CERIAS Tech Report 2009-32
Data in the Cloud: Authentication without Leaking
by Ashish Kundu
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086
DATA IN THE CLOUD: AUTHENTICATION WITHOUT LEAKING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ashish Kundu

In Partial Fulfillment of the
Requirements for the Degree

of

Doctor of Philosophy

December 2010

Purdue University

West Lafayette, Indiana
Dedicated to
My Beloved Parents
Aniruddha and Brundabati
and Grandparents Uday and Menaka
Here is another night, quietly passing by. Perhaps, the last night of my pursuit of the degree, that degree, for which I started this marathon in 2005 – the highest academic degree of Ph.D. In the air, there is a feeling of nostalgia, a feeling of emptiness, alongwith, of course, a great sense of joy and perhaps, relief. A great number of individuals have contributed to my successful run at Ph.D. and to this feeling of mine today. I wish I could mention each one of them with honor and gratitude here. Alas! This night may not last that long. But I would give it a try.

Without the constant encouragement, support, and inspiration over these years from my advisor, Prof. Elisa Bertino, it would not have been possible at all for this dissertation to materialize. I have the highest respect and gratitude in my heart for Prof. Bertino, who has painstakingly listened to my ideas and thoughts, revised my papers, and has shown me the right directions. My advisor has been instrumental in all the success during my Ph.D. I feel honored and privileged to have such a great advisor and a distinguished doctoral committee.

I would like to express my sincere-most gratitude and respect for Prof. Mikhail Atallah for being such a great teacher to me, for giving me an opportunity to work with him, for kindly serving on the doctoral committee.

I would like to express my highest regards and gratitude for Dr. Guruduth Banavar. Dr. Banavar has been a great mentor, and a constant source of inspiration.
and encouragement to me not only during my Ph.D. as a member of the doctoral committee but also since the days I started working in his group at IBM Research. Dr. Banavar has regularly monitored my progress, the direction of my research, and has played a pivotal role in my development as a researcher and individual. I would also like to thank Dr. Nevenka Dimitrova for her encouragement during my Ph.D., and for inspiring me to work on solving real world problems such as those in biology with the help of computer science.

I am highly thankful to Prof. Ninghui Li for kindly serving on my doctoral committee, and for his kind advice and feedbacks throughout my Ph.D. My sincere-most gratitude to Prof. Sunil Prabhakar, for kindly serving on my doctoral committee, for his suggestions, and encouragement over these years.

I am highly thankful to Prof. Eugene Spafford for his kind support and encouragement, and for being such a wonderful teacher to me over all these years. I would like to express my sincere-most gratitude to Professors Aditya Mathur and Jyoti Mathur, who have been very inspirational to me over all these years with a great positive influence during my graduate studies. I am highly thankful to Professors Greg Frederickson, Susanne Hambrusch, Samuel Wagstaff, Christina Nita-Rotaru, Patrick Eugster, Bharat Bhargava, Saurabh Bagchi, Kihong Park, Chris Clifton, Sonia Fahmy, and Ahmed Elmagarmid for their time, advice and help at various stages during my Ph.D. I am very much thankful to Dr. William J. Gorman for his support and help related to my progress during my Ph.D., and for his patience and perseverance towards the realization of this dissertation in this final form.

I would like to extend my sincere gratitude and respect to Dr. Ponani Gopalakrishnan, the then Director of IBM India Research Lab, for his kind support and encouragement towards my Ph.D. during 2005. I would like to extend my sincere-most thanks to Dr. Amit A. Nanavati and Ms. Hruta Nanavati for being such wonderful friends to me. Since the time Amit and I came to know each other at IBM, Amit has encouraged and guided me towards my Ph.D. and research. His honest thoughts and advice have had a great positive influence on my personal and professional de-
velopement. I have my heart-felt gratitude and regards for Dr. Sudha Krishnamurthy who have given me her invaluable thoughts and advice before and during my Ph.D. My sincere thanks to Prof. Rahul Shah, Prof. Amitabh Chaudhary, Dr. Tanu Malik, and Dr. Ashish Gehani for their insightful thoughts and advice during my Ph.D. I would like to extend my greatest thanks to my ex-colleagues at IBM, from whom I have learnt a lot – Dr. Danny Soroker, Dr. Vijay Naik, Dr. Mukesh Mohania, Dr. Satish Chandra, Dr. Neeraj Karnik, Dr. Vinayaka Pandit, Krishna P. Chitrapura, Nitendra Rajput, Girish Chaffe, Vikas Agarwal and Arun Kumar.

Whatever I am today and whatsoever success I have achieved are because of my loving family. My parents Aniruddha and Brundabati have been my finest teachers, and have given me an unlimited world full of love and compassion, a world where each horizon is colorful and full of happiness, a world of hope and optimism, a world that they have created tirelessly with their vision and great sacrifices and dedication. No word can ever describe their love for me – words have edges and boundaries, while their love for me is limitless. The love and affection of my paternal grandparents Uday and Menaka, and maternal grandparents Satyendra and Bidhumati have played a significant role in my growth as an individual. My elder brother, Chinmay, a Ph.D. in Aerospace Engineering has mentored me since college days, has been there with me always. My elder sister, Chinmayi is an epitome of love and care, and has given a sense of security in every step of my life. My brother-in-law, Prof. Narayan Jana, a Ph.D. in Geography and co-author of several books, have been of constant advice to me on how to do a Ph.D. or rather how not to do a Ph.D. Nandini, my sweet little niece, has brought joy and happiness for me always, whenever I have stared at the bottom of the horizon. And then there is some love in the life of this graduate student – Rakhi, my wife, who has been with me since I started my doctoral research, and through all the ups and downs. She has given me all her unconditional love and support, so that I could just focus on doing my research. She is the first person (other than the author), who has painstakingly reviewed this dissertation. My in-laws Dr. Satish Tyagi, and Savita Tyagi have given me a lot of encouragement and support.
towards my doctoral research. Rakhi’s grand-father Jai Dutt Tyagi has been a great source of inspiration for me. I am very much thankful to God for giving me such a loving and wonderful family.


I would like to thank Cathy Muller and Asha Thimmanna of IBM. I offer my heartiest thanks and regards to the late Amy Ingram, who used to work as the graduate office secretary in the Department of Computer Science. For the most part of my graduate studies, Amy had been of great help to me in taking care of various matters related to administration. May her soul rest in peace. I would also like to thank all the staff of the Computer Science and CERIAS for their help and support over these years.
Since my first semester, I was in search of an important problem that would form the core of my doctoral thesis. I knew of one problem in language security, from my work at IBM Research; however, I was not sure if that was of the right depth and breadth to become a topic of doctoral research.

During the Fall 2005, I was taking the class of Information Security offered by Prof. Elisa Bertino. During the discussion of secure XML dissemination in the class, I found that existing schemes leak information, especially structural information. I discussed this problem with Prof. Bertino, who suggested me to study it further as a course project. Thus I started working on the problem of “Authentication of trees without leaking”.

Our first paper in this topic was on secure dissemination of XML documents using randomized traversal numbers, which received the Best Student Paper at IEEE Enterprise Computing (2006). We were encouraged by this interest in the community, and wanted to explore this problem further. At this point, with the advice of my advisor Prof. Bertino, we decided to take this problem as the topic of my doctoral research. We developed a scheme for leakage-free authentication of trees, following which we asked ourselves – can we solve the problem for graphs, or is it the end of this problem? It seemed to be more challenging, especially for cyclic graphs. Since we found that the problem had enough depth and breadth for the purpose of doctoral research, we continued working on it.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>xii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>xiii</td>
</tr>
<tr>
<td>ABBREVIATIONS</td>
<td>xvi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xvii</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 The Problem</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Contributions</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Dissertation Roadmap</td>
<td>9</td>
</tr>
<tr>
<td>2 RELATED WORKS</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Merkle Hash Technique</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Authenticated data structures</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Transitive Signatures</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Redactable Signatures</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Sanitizable Signatures</td>
<td>14</td>
</tr>
<tr>
<td>2.6 Commitment Schemes</td>
<td>15</td>
</tr>
<tr>
<td>2.7 Proxy Signatures</td>
<td>15</td>
</tr>
<tr>
<td>2.8 Zero-Knowledge Sets and Membership Queries</td>
<td>16</td>
</tr>
<tr>
<td>2.9 Hashing and Accumulation Schemes</td>
<td>17</td>
</tr>
<tr>
<td>2.10 Traversal numbers</td>
<td>18</td>
</tr>
<tr>
<td>2.11 RSA Signatures</td>
<td>18</td>
</tr>
<tr>
<td>2.12 Privacy-preserving Authentication of Query Results</td>
<td>19</td>
</tr>
<tr>
<td>2.13 Secure Publish/Subscribe</td>
<td>21</td>
</tr>
<tr>
<td>3 STRUCTURAL LEAKAGES AND PRIVACY</td>
<td>23</td>
</tr>
<tr>
<td>3.1 Inference Attacks on Merkle Hash Technique</td>
<td>23</td>
</tr>
<tr>
<td>3.1.1 Leakages during Authentication using MHT</td>
<td>24</td>
</tr>
<tr>
<td>3.1.2 Inference Attacks</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Structural Leakages in Tree-structured Data</td>
<td>26</td>
</tr>
<tr>
<td>3.3 Structural Leakages in Graph-structured Data</td>
<td>33</td>
</tr>
<tr>
<td>4 FORMAL MODEL OF LEAKAGE-FREE REDACTABLE SIGNATURES</td>
<td>38</td>
</tr>
<tr>
<td>4.1 Preliminaries</td>
<td>38</td>
</tr>
<tr>
<td>4.1.1 Signature Schemes</td>
<td>39</td>
</tr>
<tr>
<td>4.1.2 Forgery</td>
<td>40</td>
</tr>
</tbody>
</table>
## 4.2 Leakage-Free Redactable Signatures

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 Why a New Model?</td>
<td>41</td>
</tr>
<tr>
<td>4.2.2 Definition of a General Scheme</td>
<td>42</td>
</tr>
</tbody>
</table>

## 4.3 Security of Leakage-Free Redactable Signatures

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.1 Unforgeability</td>
<td>44</td>
</tr>
<tr>
<td>4.3.2 Leakage-Free Properties</td>
<td>46</td>
</tr>
<tr>
<td>4.3.3 Privacy</td>
<td>47</td>
</tr>
<tr>
<td>4.3.4 Transparency</td>
<td>48</td>
</tr>
<tr>
<td>4.3.5 Relationships of Privacy and Transparency</td>
<td>49</td>
</tr>
</tbody>
</table>

## 4.4 Secure Names

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1 Secure Naming Schemes</td>
<td>50</td>
</tr>
<tr>
<td>4.4.2 Security of Naming Schemes</td>
<td>51</td>
</tr>
</tbody>
</table>

## 4.5 Summary

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5 Summary</td>
<td>53</td>
</tr>
</tbody>
</table>

## 5 STRUCTURAL SIGNATURES FOR TREES

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Review of Tree Traversals</td>
<td>54</td>
</tr>
<tr>
<td>5.2 Randomized Traversal Numbers</td>
<td>55</td>
</tr>
<tr>
<td>5.2.1 Computation of randomized traversal numbers</td>
<td>57</td>
</tr>
<tr>
<td>5.3 Structural Signatures</td>
<td>58</td>
</tr>
<tr>
<td>5.3.1 Signing ((rSign))</td>
<td>59</td>
</tr>
<tr>
<td>5.3.2 Distribution of a Subtree ((rRedact))</td>
<td>60</td>
</tr>
<tr>
<td>5.3.3 Distribution of a Subtree along with its Structure</td>
<td>61</td>
</tr>
<tr>
<td>5.3.4 Authentication ((rVrfy))</td>
<td>61</td>
</tr>
<tr>
<td>5.3.5 Sharing a Subtree – Only the Nodes</td>
<td>62</td>
</tr>
<tr>
<td>5.3.6 Illustration</td>
<td>66</td>
</tr>
<tr>
<td>5.4 Security Analysis</td>
<td>67</td>
</tr>
<tr>
<td>5.5 Complexity and Performance Analysis</td>
<td>69</td>
</tr>
<tr>
<td>5.5.1 Complexity Analysis</td>
<td>70</td>
</tr>
<tr>
<td>5.5.2 Performance</td>
<td>70</td>
</tr>
<tr>
<td>5.6 Dynamic Trees</td>
<td>74</td>
</tr>
<tr>
<td>5.7 Applications</td>
<td>77</td>
</tr>
<tr>
<td>5.7.1 Automatic Recovery from Structural Errors</td>
<td>77</td>
</tr>
<tr>
<td>5.8 Summary</td>
<td>78</td>
</tr>
</tbody>
</table>

