} need not be verified as they have been traversed by $pol_{hist_i}$, as can seen by the blue arrows. In the path \{R,J,K,L,M\}, nodes L and M need not be verified as they have been traversed by $pol_{hist_i}$, whereas nodes J and K need to be verified, as they have not been traversed by $pol_{hist_i}$. Since, the path \{A,B,C,D,E\} has not been traversed by $pol_{hist_i}$, all the nodes A, B, C, D, and E need to be verified.
7.4.3 Compliance Checking

The third step is to check whether the filtered paths are policy compliant. This step is of course necessary as some nodes may have passed the message along without meeting the policy conditions. Generally, our algorithm is based on examining the policy satisfiability of nodes in the propagation path using attribute-based encryption.

7.4.3.1 Baseline Method

Without loss of generality, we assume that the requester wants to verify the attributes or properties of a node $N_i$, called target node. Our baseline method relies on random message challenges. The requester creates two random messages $m_0$ and $m_1$, and flips a bit coin $b \in R \{0, 1\}$ for encrypting $m_0, m_1$ respectively under the policy $pol_{own}$ and its complementary policy $\overline{pol}_{own}$ using CP-ABE. That is, the requester encrypts $m_0$ under $pol_{own}$ for obtaining $[m_0]_{pol_{own}}$, while $m_{1-b}$ under
\(\overline{pol_{own}}\) for obtaining \([m_{1-b}]_{\overline{pol_{own}}}\). Both ciphertexts \([mb]_{pol_{own}}\) and \([m_{1-b}]_{\overline{pol_{own}}}\) are sent (in random order) to the target node for decryption.

According to our assumption, each node has a private key with respect to its attributes, which it obtains from a certificate authority. Thus, the target node can only decrypt either \([mb]_{pol_{own}}\) or \([m_{1-b}]_{\overline{pol_{own}}}\). This is because the underlying policies in \([mb]_{pol_{own}}\) and \([m_{1-b}]_{\overline{pol_{own}}}\) are complementary: If the attributes of the target node do not satisfy one underlying policy, they must satisfy the other policy. We employ complementary ciphertexts so that the node that is being verified does not lie by responding that it is unable to decrypt both the ciphertexts. We, hence, prevent the nodes in the network to be lazy. The target node attempts to decrypt both ciphertexts (but it is only able to decrypt one message among the two), and returns back the results \(m'_0\) and \(m'_1\) (in the same order) to requester. Requester verifies if \(mb = m'_b\) and \(m_{1-b} \neq m'_{1-b}\). If both the conditions hold, then the target node \(N_i\) satisfies the policy \(pol_{own}\). Thus, the node \(N_{i-1}\), which is the neighboring node and has forwarded query to \(N_i\), is considered to be non-malicious and honest as it correctly forwarded the query according to the policy established by the requester, without any disruption. Otherwise, the node \(N_i\) does not satisfy the policy and hence \(N_{i-1}\) maliciously forwarded the query to a non-satisfying node \(N_i\), hence, \(N_{i-1}\) is a malicious node.

We note that our baseline method allows to verify the policy compliance of a certain node with significant probability. The false positive exists only when the non-satisfying target node correctly guesses the random messages, which is negligible provided that the CP-ABE scheme is secure.

One limitation of this baseline method is its overhead. Specifically, our baseline method requires to verify every node (in the path) by encrypting random messages with the original and complementary policies, introducing a massive overhead for the requester. To address this issue, we propose an improved approach, which aims at reducing the computational overhead for the requester node.

### 7.4.3.2 Offloading Partial Computation to Verified Nodes

Our improved approach is based on reusing the verified nodes in the path. Specifically, if we have already verified the honesty of one node, we then trust that it can also honestly perform some (carefully crafted) delegated computations. In other
words, the idea underlying this approach is to offload the computation to verified nodes so that the requester’s overhead is minimized.

A naïve approach is to directly send the random messages \( m_0 \) and \( m_1 \) to a verified node for encryption (with \( \text{pol}_{\text{own}} \) and \( \overline{\text{pol}}_{\text{own}} \)) which eliminates the ABE computation at requester side. However, this approach is infeasible, because it requires to fully trust the verified node. That is, the naïve approach is based on the assumption that the verified node would not leak the challenge messages to target nodes. We argue this assumption is unconvincing. This is because in policy compliant verification, we can only verify the honesty of node, which is weaker than trust. Even if a verified node is honest, it can still extract or leak secret information to others. Hence the goal of our optimization is offloading computation to verified nodes, while preserving the confidentiality of both \( m_0 \) and \( m_1 \).

As mentioned before, the policy \( \text{pol}_{\text{own}} \) is in a conjunctive form, i.e., \( \text{pol}_{\text{own}} = \text{cond}_1 \land \text{cond}_2 \land \ldots \land \text{cond}_n \), where \( \text{cond}_i \) is the condition. We use \( \gamma_i \) to denote the satisfying attribute set in which the attributes satisfy \( \text{cond}_i \). We introduce a few notations in order to elaborate on our approach.

Recall that each node \( N_i \) has an attribute private key as \( sk_i \), constructed as \( (g^{(\alpha+r_i)/\beta}, \{g^{r_i H(j)^{r_i}} \}_{j \in \omega_i \cup ID_i}) \), where \( \omega_i \) is the attribute set of \( N_i \) and \( ID_i \) is the ID of node \( N_i \), \( \alpha, \beta \) are the master secrets identical with each node, \( r_i \in R \mathbb{Z}_p \) is the random number per key, \( r_j \in R \mathbb{Z}_p \) (\( j \in \omega_i \cup ID_i \)) is the random number per attribute (in a certain secret key). The parameters \( g \in \mathbb{G}, g^\beta \) and \( e(g,g)^\alpha \) are published after system setup.

Assuming the honesty of a node \( N_k \) (called the verified node) is verified, we can offload the ABE computation [136] to \( N_k \) as shown in Figure 7.5. Specifically, after generating the random test messages \( m_0, m_1 \in \mathbb{G}_T \), the requester (i.e. the query owner in the first verification step) picks integers \( s, s_1, s_2, s', s'_1, s'_2 \in R \mathbb{Z}_p \), with \( s = s_1 + s_2 \) and \( s' = s'_1 + s'_2 \). Next, the requester sends the target policy \( \text{pol}_{\text{own}} \) as well as the delegating secrets \( s_1 \) and \( s'_1 \) to the verified node for partial ABE encryption on both \( m_0 \) and \( m_1 \).

Upon receiving \( (\text{pol}_{\text{own}}, s_1, s'_1) \), the verified node respectively uses \( s_1 \) and \( s'_1 \) to compute partial ABE ciphertexts with respect to \( \text{pol}_{\text{own}} \) and \( \overline{\text{pol}}_{\text{own}} \). We illustrate this offloaded computation to the verified node in Figure 7.6. Precisely, policy \( \text{pol}_{\text{own}} \) is normalized into a tree-based structure. The root node is an “\( \land \)” gate connecting \( n \) sub-trees (corresponding to each condition \( \text{cond}_i \)). To produce a (par-
\[ \text{Requester} \quad \text{Verified Node} \]

<table>
<thead>
<tr>
<th>Pick ( s, s_1, s_2, s', s'_1, s'_2 \in \mathbb{Z}_p ) with</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s = s_1 + s_2 ) and ( s' = s'_1 + s'_2 )</td>
</tr>
</tbody>
</table>

\[
\text{(pol}_{\text{own}}^s, s_1, s'_1) \quad \text{Compute pol}_{\text{own}}^s
\]

\[
\text{Compute } \{(g^{s_1i}, \{H(\theta)^{s_1i}\}_{\theta \in \gamma_i})\}_{i=1}^n
\]

\[
\text{and } \{(g^{s'_1i}, \{H(\theta)^{s'_1i}\}_{\theta \in \gamma'_i})\}_{i=1}^{n'}
\]

\[
\{(g^{s_1i}, \{H(\theta)^{s_1i}\}_{\theta \in \gamma_i})\}_{i=1}^n
\]

\[
\text{and } \{(g^{s'_1i}, \{H(\theta)^{s'_1i}\}_{\theta \in \gamma'_i})\}_{i=1}^{n'}
\]

\[
\text{Compute } (g^{\beta s}, m_0 e(g, g)^{\alpha s}, g^{s_2}, H(ID_{k+2})^{s_2})
\]

\[
\text{and } (g^{\beta s'}, m_1 e(g, g)^{\alpha s'}, g^{s'_2}, H(ID_{k+2})^{s'_2})
\]

\[\text{Output } [m_0]_{\text{pol}_{\text{own}}^s \land c_0} \text{ and } [m_1]_{\text{pol}_{\text{own}}^s \land c_0}\]

**Figure 7.5: Workflow of Offloading Partial Computations to Verified Nodes**

Partial ciphertext with respect to \( \text{pol}_{\text{own}}^s \) (using \( s_1 \)), verified node firstly computes the satisfying attribute set \( \gamma_i \) and generates the ciphertext \( (g^{s_1i}, \{H(\theta)^{s_1i}\}_{\theta \in \gamma_i}) \) of each condition \( \text{cond}_i \) for \( i = 1, 2, \ldots, n \), where \( \sum_{i=1}^n s_{1i} = s_1 \). Finally, we can get the partial ciphertext of policy \( \text{pol}_{\text{own}}^s \) as \( \{(g^{s_1i}, \{H(\theta)^{s_1i}\}_{\theta \in \gamma_i})\}_{i=1}^n \). Similarly, verified node also computes the partial ciphertext with respect to \( \text{pol}_{\text{own}}^s \). It firstly calculates the complementary policy \( \text{pol}_{\text{own}}^\bot \) of \( \text{pol}_{\text{own}}^s \), normalizes \( \text{pol}_{\text{own}}^\bot \) into a tree-based structure and then uses \( s'_1 \) to compute the partial ciphertext \( \{(g^{s'_1i}, \{H(\theta)^{s'_1i}\}_{\theta \in \gamma'_i})\}_{i=1}^n \), where \( \sum_{i=1}^n s'_{1i} = s'_1 \), \( \gamma'_i \) is the satisfying attribute set in \( \text{pol}_{\text{own}}^\bot \) and \( n' \) is the number of conditions involved in \( \text{pol}_{\text{own}}^\bot \). Both (partial) ciphertexts are sent back to requester. The requester uses \( s_2 \) and \( s'_2 \) respectively to compute the partial ciphertexts \( (g^{s_2}, H(ID_{k+2})^{s_2}) \) and \( (g^{s'_2}, H(ID_{k+2})^{s'_2}) \) where \( ID_{k+2} \) is the ID of the node \( N_{k+2} \) whose attribute satisfaction is to be checked. After checking the satisfaction of policy by node \( N_{k+2} \), we will be able to detect the forwarding compliance of node \( N_{k+1} \). Requester also computes the encryptions \( (g^{\beta s}, m_0 e(g, g)^{\alpha s}) \) and
Figure 7.6: Offloaded Computations to the Verified Nodes

\((g^βs', m_1 e(g, g)^{αs'})\). Finally the ABE ciphertexts for \(m_0\) and \(m_1\) are output as

\[
[m_0]_{\text{pol}_{\text{own}} \wedge c_0} = (\text{pol}_{\text{own}} \wedge c_0, g^{s_2}, H(ID_{k+2})^{s_2}, m_0 e(g, g)^{αs}, g^βs, \\
\{(g^{s_i}, \{H(θ)^{s_i}\}_{θ ∈ γ_i})\}_{i=1}^n) \tag{7.1}
\]

and

\[
[m_1]_{\overline{\text{pol}}_{\text{own}} \wedge c_0} = (\overline{\text{pol}}_{\text{own}} \wedge c_0, g^{s_2'}, H(ID_{k+2})^{s_2'}, m_1 e(g, g)^{αs'}, g^βs', \\
\{(g^{s_i}, \{H(θ)^{s_i}\}_{θ ∈ γ_i})\}_{i=1}^{n'}) \tag{7.2}
\]

where \(c_0\) is the introduced clause satisfied by the attribute \(ID_{k+2}\). Both outputs are sent to the target node \(N_{k+2}\) for challenge.

Note that \([m_0]_{\text{pol}_{\text{own}} \wedge c_0}\) and \([m_1]_{\overline{\text{pol}}_{\text{own}} \wedge c_0}\) follow the same structure of ciphertexts in [131] with respect to policy \(\text{pol}_{\text{own}} \wedge c_0\) and \(\overline{\text{pol}}_{\text{own}} \wedge c_0\). At the same time, the node \(N_{k+2}\) has a private key related to \(ω_i \cup \{ID_{k+2}\}\).

Since the verified node cannot access the original test messages \(m_0, m_1\), it can only “aid the encryption process”, without compromising the protocol. Therefore, our offloading method would not degrade the security guarantee of the baseline approach. Also, there is no scope of collusion attacks for this approach as the messages are encrypted with the policy which contains the ID of the recipient node. Any recipient node will have a unique private key. A recipient node which did not satisfy the policy cannot collude with another node that satisfies the policy to decrypt the messages for it.
7.4.3.3 Reusing ABE Ciphertexts

We now discuss an alternative approach which is more efficient compared to the offloading of partial computation approach. The intuition underlying this alternative approach is reusing the ABE ciphertexts. In particular, we observe that all the test ciphertexts (i.e., encryption of \( m_0 \) and \( m_1 \)) are intended for the same policy, and the most part of the ciphertexts could be re-used for saving the computations for the following ciphertexts. We describe our approach as follows.

Let us assume that a node \( N_{i-1} \) which is a neighbor of the requester, in one of the paths in \( P_{\text{pol}} \), has correctly forwarded the query according to the policy and hence, is considered honest. Since \( N_{i-1} \) is honest, the requester delegates \( N_{i-1} \) the task of verifying the policy compliance of \( N_i \), the neighbor of \( N_{i-1} \). Assume the requester has computed the initial CP-ABE ciphertexts \( (m_0 e(g,g)^{\alpha s}, g^b s, \{(g^s, H(\theta)^s)\}_{\theta \in \gamma_i, i=1,...,n}) \) and \( (m_1 e(g,g)^{\alpha s'}, g^b s', \{(g^{s'}, H(\theta)^{s'})\}_{\theta \in \gamma_i', i=1,...,n'}) \) with respect to \( \text{pol}_{\text{own}} \) and \( \overline{\text{pol}}_{\text{own}} \), where \( \alpha, \beta \) are the master secrets, \( s, s' \) are the randomness satisfying \( \sum_{i=1}^{n} s_i = s \) and \( \sum_{i=1}^{n'} s_i' = s' \), and \( \gamma_i \) and \( \gamma_i' \) respectively denote the attribute set satisfying the conditions \( \text{cond}_i \) and \( \text{cond}_i' \) in \( \text{pol}_{\text{own}} \) and \( \overline{\text{pol}}_{\text{own}} \).

To generate the ciphertexts for the following nodes, the requester sends the delegatee node \( N_{i-1} \) the two ciphertexts as described above. The delegatee node \( N_{i-1} \) can leverage the malleable property of CP-ABE and modify existing ciphertexts. Specifically, delegatee keeps the other components of ciphertexts, but multiplies \( m_0 e(g,g)^{\alpha s} \) with a random message \( m'_0 \) and \( m_1 e(g,g)^{\alpha s'} \) with \( m'_1 \). This way generates ciphertexts for a new pair of random messages \( m_0 m'_0 \) and \( m_1 m'_1 \), that could be used for the next node’s verification. We note that since \( m'_0 \) and \( m'_1 \) are random, \( m_0 m'_0 \) and \( m_1 m'_1 \) are random as well, the target node cannot get knowledge of the challenge messages unless correctly decrypts them. The delegatee node \( N_{i-1} \) then sends the new \( [m_0 m'_0]_{\text{pol}_{\text{own}}} \) and \( [m_1 m'_1]_{\overline{\text{pol}}_{\text{own}}} \) to the node \( N_{i+1} \) whose attributes are to be verified in order to verify the policy compliance of its neighboring node \( N_i \). This procedure of verifying the attributes of alternate nodes continues until every node in the set of paths from the path information is verified.