## 6 STRUCTURAL SIGNATURES FOR GRAPHS

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Background</td>
<td>79</td>
</tr>
<tr>
<td>6.2 DAGs</td>
<td>82</td>
</tr>
<tr>
<td>6.2.1 Signing a DAG</td>
<td>84</td>
</tr>
<tr>
<td>6.2.2 Distribution</td>
<td>85</td>
</tr>
<tr>
<td>6.2.3 Authentication</td>
<td>85</td>
</tr>
<tr>
<td>6.3 Graphs with Cycles</td>
<td>88</td>
</tr>
<tr>
<td>6.3.1 Signing a Graph</td>
<td>93</td>
</tr>
<tr>
<td>6.3.2 Distribution</td>
<td>93</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>6.3.3 Authentication</td>
<td>93</td>
</tr>
<tr>
<td>6.4 Security Analysis</td>
<td>100</td>
</tr>
<tr>
<td>6.5 Complexity and Performance Results</td>
<td>104</td>
</tr>
<tr>
<td>6.5.1 Complexity</td>
<td>105</td>
</tr>
<tr>
<td>6.5.2 Performance Results</td>
<td>106</td>
</tr>
<tr>
<td>6.6 Summary</td>
<td>107</td>
</tr>
<tr>
<td>7 LEAKAGE-FREE REDACTABLE SIGNATURES</td>
<td>110</td>
</tr>
<tr>
<td>7.1 Trees</td>
<td>110</td>
</tr>
<tr>
<td>7.1.1 Preliminary Scheme (Scheme-1)</td>
<td>111</td>
</tr>
<tr>
<td>7.1.2 Efficient Scheme (Scheme-2)</td>
<td>113</td>
</tr>
<tr>
<td>7.1.3 Secure Names</td>
<td>114</td>
</tr>
<tr>
<td>7.1.4 Leakage-Free Signatures of Trees (rSign)</td>
<td>117</td>
</tr>
<tr>
<td>7.1.5 Distribution (rRedact)</td>
<td>118</td>
</tr>
<tr>
<td>7.1.6 Authentication (rVrfy)</td>
<td>119</td>
</tr>
<tr>
<td>7.2 Graphs</td>
<td>122</td>
</tr>
<tr>
<td>7.2.1 Leakage-Free Signatures for Graphs (rSign)</td>
<td>123</td>
</tr>
<tr>
<td>7.2.2 Distribution of Graphs (rRedact)</td>
<td>123</td>
</tr>
<tr>
<td>7.2.3 Authentication (rVrfy)</td>
<td>126</td>
</tr>
<tr>
<td>7.3 Single Signature Scheme</td>
<td>127</td>
</tr>
<tr>
<td>7.3.1 LFR Signature: rΠ</td>
<td>128</td>
</tr>
<tr>
<td>7.3.2 Complexity</td>
<td>130</td>
</tr>
<tr>
<td>7.4 Security Analysis</td>
<td>130</td>
</tr>
<tr>
<td>7.4.1 Secure Names</td>
<td>131</td>
</tr>
<tr>
<td>7.4.2 Trees</td>
<td>133</td>
</tr>
<tr>
<td>7.4.3 Graphs</td>
<td>134</td>
</tr>
<tr>
<td>7.4.4 Single Signature Scheme</td>
<td>134</td>
</tr>
<tr>
<td>7.5 Performance Results</td>
<td>135</td>
</tr>
<tr>
<td>7.5.1 CRSA/BGLS-based Schemes</td>
<td>139</td>
</tr>
<tr>
<td>7.5.2 Single Signature Scheme</td>
<td>140</td>
</tr>
<tr>
<td>7.6 Discussion</td>
<td>142</td>
</tr>
<tr>
<td>7.6.1 Forests</td>
<td>143</td>
</tr>
<tr>
<td>7.6.2 Encrypted Trees, Graphs and Forests</td>
<td>143</td>
</tr>
<tr>
<td>7.6.3 Dynamic Trees, Graphs and Forests</td>
<td>143</td>
</tr>
<tr>
<td>7.6.4 Automatic Recovery from Structural Errors</td>
<td>144</td>
</tr>
<tr>
<td>7.6.5 Path Queries</td>
<td>144</td>
</tr>
<tr>
<td>7.7 Summary</td>
<td>145</td>
</tr>
<tr>
<td>8 SECURE PUBLISH/SUBSCRIBE OF XML</td>
<td>146</td>
</tr>
<tr>
<td>8.1 Some Simple Observations</td>
<td>147</td>
</tr>
<tr>
<td>8.2 XML Data Model</td>
<td>147</td>
</tr>
<tr>
<td>8.3 Document Encoding and Encryption</td>
<td>148</td>
</tr>
<tr>
<td>8.4 Structure-based Routing</td>
<td>149</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.4.1 Content Routers</td>
<td>151</td>
</tr>
<tr>
<td>8.4.2 Dissemination Network</td>
<td>153</td>
</tr>
<tr>
<td>8.4.3 Content Publishing</td>
<td>155</td>
</tr>
<tr>
<td>8.4.4 Document Verification</td>
<td>157</td>
</tr>
<tr>
<td>8.4.5 Update Management</td>
<td>158</td>
</tr>
<tr>
<td>8.5 Discussion</td>
<td>158</td>
</tr>
<tr>
<td>8.6 Summary</td>
<td>161</td>
</tr>
<tr>
<td>9 AUTHENTICATION OF OBJECTS</td>
<td>162</td>
</tr>
<tr>
<td>9.1 Introduction</td>
<td>162</td>
</tr>
<tr>
<td>9.2 Objects</td>
<td>163</td>
</tr>
<tr>
<td>9.3 Object Trees</td>
<td>164</td>
</tr>
<tr>
<td>9.4 Redaction of Objects</td>
<td>166</td>
</tr>
<tr>
<td>9.5 Authentication of Objects</td>
<td>168</td>
</tr>
<tr>
<td>9.5.1 Scheme Based on Merkle Hash Technique</td>
<td>169</td>
</tr>
<tr>
<td>9.5.2 Leakage-free Scheme</td>
<td>170</td>
</tr>
<tr>
<td>9.6 Summary</td>
<td>171</td>
</tr>
<tr>
<td>10 CONCLUSIONS</td>
<td>172</td>
</tr>
<tr>
<td>10.1 Research Contributions</td>
<td>172</td>
</tr>
<tr>
<td>10.2 Open Problems</td>
<td>174</td>
</tr>
<tr>
<td>10.3 Future Research Directions</td>
<td>175</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>177</td>
</tr>
<tr>
<td>VITA</td>
<td>188</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Leakage during authentication of $T_{δ2}$ using MHT</td>
</tr>
<tr>
<td>3.2</td>
<td>Inference of sensitive information from leakages (Continued to Table 3.3)</td>
</tr>
<tr>
<td>3.3</td>
<td>Inference of sensitive information from leakages (Continued from Table 3.2)</td>
</tr>
<tr>
<td>3.4</td>
<td>XML elements for the healthcare records</td>
</tr>
<tr>
<td>3.5</td>
<td>Leakage via cross-edge $e(g_{11}, g_{12})$, subgraph: $G_{δ1}$</td>
</tr>
<tr>
<td>3.6</td>
<td>Leakage via back-edge $e(g_{14}, g_{9})$, subgraph: $G_{δ2}$</td>
</tr>
<tr>
<td>6.1</td>
<td>Acronyms and notations</td>
</tr>
<tr>
<td>6.2</td>
<td>Information leakages via edge-types</td>
</tr>
<tr>
<td>8.1</td>
<td>Encoding of XML tree in Figure 8.1</td>
</tr>
<tr>
<td>8.2</td>
<td>Information at a router for PEPON 43 in Figure 8.3</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(a) Tree $T$ and subtree $T_\delta$. (b) Graph $G$ and subgraph $G_\delta$.</td>
<td>3</td>
</tr>
<tr>
<td>3.1</td>
<td>An example tree with each node having some content.</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>XML-based Health-care record of a patient.</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>The tree representation of the HealthRecord in Figure 3.2.</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>A health-record graph; tree-edges in bold.</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Algorithm to sign a tree.</td>
<td>60</td>
</tr>
<tr>
<td>5.2</td>
<td>Protocol for trees.</td>
<td>61</td>
</tr>
<tr>
<td>5.3</td>
<td>Algorithm to compute the redacted signature of a subtree $T_\delta(V_\delta, E_\delta) \subseteq T(V, E)$.</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>Algorithm to verify the authenticity of a redacted subtree $T_\delta(V_\delta, E_\delta)$.</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Algorithm to reconstruct the structure of a tree given the set of structural positions.</td>
<td>65</td>
</tr>
<tr>
<td>5.6</td>
<td>(a) Post-order and pre-order numbers assigned to the healthcare record as (PON, RON). (b) Randomized post-order and pre-order numbers assigned to the healthcare record as (RPON, RRON).</td>
<td>66</td>
</tr>
<tr>
<td>5.7</td>
<td>CRSA: Average time to sign versus number of IVs.</td>
<td>70</td>
</tr>
<tr>
<td>5.8</td>
<td>BGLS: Average time to sign versus number of IVs.</td>
<td>71</td>
</tr>
<tr>
<td>5.9</td>
<td>CRSA: Average time to distribute versus number of IVs.</td>
<td>71</td>
</tr>
<tr>
<td>5.10</td>
<td>BGLS: Average time to distribute versus number of IVs.</td>
<td>72</td>
</tr>
<tr>
<td>5.11</td>
<td>CRSA: Average time to verify versus number of nodes.</td>
<td>72</td>
</tr>
<tr>
<td>5.12</td>
<td>BGLS: Average time to verify versus number of nodes.</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>A graph with depth-first tree in bold.</td>
<td>81</td>
</tr>
<tr>
<td>6.2</td>
<td>Algorithm to sign a DAG.</td>
<td>86</td>
</tr>
<tr>
<td>6.3</td>
<td>Algorithm to redact a DAG.</td>
<td>87</td>
</tr>
<tr>
<td>6.4</td>
<td>Protocol for DAGs.</td>
<td>87</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>6.6</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>6.9</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>6.10</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>6.11</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>6.12</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>6.13</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>6.14</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>6.15</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>7.2</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>7.3</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>7.4</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>7.5</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>7.6</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7.7</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>7.8</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>7.9</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>7.10</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>7.11</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>7.12</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>7.13</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>7.14 Average number of attempts to assign a secure name to a node; branching factor $\leq 100$.</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>7.15 Average time in micro-sec to assign a secure name to a node; branching factor $\leq 100$.</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>7.16 Average number of attempts to assign a secure name to a node; branching factor $\leq 300$.</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>7.17 Average time in micro-sec to assign a secure name to a node; branching factor $\leq 300$.</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>7.18 CRSA: Time to sign a tree.</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>7.19 CRSA: Time to redact a subtree.</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>7.20 CRSA: Time to verify a subtree.</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>7.21 Computation of signature of a tree.</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>7.22 Computation of redacted signature of a subtree.</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>7.23 Computation of verification of a subtree.</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>8.1 (a) A tree: abstract representation of an XML Document, (b) Post-order numbers associated with each node and (c) Randomized post-order numbers associated with each node.</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>8.2 Three sub-trees of the content tree are shared with three consumers: consumer 1, 2 and 3.</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>8.3 Routing of three sub-trees to consumers using RPONs.</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>9.1 Tree representations of members of objects: (a) primitive data type, (b) array of primitive types, (c) methods, and (d) instances of user-defined types.)</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>9.2 (a) Class hierarchy of Emp and RegEmp, (b) Object Tree of objRegEmp1. (c) Redacted object tree for scenario 2, (d) Redacted object tree for scenario 3.</td>
<td>169</td>
<td></td>
</tr>
</tbody>
</table>
ABBREVIATIONS

PON      Post-Order Number
RON      Pre-Order Number
ION      In-Order Number
RPON     Randomized Post-Order Number
RRON     Randomized Pre-Order Number
RION     Randomized In-Order Number
IV       Integrity Verifier
CRSA     Condensed-RSA
BGLS     Boneh Gentry Lynn Sacham (Aggregate signature scheme)
POPF-CCA Pseudo-random Order-Preserving Function Chosen Ciphertext Attacks
IND-CCA  Indistinguishability against Chosen Ciphertext Attacks
EU-CMA   Existentially Unforgeable against Chosen Message Attacks
ABSTRACT

Kundu, Ashish Ph.D., Purdue University, December 2010. Data in the Cloud: Authentication without Leaking. Major Professor: Elisa Bertino.

Third party data distribution frameworks such as the cloud are increasingly being employed in order to store, process, and publish sensitive information such as healthcare and finance information, belonging to individuals and enterprises. Such data objects are often organized as trees, graphs or even forests (e.g., XML). In third party frameworks, not only authentication of data is important but also protection of privacy and assurance of confidentiality are important. Moreover, data authenticity must be assured even when the data object that a user has access to consists of subset(s) of the signed data.

Existing solutions such as Merkle hash technique and the redactable signature schemes lead to leakages of structural information, which can be used to infer sensitive information, which in turn would lead to privacy and confidentiality breaches. So the question is: can we authenticate subset(s) of signed data objects without leaking, and if so, how efficiently such authentication can be carried out? We have reported a positive result by presenting efficient and provably secure solutions not only for trees, but also graphs and forests. We have presented a scheme that computes only one signature per tree, graph or forest.

Our schemes support encrypted data to be stored at third-party services. Our schemes can also be used to automatically recover from structural errors in tree-structured data, and for leakage-free authentication of paths (e.g., XPaths). Further, as the applications of our schemes, we have also developed a publish/subscribe model for XML – Structure-based routing, and a scheme for authentication of objects.
1 INTRODUCTION

A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine.

Alan M. Turing [130]

Third-party model of data distribution and computing has been of growing interest to the enterprises and businesses over the past decade. Such a model helps organizations achieve economies-of-scale while focusing on their core competencies, and deliver better products/services. The explosive growth in the amount of data that needs to be collected, stored, processed, analyzed, distributed, displayed, and even destroyed as and when needed, requires significant amount of technological and financial investment on the part of enterprises. Such an investment spans over the lifetime of the deployment of several technological products and services, and thus puts a continuous strain on the enterprises and their revenue. The third-party model provides viable and cost-effective alternatives for the said requirements while minimizing such strains and allowing enterprises focus on their core-competencies. As a result, recently, cloud computing [132] has emerged as the umbrella area that covers all the business and computing aspects of third party data distribution and computing models.

In the emerging cloud-computing paradigms, which are increasingly being employed in order to store, process, and distribute sensitive information belonging to individuals and enterprises, protection of privacy and confidentiality are as important as assuring authenticity of such data (e.g., [7, 24]). As it is, privacy is an important problem in data publishing itself(e.g., [91, 93]). In a third-party model, there is an authorized owner of the data, which maybe the source of the data, one or more third-party services (or distributors), and one or more queriers/subscribers (collectively called as users or data consumers). In the cloud, the distribution of data is carried out by third party services (such as in “Database as a Service” [69]). Such third-
party services may not be trusted (e.g., Amazon EC2 and Amazon Web Services: AWS [1]). In such third-party data distribution setting, an important requirement is to assure data authenticity. An authentication scheme is used to verify (1) the integrity of data, and (2) that the claimed owner is in fact the authorized owner of the data. Authenticity is typically assured by message authentication codes or digital signatures computed by the authorized owner of the data, which in third-party distribution setting, are different from the party distributing the data (referred to as distributor). Data objects in the context of third-party architectures are very often organized as trees, graphs or even forests (set of disconnected trees/graphs); for example, data organized according to XML schemas. Often users receive part of the data that is stored at the database(s), as users maybe authorized to access only a subset of the data. For example, a querier receives a part of an XML document or a relational table from a database instead of the complete XML document or the table. Consequently, data authenticity must be assured even when the data that a user can access, is a subset of the signed data. A crucial requirement is to ensure that the techniques are used for verifying the data authenticity do not result in data leakages in order to protect privacy of the users and confidentiality of data. Such leakages can be used to infer sensitive information that is not part of the received data, which in turn would lead to privacy and confidentiality breaches.

When addressing the problem of authentication of trees, graphs and forests, it is important to notice that each node may contain some contents and that the edges and ordering between nodes may establish some relationships between the contents in these nodes. Such relationships may be defined according to properties such as classification, indexing, temporal-orientation and sensitivity of the contents [60]. Integrity of such edges and ordering between the nodes is referred to as structural integrity, whereas the integrity of the contents is referred to as content integrity. An authentication scheme for tree structures must thus preserve both content and structural integrity. An additional requirement for authentication of sensitive data is to maintain the confidentiality of the content and the structural information [134]. By
confidentiality, we mean that: (i) a user receives only those subtree(s) that the user is authorized to access, according to the stated access control policies, (ii) a user should not receive nor should be able to infer any information about the nodes and edges that the user is not authorized to access. Nodes and edges that are in a tree/graph but not in the subtree(s)/subgraph(s) that a user receives are referred to as *extraneous information*. Let us consider two application scenarios.

**Healthcare:** XML is the de-facto standard for specification of healthcare records. A doctor’s system uploads an XML-based healthcare record to a third-party service, and a query from an end-user such as nurse would be processed by the third-party service. The query result may be one or more sub-documents in the XML document. In such a scenario, the user should be able to verify the authenticity of the query result, and due to privacy requirements (such as HIPAA [2] requirements), the user should not be able to infer any information about the remaining nodes (diseases/treatments) and edges (types of diseases) in the source XML healthcare record.

**Finance:** XML is used to specify financial information of individuals and financial accounts. When a user receives part of the financial document of another individual or enterprise, she should not be able to infer existence of any other nodes (bank accounts/credit cards/loans) and edges (types of accounts/cards/loans) in the source financial record.
Before we describe the problem, let us describe the third-party model informally.
The formal security model is given elsewhere (Chapter 4).

Data

Trees, Graphs and Forests: A directed (rooted) tree $T(V,E)$ or a directed graph
$G(V,E)$ is a data object, where $V$ and $E$ are the sets of vertices and edges in $T$. A
node $x$ represents an atomic unit of the data referred to as $c_x$. $e(x,w)$ represents a
directed edge from $x$ to node $w$. The fact that $x$ precedes one of its siblings $y$ in
a DAG is denoted by $x \prec y$. A subtree $T_\delta(V_\delta,E_\delta)$ that is shared with a user as a
result of a query on the tree $T(V,E)$ is a subtree $T_\delta(V_\delta,E_\delta) \subseteq T(V,E)$. Likewise, a
subgraph $G_\delta(V_\delta,E_\delta) \subseteq$ graph $G(V,E)$. $\Upsilon(V,E)$ represents either a tree or a graph.

Third-Party Model

Authorized owner: Alice, the authorized owner of one or more data object(s) digitally
signs each data object. The signature is referred to as $\sigma_\Upsilon(V,E)$, where $\Upsilon(V,E)$
denotes the data object. After signing $\Upsilon$, Alice may delegate the job of publishing
$\Upsilon$ or processing queries over $\Upsilon$ to one or more third-party distributors $D$ (such as a
cloud server).

Users: A user (also referred to as Bob) receives a subtree/subgraph/sub-forest from
a third-party distributor and verifies its authenticity.

Third-party distributors: A third-party distributor $D$ processes queries from a clien-
t/user Bob on the data objects, and sends the query result(s) to Bob. The data
object that a user receives as a result of its query on a tree or a graph is a subtree
or a subgraph, respectively. $D$ also sends a signature $\sigma_\Upsilon_\delta$ for $\Upsilon_\delta$ so that the user can
verify the authenticity of the $\Upsilon_\delta$. 
Authorized owner is trusted. The third-party distributors are untrusted in the following sense: a distributor can carry out data tampering attacks, it maybe vulnerable to disasters or security compromises, and it does not have any signature authority on behalf of the authorized owner.

Threats

Throughout the dissertation, we assume a probabilistic polynomial time (PPT) adversary [61]. There are two types of threats.

1. **Data tampering attack**: by an adversary over the communication channel: main-in-the-middle attack or at the distributor. The adversary may tamper the content of one or more nodes, the structural order and/or the type of structural relation (edge) between two or more nodes of a tree/subtree.

2. **Inference attack**: A user, who receives $\Upsilon_\delta$ that is a portion of the data object $\Upsilon$, infers information (at least one bit of information) about $(\Upsilon \setminus \Upsilon_\delta)$ from the signature $\sigma_{\Upsilon}$ for $\Upsilon_\delta$ that it receives from $D$.

In order to protect from data tampering attacks as well as inference attacks, we need to develop “leakage-free” data authentication schemes.