In this approach, the requester just needs to compute one pair of CP-ABE encryption, each for \( \text{pol}_{\text{own}} \) and \( \overline{\text{pol}}_{\text{own}} \) respectively, and the the ciphertexts resulting from the two CP-ABE encryptions are modified by the delegatee nodes to carry on the verification of rest of the nodes in the path. Nevertheless, we note that this
approach essentially trades security for efficiency, and could suffer from a single-point-of-failure. In particular, since our approach re-uses the same randomness (i.e., $s$ and $s'$) involved in the ABE ciphertexts for verification, it inevitably suffers from failure if nodes collude on $s$ and $s'$. If some node passes the verification process, but leaks the common randomness in the form of $e(g, g)^{as}$ or $e(g, g)^{as'}$, all the following nodes can thus directly recover the message without any decryption.

Our schemes would work for both static and dynamic networks. In the context of dynamic networks, when a new node is added to the network, the requester node need not verify the compliance of this node. The reason is intuitive; the newly added node could not have participated in the query forwarding procedure prior to compliance verification. If a node has participated in the query forwarding process but disconnects from the network in order to not participate in the verification process, we flag such a node malicious.

### 7.4.4 Alternate Probabilistic model

We present an alternate probabilistic model for reducing computational overhead with respect to the requester. Instead of verifying the policy compliance of all the filtered paths, we determine the number of nodes to verify which would suffice, without the requester missing catching a malicious node.

### 7.4.5 Determining the Number of Nodes to verify

In this section, we analyze the number of nodes the requester should check to achieve $\alpha$-compliance (i.e., at least a percentage $\alpha$ of the nodes in path satisfies the policy), for both cases.

We model our problem as follows. Suppose a query SQ has been resolved by a given $Path$, where $|Path| = n$ indicates that $n$ unique nodes were involved during this search. Assume that an arbitrary number of nodes $m$ ($m < n$) in the path does not satisfy our policy requirement. Our aim is to estimate the number of nodes to be checked to detect this dishonest behavior with a confidence greater than $\alpha^3$. Note that our detection model follows the “once for all” philosophy. That is, if only one non-compliant node is found, we consider the full search dishonest.

---

3The symbol $\alpha$ is abused here to denote the confidence threshold in dishonesty detection
Non-iterative Probabilistic Model. In the first verification model, discussed in Section 7.4.3, we assume that the requester generates all the target nodes (constitute the target set \( \text{target} \)) to check at once. The requester does not obtain any feedback about the intermediate checking results before it generates all the target nodes. Suppose the number of nodes to be checked is \( x \) (i.e., \( |\text{target}| = x \) in this case)\(^4\).

Given known values of \( n \) and \( m \), we can compute the minimum value of \( x \) by resolving the following inequality which contains only \( x \) as an unknown value.

\[
1 - \frac{\binom{n-m}{x}}{\binom{n}{x}} \geq \alpha \Rightarrow 1 - \frac{(n-m)!(n-x)!}{n!(n-m-x)!} \geq \alpha
\]  
(7.3)

The equation is easily understood. Our problem consists of selecting \( x \) nodes at once from \( n \) path nodes to be checked, with \( \binom{n}{x} \) possibilities. Assume the \( x \) target nodes to check are all selected from the \( n-m \) satisfying nodes, which has \( \binom{n-m}{x} \) possibilities.

Then, we can compute the probability of not detecting a dishonest node by randomly checking \( x \) nodes as \( \frac{n-m}{\binom{n}{x}} \). Thus, \( 1 - \frac{\binom{n-m}{x}}{\binom{n}{x}} \) is the probability of detecting any non-compliant node by checking \( x \) nodes.

Iterative probabilistic Model. In the iterative probabilistic model presented in Section 7.4.3, we assume that the requester is able to adaptively generate the target node to be checked. Since in this scheme, the requester knows the intermediate results obtained from previous checks, it can decide accordingly which nodes are to be checked. Suppose the number of nodes to be checked is \( x \). Suppose \( A_k \) is the probability for detecting dishonesty by checking \( k \) nodes in \text{Path}. It is clear that \( A_1 = \frac{m}{n} \) and

\[
A_k = (1 - \sum_{i=1}^{k-1} A_i) \frac{m}{n-k+1}
\]  
(7.4)

In what follows we explain the above equation. Since the requester can adaptively generate the target node, the probability of selecting the satisfying node is not identical each time a check is performed. For example, if the requester successfully selects a non-compliant node the first time, and detects a dishonest node,

\(^4\)We need to restrict that \( x \leq n - m \) in our models. This is because, we can always detect non-compliant nodes if we test more than \( n-m \) nodes.
then $A_1 = \frac{m}{n}$. The next time, in the adaptive case, the probability of catching a non-compliant node becomes $\frac{m}{n-1}$, because one satisfying node has been verified already, and should be removed for all the subsequent selections. Thus, the probability $\frac{m}{n-1}$ holds in the case that the non-compliant node is not caught in the first time, having probability $1 - A_1$.

Accordingly, the probability of catching non-compliant nodes in the second check is computed as $A_2 = (1 - A_1)\frac{m}{n-1}$. Recursively, for $A_k$, $(1 - \sum_{i=1}^{k-1} A_i)$ is the probability that any non-compliant node is not caught in the first $k-1$ times of checking. At the $k$th round, $k-1$ satisfying nodes are removed due to the inability of catching non-compliant nodes in the first $k-1$ times, and thus the probability of catching a dishonest node for the $k$ th time is $\frac{m}{n-k+1}$. Finally, we can get the probability $A_k = (1 - \sum_{i=1}^{k-1} A_i)\frac{m}{n-k+1}$.

We are to solve the following inequality with respect to unknown $x$:

$$\sum_{i=1}^{x} A_i \geq \alpha \quad (7.5)$$

Interestingly, although our proposed non-iterative and iterative models work in a different manner, they achieve the same probability of catching dishonest nodes, assuming they check the same number of nodes. This finding can be demonstrated by solving the general formula (7.4) and comparing the result $\sum_{i=1}^{x} A_i$ with the probability of non-iterative model (i.e., the left part of inequality (1)). In what follows, we provide a detailed proof that the left part of Equation (3) equals to the left part of Equation (1). That is, for a fixed value of $x$, $\sum_{i=1}^{x} A_i = 1 - \frac{(n-m)!(n-x)}{n!(n-m-x)!}$ where $A_k = (1 - \sum_{i=1}^{k-1} A_i)\frac{m}{n-k+1}$ for $k = 2, 3, \ldots, x$.

Without loss of generality, we denote $\text{prob}_{\text{nonitera}}(k) = 1 - \frac{(n-m)!(n-x)!}{n!(n-m-k)!}$ indicating the probability of catching dishonest nodes in the non-iterative model when checking $x$ nodes. Similarly, $\text{prob}_{\text{itera}}(x) = \sum_{i=1}^{x} A_i$ is the probability of catching dishonest nodes in the interactive model. It is clear that in the iterative model $A_x = \text{prob}_{\text{itera}}(x) - \text{prob}_{\text{itera}}(x-1)$, and we substitute this expression into equation (7.4) to obtain

$$\text{prob}_{\text{itera}}(k) - \text{prob}_{\text{itera}}(k-1) = (1 - \text{prob}_{\text{itera}}(k-1))\frac{m}{n-k+1}$$

$$1 - \text{prob}_{\text{itera}}(k) = (1 - \frac{m}{n-k+1})(1 - \text{prob}_{\text{itera}}(k-1)) \quad (7.6)$$
Then, our aim is to recursively solve the equation (7.6) to obtain \( \text{prob}_{\text{itera}}(k) \), with the condition that \( \text{prob}_{\text{itera}}(1) = A_1 = \frac{m}{n} \). To this end, we iterate the variable \( k \) in equation (7.6) from \( k \) down to 2 to get a series of \( k - 1 \) equations as follows.

\[
1 - \text{prob}_{\text{itera}}(k) = \left(1 - \frac{m}{n-k+1}\right)(1 - \text{prob}_{\text{itera}}(k-1))
\]

\[\ldots \ldots\]

\[
1 - \text{prob}_{\text{itera}}(2) = \left(1 - \frac{m}{n-2+1}\right)(1 - \text{prob}_{\text{itera}}(1))
\]

We then multiply these \( k - 1 \) equations together to get

\[
1 - \text{prob}_{\text{itera}}(k) = (1 - \text{prob}_{\text{itera}}(1)) \times \frac{\Pi_{i=2}^{k}(n - i + 1 - m)}{\Pi_{i=2}^{k}(n - i + 1)}
\]

(7.7)

It is clear that \( \Pi_{i=2}^{k}(n - i + 1 - m) = (n - m - k + 1) \ldots (n - m - 1) = \frac{(n-m-1)!}{(n-m-k)!} \) and \( \Pi_{i=2}^{k}(n - i + 1) = (n - k + 1) \ldots (n - 1) = \frac{(n-1)!}{(n-k)!} \). We further substitute both equations as well as \( \text{prob}_{\text{itera}}(1) = \frac{m}{n} \) into equation (7.7).

\[
1 - \text{prob}_{\text{itera}}(k) = 1 - \frac{n - m (n-m-1)(n-k)!}{n(n-1)!(n-m-k)!}
\]

\[
\text{prob}_{\text{itera}}(k) = 1 - \frac{(n-m)! (n-k)!}{n!(n-m-k)!} = \text{prob}_{\text{nonitera}}(k)
\]

In Figure 7.7, we provide some numerical examples about probability of detecting dishonest nodes proportional to the number of nodes to be verified. In this numerical example, the path consists of 100 loop-free nodes. We present three cases wherein we assume there are respectively 15%, 20% and 25% nodes in the
path that do not satisfy the requester’s policy, and show the confidence of detecting dishonest nodes. It is clear from Figure 7.7 that, in order to achieve a detection confidence of 0.9, we only need to check a small set of nodes in the path. Precisely, the requester will need to check 8, 11 and 15 nodes in the 15%, 20% and 25% case for ensuring 0.9 confidence. Only a subset of the nodes in the path are to be verified for high confidence results, and we can check only part of the nodes in the path to save computing and networking resources.

7.4.6 Attacks and Countermeasures

In this subsection, we outline some common attacks against our scheme, along with some potential countermeasures.

- In the first potential attack, since the metadata (i.e., the hop list \( L \) and aggregated signature \( \sigma \)) in resource discovery phase would be transferred back to the requester, a malicious node could record this information, and use it for launching replay attack in the future. For example, suppose a requester requests for resources, with the same policy twice. Since the policy is the same, it would follow the same path. A malicious node could record the hop list as well as the aggregated signature returned back in the first time, and use it for cheating the next time of search. Specifically, in the second time of search, even if the malicious node does not forward the query to the policy satisfying neighbor node, it can send back the recorded hop list and aggregated signature in the first time to cheat that it has forwarded the query in the correct way. A simple countermeasure to this attack is to append another time entry, a nonce, in the hop item in \( L \) and ask each node to sign on the hop item including not only previous, current and next node, but also a time period to distinguish the signatures for two times of search. In this way, during verification, the requester can easily detect the old metadata and catch the dishonest node.

- The second potential attack originates from the fact that a malicious node is lazy, which does not forward the search query to satisfying neighbor node and cheat that none of the neighbor nodes satisfy the policy. Suppose \( A \) is a lazy node adjacent to the requester \( r \), and we can detect this lazy node in the following ways. The requester node \( r \) compares the policy with the
bloom filter, $BF$, received from a neighbor node $A$, and notes down the value $x$ that lies in the corresponding positions related to the policy. This value gives the number of $A$’s neighbor nodes satisfying the search policy. The requester expects to receive $x$ aggregated signatures from its neighbor $A$. If it did not receive at least $x$ aggregated signatures (and an exhaustive search was implemented), then it concludes that $A$ is lazy or that it has dropped $SQ^5$.

- In the third potential attack, a malicious node (say $A$), upon receiving a search query, could cheat that none of the neighbor nodes satisfies the policy and return back the updated $(L, \sigma)$ to requester, but forwards the search query to a policy unsatisfying neighbor node (say $B$), which will drop the forwarding of the query and/or does not send the aggregated signature to the requester. We point out that, this attack is challenging to be detected, since the malicious nodes ($A$ and $B$) are adjacent. In this case, some additional controls are needed, in addition to the scheme discussed in this paper. A simple approach to fully prevent the policy unsatisfying nodes from accessing the search query, is for the requester to encrypt the content of the query using attribute-based encryption, such that only the policy satisfying nodes are able to decrypt and access the query. In this way, even if a policy unsatisfying node receives the search query, it is not able to learn the content of query.

In this chapter, we presented an efficient and yet secure solution for policy compliance verification of distributed search in networks hosting content, such as peer-to-peer networks. Our proposed solution includes methods to both reduce the number of nodes to be verified, and make the actual node verification mechanism efficient and yet secure by leveraging a secure cryptographic scheme.

---

⁵If a non-exhaustive search algorithm is used, the requestor would expect at least $k$ responses, where $k$ is to be determined according to the routing scheme employed by the network.
Chapter 8 | Experiments and Complexity

In this chapter, we analyze the complexity of our proposed centralized framework and discuss various experiments we conducted on the ranking (Chapter 4), the privacy-preserving web service selection (Chapter 5), P2P network routing (Chapter 6) and policy-compliance verification (Chapter 7) approaches.

8.1 Complexity

Concerning the ranking component of our framework, we analyze the complexity of the multi-criteria ranking algorithm including the various selection strategies proposed. With respect to the Web Service selection, we consider separately (a) the search criteria and (b) provisioning rules enforcement.

**Ranking Complexity** We now evaluate the complexity of the three ranking strategies. In general, the time complexity of the algorithms belonging to the first strategy is $O(N)$ where $N$ is the number of services or nodes in the graph. This is mainly because, we traverse each node in the graph to assign satisfaction levels, contributing to $O(N)$ time complexity. Our second strategy has a time complexity of $O(C \times N)$, where $C$ is the number of user criteria. The complexity involves the number of user criteria because we run the algorithm for as many iterations as the number of user criteria. As the number of user criteria increases, the first strategy performs better compared to the second strategy, in terms of time. The complexity of the third strategy depends on the strategy (first or second) that is employed within it. That is, the complexity is either $O(N)$ or $O(C \times N)$.

**Private Search Criteria Complexity** The complexity analysis of our private search component is based on the complexity analysis reported in [116, 137]. Here,
the two party private intersection process is estimated to require $O(\text{MaxT})$ time, where \text{MaxT} is the number of elements of the largest set between the two sets to be intersected. Multi-party complexity is estimated as $O((\text{NMP}^2)\lambda + (\text{NMP}\lambda + \text{NMP}^2)\text{amp})$, where \text{NMP} is the number of parties participating in the multi-party process, \text{amp} is the maximum number of attribute values an attribute can have, and $\lambda$ is a security parameter (see [137] for more details).

Let us start by considering the enforcement of user private search criteria, where with \text{SearchConds} we denote the set of conditions specified in the search criteria. The approach implies running a two-party protocol for each condition \text{Cond} $\in$ \text{SearchConds}. As such, each and every condition, \text{Cond}, is actually mapped into a range of acceptable values, denoted as $\text{RS}_{\text{Cond}}$, and is intersected with the corresponding profile attribute values of the service provider. For example, \text{Cond} = \text{Price}<400 is mapped into a range $\text{RS}_{\text{Cond}}= [0, 400]$.

Thus, since the two party protocol has to be run for each condition in \text{SearchConds}, search criteria enforcement is estimated as: \(O(|\text{SearchConds}| \ast \text{MaxSearch}_\text{Cond})\), where \text{MaxSearch}_\text{Cond} is the maximum number of values a condition can hold, that is $\text{MaxSearch}_\text{Cond} = \max\{|\text{RS}_{\text{Cond}}|\text{s.t.} \text{Cond} \in \text{SearchConds}\}$. Notice that $\text{RS}_{\text{Cond}}$, in the worst case scenario, can be of the same order of the domain of the attribute appearing in $\text{MaxSearch}_\text{Cond}$.