1.1 The Problem

The problem is how does Alice (the data owner) sign the data (a tree, a graph, or a forest) once, so that the authenticity of a subset of the data (subtree(s)/subgraph(s)/sub-forest(s)) can be verified without leaking any information about the remaining part of the data. A leakage-free signature scheme must make it possible for the receiver of one or more subtree(s)/subgraph(s)/sub-forest(s) in order to verify the authenticity of data with the following crucial requirement: the receiver should not be able to infer any extraneous information; extraneous information in a tree/graph/forest $\Upsilon(V, E)$ with respect to its subset $\Upsilon_\delta(V_\delta, E_\delta)$ comprises of $(\Upsilon \setminus \Upsilon_\delta)$ the nodes and edges that
are in $\mathcal{Y}$ but not in $\mathcal{Y}_d$ (such as $b, f, g$ and $h$, edges: $e(d, b)$, $e(h, d)$, $e(h, g)$, and $e(g, f)$ w.r.t. $T_d$ in Figure 1.1).

Consider the tree in Figure 1.1(a) and suppose that the user receives the subtree $T_d$. The user should neither receive nor should be able to infer anything about the nodes $b, f, g$ and $h$, and edges $e(d, b)$, $e(h, d)$, $e(h, g)$, and $e(g, f)$. In the case of graphs, another basic form of leakage is about immediate ancestors of a node. For example, assume that the user receives $G_d$, subgraph of $G$ in Figure 1.1(b). The user should not learn that node $c$ has another immediate ancestor other than $d$ (which is $h$), which is in $G_d$. For nodes $a, b$ and $d$, the user should not learn whether they have any other immediate ancestors in the graph.

The notion of redactable signatures and sanitizable signatures are used for authentication of subsets of a data object that is signed by an authorized owner. However, existing solutions (See Chapter 2) leak structural information; so does the widely used Merkle Hash Technique, which is used for authentication of trees and has also been extended for DAGs.

_A Straightforward Scheme_: A straightforward scheme for trees and graphs is to sign each node as well as each edge (Brzuska et al. [29] proposed such a scheme for trees). When a user has access to a subgraph (including a subtree), the signatures of the nodes and edges in the subgraph are also sent to the user. However, such a scheme has two major drawbacks: (1) If the graph has an ordering between the siblings, it would not be possible for the user to verify such ordering; (2) the total number of signatures computed and stored in the worst case is $O(n^2)$, where $n$ is the number of nodes in the graph. This is because, a graph may have $O(n^2)$ number of edges. For a graph of even medium size such as one thousand nodes, about a million signatures need to be stored and managed by the distributor, which is quite large. For a tree, it would have $2n - 1$ signatures. The question is can we support (1) while using less number of signatures?
1.2 Contributions

Highlights of our contributions are as follows. As far as we know, this work is the first such work that addresses authentication of data (ordered trees and graphs) and the related leakage-free requirements in a holistic manner.

1. We have characterized several inference attacks that can be carried out on the widely used Merkle hash technique [87].

2. We have shown that leakages of structural information can lead to privacy and confidentiality breaches, in several scenarios such as in healthcare. We have demonstrated such leakages and breaches in the context of healthcare scenarios [84,86].

3. Existing formal models in security do not cover the notion of leakage-free authentication of trees, graphs and forests. We have provided the first formal security model for leakage-free redactable signatures. Security notions of unforgeability, privacy and transparency have been formally defined as part of this model [81].

4. A signature scheme called as “structural signatures” based on the traversal numbers of a tree has been proposed that can be used to authenticate a subtree of a tree in a leakage-free manner. For this purpose, we have defined the notion of randomized traversal numbers [84]. Structural signatures scheme can be used for leakage-free authentication of induced subtrees.

5. Structural signature schemes for graphs – both directed acyclic graphs, and graphs with cycles, have been proposed that facilitate authentication of a subgraph in a leakage-free manner [86].

6. In order to facilitate the leakage-free authentication of multiple subtrees/subgraphs, the notion of secure names have been developed and a cryptographic construction has been presented [80].
7. Based on the secure names and an existing redactable signature scheme for sets, we have defined a highly efficient and generic leakage-free redactable signature scheme that can be used not only for trees, but also for graphs and forests. Our scheme compute only one signature for any tree/graph/forest, thus is an optimal leakage-free redactable signature scheme [81].

8. Our schemes for leakage-free authentication can also be used when for encrypted data, when nodes are encrypted.

9. We have shown how the structural signatures can be used to automatically recover from structural errors in tree-structured data. Such a scheme has applications in several scenarios such as satellite-based data transmission and sensors [87].

10. Our schemes can be used for leakage-free authentication of paths, and has applications in secure evaluation of XPath [6] queries.

11. As an application of the structural signatures, we have developed a publish/subscribe scheme for XML distribution called as “structure-based routing”, which uses randomized traversal numbers not only for verification structural integrity, but also for efficient routing of contents [82, 83].

12. Objects are a primary model of data representation in object-oriented software, web services, and object-oriented databases. Serialized objects are stored and/or transmitted in environments that involves untrusted third-party entities. In such scenarios, leakage-free authentication of objects is an important security requirement. We have applied our signature schemes in this context, and have developed the first such scheme for this problem [85].

13. We have presented the security analysis, and performance analysis of the proposed signature schemes wherever it is required.
1.3 Dissertation Roadmap

Chapter 2 describes the related works. Inference attacks on the Merkle hash technique, and leakages of structural information in trees and graphs in a healthcare context are described in the next chapter - Chapter 3. A formal security model for leakage-free redactable signatures with the notions of unforgeability, privacy and transparency, is presented in Chapter 4, which also describes the formal notion of secure names. Structural signatures for trees are described in the next chapter, and structural signatures for graphs are given in Chapter 6. Leakage-free redactable signatures for trees, graphs and forests are given in the next Chapter 7. An application of structural signatures: structure-based routing of XML documents in a publish/subscribe model is given in Chapter 8, and a scheme for authentication of objects using the Merkle hash technique and our leakage-free scheme is given in Chapter 9. Chapter 10 summarizes the dissertation, describes some open problems and future research directions.
2 RELATED WORKS

An algorithm must be seen to be believed.
Donald E. Knuth [77]

2.1 Merkle Hash Technique

Merkle [99, 100] proposed a digital signature scheme based on a secure conventional encryption function over a hierarchy (tree) of document fragments. Since then, this technique has been used widely, but always with an authentication path of logarithmic size to verify even a single document fragment. It also leads to leakage of information (discussed in Section 3.1). Buldas and Laur [32] have also found that Merkle trees are binding (integrity-preserving) but not hiding (confidentiality-preserving).

The use of commutative hash operations (one-way accumulators [17, 65]) to compute the Merkle hash signature prevents leakage related to the ordering among the siblings. However, it cannot prevent the leakage of signatures of a node and the structural relationships with its descendants or ancestors. Moreover, one-way accumulation is very expensive (due to modular exponentiation) in comparison to the one-way hash operation. By salting [76], the Merkle hash of each node is secured against the known-plaintext and known-ciphertext inference attacks. However, salting cannot prevent structural inference attacks completely even when coupled with one-way accumulators, because structural information would still be leaked. There have been several works [18, 73, 94] in order to optimize traversal of Merkle hash trees so that the auxiliary information required for authentication can be determined efficiently. Improved (more efficient) versions of MHT have also been proposed [31].
The MHT has been widely used for data authentication [19,53,54,68,92,113,124]. Bertino et al. [20] proposed a technique based on the Merkle hash technique for selective dissemination of XML data in a third party distribution framework. In [19], Bertino et al. have defined an authentication scheme using MHT for UDDI repositories towards building secure web services directories. Using techniques from incremental cryptography, Naor and Nissim [106] dynamize hash trees to support the insertion and deletion of elements. In their scheme, the source and the directory maintain identically implemented $2^{-3}$ trees. Each leaf of such a $2^{-3}$ tree stores an element of set, and each internal node stores a one-way hash of its children’s values. Hence the source-to-directory communication is reduced to $O(1)$ items, but the directory-to-user communication remains at $O(\log n)$. Thus, their solution is however not size oblivious. Martel et al. [98] proposed an authentication technique for data structures - directed acyclic graphs referred to as “Search DAGs” in third party distribution frameworks. However their technique uses Merkle hash technique [100], which leaks information during authentication [32,82–84,87]. Moreover the “Search DAGs” technique only covers DAGs, not the general directed graphs. For secure multicast, Perrig uses static data ordering over symmetric encryption [117]. Li et al. [92] propose a scheme for trustworthy verification (authentication) of write-only read-many data based on the MHT. Singh and Prabhakar [124] developed an authentication scheme for uncertain databases using MHT. Kocher [78] presented a scheme using Merkle hash technique for data distribution using third party models.

2.2 Authenticated data structures

Authenticated data structures (ADS’s) have been used to verify the authenticity of data in a static as well as in the dynamic context. Goodrich and Tamassia [63] proposed authenticated dictionaries using skip lists and commutative hashing (one-way accumulators). Goodrich et al. [67] proposed techniques to authenticate graphs with specific path queries and geometric searching. Goodrich et al. [67] define a generalized
Merkle Hash tree for paths in graphs, and propose a mechanism for authenticating path queries in graphs. They leak structural information because their notion of “path hash accumulator” and their use of generalized merkle hash technique. Martel et al. also proposed Search DAGs [98] using the Merkle hash technique, which however leaks information. Papamanthou et al. [115] propose authenticated hash tables, where leakage-free property is not achieved, rather efficiency is of primary concern. Goodrich et al. [67] provides an excellent related works section on authenticated data structures and we refer interested readers to this paper. Atallah et al. [8] have developed a scheme for authentication of 2-dimensional data using only one aggregate signature. Atallah et al. [9] developed schemes for covering a tree with minimum number of paths such that the total number of nodes in all the paths is minimized. Such a scheme has applications in verification of integrity of trees and paths. However, it is not obvious on how to develop a leakage-free authentication scheme for trees or paths using such solutions.

2.3 Transitive Signatures

Transitive signatures were originally conceptualized by Rivest in his seminal talk in 2001 [119], and a concrete scheme for undirected graphs were developed by Micali and Rivest [102]. In such signature schemes, the objective is to use the signatures of two or more messages and compute the signature of the messages that is derived from these messages. For example, in a binary tree (in a network route topology), given the signature of two child nodes (routers), the objective is to compute the signature of their parent node (router) [42].

The notion of transitive signatures by Micali and Rivest [102] helps computing a graph in a dynamic manner from authenticated nodes and edges so that the graph built at any point of time is an authenticated one. There are a number of interesting works on directed trees and their transitive signatures. However, they leak information about the history of the transitive signature - from the paths it has been
computed [14, 15, 108, 135, 136]. By history independence, it means that verification of a transitive signature for \((i, k)\) would not leak information about node \(j\) and/or its signature. The latest paper by Xu [135] use the notion of RGGMs developed for redactable signatures by Chang et al. [41] in order to develop transitive signatures on a path, where it is history independent. However, it uses RGGM trees and thus are quite expensive: for a given path, it computes one RGGM tree.

2.4 Redactable Signatures

Redactable signature schemes (e.g., [74]) inspired by transitive signatures support computation of valid signatures for modified messages from the signatures of the source messages. Anyone with the knowledge of a message, its signature and the associated public key can compute a signature of a redacted message. (In contrast, in case of sanitizable signatures, only a designated sanitizer can compute such a signature.) However, such schemes have been developed only for documents that are linear sequences of sub-documents, and leak structural information. On exception is the redactable signature scheme for sets by Johnson et al. [74]. They use RSA-based one-way accumulators and compute a single signature for any set, which allows anyone to computed the signature of the subset without leaking any information about the remaining elements. In this dissertation, we have developed leakage-free redactable signatures for non-linear and complex data objects such as trees and graphs.

**Formalization of our work:** Brzuska et al. [29] extended our work [84] and attempted to formalize the notion of structural signatures for trees. However, their formal model is quite restrictive: does not support query results to be disconnected subtrees, and do not address graphs and forests. Moreover, its definition of integrity is flawed: it claims to support security as strong as “existentially unforgeable under adaptive chosen message attack”, which however cannot be supported by redactable signatures as shown by Johnson et al. [74]. Moreover, their construction is quite expensive: computes a linear number of signatures for the tree, and for authenticating
the ordering among sibling, computes a quadric number of signatures. An open problem Brzuska et al. pose is to find a more efficient solution for trees, which we solve in this dissertation.

*Content Extraction Signatures* Steinfeld [126] proposed content extraction signatures in which they use RSA homomorphic property to build a redactable signature on documents: the messages are deleted but the *structural position is leaked*. This scheme is a form of redactable signatures. Bull et al. [33] proposed content extraction signatures for XML documents, which however, leaks structural information.

2.5 Sanitizable Signatures

: There are two types of sanitization of documents, where a document is a collection of several messages/sub-documents: one that allows modification of messages in a document (e.g., [10], [30]), and another that allows only deletion of the messages [103]. Such signature schemes focus on hiding only the messages being sanitized, but not their structural position in the document and the structural relationships they have with other messages. Brzuska et al. [30] propose a notion of “unlinkability” between sanitized documents. Unlinkability means that the hidden/sanitized messages in a sanitized version cannot be recovered even with the knowledge of two or more sanitized versions of the same document. On the other hand, our notion of “leakage-free” signatures focus on hiding not only contents (nodes as messages) but also structure of the original document. Moreover, some of these schemes have a designated entity called “sanitizers”. Leakage-free signatures are redactable in nature: applicable to a more relaxed notion of security models that involve untrusted third-parties. Haber et al. [68] proposed a signature scheme that allows redaction, and pseudonymization of parts of data (messages/sub-documents), as well as controls redaction of a document by prohibiting it. It does not leak information about messages that have been redacted/pseudonymized, however it leaks information about the existence of such
messages and their structural positions. Hiding not only the messages but also the structural information is the purpose of our leakage-free signature scheme.

2.6 Commitment Schemes

Commitment schemes [116] are not signature schemes. Let \( c \) is a commitment of value \( m \); \( c \) is not a signature of \( m \), because (1) \( m \) remains hidden until \( c \) is opened; (2) \( c \) can be forged easily as everyone knows the public key being used to compute \( c \) from \( m \); (3) in order to verify the so called “signature” \( c \) of \( m \), the signer has to release its private key, which is not a case in digital signatures. Set commitment schemes prove that an element is in fact an element of a set in a zero knowledge manner [101]. In commitment schemes, an important issue is who generates the commitment key pair \((pk, sk)\): is it the sender (who commits), the recipient (who receives) or a trusted third party. There are solutions using trusted third party. But in our setting in the dissertation, third party is not “trusted”. The primary objective is hiding, and then proving, whereas in a signature scheme, the primary objective is proving.

2.7 Proxy Signatures

There have been several works on proxy signatures (e.g., [27, 88, 97]), where the data owner delegates the signing of a data object to a proxy, who computes a proxy signature of the data. A verifier uses the public key of the proxy and validates the data against the proxy signature. Such a model is similar but different from the third-party model that is being used in this work. One can assign a proxy key pair to each third party server, who would sign each subtree/subgraph that is the result of the graph (that it has not signed earlier). However, the proxy in the signature may not be trusted but not malicious as the third-party servers in our model are assumed to be. Moreover, signing each subtree/subgraph as a query result on behalf of the data owner dynamically exposes the server to side channel attacks, key recovery attacks as well as denial of service attacks.
2.8 Zero-Knowledge Sets and Membership Queries

ZKS (Zero-Knowledge Sets) have been proposed by Micali, Rabin and Killian [101], later extended by [40]. The setting of ZKS has a prover and a verifier engaging in non-interactive zero-knowledge proofs about membership of a value in a set. The prover commits a set, and then generates proof for an element without revealing anything about the other elements, even the size of the set. Such a scheme comes quite close to what we want to achieve, but there are several differences: (1) the setting is different: in our case, the data owner needs to sign a tree/graph and give it to an untrusted server(s), who then evaluate queries from users. (2) the user needs proofs of subtree/subgraph relationships instead of a set membership. (3) we need a digital signature instead of a commitment scheme. (4) In our case, the untrusted server should be able to compute a proof for the subtree/subgraph the user receives using the digital signature computed by the owner, and should not be able to forge such proofs.