**Service provisioning Complexity** Concerning enforcement of service provisioning rules, as anticipated, the most computationally expensive case is that of a WF, whereby the provisioning rules which are to be matched against the user’s profile attributes are composed by means of a combination of two party and multi-party protocols. The approach includes two phases. A first phase (b1) requires running a set of two-party protocols among the service providers in WF, so as to determine the conditions specified against common attributes among the service providers. The second phase, (b2), involves a set of multi-party protocols computing the intersection of admitted attribute values, for the conditions identified at step (b1).

Let us start by estimating (b1). Assume a workflow, WF consisting of WFA activities, where each activity is performed by a distinct service provider, so that the involved service providers are given an ordered set \{\text{SP}_1, \text{SP}_2, \ldots, \text{SP}_{|\text{WFA}|}\}. The two party cascade protocol is implemented by each service provider \text{SP}_i with any \text{SP}_j, where $i \in \{\text{SP}_1, \ldots, \text{SP}_{|\text{WFA}|}\}$ and $j \in \{\text{SP}_{i+1}, \ldots, \text{SP}_{|\text{WFA}|}\}$. There-
Therefore, we have \( \sum_{k=1}^{\lfloor WFA \rfloor} |WFA| - k \) executions of the two party protocol in total. Moreover, since the aim is to find the common attributes of each provider’s provisioning rule conditions, the worst-case time required by protocol execution for one service provider would be \( O(|WFA| \times MaxSet) \), where \( MaxSet \) is the maximum number of attributes a service provider can hold. Therefore, the total cost implied by (b1) can be estimated as \( O(|WFA|^2 \times MaxSet) \) (i.e., this complexity takes into consideration all the service providers).

In the second phase (b2), a distinct multi-party set intersection is run by the service providers for each common attribute determined in the first phase. The total time it takes to run the multi-party processes by a service provider depends on the total number of conditions of the service providers that are involved in the multi-party processes, the number of service providers that hold each of these conditions, and the number of attribute values contained in these conditions. The worst case number of times the multi-party private intersection process would be run by a service provider, occurs when all the attributes (\( MaxSet \)) of the service provider are in common with attributes of the other maximum number of service providers.

According to the complexity analysis provided in [137], each multi-party execution would take \( O(|WFA|^2 \lambda + (|WFA| \lambda + |WFA|^2)amp) \) number of computations, as mentioned previously except that NMP is replaced by \( |WFA| \). The final cost of (b2) is given by \( O((|WFA|^2 \lambda + (|WFA| \lambda + |WFA|^2)amp) \times MaxSet) \). As such, the final complexity of the overall protocol is estimated as \( O(|SearchConds| \times MaxSearch\_Cond) + O((|WFA|^2 \lambda + (|WFA| \lambda + |WFA|^2)amp) \times MaxSet) \).

Notice that, while the overall complexity is quadratic with respect to \( |WFA| \), it is reasonable to expect the number of WF activities to be bounded by a constant. Similarly, we expect the number of conditions, \( SearchCond \) to be relatively small, unless extremely sophisticated rules are applied by any of the providers. Hence, in the general case, the complexity is linear with respect to \( MaxSearch\_Cond \) and \( MaxSet \), which could have a potentially large cardinality. These dimensions can be greatly decreased by adopting strategies for attribute domain reduction [138]. More precisely, domain reduction can be obtained by following generalization techniques used in privacy-preserving techniques (e.g., k-anonymity). Clearly, reducing attribute domain limits the complexity, although it impacts the attribute accuracy. The optimal trade-off between efficiency and accuracy depends on several aspects,
including the specific service and business rules underlying the provisioning, the possible legal constraints and business needs, and the user’s desired quality of service.

8.2 Experiments

In what follows, we discuss various experiments we conducted on the approaches we proposed in this dissertation.

8.2.1 Service and Content Selection in Centralized Systems

The experiments for the ranking algorithms were conducted on an Intel core i7 2630QM CPU @2.00 GHz, 8GB RAM, Windows 7 Operating System and the experiments for the private negotiator protocols were conducted on an Intel core 2 quad CPU, Q6700 @ 2.66GHz, 4 CPU cores, 3 GB RAM, Red Hat Enterprise Linux 5.6 operating system.

As depicted in Figure 6.1, the proposed framework’s prototype consists of the following components: a client, set of Service Providers, a UDDI registry entity hosted by a UDDI provider, a planner module and the core components of our architecture, the Private Negotiator (PN) and the Ranker. The Ranker strategies are implemented in Java and the PN main algorithms are written in a high-level definition language called Secure Function Definition Language (SFDL) adapted from [116]. The intersection function, written in SFDL, is given as input to the two-party secure function evaluation core algorithm or to the multi-party secure function evaluation algorithm [139].

In what follows, we report our results concerning the algorithms implemented by the Web service Ranker, and by the PN.

8.2.1.1 Ranking Algorithms

We assessed both accuracy and overhead introduced by our ranking algorithms (Chapter 4). For both types of tests, we compare our results with state-of-the-art algorithms using real world data sets.

Comparison with State-of-The-Art We compared our recommendation algorithm with the Web service classification of the QWS dataset [46]. The dataset is a
Figure 8.1: Comparison with existing classification: comparing our approach with QWS classification, which classifies services into four classes.

Figure 8.2: Performance test for our approaches: Weighted Average (WA) and Single Criteria Preference (H).

collection of quality of service information for 9 criteria of 365 Web services which are collected using a Web service Crawler Engine (WSCE) [140]. The classification is made according to the Web service Relevancy Function [46], which normalizes the service capability values. The services are classified into four different classes: \{1,2,3,4\}, where 1 represents services of high quality, and 4 represents services of low quality. We could compare the provided classification with the results obtained by running a simplified version of our first strategy, that is, the Weighted
Average of Criteria. We did not compare either the priority-based strategy or the bounded level strategy, due to the lack of user input on the classified dataset. Since the QWS dataset does not account for composite Web services, we artificially created 20 service combinations from the dataset. Each combination consists of 4 services, combined together as a potential composite service. We use 9 different user criteria for ranking purposes. The aim is to maximize the satisfaction of the user’s criteria. To ensure fairness, we set the priority levels of all the criteria of the user as medium, as these are not included in the QWS classification. We tracked the average class of the four services of each combination from the ranked set of combinations (that is, each ranked combination is assigned an average class, based on the individual services’ original class). Our results show that the top 10 combinations ranked by our algorithm consistently have an average class value higher than those of the combinations ranked among the ranks 10 to 20, in line with the classification of the QWS dataset.

For the second experiment, the input to the Ranker consists of 5 paths or combinations of possible compositions of varying length (i.e., varying number of service providers). For example, 5 paths of length 2 are: \((WS_1, WS_2)\), \((WS_2, WS_3)\), \((WS_3, WS_9)\), \((WS_4, WS_2)\), and \((WS_5, WS_2)\). The corresponding composition size is of 10 nodes (each node is a unique Web service). For each group of 5 combinations, we considered services all of which belong to class 1, and then similarly classes 2, 3 and 4. For example, we compute the average PEF of the combinations, \((WS_1, WS_2)\), \((WS_5, WS_3)\), \((WS_3, WS_9)\), \((WS_4, WS_2)\), and \((WS_5, WS_2)\) which belong to class 1, and similarly we consider combinations belonging to other classes and compute their average PEF. As mentioned in Chapter 4, PEF is the path evaluation function, which quantifies the satisfaction of the user requirements or criteria by the services in a given composite service. For each of the classes, we further varied the size of the composition graph, ranging from 10 to 100, that is, from 2 to 20 services in each of the 5 composite services. As reported in Figure 8.1, the set of combinations belonging to class 1, maintain higher average PEF values than the combinations belonging to other classes, regardless of the number of combinations. Hence, the results confirm that the recommender system ranks the services using a comparable order of the one used in the QWS dataset, and that the PEF is consistent regardless of the size of the combinations. As shown in Figure 8.1, we range the number of services in each composite services up to 20, as
we observed almost a negligible increase in the PEF values as we further increase the number of services which followed the same pattern as for service up to 20.

**Performance Test.** We tested the performance of our three strategies by running three separate tests each. Each test measures the time taken to rank and recommend service combinations to the user by varying the input size in terms of number of paths in the composition graph and number of user criteria. Each composition path consists of 4 services. As shown in Figure 8.2, the time for ranking and recommending the Web services according to the user criteria, increases linearly as we increase the number of combinations input to Ranker. We considered composite services ranging from 10 to 90 as these numbers are typical for an experimental setting with respect to number of composite services, as shown by previous work on service composition [141]. The time increases linearly as we increase the number of user criteria. The overall results show that the Ranker is extremely efficient: the time taken to compute the ranks for 90 paths is utmost 1 millisecond. That is, a graph with 360 nodes (i.e., Web services) is processed in only 1 ms. The results obtained for running the bounded values strategy are similar to the first strategy. The bounded value strategy in fact requires completing additional checks for satisfaction of global criteria, with a negligible additional time.

**Performance Comparison with Integer Programming Method.** We compared the computation time of our ranking strategies with the integer programming (IP) method when it is used for ranking service compositions. Integer Programming is a popular method used to optimize a specific objective function which is subject to a set of linear constraints over integer variables. In the context of Web services, it used for selection of a service that best satisfies the given requirements or that has the best QoS attribute values. [142] and [52] are works on Web service selection and composition respectively. In order to compare both methods for generating ranked compositions, we generated all possible service combinations for a given user’s input and output. These set of combinations are generated by syntactic matching of the inputs and outputs of the services. The combinations are provided to our ranking strategies as input. The output is a set of ranked combinations of services. In order to rank a set of service combinations using IP method, we ran the IP method for all possible candidate services for each task, as many times as the number of ranked service combinations desired in the output.
For an input of 50 candidate services, and 4 services in each combination, that is, for a set of around $6 \times 10^6$ service combinations, the IP method consumes time in the order of $6 \times 10^6$ seconds, while our method consumes around 70 seconds. This is because the IP method generates only one service combination every time it is run. In order to rank $n$ combinations, the IP method should be run $n$ times, each time removing the combination resulted in previous run, which contributes to the high computation time for ranking. Hence, while the IP method is very efficient in generating a single service combination [52], it is not suitable for ranking service combinations, especially, when ranking of huge number of combinations is required.

8.2.1.2 Private Service Negotiation

We analyze the overhead incurred by running private selection of Web service (Chapter 5), in case of simple and composite Web services. As part of the analysis of private search, we also extensively tested the overhead added by the parser, to estimate whether it significantly affects the search time for a client.

**Private Selection** We recall that enforcing search criteria and provision rules while selecting a simple service requires the execution of two-party private intersection protocol. The overall results for running two party set intersection protocols

![Figure 8.3: Provision rule comparison in private fashion: Finding common attributes by performing two-party set intersection, with two and four service providers](image)
are reported in Figure 8.3. As anticipated, the overhead is linear with respect to the size of the set intersection input, i.e. the number of attributes considered. In this experiment, we limit the number of attributes to 150 as it might not be realistic for a service provider or a user to hold number of attributes beyond 150.

To estimate the overhead added by private selection in case of composite Web services, we completed two experiments on provisioning rule composition. There are two main steps to be completed, (1) to identify the common attributes and (2) check their satisfiability ranges.

First, we calculated the time required to identify the common attribute sets. In Figure 8.3, the actual results for the case of a workflow with 4 providers is also considered. In this experiment, we let the service providers execute the two-party protocols in cascade. If the protocols are executed in parallel, the overall execution time is of the same order of the time for single two-party process, otherwise it increases by a factor equal to the number of providers participating in the process. Next, we considered the overhead added by computing private intersection among range of satisfiability of conditions held by multiple providers on common attribute domains (Figure 8.4). Our results show that with increasing number of service providers participating in a multi-party process, the run time of the multi-

![Figure 8.4: Provision rule comparison in private fashion: Finding common attribute values by performing multiparty set intersection where the service providers are the multiple parties participating](image-url)
party algorithm quadratically increases. This result is in line with the complexity estimated in Section 8.1. The execution time of the multi-party processes can be substantially reduced if the parties execute the multi-party intersection protocols in parallel, rather than sequentially.

**WSDL Parser.** Precisely, for this experiment, we performed several rounds of tests for different set of actual WSDL documents by varying the number of attributes/search keywords in a WSDL document and tracked the time taken to parse these documents. At each round, we submitted a certain number of documents consisting of the same number of attributes operations of the WSDL, that are the function names of the interface of a Web service as input to the parser and recorded the time taken to parse these documents. In the same round, we considered the same number of documents but with a different number of attributes and tracked the time taken to parse these documents. In the consecutive rounds, we gradually increased the number of documents with the same test procedure, from 20 to 200 (in tandem with the experimental settings with respect to number of service providers participating in a composite service [141], as in this experiment, a document represents a service provider’s attributes), for each round as described. In Figure 8.5, we present the test results for this experiment. The time to parse a
set of documents slightly increases as the number of attributes in the documents increases and is always below 1000 ms (for 200 documents).

8.2.2 Service and Content Selection in Decentralized Systems

We performed several experiments to assess performance and accuracy of our approaches: query routing (Chapter 6) and node-compliance verification (Chapter 7). The experiments were conducted on an Intel core i7 2630QM CPU @2.00 GHz, 8GB RAM, Windows OS.

8.2.2.1 Policy-compliant Query Routing

We conducted the experiments on an instance of P2P network topology provided by http://snap.stanford.edu/data/#web. The network consists of up to 10,000 nodes. Out of these 10,000 nodes we assume 100 of these have a resource, meeting the service request, whereas all nodes in the network maintain filtering policies. In the first three experiments, we assume that filtering policies do not block the query from being forwarded, whereas we change our assumptions in the remaining experiments, when measuring performance. We defined policies as a disjunction of up to 5 clauses. Each clause has a set of up to 6 conditions.

In all our experiments, we compared our approach to the random walk approach, wherein the $k$ nodes used to forward the search query are selected randomly. The random walk approach is used in the traditional peer to peer networks to initiate a search for a resource [143].

In the first experiment, we checked the efficiency of our approach in terms of the satisfaction ratios of the policy by the nodes in the network. In specific, we compared the policy satisfaction of the nodes in the network that are asked to evaluate or forward the search query, using our approach and the random walk approach. We increase the number of nodes in the network from 1000 to 9000 (in specific, we test for 1000, 5000 and 9000 nodes in order to make it even with respect to increasing the number of nodes from the 10,000 node network we considered), and measure the average satisfaction value of the nodes that forward the search query. We keep the size of the policy constant, that is, we maintain the number of clauses to 3 throughout the experiment. On repeated trials, the satisfaction ratios corresponding to our approach are consistently higher than those of the
Figure 8.6: Satisfaction ratios in case of equal number of nodes traversed, for networks of 1000, 5000, and 9000 nodes. Our approach produces higher satisfaction values compared to random walk approach. This graph is a zoomed-in version showing clear differences between the lines.

random walk approach. Specifically, satisfaction levels are always above 0.6, in our approach, whereas in the case of random walk an average of 0.4 of satisfaction rate is achieved.

In addition to the above, to check whether the satisfaction is related to the number of nodes being visited (that is, number of nodes forwarding the search query), we compute the satisfaction levels achieved in a network of fixed size and the number of nodes that forward the query (or number of nodes traversed by the search query) constant in both approaches. We repeat this procedure by increasing the size of the network and the number of nodes traversed by the query. Results are reported in Figure 8.6 and Figure 8.7. We can observe that even when the number of nodes traversed by the search query is constant in both approaches, the policy driven approach has better satisfaction values than the random approach, for all three different sizes of the network. We also observe that the satisfaction values of the policy driven approach are greater on an average, whereas, those of the random walk approach are lesser on an average.

In order to find the statistical significance of our approach, we performed a t-test for the results shown in Figure 8.6 and Figure 8.7. We observed that the
Figure 8.7: Satisfaction ratios in case of equal number of nodes traversed, for networks of 1000, 5000, and 9000 nodes. Our approach produces higher satisfaction values compared to random walk approach. This graph is the version showing the scale 0 to 1 for satisfaction values.

distributions between our method and the random walk approach is statistically significant for networks with at least 5000 nodes. The p-values for networks with 5000 and 9000 nodes are 0.0267 and 0.0001 respectively.