Membership queries and associated proofs of their answers have been studied by [101]. Ostrovsky et al. [111] proposed a consistency proof for membership queries in a database that also protects privacy. In their model, a database is a set of key-value pairs. Privacy means that the user can only know the answer it receives and it cannot learn any other information about the other key-value pairs in the database. However, they cannot hide the size of the database ( [111]: Page: 7). Our leakage-free signatures achieve this: hiding the size of a tree/graph or a database of trees/graphs. They also use a notion of “explicit hashing” which is constructed such that the user always learns that the same number of paths irrespective of the query it issues. It is achieved by issuing dummy pre-images. Moreover, their protocol is used for membership queries in sets and range queries. It is not trivial as how to apply their scheme to solve the problem for trees and graphs. Moreover, their scheme is a commitment scheme, which is different from digital signature schemes.
2.9 Hashing and Accumulation Schemes

Merkle-Damgard [50,99] (MD) model of hash functions (conceptually modeled as random oracles [16]) is widely used in designing modern hash functions such as SHA1 and MD5. Their proposal is to divide an arbitrary length message into blocks of identical size $k$ (pad the last block if necessary), and apply a compression function such as a block cipher on the blocks sequentially with the output of the compression of block $b_i$ given as an input to the compression of the next block $b_{i+1}$. At the end, a finalization stage that appends bits and information related to the size of the entire message to the output compression of the last block $b_n$. Coron et al. [46] define a new security notion for random oracles and the hash functions defined thereof: for a hash function $H$ to behave as a random oracle, the underlying fixed-size block cipher must behave as a random oracle as well. Such a property is stronger than the standard collision resistant property of hash functions. Coron et al. show that the MD model does not satisfy this property of hash function. Dodis et al. [56] show that a pre-image attack can be carried out on MD-hash functions by knowing the padding used. They propose a methodology for how to strengthen the one-way compression function in order to prevent such attacks.

Accumulators: Exponential accumulators have been proposed by Benaloh and Mare' [17]. The core scheme works as follows: a seed $y_0$ is used such that $y_0 > 1$, $y_0$ and $N$ are relatively prime, $N = pq$, as in RSA. Accumulator $H(y_0, x_1) = y_0^{x_1} \mod N$; $H(y_{i-1}, x_i) = (y_{i-1})^{x_i} \mod N$. It is commutative, but not associative [63]. So incremental computations are not feasible. It is important for its implementation that the seed $y_0$ of the accumulator should be relatively prime to the RSA primes $p$, and $q$ [121]. This work has been improved by Baric and Pfitzmann [13] with efficient constructions using strong RSA assumptions. Baric and Pfitzmann developed two RSA-based accumulators: one without random oracles, and the other one using random oracles and they prove that they are collision-resistant assuming that the RSA is
an one-way function. They also proposed fail-stop signature schemes based on the accumulators. The public keys are accumulated and the messages are also accumulated, if there are more than one message. However, the update operations on such accumulators were not size-oblivious (size of the set being accumulated). Camenisch and Lysyanskaya [34] made the update operations size-oblivious. Li et al. [90] built upon this work and developed accumulators using which non-membership can be proven efficiently, and memberships can be revoked in anonymous manner.

2.10 Traversal numbers

Traversal numbers have been used for querying and navigation of XML data by Zezula et al. [137]. However, they do not address any security issues. Wang et al. [134] have used a notion similar to traversal numbers in defining the structural index in XML databases in order to be able to locate encryption blocks as well as their unencrypted data nodes that satisfy user query. They use real intervals [0, 1] for root and every child of the root is assigned a sub-interval such as [0.5, 0.6]. The first entry in the interval can be assumed to be referring to the pre-order number and the second one to the post-order number. However, they do not derive such an interval from traversal numbers nor do they use traversal numbers for signing trees. None of the previous approaches propose the use of randomized traversal numbers for the signature of trees. As such, the previous approaches do not include security analysis, performance evaluations nor detailed comparison with the MHT and other secure data publishing techniques derived from the MHT.

2.11 RSA Signatures

The Condensed-RSA signature scheme by Mykletun et al. [105] is an aggregate signature scheme. They compare Condensed-RSA and the aggregate signature scheme by Boneh et al. [28] in order to efficiently authenticate outsourced databases. Condensed-RSA is secure as against another similar signature scheme: Batch-RSA.
Batch verification of RSA signatures were first proposed by Harn [70] and was thought to support secure signature aggregation. However, such a scheme is insecure (Hwang et al. [72] first showed a forgery attack). Later in another paper [71] Hwang et al. proposed a new scheme overcoming the weakness of Harn’s scheme. However, Bao et al. [12] have shown that even this scheme is not secure. The scheme proposed in [12] has also not been proven to be secure. The Condensed-RSA signature scheme is secure under the RSA assumption, because the user does not receive individual signatures [105]. The aggregate signature scheme by Boneh et al. [28] is based on elliptic curves and bilinear maps, and is proven to be secure.

2.12 Privacy-preserving Authentication of Query Results

Confidentiality and authenticity are important requirements in secure data management and publishing [22, 25]. Some of the other notable authentication schemes in the literature are by Li et al. [89], Mouratidis et al. [104], Pang and Mouratidis [113], and by Pang and Tan [114]. The authentication scheme for completeness of query results proposed by Pang et al. [112] leaks even in the simple case of “greater than” predicates. In their scheme, if $a_3$, $a_4$, and $a_5$ satisfy the “greater than” predicate among elements $a_1, a_2, \ldots, a_{10}$, then the user uses auxiliary information about $a_2$ and $a_6$ in order to authenticate the results; such extra information leads to leakage about the existence and structural position of these two elements in the data object, which as we have seen earlier could be sensitive information or could lead to inference of sensitive information. Mykletun et al. [105] use aggregation of signatures for efficient authentication of tuples in outsourced databases. Ma et al. [96] proposes an approach using Merkle hash technique and aggregate signatures [28, 105] in order to authenticate query results in databases. It computes an attr-MHT for each tuple and signs the Merkle hash for the tuple; signatures of the tuples are aggregated to compute the signature of the whole table. However, this scheme leaks: consider the case in which a querier does not receive all (possibly some of these are sensitive at-
tributes), but receives only some attributes of a tuple in the result set. Narasimha and Tsudik [107] proposed another scheme for authenticated database outsourcing. However, this solution computes a signature for each tuple: a hash $H$ is signed where $H$ is the hash of the concatenation of the hash of the tuple and hash of each its immediate preceding tuple in each searchable dimension. Clearly this scheme also leaks: consider the case when a querier has access to only on tuple, but it receives auxiliary information and learns about other tuples. Even though each of these techniques provides authenticity, yet they leak. Moreover, none of these works address the problem of authentication of tree structures.

The PADS framework by Thompson et al. [128] uses Pedersen’s homomorphic commitment scheme [116], Shamir’s $k$-out-of-$n$ secret sharing scheme [123] and Merkle hash technique in order to prevent leakage of micro-data (value of an attribute) while (1) carrying out aggregate query evaluation at multiple third-party servers (in a privacy-preserving manner) and (2) supporting authentication of query results by the client. However, the PADS scheme leaks information about the size of the table and about the existence and positions of other attributes in the table that are not part of the query evaluation. Pang et al. [112] propose completeness verification and authentication of data without revealing micro-data, however, they still leak structural information.

Chatvichienchai and Iwaihara [43] proposed mechanisms for secure updates of XML, without leading to information leakages. However such mechanism does not address the problem of information leakages during verification of integrity of partial XML documents. Wang et al. [134] treat structure and content as first-class protection units. However they focus on a sharing model, in which the receiver of the data has access to only the content (nodes) and not to the structural relationship between them. The paper proposed a scheme for securing structural information in XML databases: how to process queries on an encrypted XML database such that individual element content and structural relations are kept confidential if the security constraint
specified requires so. In our case, we allow the receiver to have access to both nodes and the structural relationships between them.

Graphs are commonly used to model relationships between data items such as documents such as text, XML, richtext, images [58], geographic information [118], healthcare and biological data [57]. Querying, management [44, 57, 129] and mining [59, 122] of graph-structured data as well as privacy-preserving graph publishing and mining techniques [95, 139] have recently emerged as important topics in both academia and industry.

2.13 Secure Publish/Subscribe

The main research efforts related with our work are in the area of secure dissemination of XML data and in the area of secure publish-subscribe systems.

In the first area, the only approach supporting access control in both pull and push based distribution of data has been proposed by Bertino and Ferrari [21]. Such an approach relies on encrypting different portions of the data with different keys and then distributing the keys to data consumers according to the access control policies. Bertino et al. [20] have also investigated the problem of integrity of XML data by using the notion of Merkle hash. Those approaches have however some major drawbacks in that they are not scalable and do not remove extraneous data from contents. These drawbacks are fully addressed by the approach proposed in this paper.

In the second area, several approaches have been proposed to address efficiency issues concerning pub/sub systems ([37–39, 47–49, 52, 110]). Several highly efficient publish/subscribe systems (e.g., IBM Gryphon [127], JMS-based systems) have been developed. Multicast-based content dissemination [11, 109] techniques have been developed in order to support efficient dissemination of contents to subscribers. Most approaches (e.g., [38, 48, 49]) use a spanning tree structure for event routing. In order to reduce the matching that has to be performed by brokers from the root to the leaves, several optimization techniques have been proposed. Virtual groups are
used to reduce the matching performed by brokers [138]. However, security issues in content-based pub/sub systems have not been investigated. The only exceptions are the approaches by Srivatsa and Liu [125], that focuses only on resiliency, and by Opyrchal and Prakash [110], which is very inefficient and is not flexible with respect to access control policies. By contrast our approach addresses a larger spectrum of security requirements while being at the same time efficient and scalable.
3 STRUCTURAL LEAKAGES AND PRIVACY

Not all bits have equal values.
Carl Sagan [120]

Structural information is leaked when existing data authentication schemes such as Merkle hash technique is used. Most existing works are not designed to protect such information with the assumption that structural information does not lead to any privacy/confidentiality breaches. In this chapter, we have demonstrated the relationship between privacy and structural leakages in the healthcare context. We have also described the possible inference attacks on Merkle Hash Technique, which is widely used for authentication of tree-structured, and graph-structured data.

3.1 Inference Attacks on Merkle Hash Technique

In the MHT, a tree is signed by signing the Merkle hash (MH) of its root, which is computed as described below. Common notations are defined in Table 1. In what follows, $\| \|$ denotes concatenation.

The MHT works bottom-up. For a node $x$ in tree $T(V,E)$, it computes a MH $mh(x)$ as follows: if $x$ is a leaf node, then $mh(x) \leftarrow H(c_x)$; else $mh(x) \leftarrow H(mh(y_1) \| \ldots \| mh(y_m))$, where $y_1, \ldots, y_m$ are the $m$ children of $x$ in $T$ in that order from left to right. For example, the MH of the tree in Figure 3.1(a) is computed as follows. The MHes of $e$ and $f$ are: $H(c_e)$ and $H(c_f)$, respectively, which are then used to compute the MH of $d$ as $mh(d) \leftarrow H(mh(e) \| mh(f))$. The MH of $b$ is $H(mh(d))$. Similarly, the MHes of $c$ and $a$ are computed as $H(c_c)$ and $H(mh(b) \| mh(c))$, respectively.

By using such a technique, only the contents of the leaf nodes can be authenticated. In case, the non-leaf (root/intermediate) nodes have contents, the computation of the
Figure 3.1. An example tree with each node having some content.

MH of a non-leaf node $x$ involves the MH of all of its children, and either (a) the content of $x$, or (b) hash of the content of $x$ (used for Figure 3.1(a)).

3.1.1 Leakages during Authentication using MHT

Let $T_{\delta}$ be a subtree of tree $T$ to be shared with Bob. Bob needs the following auxiliary information for authentication of $T_{\delta}$. Such auxiliary information constitute the information leakages by MHT (Figure 3.1). The user then computes the MH of the whole tree using such information (the subtree and auxiliary information) and verifies the signature of the original tree using this MH.

1. Let $x$ be a node in $T_{\delta}$. The MH of each sibling of $x$ that is in $T$ but not in $T_{\delta}$.
   (e.g., $mh(f)$ w.r.t. $e$.)

2. The MH of each sibling $y$ of each ancestor of $x$, such that $y$ is not in $T_{\delta}$. (E.g., $mh(c)$ and $mh(f)$ w.r.t. $e$.)

3. The hash of the content of each ancestor of $x$. (e.g., $\mathcal{H}(c_a)$ and $\mathcal{H}(c_b)$ with respect to $x$.)

4. (i) The structural order between $x$ and its those sibling(s) that are not in $T_{\delta}$, and (ii) the structural order between those siblings of $x$ that are not in $T_{\delta}$. (e.g., (i) the order between $e$ and $f$, and (ii) the order between $b$ and $c$.)
5. (i) The parent-child/ancestor-descendant relationship(s) between a node in $T_δ$ and another node not in $T_δ$, and (ii) those between the nodes that are not in $T_δ$. (e.g., the relationships (i) between $b$ and $d$, (ii) between $a$ and $b$, and between $a$ and $c$).

6. The fact that a given node is the root of the tree $T$ (even if it is the root of $T_δ$).

3.1.2 Inference Attacks

The attacks on the MHT are based on the set of auxiliary information sent to the user. By exploiting these leakages, the following inference attacks can be carried out on MHT.

- **Dictionary attack-1**: It exploits the information (1), (2), (3), and (6). By comparing the MH of a node $e$ in the shared subtree with the MH of another node $f$ received as part of the auxiliary information, the user can infer whether contents of $e$ is same as that of $f$ and if the subtree with root $e$ is identical to the subtree with root $f$. With the auxiliary information (6), the user can also infer (a) whether the received subtree is in fact the original tree, and (b) whether the root of received subtree is in fact the root of the tree.

- **Dictionary attack-2**: It exploits the information (1), (2), (3), and (6). By comparing the Merkle hashes of two nodes ($c$ and $f$) that are received as part of the auxiliary information, the user can infer whether the contents of $c$ is same as that of $f$ and whether the subtree with root $c$ is identical to the subtree with root $f$. With the auxiliary information (6), the user can also infer whether the received subtree is in fact the original tree.

- **Structural inference attack**: It exploits the information (1), (2), (3), and (5). The user infers the number of nodes that are not in the received subtree, and the structure of the original tree from the auxiliary information and shared subtree. If the user receives non-empty auxiliary information, then it infers
that the shared subtree is a proper subtree of the original tree and there are
nodes it has not received. In some cases, the user can also infer the exact size
of the original tree (such as in the case of our example in Figure 3.1).

- **Missing-siblings inference attack**: It exploits the information (1) and (2). From
  the auxiliary information, the user infers the number of siblings of a received
  node $e$ that are not in the shared subtree. If the shared subtree has $x$ and $y$ as
  siblings, the user also infers the number of siblings that are not in the shared
  subtree but are to the right of $x$ and to the left of $y$ in the original tree.

- **Structural-order inference attack**: It exploits the information (4) and (5). The
  user infers structural order between siblings involving one or more nodes that
  the user does not have access to. In our example, the user learns that $b$ and $e$
  are left siblings of $c$ and $f$, respectively.

- **Parent-child inference attack**: It exploits the information (4) and (5). The user
  infers the parent-child relationships involving one or more nodes that the user
  does not have access to. In our example, the user learns that $b$ and $d$ are the
  parents of $e$ and $f$, respectively. The user also learns that $b$ is an ancestor of $e$
  and $f$.

The MHT requires certain auxiliary information about the contents and/or the
structure of the signed tree in order to authenticate a subtree. This inherent require-
ment leads to leakages and cannot be completely prevented by the use of salting, or
of one-way accumulators. In the next section, we give examples in order to illustrate
the significance of such leakages and related inference attacks.