We used breadth first search algorithm to sample the nodes from the network. We observed that the connected nodes are around 75%, 85%, and 85% for 1000, 5000, and 9000 nodes we sampled, respectively. The satisfaction values we present in the graph are an average of the satisfaction values of all the nodes that are being traversed by the query, and we do not include the satisfaction values of the nodes that are not connected.

Next, we compared the policy satisfaction ratios of our approach and the random walk approach by varying the number of atomic conditions in each of the 5 clauses of the policy associated with the search query. We keep the number of nodes constant throughout the experiment, that is, we conducted the experiment with 5000 nodes. Figure 8.8 shows the graph for this experiment. The satisfaction ratios remain almost constant even though the number of conditions in a policy is increased (and hence, we show the results for up to 9 conditions only).
to experiment 1, in this experiment, we observe that the satisfaction ratios corresponding to our approach are higher than those of the random walk approach.

We also conducted an experiment to test the execution time of the policy driven approach by introducing filtering policies for all the nodes in the network. We computed the total execution time taken to forward a search query from the requester to a resource node in the network. When a search request arrives at a node, the node compares its filtering policy against the categories of the query and the profile of the requesting node. If the filtering policy is not satisfied, the requesting node runs a module that chooses the next neighbor from its suitable neighbor set. The time taken to verify if the filtering policy is satisfied or not, and the time taken to pick a new neighbor if not satisfied is about 9 micro seconds for one node. The time taken to compare the forwarding policy is about 500 micro seconds for one node. Hence, the total time taken for comparing filtering policy is negligible compared to the total execution time taken to forward the query from the requester to a resource. Note that we only compute the computational time but do not compute the network delays.
8.2.2.2 Policy-compliance Verification

Path authentication

Our first experiment involves testing for the computational times of the first step of our protocol (Chapter 7), that is, secure proof of identities of the path of the search query (see Section 7.4.1). This experiment has two parts to it. The first part measures the computational times for the aggregated signature and search query traversal through the network. The second part measures the computational times for the verification by the requester, of the aggregated signatures of the paths that the search query had taken.

First, we vary the path length traversed by the search query and observe the respective computational times. Path length is the number of nodes traversed by the search query in a path. From the graph in Figure 8.9, we observe that as the path length increases, the time for computing the aggregated signatures increases. We observe the same pattern as we increase the number of nodes in a path, and hence, we show the graph for up to 300 nodes only. Next, in the second part, we vary the path length and observe the respective computational times. Interestingly, from the graph in Figure 8.10, we observe that even though as the number of nodes in a path increases, the time to verify the aggregated signatures increases very negligibly, in the order of milliseconds (we observe the same pattern as we increase
the number of nodes in a path, and hence, we show the graph for up to 300 nodes only). This confirms that using aggregated signatures for secure proof of identities of a path is efficient when compared to sending individual signatures by each node in the path to the requester. This is because as the number of nodes increases in a path, the number of individual signatures to verify will increase for the verifier.
Hence, receiving individual signatures from every node would drastically increase
the communication overhead of the protocol.

Our second set of experiments test the policy compliance of the nodes in the
paths taken by the search query, that is, to test the phase where the requester
uses attribute based encryption. First, we compute the computational times of
the encryption of messages performed by the requester or the owner of the query
for each node in a path taken by the query. If there are \( n \) nodes in a path,
then the requester encrypts \( n \) messages with the ABE protocol. We compute the
computational times by varying the number of nodes in a path, and also we perform
the same experiment for different number of attributes in the encryption policy of
the requester. From the graph in Figure 8.11, we observe that as the number of
nodes in a path increases, the computational time linearly increases for encrypting
the messages with ABE (we observe the same pattern as we increase the number of
nodes in a path, and hence, we show the graph for up to 300 nodes only). We also
observe that, as the number of attributes in a policy increases, the computational
time linearly increases. Next, we also compute the times of the decryption of
messages by all the nodes in a path. That is, in this experiment, each node in the
path sequentially decrypts the message encrypted by the requester for the node,
with the requester’s policy. We compute the computational times by varying the
number of nodes in a path. From the graph in Figure 8.12, we observe that the time for all the nodes in a path to decrypt the ABE message linearly increases as the number of nodes in a path increases (we observe the same pattern as we increase the number of nodes in a path, and hence, we show the graph for up to 300 nodes only). In this experiment, the number of attributes in the policy does not affect the computational time for decrypting the message, as for decryption, each node uses its own private key to decrypt the message, and the private key is not associated with the number of attributes in the encryption policy.

Path Filtering

Our first experiment in the path filtering phase (Chapter 7) aims to test the computational overhead of the policy comparison, where we determine the policies in the history that subsume $\text{pol}_{\text{own}}$ or are a subsumption of $\text{pol}_{\text{own}}$. We conducted the experiment for 1000 policies in the history to be compared with $\text{pol}_{\text{own}}$, and for up to 100 attributes or conditions in each policy of the history and $\text{pol}_{\text{own}}$. In this experiment, we limit the number of attributes to 100 as it might not be realistic for a service provider or a user to hold number of attributes beyond 100 to 150. The time taken to find the policies that have a subsumption relation with $\text{pol}_{\text{own}}$ grows linearly with the number of attribute conditions to check as seen in Figure 8.13. With 1000 policies in the history, the overhead ranges from less than 5 milliseconds to around 400 milliseconds for 100 attributes in each policy.

Our second experiment consists of computing the overhead for filtering out the nodes in $\mathcal{P}_{\text{pol}_{\text{own}}}$ that do not require any further verification (Section 7.4.2). We vary the number of nodes in both the sets exponentially, as seen in Figure 8.14. Our approach scales well, as the time taken to find the intersection of nodes is in the order of milliseconds. Even for a huge number of nodes in the path sets such as 10000000, the computational overhead is in the order of milliseconds.

Policy Compliance

In this experiment, we measure the computational overhead of policy compliance verification protocol (Chapter 7) using our two approaches. We also compare the overhead introduced by our approaches with that of the basic CP-ABE approach. Results are shown in Figure 8.15.

For the optimization based on “Resuing Ciphertexts”, we compute the time taken for a single CP-ABE encryption which is computed by the requester node. The time taken for the modification of the encrypted CP-ABE component (i.e.
finite-field multiplication) in order to generate new random messages $m_0$ and $m_1$ by the delegatee nodes in the paths, is negligible. The entire verification process
does not even consume 1 second which is very less as compared to the other two approaches.

For our “Offloading Partial Computation” approach, the computational overhead for the requester just includes three modular exponentiations and one bilinear mapping for a single CP-ABE encryption. If the requester encrypts the messages using CP-ABE for every node that has to be verified (as with baseline approach), the computational time explodes as the number of nodes to be verified increases. Since in our approach the requester needs to perform only three modular exponentiations and one bilinear mapping instead of a whole CP-ABE encryption operation, as part of the encryption is delegated to “honesty-verified” nodes, there is a huge difference in the computational overhead compared to the baseline CP-ABE approach. For all the three approaches, the number of attributes in the policy is equal to 8, while we increased the number of nodes to verify, as shown in Figure 8.15.
We ran the same experiment multiple times by increasing the number of attributes in the policy and observed the difference of computational overhead between the offloading and the baseline approach. As the number of attributes in the policy increases, the computational overhead of the baseline CP-ABE approach increases, while that of our offloading approach remains the same irrespective of the number of attributes in the policy. This is because our approach requires the requester to compute the partial cipher-text for one attribute only which is the ID of the recipient node. The rest of the cipher-text computation for the attributes in the policy is offloaded to the honest nodes. Hence, as the number of attributes in the policy increases, we can observe a huge difference in the computational overhead between our offloading approach and the baseline approach.

8.3 Summary

In summary, the following are the main observations from the results we obtained from running multiple experiments. The satisfaction values we obtained for our ranking algorithms confirm that our algorithms rank the services using a comparable order of the one used in the QWS dataset, and the ranking generated by our approaches is in-line with those of the QWS dataset, and also, our ranking approaches perform very well and outperform the integer programming technique. We have presented the computational overhead introduced by the private matching protocols. The computational overhead is further reduced if the protocols are run in parallel. The computational overhead introduced by our WSDL parser is very low, which is around 1 second for up to 200 WSDL documents and for each document consisting of up to 150 attributes. With respect to our policy-driven routing approach, we observed that our approach is effective compared to the baseline approach, that is, the random walk approach. The path filtering technique we proposed in the context of policy-compliance verification, introduces very little overhead which is in the order of milliseconds. The two approaches we proposed, the partial-offload and the cipher-reuse outperform the baseline verification approach we proposed which leverages cryptographic algorithms.
Chapter 9  
Conclusion and Future Work

9.1 Conclusions

The Internet offers users a large variety of content to select from. In this dissertation, we focus on formal channels of content selection platforms and their security and privacy issues. Web service selection is an important and critical phase in the Web service life cycle as it is the phase where a user is able to select a suitable service of his or her requirement. As our study confirms, Web services are associated with security vulnerabilities, and these vulnerabilities are wide spread among the Web services that are currently on the World Wide Web.

It is very critical for the service providers to protect the services from attacks, and also it is critical for the users to protect their information when they select and invoke a Web service. Hence, it is of best interest for the users to select services with no vulnerabilities in order to avoid security risks to the users’ data. Also, the service providers must not collaborate with other service providers whose services are prone to vulnerabilities. We are the first ones to propose two types of frameworks or approaches, one that is related to centralized systems, and the other that is related to decentralized systems, which secure the service and content selection process. With respect to centralized systems, we design a framework which enables the user to well express his or her preferences or constraints to select a Web service, and to recommend a ranked list of Web services selected according to the user’s security and privacy preferences. It is challenging to address the ranking and recommendation problem as it is not very realistic to find services that exactly match the user preferences, and hence a distance-based approach
is highly needed. Previous works [47–49] have mainly focused on recommending non-composite services, that is, single Web services. We take a step forward and propose an approach to rank and recommend composite services. Also, the possibly large amount of sensitive information exchange between the users and the service providers, and among the service providers themselves during service selection and provisioning, leads to privacy issues both at the users’ and the service providers’ end. Works to date [69,72,73] have taken into consideration the privacy of the client’s information during the service provisioning phase and later, but do not consider the client’s information privacy during the service selection phase. Users’ search criteria information is as vulnerable to secondary usage as other data of the client stored at the service provider’s side. Our framework enables the users and the service providers to expose their information to each other only to a minimum, that is, only the information that is highly required is exposed. Designing an approach to address this problem is non-trivial as an efficient solution is needed since there are multiple providers and users involved in service selection, and each possesses sensitive information that is to be kept private.

With respect to decentralized systems, we propose a framework that routes the search queries through content hosting networks according to a user-generated policy. Previous works (e.g., [43], [42], [44]) have mainly focused on forwarding content based on the properties of the neighboring nodes. In addition to considering the properties of the neighboring nodes, we consider the properties of nodes in further levels down, which gives a better insight into which neighboring directions best satisfy the policy. Also, in this framework, we generate authenticated paths that the search query has traversed through, and verify the policy compliance of the nodes in the network so as to detect malicious query forwarding in these networks. Prior works on P2P networks [100–104] have mainly focused on efficient query processing and policy-specification and policy-compliant routing [42–44,106], and do not tackle the problem of detecting malicious forwarding in the context of P2P networks. Addressing secure selection in decentralized systems is non-trivial because there is no centralized server in place that could maintain trustworthy information related to nodes in the network, hence requiring distributed protocols.

We discuss the limitations of our work in what follows. With respect to matching capabilities in the form of categorical values for recommending services, we assume the existence of classifications, ontologies and semantic matching algo-
rithms. Semantic algorithms as mentioned in [144] could be used. In specific, categorical capability values are specified mainly in the context of security and privacy related constraints. In order to take these values into account, security and privacy ontologies such as those proposed in [145,146] can be used.

In computing the overhead of our decentralized protocols, network delays are not included. When network delays are added to our computational overheads, the actual overhead needed for the nodes in the network to employ our approaches would be known. Network delays vary based on the type of network in use, and hence the total overhead would depend on the type of network in use.

In regards to decentralized systems, we assume that the attributes or the properties of the nodes are static, that is, they do not change. In some networks, it could be the case that the properties of the nodes change. For example, in mobile networks, the properties such as location, and hence other properties related to location, such as cost, delays, and so forth could also change. The nodes that changed their properties would need to get a new key with respect to their new properties, and the old key needs to be revoked. For dynamically changing properties, we could employ techniques where a property verification is done through trusted computing platforms as proposed in [147].

We assume that our network is not very dynamic in nature, that is, there is no high frequency of new nodes joining or leaving the network. In the case of highly dynamic networks, leveraging policy and query path history, as suggested in our proposal, becomes difficult. Also, in the case of highly dynamic networks where new nodes are constantly added, the attenuated bloom filters, carrying the attributes of nodes at different neighboring-branch levels, might not reach the required nodes in a timely fashion when a query propagation is taking place. However, at least the nodes nearer to the newly added node would be able to receive the attenuated bloom filter from the newly added node, and thus are able to include the properties of the newly added node in the decision-making process for query-forwarding. Finally, in the case where nodes are constantly removing themselves from the network, the attenuated bloom filters distributed while a query propagation is taking place, would not be valid. We can overcome this problem by requiring those nodes which newly joined the network, and which want to leave the network, to not send their attenuated bloom filters after a specific time period at the end of which the node wants to leave. If an old node leaves the network,
its neighboring nodes could send an updated attenuated bloom filter or a single
bloom filter associated to the node that left, so that the rest of the nodes in the
network are able to update their attenuated bloom filters.

9.2 Future Work

Our framework for service and content selection in centralized systems, takes into
account both search criteria and service provisioning rules, and is suitable for
both single and composite services. Our current solution can be extended along
two dimensions. First, we extend the applicability of our approach toward more
complex rules, and will consider the case where providers handle several rules. This
extension will require us to revisit our algorithms to control the complexity added
by private selection over complex data structures.

Second, we will conduct a formal security analysis to estimate the amount of
information possibly inferred by service providers and users during the protocols.
For instance, a user or a service provider could produce fake attribute values just
to know if the other party has those attributes, in the context of private matching
of attributes. So, in such cases, the actual amount of information exposed is
higher than normal. By conducting a security analysis, we would be able to detect
what kind of information is more prone to exposure, what type of services are
more prone to attacks, and what types of solutions better suit for addressing the
problems at hand. Also, we would deploy a GUI for entering service description,
criteria, and priority values, and would work on enabling good visualization of
the service recommendations. Next, we will consider dependencies among user
specified criteria. For example, if cost of a service (output) depends on the location
of the service (input), then the user needs to reveal in clear the exact locations he
or she needs the service at, which is a privacy breach for the user. We will work
on reducing information exposure in spite of dependencies among user specified
criteria.

For our decentralized approach, for comparing policies of the nodes, we plan
to design an in-depth scheme for measuring distance between categorical attribute
values. It is difficult to quantify the difference or distance between categorical
values unlike quantitative values, as clearly categorical values are not numbers.
We plan to leverage existing ontologies on categorical values to build a scheme for
measuring distance between categorical attribute values. We will also study if the topology of the P2P network affects the policy satisfaction levels for a query. That is, we want to observe if the change in the way the network is structured affects the policy satisfaction levels for a given query. The policy satisfaction levels might be different for different networks. Collusion is a usual problem with key exchange protocols. With respect to verifying policy-compliance of nodes in the network, we aim to efficiently address the collusion problem that might exist in regards to one of our approaches, that is, the cipher-reuse approach.
Bibliography


URL http://doi.acm.org/10.1145/514191.514206


URL http://dx.doi.org/10.1002/ett.1017


URL http://dx.doi.org/10.1007/s10660-010-9064-y


Vita

Sushama Karumanchi

Sushama Karumanchi received her bachelors degree in Computer Science and Engineering in 2008 from Andhra University, Visakhapatnam, India. She started pursuing her Master of Science in Computer Science at the University of Kentucky in 2008. After that, she joined the PhD program in the College of Information Sciences and Technology at the Pennsylvania State University. Her research interests in general include information security and privacy in distributed systems. During her PhD, she worked as an intern at Microsoft Research, Intel Labs, and IBM Research.

Selected Publications