3.2 Structural Leakages in Tree-structured Data

Our running example is in the area of XML data management. XML organizes
data according to the tree structure; integrity and confidentiality of XML data is
an important requirement, given the widespread adoption of XML for distributed
<HealthRecord>
  ...
  <PatientID id=2345S> ... </PatientID>
  <Contact> ... </Contact>
  <CriticalDiseases>
    <Disease name=Cancer>
      <Surgery> ... </Surgery>
      <Chemotherapy>
        <Treatment instance=1>
          <Doctor name=Dr. S. Stevens/>
          <DateTime date=...time=.../>
        </Treatment>
        <Treatment instance=2>
          <Doctor name=Dr. M. Paul/>
          <DateTime date=...time=.../>
        </Treatment>
        </Chemotherapy>
        <Medication> ... </Medication>
    </Disease>
    <Disease name=KidneyFailure> ... </Disease>
  </CriticalDiseases>
  ...
</HealthRecord>

Figure 3.2. XML-based Health-care record of a patient.

web-based applications. As such, XML is an important application domain for the techniques presented in the paper.
The XML document in Figure 3.2 is a fictitious health-care record of a patient and thus contains sensitive information. Assume that such record is stored in a hospital database and that its schema, referred to as HealthRecord, is defined as follows. The HealthRecord element, that is, the root of the tree has a child for each of the following elements: CriticalDiseases, PatientID, and Contact. The CriticalDiseases element lists all the critical diseases a patient suffers from; information about a specific critical disease is specified as its child by the element Disease. Each Disease element lists the types of treatment that the patient has gone through for that same disease. For Cancer, the types of treatment are specified by the following elements: Surgery, Chemotherapy, and Medication. Each type of administered treatment is specified as a child node of the node specific to the treatment type and is an instance of Treatment element. It contains an attribute instance, which refers to the specific instance of the treatment, and child elements to specify the date and time of administering (DateTime), and the name of the doctor who administered the treatment (Doctor).
Table 3.1
Leakage during authentication of $T_{δ_2}$ using MHT

<table>
<thead>
<tr>
<th>Node $x$</th>
<th>Nodes whose $MH$ used in this order to compute $MH(x)$</th>
<th>Distinct leakages during computation of $MH(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{13}$</td>
<td>$a_{13}$</td>
<td>none</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$a_{12}$</td>
<td>none</td>
</tr>
<tr>
<td>$a_{10}$</td>
<td>$a_{12}, a_{13}, a_{10}$</td>
<td>none</td>
</tr>
<tr>
<td>$a_8$</td>
<td>$a_{10}, a_{11}, a_8$</td>
<td>hash of $a_{11}$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_{11}$ is sibling of $a_{10}$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_{11}$ is child of $a_8$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_{11}$ is to the right of $a_{10}$</td>
</tr>
<tr>
<td>$a_5$</td>
<td>$a_7, a_8, a_9, a_5$</td>
<td>$a_7$-specific leakage:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hash of $a_7$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_7$ is sibling of $a_8$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_7$ is child of $a_5$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_7$ is to the left of $a_8$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_9$-specific leakage:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hash of $a_9$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_9$ is sibling of $a_8$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_9$ is child of $a_5$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_9$ is to the right of $a_8$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>$a_5, a_6, a_4$</td>
<td>hash of $a_6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_6$ is sibling of $a_5$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_6$ is child of $a_4$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a_6$ is to the right of $a_5$</td>
</tr>
</tbody>
</table>

A patient may have received treatments from different doctors, each related to a different instance of the same type of treatment or instance of a different type of
Table 3.2
Inference of sensitive information from leakages (Continued to Table 3.3)

<table>
<thead>
<tr>
<th>Leakage during computation of $MH(x)$</th>
<th>Inference from the leakage in the health-care context</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_8$: hash of $a_{11}$ AND ($a_{11}$ is sibling of $a_{10}$ OR $a_{11}$ is child of $a_8$)</td>
<td>Patient has gone through another Chemotherapy.</td>
</tr>
<tr>
<td>$a_8$: $a_{11}$ as to the right of $a_{10}$</td>
<td>If sibling-order represents certain information such as temporal order, then it can be inferred if the chemotherapy referred to by node $a_{11}$ was administered earlier or later than the one referred to by $a_{10}$.</td>
</tr>
<tr>
<td>$a_5$: hash of $a_{7}$ AND ($a_7$ is sibling of $a_8$ OR $a_7$ is child of $a_5$)</td>
<td>Patient has gone through another type of treatment; also inferred is: it is either Surgery or Medication.</td>
</tr>
<tr>
<td>$a_5$: $a_7$ is to the left of $a_8$</td>
<td>More leakage related to the order such as temporal order.</td>
</tr>
<tr>
<td>$a_5$: hash of $a_9$ AND ($a_9$ is sibling of $a_8$ OR $a_9$ is child of $a_5$)</td>
<td>Patient has gone through another type of treatment; also inferred is: it is either Surgery or Medication.</td>
</tr>
<tr>
<td>$a_5$: $a_9$ is to the right of $a_8$</td>
<td>More leakage related to the order such as temporal order.</td>
</tr>
</tbody>
</table>

For expository purposes, we associate a label with each node in the health record in Figure 3.3; for example, the node Chemotherapy is labeled by $a_8$. 
Table 3.3
Inference of sensitive information from leakages (Continued from Table 3.2)

<table>
<thead>
<tr>
<th>Leakage during computation of $MH(x)$</th>
<th>Inference from the leakage in the health-care context</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_4$: hash of $a_6$ AND $(a_6$ is sibling of $a_5$ OR $a_6$ is child of $a_4$)</td>
<td>Patient suffers from another critical disease, whose type can be inferred from the hospital specialization.</td>
</tr>
<tr>
<td>$a_4$: $a_6$ is to the right of $a_5$</td>
<td>More leakage related to the order such as temporal order: time of treatment of this disease in this hospital relative to the time of treatment of Cancer.</td>
</tr>
</tbody>
</table>

The hospital database, which can be accessed remotely, stores all such patient health records. The Merkle hash technique is used to sign the tree and support authentication of the data by users. Table 3.1 lists the details about how to compute and verify the Merkle hash for each node in the tree in Figure 3.3. The third column of such table also lists the information which is leaked when the integrity of a node is verified. Table 3.2 lists the inference attacks that can be carried out due to the leakages.

Consider the following scenario. A cashier has access to the subtree $T_{51}$, shown in Figure 3.3, including the root $a_1$, that is essential for financial and administrative purposes. She does not have access to $a_4$ and its content that refers to Critical Diseases. An access to the health-record in Figure 3.2 leads to the integrity verification of this portion of the health record at the side of the cashier. During this process, the cashier receives the Merkle hash of a node $a_4$ and she also receives the following information: $a_4$ is a child of $a_1$ and is on the left side of $a_2$ and $a_3$. By knowledge of
the schema, the cashier determines that a node at such position must be the *CriticalDiseases* node. Thus the cashier infers that the patient is definitely suffering from some critical disease. If the hospital specializes in some specific critical disease(s), the cashier can further infer which (possible) disease the patient is suffering from. Each of these inferences leads to disclosure of information that is sensitive for the patient.

We now consider another scenario, which leads to leakage of more detailed sensitive information. A nurse has access to $T_{\delta_2}$ and $a_1$ from the record in Figure 3.2. He has access to $T_{\delta_2}$, because he works with the doctor *S. Stevenson*, who prescribed the administering of this treatment. The nurse receives $T_{\delta_2}$ and $a_1$, and the corresponding hashes from the remote database. In order to be able to verify the integrity of $T_{\delta_2}$, he also receives the hashes of $a_6$, $a_7$, $a_9$ and $a_{11}$.

The schema of *HealthRecord* specifies that a child of *CriticalDiseases* refers to a critical disease from which a patient suffers from. By receiving the hash for $a_6$, which is a child of the node $a_4$ (element *CriticalDiseases*), the nurse infers that the patient is suffering from another critical disease different from cancer. This is a disclosure of private information, to which the nurse does not have access to. Assume that the hospital of our example specializes on the treatment of only a limited number of critical diseases. It is thus easy to infer what the other disease is. Furthermore, by inferring that $a_8$ has two siblings, that is, $a_7$ and $a_9$, the nurse is able to infer that the patient has gone through two other treatments other than chemotherapy. Such inference may easily lead to determine the seriousness of the illness. In addition, from the schema, the nurse can infer that these nodes refer to *Surgery* and *Medication*, which reveals that the patient has been received either or both of these treatments. If the hospital has two doctors who specialize in *Surgery* or *Medication*, then the knowledge that the patient has been treated with *Surgery* or *Medication* leads to more information, such as that he has been treated by more than one doctors and who (possibly) has been his doctor.

Furthermore, by the disclosure of the hash of node $a_{11}$ and its structural relationship with $a_8$ as its child, and by knowing that children of a *Chemotherapy* element refer
to the treatment instances, the nurse is sure that the patient went through another Chemotherapy treatment and possibly with another doctor. Additional knowledge about doctors and the hospital could lead to more leakage.

3.3 Structural Leakages in Graph-structured Data

The “IDREF” attribute or a user-defined attribute in XML is used to refer to another element (node) in an XML document, which leads to a non-tree edge (either a back-edge, cross-edge, or a forward-edge). The data object in Figure 3.4 is a graph-based representation of XML-based health-care record of a patient, which typically contains sensitive information. The XML schema elements are described in Table 3.4. The hospital database, which can be accessed remotely, stores all such patient health records.
<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Semantics</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>HealthRecord</td>
<td>Root of the XML document</td>
</tr>
<tr>
<td>Symptoms</td>
<td>Lists the symptoms of a patient</td>
</tr>
<tr>
<td>CriticalDiseases</td>
<td>Critical diseases the patient suffers/suffered from.</td>
</tr>
<tr>
<td>Symptom</td>
<td>Specifies a symptom. Attributes. <em>type</em> refers to the type a symptom. <em>idref</em> refers to the related disease.</td>
</tr>
<tr>
<td>Disease</td>
<td>Information about a specific critical disease; Attributes: <em>idref</em> refers to PostRecoveryProblems.</td>
</tr>
<tr>
<td>Treatment</td>
<td>Type of treatment administered on the patient. Attributes: <em>name</em> specifies the treatment. <em>idref</em> refers to the symptom that a medication may affect.</td>
</tr>
<tr>
<td>Doctor</td>
<td>Name of the doctor who administered the treatment.</td>
</tr>
<tr>
<td>PostRecoveryProblems</td>
<td>Post-recovery problems that a patient goes through. The node is created only once for the first recovery. For any later disease and recoveries from it, this node is referred to from that Disease node.</td>
</tr>
<tr>
<td>Problem</td>
<td>A post-recovery problem. Attribute: <em>type</em> specifies the type of the problem.</td>
</tr>
<tr>
<td>PatientID</td>
<td>Identifier of the patient. Attributes: <em>id</em> and <em>name</em></td>
</tr>
<tr>
<td>ContactDetails</td>
<td>Contact details of the patient.</td>
</tr>
</tbody>
</table>

Consider the following scenario. An insurance manager handling the insurance claim specifically towards the treatment of the disease *KidneyStone* has access to
Table 3.5
Leakage via cross-edge \( e(g_{11}, g_{12}) \), subgraph: \( G_{\delta 1} \).

<table>
<thead>
<tr>
<th>Distinct leakages</th>
<th>Related sensitive information leaked</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-degree of ( g_{12} \geq 2 )</td>
<td>The patient has suffered from at least one more disease.</td>
</tr>
<tr>
<td>There is at least one more path from the node ( CriticalDiseases ) to ( g_{12} )</td>
<td>(1) This disease is a critical disease. (2) The patient has already recovered from this disease.</td>
</tr>
<tr>
<td>The graph is larger than ( G_{\delta 1} )</td>
<td>(1) The patient has been treated by another doctor. (2) She has been associated with this hospital earlier. (3) HealthRecord contains more disease-related data.</td>
</tr>
</tbody>
</table>

subgraph \( G_{\delta 1} \). As a result of querying the database, the manager receives \( G_{\delta 1} \) and a set of authenticity verification items. This portion includes the nodes \( g_1, g_3, g_{11}, g_{12}, g_{14}, g_{15}, g_{16} \) and \( g_{18} \). One of the cross-edges received is \( e(g_{11}, g_{12}) \). By knowing that \( e(g_{11}, g_{12}) \) is a cross-edge, the manager also learns that there is a tree-edge (to which the manager has no access) to the node representing \( PostRecoveryProblems \), which implies that the patient has also suffered from another disease.

By knowledge of the schema, the insurance manager determines that an edge to such a node must be from a Disease node, which is a child of the \( CriticalDiseases \) node. Thus the manager infers that the patient is definitely suffering from some other critical disease. If the hospital specializes in some specific critical disease(s), which is mostly the case in reality, the manager can further infer which (possible) disease the patient is suffering from. Moreover, the manager can definitely infer that
Table 3.6
Leakage via back-edge $e(g_{14}, g_{9})$, subgraph: $G_{δ^2}$.

<table>
<thead>
<tr>
<th>Distinct leakages</th>
<th>Related sensitive information leaked</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) There is at least one cycle in the graph; (b) so there is at least one path from $g_9$ to $g_{14}$</td>
<td>(1) Symptom UrinationPain led to a disease, which is being treated with medication. (2) Since the treatment is medication, with high probability, this disease is not in a serious stage.</td>
</tr>
<tr>
<td>The graph is larger than $G_{δ^2}$</td>
<td>HealthRecord contains more disease-related data.</td>
</tr>
</tbody>
</table>

such a disease was cured (and thus diagnosed) before the KidneyStone disease; this is because the node of PostRecoveryProblems already exists. Each of these inferences leads to disclosure of information that is sensitive for the patient. Leakages inferred from the knowledge that $e(g_{11}, g_{12})$ is a cross-edge are listed in Table 3.5.

We now consider another scenario, which also leads to leakage of some sensitive information by the knowledge that an edge is a back-edge. A pharmacist has access to $G_{δ^2}$ from the record in Figure 3.4. She has access to $G_{δ^2}$, because she provides this medication (specified by node $g_{14}$) to the patient. The pharmacist receives $G_{δ^2}$ and the corresponding verification items from the remote database to verify authenticity. If she somehow learns that the $e(g_{14}, g_{9})$ is a back-edge from the received information, she would learn that there is a path from $g_9$ to $g_{14}$ in the health-record. The schema of HealthRecord specifies that a symptom node is related to a critical disease from which a patient suffers from, that is there is an edge from the symptom node to a disease node. The disease node further has an edge to the Medication node. The only path that can exist from $g_9$ to $g_{14}$ is through a disease node. It is thus apparent
that the patient suffers from a critical disease related to the symptom *UrinationPain*. The symptom name further leads to the possible name of the disease. If the hospital specializes in a limited number of critical diseases, the pharmacist can determine the name of the disease. The treatment type may easily lead to determine the seriousness of the illness (Table 3.6).
In this chapter, we propose the formal model of leakage-free redactable signatures that is general and supports not only trees, but also for graphs and forests; moreover, it supports authentication of disconnected subtrees/subgraphs/sub-forests.

4.1 Preliminaries

Trees and Graphs: A directed (a directed graph $G(V, E)$ is a set of nodes (or vertices) $V$ and a set of edges $E$ between these nodes: $e(x, y)$ is an edge from $x$ to $y$, $(x, y) \in V \times V$. A node $x$ represents an atomic unit of the data, which is always shared as a whole or is not shared at all. A node $x$ is called the ancestor of a node $y$ iff there exists a path consisting of one or more edges from $x$ to $y$. Node $x$ is an immediate ancestor of $y$ in $G$ iff there exists an edge $e(x, y)$ in $E$. Nodes having a common immediate ancestor are called siblings. If there is a specific order between siblings in $G$, then $G$ is an ordered graph. The fact that $x$ precedes one of its siblings $y$ in an ordered graph is denoted by $x \prec y$; i.e., $x$ is to the left of $y$. The contents, if any, of a node $x$ is referred to as $c_x$. A redacted subgraph graph $G(V, E)$ is denoted by $G_\delta(V_\delta, E_\delta)$, and $G_\delta(V_\delta, E_\delta) \subseteq G(V, E)$. $G_\delta(V_\delta, E_\delta) \subseteq G(V, E)$ if and only if $V_\delta \subseteq V$ and $E_\delta \subseteq E$. $G_\delta(V_\delta, E_\delta) \subset G(V, E)$ if and only if $V_\delta \cup E_\delta \subset V \cup E$. Redacted subgraph $G_\delta(V_\delta, E_\delta)$ is derived from the graph $G(V, E)$ by redacting the set of nodes $V \setminus V_\delta$ and the set of edges $E \setminus E_\delta$ from $G$. A directed tree $T(V, E)$ is a directed graph with the following constraint: removal of any edge $e(x, y)$ from $E$ leads to two disconnected trees with no edge or path between nodes $x$ and $y$. Two nodes are siblings, if they
have a common parent; \( x \) is a parent of node \( y \) iff there is an edge \( e(x, y) \) in \( T \); \( y \) is called the child of its parent \( x \). In a tree, a parent node is nothing but the immediate ancestor of all its child nodes. An ordered tree is one where each pair of siblings have an order between them. A rooted tree has a special node called root, for which there is no parent node. As in the case of graphs, a redacted subtree of tree \( T(V, E) \) is denoted by \( T_\delta(V_\delta, E_\delta) \) such that \( T_\delta(V_\delta, E_\delta) \subseteq T(V, E) \). \( T_\delta(V_\delta, E_\delta) \subseteq T(V, E) \) denotes that \( V_\delta \subseteq V \) and \( E_\delta \subseteq E \). Redacted subgraph \( T_\delta(V_\delta, E_\delta) \) is derived from the tree \( T(V, E) \) by redacting the set of nodes \( V \setminus V_\delta \) and the set of edges \( E \setminus E_\delta \) from \( T \). Two trees/graphs/forests with the same nodes and edges, but different ordering between at least one pair of siblings are different trees/graphs/forests. In the rest of the paper, a trees refers to an ordered directed rooted tree, and a graph refers to an ordered directed graph, a subtree/subgraph refers to a redacted subtree/subgraph of a tree/graph unless otherwise stated. \( \Upsilon(V, E) \) refers to either a tree \( T(V, E) \) or a graph \( G(V, E) \).

4.1.1 Signature Schemes

In this section, we review the standard definition of digital signatures (adopted from [76]). A standard digital signature scheme \( \Pi \) is a tuple \((\text{Gen}, \text{Sign}, \text{Vrfy})\).

**Definition 4.1.1 (Standard digital signature scheme)** A digital signature scheme \( \Pi \) consists of two probabilistic polynomial-time algorithms and one deterministic polynomial-time algorithm \( \Pi \equiv (\text{Gen}, \text{Sign}, \text{Vrfy}) \) satisfying the following requirements:

**KEY GENERATION:** The probabilistic key generation algorithm \( \text{Gen} \) takes as input a security parameter \( 1^n \) and outputs a pair of keys \((\text{pk}, \text{sk})\), where \( \text{pk} \) and \( \text{sk} \) are the public and private keys, respectively. We assume for convention that each of these keys has length \( n \) bits, and that \( n \) can be determined from \((\text{pk}, \text{sk})\).

\[(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^n).\]
SIGNING: The probabilistic signing algorithm $\text{Sign}$ takes as input a private key $sk$ and a message $M$, and outputs a signature $\sigma$.

$$\sigma \leftarrow \text{Sign}_{sk}(M)$$

VERIFICATION: The deterministic verification algorithm $\text{Vrfy}$ takes as input a public key $pk$, a message $M$, and a signature $\sigma$, and outputs a bit $b$, with $b = 1$ meaning valid (i.e., $\sigma$ is a valid signature of message $M$), and $b = 0$ meaning invalid (i.e., $\sigma$ is not a valid signature of message $M$).

$$b \leftarrow \text{Vrfy}_{pk}(\sigma, M)$$

4.1.2 Forgery

The strongest form of security for a digital signature scheme $\Pi$ is existentially unforgeable against chosen message attack (EU-CMA). The following experiment $\text{Sig-Forge}^{\text{eu-cma}}_{A,\Pi}(\lambda)$ and definition review the unforgeability property of digital signatures (originally defined in [62]).

Unforgeability: Let us consider the following signature forging experiment. $A$ is a probabilistic polynomial time (PPT) adversary with the knowledge of $pk$.

Signature Forging Experiment: $\text{Sig-Forge}^{\text{eu-cma}}_{A,\Pi}(\lambda)$

1. $(pk, sk) \leftarrow \text{Gen}(1^\lambda)$

2. $A$ may know $pk$ and has oracle accesses to $\text{Sign}_{sk}(\cdot)$ and $\text{rVrfy}_{sk}(\cdot)$. Let $Q$ be the set of messages for which $A$ queries $\text{Sign}_{sk}(\cdot)$.

3. $A$ outputs $(\sigma, M)$, where $M \notin Q$.

4. The output of the experiment is 1 if and only if $\text{Vrfy}_{pk}(\sigma, M)$ outputs 1, else the experiment outputs 0.
Definition 4.1.2  The digital signature scheme \( \Pi \) is \((t,q,\epsilon)\)-secure if the experiment \( \text{Sig-Forge}^{e_{\Pi}}_{\text{eu-cma}}(\lambda) \) outputs 1 with probability \( \epsilon(\lambda) = \frac{1}{2} + \epsilon(\lambda) \), where \( \epsilon(\lambda) \) is a negligible function in terms of \( \lambda \); i.e.,

\[
Pr[\text{Sig-Forge}^{e_{\Pi}}_{\text{eu-cma}}(\lambda) = 1] \leq \epsilon(\lambda) = \frac{1}{2} + \epsilon(\lambda)
\]

4.2  Leakage-Free Redactable Signatures

In this section, we present the formal model of the leakage-free redactable (LFR) signatures for trees, graphs and forests.

4.2.1  Why a New Model?

The standard notion of digital signatures (Section 4.1) has several shortcomings in our context. (1) It is not suitable for cases when a user has access only to a part of the message, and not the full message. (2) It is not suitable for messages that are trees, graphs or forests. Therefore, they also do not define the notion of “leakage-free”. Redactable signatures have been proposed to address the first shortcoming. Such schemes allow signature of (a designated) part of a message to be computed from the signature of the complete message using the knowledge of the public key. Existing models for redactable and sanitizable signatures do not capture all our requirements in a holistic manner, and thus their security definitions are not applicable here as they are.

Brzuska et al [29] have defined formal security definitions of redactable structural signatures for tree data structures. Their definition is restrictive and cannot be applied to our general case. First of all, their scheme is defined only for trees, and can support neither graphs nor forests. Second, their scheme is restrictive for trees as well. They define an operation called \( sCut(\cdot) \), which “cuts” a leaf node from a tree, and can be applied multiple times in order to compute the redacted subtree and its signature. However, by this cut operation, a tree can be reduced to only a connected
subtree, not to a set of disconnected subtrees. The cases, in which a receiver that must receive multiple disconnected subtrees of a tree and verify their authenticity cannot be supported by such a definition. For example, consider the tree in Figure 1.1(a). Consider that a query on tree $T$ evaluates to a subtree $T_δ$ that is verified by the querier for its authenticity. By the definition of Brzuska et al., the $sCut(\cdot)$ operation is applied on leaf $b$, and iteratively on leaf $f$. It then results on the subtree $T_δ$ as shown shaded in the figure. The signature of $T_δ$ is computed from the signature of the $T$ without the knowledge of the secret key, with which $T$ was signed. Now consider another scenario – a query on tree $T$ evaluates to two disconnected subtrees: $T_δ$ and the subtree $g \rightarrow f$. Such a scenario cannot be modeled by this definition. However, such scenarios occur quite frequently in practical usages and databases, and cannot be ignored by a security model. Therefore, there is a need for a completely general security model, which should also support graph-structured data (XML graphs, biological and genetic data, which have private and sensitive information encoded in them).

4.2.2 Definition of a General Scheme

In the following definition, $\Upsilon(V,E)$ refers to a tree, a graph, or a forest (a set of disconnected trees/graphs) with the set of vertices and edges being denoted by $V$ and $E$, respectively.

**Definition 4.2.1 (Leakage-Free Redactable Signature Scheme $rΠ$)** A leakage-free redactable signature scheme $rΠ$ consists of four polynomial algorithms $rΠ \equiv (rGen, rSign, rRedact, rVrfy)$ satisfying the following requirements. Let $\Upsilon(V,E)$ refer to either a tree, graph, or a forest, and let $\Upsilon_δ(V_δ,E_δ) \subseteq \Upsilon(V,E)$, i.e., $\Upsilon_δ(V_δ,E_δ)$ is a redacted subtree/subgraph/sub-forest derived from $\Upsilon(V,E)$.

**KEY GENERATION:** A key generation algorithm $rGen$ takes as input a security parameter $1^\lambda$ and outputs a pair of keys $(pk, sk)$, where $pk$ and $sk$ are the
public and private keys, respectively. We assume for convention that each of these keys has length $\lambda$, and that $\lambda$ can be determined from $pk$ and $sk$.

$$(pk, sk) \leftarrow rGen(1^\lambda).$$

**SIGNING:** The signing algorithm $rSign$ takes as input a private key $sk$ and a tree/graph/forest $\Upsilon(V, E)$, where the content $c_x$ of each node $x \in V$ is such that $c_x \in \{0, 1\}^*$. It outputs a signature $\sigma_{\Upsilon}$.

$$((\sigma_{\Upsilon}, \Upsilon(V, E)) \leftarrow rSign_{sk}(\Upsilon(V, E)).$$

**REDACTION:** The redaction algorithm $rRedact$ takes as input $\Upsilon(V, E)$, its signature $\sigma_{\Upsilon}$, and a set of nodes $V_\delta'$ and edges $E_\delta'$, where $V_\delta' \subseteq V$ and $E_\delta' \subseteq E$. $rRedact$ outputs a redacted signature $\sigma_{\Upsilon_\delta(V_\delta, E_\delta)}$ for $\Upsilon_\delta(V_\delta, E_\delta)$, where $\Upsilon_\delta(V_\delta, E_\delta)$ is a tree/graph/forest derived from $\Upsilon(V, E)$ consisting of vertices in $V_\delta = V \setminus V_\delta'$, and edges $E_\delta = E \setminus E_\delta'$.

$$((\sigma_{\Upsilon_\delta}, \Upsilon_\delta(V_\delta, E_\delta)) \leftarrow rRedact(pk, \sigma_{\Upsilon}, \Upsilon(V, E), V_\delta', E_\delta').$$

**VERIFICATION:** The verification algorithm $rVrfy$ takes as input a public key $pk$, a tree/graph/forest $\Upsilon(V, E)$, whose authenticity needs to be verified, and a signature $\sigma$. It outputs a bit $b$, with $b = 1$ meaning valid (i.e., $\sigma$ is a valid signature of $\Upsilon(V, E)$) and $b = 0$ meaning invalid (i.e., $\sigma$ is not a valid signature of $\Upsilon(V, E)$).

$$b \leftarrow rVrfy_{pk}(\sigma, \Upsilon(V, E))$$

An LFR signature scheme is correct if the following properties hold.

**Signing Correctness:** For any tree/graph/forest $\Upsilon(V, E)$, any positive integer value of $\lambda$, any key pair $(pk, sk) \leftarrow rGen(\lambda)$, and any $(\sigma_{\Upsilon(V, E)}, \Upsilon(V, E)) \leftarrow rSign_{sk}(\Upsilon(V, E))$, $rVrfy_{pk}(\sigma_{\Upsilon(V, E)}, \Upsilon(V, E))$ always outputs 1.
Redaction Correctness: For any tree/graph/forest $\Upsilon(V, E)$, any positive integer value of $\lambda$, any key pair $(pk, sk) \leftarrow r\text{Gen}(\lambda)$, any $(\sigma_{\Upsilon(V, E)}, \Upsilon(V, E)) \leftarrow r\text{Sign}_{sk}(\Upsilon(V, E))$, any subset of vertices $V'_\delta \subseteq V$, any subset of edges $E'_\delta \subseteq E$ such that $V_\delta = V \setminus V'_\delta$, and $E_\delta = E \setminus E'_\delta$, and any $(\sigma_{\Upsilon_\delta}, \Upsilon_\delta(V_\delta, E_\delta)) \leftarrow r\text{Redact}(pk, \sigma_{\Upsilon}, \Upsilon(V, E), V'_\delta)E'_\delta$, $r\text{Vrfy}_{pk}(\sigma_{\Upsilon_\delta}, \Upsilon_\delta(V_\delta, E_\delta))$ always outputs 1.

4.3 Security of Leakage-Free Redactable Signatures

In this section, we define the security requirements of the leakage-free redactable signature scheme $r\Pi$. Informally, these requirements are:

Unforgeability: Someone who does not know the secret key and does not know a valid signature for a tree/graph/forest $\Upsilon$ should not be able to compute a valid signature for $\Upsilon$ provided that (s)he does not know the valid signature for any tree/graph/forest $\Upsilon'$ such that $\Upsilon \subset \Upsilon'$. This is a bit different from the traditional notion of EU-CMA (More on this in Section 4.3.1).

Privacy: Someone who has access to $\Upsilon \subset \Upsilon'$ but not to $\Upsilon'$ should not be able to infer any information about the redacted nodes and edges (in $\Upsilon'$ but not in $\Upsilon$) from the leakage-free redactable signature of $\Upsilon$.

Transparency: Someone who has access to $\Upsilon \subset \Upsilon'$ but not to $\Upsilon'$ should not be able to infer whether the signature of $\Upsilon$ has been computed from scratch or through the process of redaction.

4.3.1 Unforgeability

In standard digital signature schemes, only the signer can compute the signature of a given message. However, a redactable signature scheme allows “anyone” with the public key $pk$ to be able to compute the signature of a (designated) part of the data (In contrast, for sanitizable signatures, a designated sanitizer can only compute
signatures on altered documents). Redactable signatures by Johnson et al. [74] compute the signature of a message $M$, which consists of $n$ parts: $m_1, m_2, m_3, \ldots, m_n$. It allows anyone, who has a knowledge of the public key to compute the signature of a redacted document $M'$ that consists of a subset of $m_i$'s. Due to such a property, redactable signature schemes cannot support the strongest notion of unforgeability, i.e., existentially unforgeable under adaptive chosen-message attack, as it is. It has to be adapted (in other words, weakened) in order incorporate the fact that the signatures of any redacted sub-document $M'$ can be “legally forge” with respect to the operation that can be used to compute these sub-documents from the original document $M$ (cf., [74]). Therefore, the notion of unforgeability proposed by Brzuska et al ( [29]: Definition 3) for redactable structural signatures for trees, is rather too strong to be supported by any redactable signature scheme. Moreover, their definition does not formally define what a “tree attack” means.

In what follows, we make use of the results of Johnson et al. [74] in defining the notion of unforgeability for the leakage-free redactable signature scheme $r\Pi$ defined in the previous section. The operation related to our notion of redaction is the subset $\subset$ operation: a redacted subtree $T_\delta$ is a tree with subsets of nodes and edges of the original tree $T$. Intuitively this notion means that if an adversary $A$ has the knowledge of a subtree/subgraph/sub-forest $\Upsilon_\delta$, its valid signature $\sigma_{\Upsilon_\delta}$, and the public key, then $A$ can in fact compute the valid signature of any subtree/subgraph/sub-forest of $\Upsilon_\delta$, and these are the only “messages” for which $A$ can “forge” a signature without the knowledge of the secret key.

**Existentially unforgeable under adaptive chosen-message attack with respect to the $\subset$ operation** (EU-CMA-$\subset$): Let us consider the following signature forging experiment. As earlier, $A$ is a PPT adversary has access to the signing oracle $r\text{Sign}_{sk}(\cdot)$ with the knowledge of $pk$. $A$ should not be able to forge a valid signature $\sigma$ for a “new” tree/graph/forest even after it has adaptively queried the signing oracle $r\text{Sign}_{sk}(\cdot)$. A “new” tree/graph/forest denotes a tree/graph/forest $\Upsilon'$, for which the $A$ has not received a valid signature from the signing oracle and $\Upsilon'$ is not a redacted
subtree/subgraph/sub-forest of another $\Upsilon$ for which $\mathcal{A}$ has received a valid signature from the signing oracle.

**Signature Forging Experiment: Sig-Forge$^{eu-cma-c}_{\mathcal{A}, r\Pi}(\lambda)$**

1. $(pk, sk) \leftarrow \text{Gen}(1^{\lambda})$

2. $\mathcal{A}$ is given the $pk$, and is allowed oracle accesses to $r\text{Sign}_{sk}(\cdot)$, $r\text{Redact}_{pk}(\cdot, \cdot)$, and $r\text{Vrfy}_{sk}(\cdot)$. Let $Q$ be the set of all trees/graphs/forests for which $\mathcal{A}$ queries $r\text{Sign}_{sk}(\cdot)$ for their signatures.

3. $\mathcal{A}$ outputs $(\sigma, \Upsilon')$, such that
   
   (a) $\Upsilon' \not\in Q$: $\Upsilon'$ is a tree/graph/forest whose signature the $\mathcal{A}$ has not queried from the signing oracle, and
   
   (b) $\Upsilon' \not\in \bigcup_{\Upsilon \in Q} \Theta_{\Upsilon}$, where $\Theta_{\Upsilon} = \{ \Upsilon'' | \Upsilon'' \subset \Upsilon \}$: $\Upsilon'$ is not a subtree/subgraph/sub-forest of any $\Upsilon(V,E)$, whose signature has been queried by $\mathcal{A}$ from $r\text{Sign}_{sk}(\cdot)$.

4. The output of the experiment is 1 if and only if $\text{Vrfy}_{pk}(\sigma, \Upsilon') = 1$, else the experiment outputs 0.

**Definition 4.3.1 (LFR Signature: Integrity)** The leakage-free redactable signature scheme $r\Pi$ is existentially unforgeable under adaptive chosen-message attack over $\subset$ operation, if the experiment $\text{Sig-Forge}^{eu-cma-c}_{\mathcal{A}, r\Pi}(\lambda)$ outputs 1 with probability $\epsilon(\lambda) = \frac{1}{2} + \epsilon(\lambda)$:

$$Pr(\text{Sig-Forge}^{eu-cma-c}_{\mathcal{A}, r\Pi}(\lambda) = 1) \leq \epsilon(\lambda) = \frac{1}{2} + \epsilon(\lambda).$$

**4.3.2 Leakage-Free Properties**

There are two leakage-free properties: privacy and transparency. Transparency is the stronger property and subsumes privacy.
4.3.3 Privacy

An LFR signature scheme supporting privacy does not lead to leakage of the contents of the nodes that have been redacted. Given a redacted subtree/subgraph/subforest and its LFR signature, one cannot infer any of the contents of the redacted nodes. For example, given a subtree from a financial XML document that lists credit-card information, and given two source XML documents, one cannot identify which source tree the subtree has been redacted from.

In the following adversarial experiment, a PPT adversary \( A \) outputs two trees/forests \( \Upsilon_0 \) and \( \Upsilon_1 \) such that (a) \( A \) has not queried the signatures for them nor has queried signatures for super-trees/super-graphs/super-forests of them; (b) redacting a given set of vertices \( V'_\delta \) and edges \( E'_\delta \) from either of them lead to the same subtree/subgraph/sub-forest \( \Upsilon'_\delta(V'_\delta,E'_\delta) \). The experiment randomly chooses either \( \Upsilon_b \) (\( b = 0 \) or \( 1 \)) to be signed, and then computes the signature for the redacted subtree \( \Upsilon'_\delta \). A challenge consisting of the redacted subtree and its signature is given to the adversary. The experiment is successful, if the adversary can infer the \( \Upsilon_b \) from which \( \Upsilon'_\delta \) has been redacted with a probability non-negligibly larger than \( \frac{1}{2} \).

**Privacy Experiment:** \( \text{Sig-Priv}^{\text{priv}}_{A,r\Pi}(\lambda) \) Let us consider the following experiment on privacy of the LFR signature scheme \( r\Pi \):

1. \( r\text{Gen}(1^\lambda) \) is run to obtain keys \((pk, sk)\).
2. Probabilistic polynomial-time adversary \( A \) is given \( pk \) and oracle access to \( r\text{Sign}_{sk}(\cdot) \), \( r\text{Redact}(\cdot) \) and \( r\text{Vrfy}_{pk}(\cdot) \). Let \( Q \) be the set of all trees/graphs/forests for which \( A \) queries \( r\text{Sign}_{sk}(\cdot) \) for their signatures.
3. \( A \) outputs two trees/forests \( \Upsilon_0(V_0,E_0) \), and \( \Upsilon_1(V_1,E_1) \), and \( \Upsilon_\delta(V_\delta,E_\delta) \), where \( \Upsilon_\delta(V_\delta,E_\delta) \subset \Upsilon_0(V_0,E_0) \) and \( \Upsilon_\delta(V_\delta,E_\delta) \subset \Upsilon_1(V_1,E_1) \).
4. Draw a random \( b \) uniformly from \( \{0,1\} \). \((\sigma_{\Upsilon_b}, \Upsilon_b(V_b,E_b)) \leftarrow r\text{Sign}_{sk}(\Upsilon_b(V_b,E_b)) \).
   The signature for redacted subtree \( \Upsilon_\delta \) is computed from \( \sigma_{\Upsilon_b} \): \((\sigma_{\Upsilon_\delta}, \Upsilon_\delta) \leftarrow r\text{Redact}(pk, \sigma_{\Upsilon_b}, \Upsilon_b(V_b,E_b), V_b \setminus V_\delta, E_b \setminus E_\delta) \). Then the challenge \((\sigma_{\Upsilon_\delta}, \Upsilon_\delta) \) is given to \( A \).
5. The adversary $A$ continues to have oracle access to $r\text{Sign}_{sk}(\cdot)$, $r\text{Redact}_{pk}(\cdot, \cdot)$, and $r\text{Vrfy}_{pk}(\cdot)$. Eventually, $A$ outputs a bit $b'$ (for the challenge $(\sigma_{\Upsilon_0}, \Upsilon_{\delta})$).

6. The output of the experiment is 1 if $b' = b$, and 0 otherwise.

**Definition 4.3.2 (LFR Signature: Privacy)** The LFR signature scheme $r\Pi$ preserves privacy if $\Pr(Sig-Priv_{A,r\Pi}(\lambda) = 1) \leq \frac{1}{2} + \epsilon(\lambda)$.

4.3.4 Transparency

In this section, we formally define a transparent LFR scheme. In many applications, it is expected that the end-user should not be able to infer whether the received subtree/subgraph/sub-forest is a redacted one or not. If such information can be inferred, the adversary can learn that the source tree/graph/forest has a higher number of vertices/edges (which might represent the number of bank accounts one has, or the number of critical diseases one has suffered from). Therefore, the leakage-free redactable signature scheme should prevent inference of such information.

A similar notion exists in the literature for sanitizable signatures in the name of “signature transparency” [74], which however, is different from our notion of signature-indistinguishability in the sense that in our case, the adversary should not be able to infer any extraneous information (either contents or structural) in $\Upsilon$ with respect to $\Upsilon_{\delta}$ (extraneous information is defined in Section 4.2). The existing notion transparency does not capture leakage of structural information.

In the following adversarial experiment, a PPT adversary $A$ outputs two trees/forests $\Upsilon_0$ and $\Upsilon_1$ such that $\Upsilon_0 \subset \Upsilon_1$. The experiment randomly chooses either $\Upsilon_b$ ($b = 0$ or 1) to be signed; if $\Upsilon_1$ is signed, then the signature for $\Upsilon_0$ is computed by redaction from that of $\Upsilon_1$. A challenge consisting of $\Upsilon_0$ and its signature is given to the adversary. The experiment is successful (returns 1) if the adversary can infer whether the signature of $\Upsilon_b$ was computed by the signing oracle or by the redaction oracle, with a probability non-negligibly larger than $\frac{1}{2}$. 
**Transparency Experiment:** Sig-Transp\textsubscript{transp}\textsubscript{rΠ}(\lambda) Let us consider the following experiment on transparency of the LFR signature scheme rΠ:

1. \text{rGen}(1^\lambda) is run to obtain keys (pk, sk).

2. Probabilistic polynomial-time adversary A is given pk and oracle access to \text{rSign}_{sk}(\cdot), \text{rRedact}(\cdot) and \text{rVrfy}_{pk}(\cdot). Let \mathcal{Q} be the set of all trees/graphs/forests for which A queries \text{rSign}_{sk}(\cdot) for their signatures.

3. A outputs two trees/graphs/forests \( \Upsilon_0(V_0, E_0) \), and \( \Upsilon_1(V_1, E_1) \), such that \( \Upsilon_0(V_0, E_0) \subset \Upsilon_1(V_1, E_1) \).

4. Draw a random \( b \) uniformly from \{0,1\}. \( (\sigma_{\Upsilon_b}, \Upsilon_b(V_b, E_b)) \leftarrow \text{rSign}_{sk}(\Upsilon_b(V_b, E_b)) \).
   If \( b=1 \), then the signature for \( \Upsilon_0(V_0, E_0) \) is computed from \( \sigma_{\Upsilon_1} : (\sigma_{\Upsilon_0}, \Upsilon_0(V_0, E_0)) \leftarrow \text{rRedact}_{pk}(\sigma_{\Upsilon_1}, \Upsilon_1(V_1, E_1), V_1 \setminus V_0) E_1 \setminus E_0 \). Then the challenge \( (\sigma_{\Upsilon_0}, \Upsilon_0(V_0, E_0)) \) is given to A.

5. The adversary A continues to have oracle access to \text{rSign}_{sk}(\cdot), \text{rRedact}_{pk}(\cdot, \cdot), and \text{rVrfy}_{pk}(\cdot). Eventually, A outputs a bit \( b' \) (for the challenge \( (\sigma_{\Upsilon_0}, \Upsilon_0) \)).

6. The output of the experiment is 1 if \( b' = b \), and 0 otherwise.

**Definition 4.3.3 (LFR Signature: Transparency)** The LFR signature scheme rΠ preserves transparency, if \( Pr(\text{Sig-Transp}_{A,rΠ}(\lambda) = 1) \leq \frac{1}{2} + \epsilon(\lambda) \).

4.3.5 Relationships of Privacy and Transparency

In this section, we analyze the relationships between the two leakage-free properties of LFR signature schemes: privacy and transparency. In whatever follows, any mention of tree/subtree can be substituted by the term graph/subgraph or forest/subforest without any change to the semantics of the context.
Proposition 4.3.1 \textit{Privacy} \not\Rightarrow \textit{Transparency}.

Consider two trees: $\Sigma_1$: $x$ is the parent of $y$ and $z$, and $y < z; \Sigma_2$: $x$ is the parent of $y$ and $w$, and $y < w$. Consider the case in which a party receives subtree $x$, $y$, and the edge from $x$ and $y$, and its signature. If the signature scheme supports privacy, the signature does not leak contents of $z$ or $w$, but may provide the fact that $y$ has a right sibling. Such a scheme is privacy-preserving, but not transparent, because someone who knows $\Sigma_1$, and the subtree and its signature, can infer that the signature is a redacted signature from $\Sigma_1$.

Proposition 4.3.2 \textit{Transparency} \Rightarrow \textit{Privacy}.

Someone who cannot determine if a signature is a redacted signature, cannot in fact determine if the subtree is a redacted subtree or a source tree by itself. By taking negation of the implication, if an LFR scheme does not support privacy, then one can determine the source tree from whose signature, the signature of a subtree has been derived. Such inference in fact implies that the signature of the subtree has not be computed from scratch, but is a redacted signature, which is why transparency also does not hold for such a scheme. \textit{The transparency property of a redactable signature scheme is indeed its leakage-free property.}

4.4 Secure Names

We would now define a formal notion of secure names, and secure naming schemes. Secure names are assigned with nodes in order to prove/disprove the alleged order between a pair of nodes.

4.4.1 Secure Naming Schemes

Each secure (i.e., leakage-free) naming scheme $rN \equiv (rNameGen, rNameVrfy)$ has two components: generation of secure names and verification of the order between secure names, and is defined as follows.
Definition 4.4.1 (Secure Naming Scheme $rN$) A leakage-free naming scheme $rN$ consists of two polynomial algorithms $rN \equiv (rNameGen, rNameVrfy)$ satisfying the following requirements. Let $V$ refer to a set of vertices such that $V = \{x_i | 1 \leq i \leq |V|\}$, and $x_i \prec x_j$, for all $1 \leq i < j \leq |\Theta|$. Let $V_\delta \subseteq V$, i.e., $V_\delta$ is a redacted set of vertices derived from $V$.

**NAME GENERATION:** A name generation algorithm $rNameGen$ takes as input two or less security parameters $1^{\lambda_1}$ and $1^{\lambda_2}$ and $V$ and outputs a set of $\Theta = \{\eta_i | 1 \leq i \leq |V|\}$ such that $\eta_i$ is a secure name for node $x_i$. We assume for convention that each of these secure names has length $\lambda$, and that $\lambda$ can be determined from $\eta_i$’s. In case of one security parameter, $1^{\lambda_2}$ would just be replaced by 0.

$$\Theta \leftarrow rNameGen(1^{\lambda_1}, 1^{\lambda_2}, V).$$

**ORDER VERIFICATION:** The order verification algorithm $rNameVrfy$ takes as input two or less security parameters $1^{\lambda_1}$ and $1^{\lambda_2}$ and two pairs $(x_i, \eta_i)$ and $(x_j, \eta_j)$, where a pair is: (node $x$, secure name $\eta_x$), and outputs a bit $b$, with $b = 1$ meaning valid (i.e., $x_i \prec x_j$) and $b = 0$ meaning invalid (i.e., $x_i \not\prec x_j$).

$$b \leftarrow rNameVrfy(1^{\lambda_1}, 1^{\lambda_2}, (x_i, \eta_i), (x_j, \eta_j))$$

An LFR signature scheme is correct if the generated secure names are verified to be order-preserving.

**Naming Correctness:** For any set of vertices $V$ such that between a pair of vertices $x, y \in V$, $x \prec y$, any positive integer values of $\lambda_1$, and $\lambda_2$, and any $\Theta \leftarrow rNameGen(1^{\lambda_1}, 1^{\lambda_2}, V)$, $rNameVrfy(1^{\lambda_1}, 1^{\lambda_2}, (x, \eta_x), (y, \eta_y))$ always outputs 1.

4.4.2 Security of Naming Schemes

In this section, we define the security requirement of the secure names. Informally, this requirement is as follows.
Name-Transparency: Someone who has access to an ordered set of secure names \( \Theta \delta \subset \Theta \) but not to \( \Theta \) should not be able to infer whether \( \Theta = \Theta \delta \) or \( \Theta \neq \Theta \delta \).

Let us formally define a transparency notion for secure names. Such a notion is similar to that of transparency (Section 4.3.4).

In the following adversarial experiment, a PPT adversary \( A \) outputs two sets of vertices \( V_0 \) and \( V_1 \) such that \( V_0 \subset V_1 \), and \( x_i \prec x_j \), where \( 1 \leq i < j \leq |V_1| \). The experiment randomly chooses either \( V_b \) (\( b = 0 \) or \( 1 \)) for which secure names are to be generated; if \( V_1 \) is chosen, then the secure names for the vertices in \( V_0 \) are selected from \( V_1 \). A challenge consisting of \( V_0 \) and its set of secure names \( \Theta_0 \) is given to the adversary. The experiment is successful (returns 1) if the adversary can infer whether the secure names in \( \Theta_0 \) of \( \Upsilon_b \) was computed from scratch or was selected from the \( \Theta_1 \) after computing \( \Theta_1 \), with a probability non-negligibly larger than \( \frac{1}{2} \).

Name-Transparency Experiment: Define \( \text{Name-Transp}_{\lambda, \Pi}^{\text{transp}}(\lambda) \) Let us consider the following experiment on name-transparency of the secure names \( \Pi \):

1. Probabilistic polynomial-time adversary \( A \) is given oracle access to \( r\text{NameGen}(\lambda_1, \lambda_2, \cdot) \) and \( r\text{NameVrfy}(\lambda_1, \lambda_2, \cdot) \).

2. \( A \) outputs two sets of vertices \( V_0 \), and \( V_1 \), such that \( V_0 \subset V_1 \).

3. Draw a random \( b \) uniformly from \( \{0, 1\} \). \( \Theta_b \leftarrow r\text{NameGen}(1^{\lambda_1}, 1^{\lambda_2}, V_b) \). If \( b = 1 \), then the set of secure names \( \Theta_0 \) for the vertices in \( V_0 \) are selected from \( \Theta_1 \). Then the challenge \( (\Theta_0, V_0) \) is given to \( A \).

4. The adversary \( A \) continues to have oracle access to \( r\text{NameGen}(\lambda_1, \lambda_2, \cdot) \) and \( r\text{NameVrfy}(\lambda_1, \lambda_2, \cdot) \). Eventually, \( A \) outputs a bit \( b' \) (for the challenge \( (\Theta_0, V_0) \)).

5. The output of the experiment is 1 if \( b' = b \), and 0 otherwise.

**Definition 4.4.2 (Secure Naming Scheme: Name-Transparency)** The secure naming scheme \( \Pi \) preserves transparency, if \( \Pr(\text{Name-Transp}_{\lambda, \Pi}^{\text{transp}}(\lambda) = 1) \leq 1/2 + \max(\epsilon(\lambda_1, \epsilon(\lambda_2)), \text{where } \max(i, j) \text{ returns the larger of } i \text{ and } j. \)
4.5 Summary

Redactable signatures for linear-structured data such as strings have been studied. Redactable signatures for non-linear-structured data such as trees and graphs have also been recently developed with a stronger security requirement - that is - such signatures should be completely leakage-free. However, it is restrictive and is flawed. We proposed the first fully general formal model for leakage-free redactable signatures for trees, graphs and forests. We also defined a notion of security of secure names for ordered vertices.
5 STRUCTURAL SIGNATURES FOR TREES

Whether you can observe a thing or not depends on the theory which you use. It is the theory which decides what can be observed.

Albert Einstein (during Heisenberg’s 1926 lecture at Berlin)

In this chapter, we present a solution to our problem of leakage-free authentication of trees, for scenarios when a user receives “only one subtree”. This scheme is based on the notion of randomized traversal numbers and post-order/pre-order/in-order traversals of trees, and aggregate signature schemes. We then provide security analysis, and complexity as well as performance analysis of this scheme.

For scenarios, in which a receiver receives more than one subtrees as a result of one or more queries, this scheme leaks path-related information between each pair of subtrees.

5.1 Review of Tree Traversals

Post-order, pre-order, and in-order tree traversals are defined in [77]. While post-order and pre-order traversals are defined for all types of trees, in-order traversal is defined only for binary trees. Binary trees are the trees in which each non-leaf node has at most two children, and they are left and right children. A node can have a right child without having a left child and vice versa. In each of these traversals, the first node visited is assigned 1 as its visit count. For every subsequent vertex visited, the visit count is incremented by 1 and is assigned to the vertex. This sequence of numbers is called the sequence of post-order (PON), pre-order (RON), or in-order (ION) numbers for the tree $T$, depending on the particular type of traversal.

Properties of traversal numbers: The post-order number of a node is smaller than that of its parent. The pre-order number of a node is greater than that of its parent.
The in-order number of a node in a binary tree is greater than that of its left child and smaller than that of its right child. A specific traversal number of a node $x$ is always smaller than that of its right sibling $y$. The distribution and range of the traversal numbers are uniform and deterministic ($[1, 2, \ldots, |V|]$). The determinism of the distribution and range of the traversal numbers make them unsuitable for our purposes as they reveal information about the approximate size of the data and the position of the subset of data in the data set. It is possible for an adversary to exploit this information and replace a signed node with a compromised or a different node altogether by assigning to it the original pre-order number. Siblings can be interchanged and the corresponding visit counts could also be interchanged while satisfying the specific properties. A non-binary tree can be uniquely reconstructed from the sequences of pre-order and post-order numbers of its nodes. A binary tree can be uniquely reconstructed from the sequences of in-order and either pre-order or post-order numbers of its nodes.

In order to overcome the above limitations of the traversal numbers, we propose the notion of *randomized traversal numbers*.

### 5.2 Randomized Traversal Numbers

We transform a set of traversal numbers (of a specific traversal) into a set of distinct random numbers such that the order between the traversal numbers is preserved. By preserving the order of their original counterparts, these randomized traversal numbers (RTNs) preserve their properties. The distribution and range of randomized traversal numbers is non-uniform and non-deterministic. The randomized post-order, pre-order and in-order numbers are called randomized post-order (RPON), randomized pre-order (RRON), and randomized in-order (RION) numbers. RPON, RRON, and RION for a node $x$ are denoted by $p_x$, $r_x$, and $i_x$, respectively.

Before we define the randomized traversal numbers formally, we would need to introduce the security notion of POPF-CCA introduced by Boldyreva et al. [26] in
order to develop secure order-preserving encryption schemes. Boldyreva et al observe that the standard indistinguishability notion of IND-CCA (indistinguishable against chosen-ciphertext attack) cannot be supported by any order-preserving encryption (OPE), because any OPE scheme (1) has to be deterministic in order to support determination of equality among plaintexts, (2) leaks equality, and (3) order among plaintexts. Our notion of randomized traversal numbers in fact can be defined to be the ciphertexts of such OPE scheme that is secure under the new notion of indistinguishability as defined in [26]: pseudo-random order-preserving function against chosen-ciphertext attack (POPF-CCA). We should note that POPF-CCA is weaker than the strongest notion of indistinguishability - indistinguishable against chosen-ciphertext attack (IND-CCA). We refer the reader to [26] for detailed description on this notion.

**Definition 5.2.1** Let $T(V, E)$ be a tree. Let $V_δ$ be a subset of $V$. Let $o_x$ be the traversal number of a node $x \in V$, $O_V = \{o_x | x \in V\}$. Let $O'_V = \{o'_x | x \in V\}$, where $o'_x$ is the randomized traversal number associated with $o_x$ for node $x$ if and only if the following hold true. Let $\odot$ denote either $<,=,>$.  

1. $o_x \odot o_y \iff o'_x \odot o'_y$.

2. Let $Enc_K(\cdot)$ be a POPF-CCA secure order-preserving encryption, which is used as follows: $O'_V \leftarrow Enc_K(O_V)$.

From the definition, it is clear that given the knowledge of any subset $O'_{V_δ}$ of randomized traversal numbers $O'_V$ in a tree $T(V, E)$, a PPT adversary $A$ has a negligible advantage (as defined in [26]) in determining whether $V \setminus V_δ$ is empty or non-empty under the POPF-CCA notion of security. In other words, $Enc_K(\cdot)$ is transparent (Section 4.3.4).

The following lemmas provide the basis for developing the structural signatures for trees.
Lemma 5.2.1 The pair of randomized in-order number and either post-order or pre-order number for a node in a binary tree correctly and uniquely determines the position of the node in the structure of the tree, where the position of a node is defined by its parent and its status as the left or right child of that parent.

Proof From the in-order and either post-order or pre-order traversal sequences of the vertices, it is possible to uniquely re-construct a binary tree [75]. Thus from these sequences or from their randomized counterparts, for a node, it is possible to correctly identify its parent and its status as left or right child of that parent in the tree. Thus the lemma is proven. Follows from [75].

Lemma 5.2.2 The pair of randomized post-order number and pre-order number for a node in a (non-binary) tree uniquely determines the position of the node in the structure of the tree, where the position of a node is defined by its parent and its siblings to its immediate left and right in the tree.

Proof Follows from [51].

5.2.1 Computation of randomized traversal numbers

In order to compute the randomized traversal numbers, we refer to the secure scheme presented in the literature.

1. By using Boldyreva et al’s order-preserving encryption: Boldyreva et al. [26] recently developed a symmetric order-preserving encryption (OPE) scheme that is secure with respect to a security notion - POPF-CCA This scheme is based on hypergeometric (HG) and negative hypergeometric (NHG) probability distributions, and their relation with order-preserving encryption. For our purpose, let \( Enc_K([i,j]) \) refer to the randomized order-preserving encryption of the list of integers \([i,j]\) using OPE scheme and the key \( K \). For our purpose, we just
need to carry out verification of the order among them, there is no decryption of the encrypted integers. These encrypted numbers can be used as the randomized traversal numbers. However, order-preserving encryption is more expensive than (1) and (2) as it provides a trapdoor to compute the plaintext from the ciphertext. We do not need such a property here. We should note that Boldyreva et al. have not characterized the leakage and its amount due to their OPE scheme ([26]: page: 5). However, this is the state-of-the-art scheme. Existing order-preserving hash functions are not related to security in any sense ([26]: page: 5).

2. Not provably secure: One scheme is to draw as many randoms as the number of the nodes, and sort them. The sorted (increasing-order) random numbers are assigned as the randomized post-order numbers. This process is repeated for randomized pre-order numbers.

We mention of the second scheme because we believe that it provides security for trees with small arity such as 2 (however, we note that we do not have a proof of security in terms of transparency as of now).

5.3 Structural Signatures

In this section, we develop the protocol of structural signature scheme for trees. Structural signatures for non-binary trees are based on post-order and pre-order numbers, and those of binary trees are defined identically to that of non-binary trees except that in-order traversals are used in place of either the post-order and pre-order traversals (Lemma 5.2.1). For simplicity of exposition, we focus primarily on non-binary trees. The structural signature scheme is referred to as $r\Pi \equiv (r\text{Gen}, r\text{Sign}, r\text{Redact}, r\text{Vrfy})$, and is presented in the following sections.
5.3.1 Signing (rSign)

**Structural Position:** A structural position uniquely identifies a node in a tree structure. It is defined as a pair of the RPON and the RRON of a node and for binary trees it is defined as a pair of the RION and RPON (or RRON) of the node (Definition 5.3.1). An integrity verifier (IV) is different from an authenticity verifier (VO), in the sense that the latter is defined from IVs, and can be used to verify authenticity of a subtree (as described later). A structural position is essential a secure-name of a node.

**Definition 5.3.1 (Structural Position and Integrity Verifier)** Let $x$ be a node in tree $T(V,E)$. Its structural position, denoted by $\eta_x$, is defined as a pair of its RPON $p_x$ and RRON $r_x$, that is, $\eta_x = (p_x, r_x)$. Let $c_x$ denote the content of node $x$. Its structural integrity verifier (IV) denoted by $\xi_x$, is defined as: $\xi_x = H(\eta_x \parallel c_x)$.

Using the IVs, we define the structural signature $\sigma_{T(V,E)}$ of $T(V,E)$. Such signatures are leakage-free. In cases when “the received subtree (sent to the user) is the same as the original tree” is a sensitive information, the signature of a tree may be salted using a random value in order to protect this fact. The (salted) tree signature is publicly available or passed to the user along with the subtree that the user has access to. $\sigma_{T(V,E)}$ is an aggregate signature, computed over the IVs of its nodes. We define two types of signatures for trees: one based on the condensed-RSA signatures [105] and the other based on bilinear maps [28].

**Definition 5.3.2 (Structural Signature using CRSA)** Let $T(V,E)$ be a tree. Let $\mathcal{H}$ denote a random oracle. Let the RSA signature $\sigma_x$ of each node $x$ be defined as follows $\sigma_x \leftarrow \xi_x^d \mod \bar{n}$, where $\xi_x$ is the IV of $x$. Let $\omega_T$ be a random. The leakage-free signature of $T$, denoted by $\sigma_{T(V,E)}$, is defined as

$$\sigma_{T(V,E)} = (\omega_T \prod_{x \in V} \xi_x)^d \mod \bar{n}. \quad (5.1)$$
**rSign**: Sign tree $T(V, E)$.

1. Compute the post-order and pre-order numbers for each node in $T$.

2. For each node $x$ in $T$: transform the post-order and pre-numbers into post-order and pre-order numbers denoted, respectively, as $p_x$ and $r_x$.

3. For each node $x$,
   
   (a) Assign $(p_x, r_x)$ to $x$ as its structural position $\eta_x$.

   (b) Compute the IV $\xi_x$ of $x$: $\xi_x \leftarrow \mathcal{H}(\eta_x \Vert c_x)$.

4. Compute the leakage-free signature $\sigma_{T(V, E)}$ using either Condensed-RSA (Eq.5.1) or BGLS (Eq.5.2).

**Figure 5.1. Algorithm to sign a tree.**

**Definition 5.3.3 (Structural Signature using BGLS)** Let $T(V, E)$ be a tree. Let $\mathcal{H}$ denote a random oracle. Let the RSA signature $\sigma_x$ of each node $x$ be defined as follows $\sigma_x \leftarrow \text{sk}\xi_x$, where $\xi_x$ is the IV of $x$. Let $\omega_T$ be a random. The leakage-free signature of $T$, denoted by $\sigma_{T(V, E)}$, is defined as

$$\sigma_{T(V, E)} \leftarrow (P, \text{sk}(\omega_T + \sum_{x \in V} \xi_x)).$$

(5.2)

The signing algorithm for a tree $T(V, E)$ is given below in Figure 5.1.

5.3.2 Distribution of a Subtree (**rRedact**)

Suppose that Bob receives $T_\delta(V_\delta, E_\delta)$, a subtree of $T(V, E)$. $T_\delta(V_\delta, E_\delta)$ can be shared with Bob according to two different strategies: (1) by sharing the signed subtree - its nodes and the structure; (2) by sharing the signed nodes in the subtree.
and letting Bob reconstruct the subtree using the RPON’s and RRON’s of the nodes. We describe both options in the following sections. By use of either CRSA or BGLS, the distributor $D$ needs to only send $O(1)$ authenticity verifiers to the user.

5.3.3 Distribution of a Subtree along with its Structure

Let $T_{\delta}(V_{\delta}, E_{\delta})$ be the subtree of $T(V, E)$ that is to be shared with user Bob by the distributor. The distributor $D$ sends the information about the parent-child relationships and ordering between siblings (e.g. as an adjacency matrix) in $T_{\delta}(V_{\delta}, E_{\delta})$ and $\sigma'_{T_{\delta}}$ to Bob: $\sigma'_{T_{\delta}} = (\sigma_{T(V,E)}, \sigma_{T_{\delta}}, VO, \{\eta_x, \sigma_x | x \in V_{\delta}\})$. The redaction procedure is given in Figure 5.3.3.

The purpose of sending $\sigma'_{T_{\delta}}$ to each user is to prevent injection of spurious nodes to the subtree either at the distributor (cloud server), and/or in the communication channel. The purpose of sending the verification object $VO$ is to prevent an attacker from deleting nodes from the subtree.

5.3.4 Authentication ($rVrfy$)

Figure 5.2 summarizes the interactions between Alice, $D$ and Bob. Bob receives the subtree $T_{\delta}(V_{\delta}, E_{\delta})$ (with a different name in order avoid any ambiguity), the structural position $\eta_x$ of each node $x$, $VO$, and the signature of the tree $\sigma_T$. It verifies the authenticity of the contents; if they are authentic then the integrity of structural relations are verified.
**RRedact:** Compute signature of subtree(s) \( T_\delta(V_\delta, E_\delta) \subset T(V, E) \).

1. Aggregate signature of the nodes: \( \sigma_{T_\delta} \) computed as follows:
   - in CRSA, \( \sigma_{T_\delta} = \prod_{x \in (V_\delta)} \sigma_x \mod \bar{n} \); in BGLS, \( \sigma_{T_\delta} = (P, \sum_{x \in (V_\delta)} \sigma_x) \).

2. Verification object \( VO \) computed as follows:
   - CRSA: \( VO \leftarrow \omega_T \prod_{x \in (V - V_\delta)} \sigma_x \mod \bar{n} \);
   - BGLS: \( VO \leftarrow \omega_T + \sum_{x \in (V - V_\delta)} \sigma_x \).

Figure 5.3. Algorithm to compute the redacted signature of a subtree \( T_\delta(V_\delta, E_\delta) \subset T(V, E) \).

*Authentication of Contents:* Authentication of contents and structural positions of the subtree received verifies (1) its integrity and, (2) the authorized owner (signer) of the subtree. Verification of structural relations is carried out in the next step.

*Verification of Structural Relations:* The integrity verification of structural relations in a tree involves traversing the tree and comparing the RPON (RRON) of each node with the RPON (RRON) of its parent or its sibling (Figure 5.3.4).

5.3.5 Sharing a Subtree – Only the Nodes

An advantage of the use of structural signatures is that there is no need to supply the user with the structure of the subtree it is receiving. It reduces the amount of data that needs to be transmitted from the distributor to the users and thus improves the efficiency of the data distribution. The structure can be reconstructed from the pre-order and post-order traversals (for non-binary trees [51]) (in-order and pre/post-traversals for binary trees [75]).
Authentication and Reconstruction of a Subtree: If all the node contents are verified to be authentic, the subtree $T_δ$ is re-constructed using RPONs and RRONs using the algorithm in Figure 5.3.5. Structural integrity is automatically verified during the subtree re-construction process. This algorithm has a lower-bound complexity of $O(n\log n)$ due to sorting.
rvrfy: Verify authenticity of $T_\delta(V_\delta, E_\delta)$.

1. Authentication of contents:

   (a) For each $y \in V_\delta$ (received nodes), $\xi_y \leftarrow \mathcal{H}(\eta_y \| c_y)$.

   (b) CRSA: $\prod_{y \in V_\delta} \xi_y \overset{?}{=} (\sigma_{T_\delta})^e \mod \bar{n}$.
       BGLS: $(Q, \sum_{y \in V_\delta} \xi_y) \overset{?}{=} \sigma_{T_\delta}$.

   (c) CRSA: $VO \prod_{y \in V_\delta} \xi_y \overset{?}{=} (\sigma_T)^e \mod \bar{n}$.
       BGLS: $(Q, VO + \sum_{y \in V_\delta} \xi_y) \overset{?}{=} \sigma_T$.

   (d) If (b) and (c) are true, the contents and structural positions of $T_\delta(V_\delta, E_\delta)$ are authenticated.

   (e) Else if (c) is true and (b) is false, at least one of the nodes in $V_\delta$ is an unauthorized node.

   (f) Else if (b) is true and (c) is false, at least one of the nodes in $T(V_\delta, E_\delta)$ has been dropped in an unauthorized manner from $V_\delta$ or from the computation of $\mathcal{VO}$.

2. Verification of structural relations:

   (a) Carry out a pre-order traversal on $T_\delta$.

   (b) Let $x$ be the parent of $z$; if ($(p_x \leq p_z)$ or $(r_x \geq r_z)$), then parent-child relationship between $x$ and $z$ is incorrect.

   (c) For ordered trees, let $y$ be the right sibling of $z$; if ($(p_x \geq p_y)$ or $(r_x \geq r_y)$), then the left-right order among the siblings $y$ and $z$ is incorrect.

Figure 5.4. Algorithm to verify the authenticity of a redacted subtree $T_\delta(V_\delta, E_\delta)$.
1. Initialize a stack $S_p$, and subtree $T_δ$ as empty.

2. $SortedV \leftarrow$ Sort the vertices with increasing RRONs.

3. $x \leftarrow$ remove first item from $SortedV$.

4. Push($S_p, x$). Assign $x$ as the root of $T_δ$.

5. $y \leftarrow$ remove next item from $SortedV$; (i.e., $r_y > r_x$).

6. Push($S_p, y$). Create edge $e(x, y)$.

7. While $SortedV$ is not empty

   (a) $z \leftarrow$ remove next item from $SortedV$.

   (b) Let $y$ refer to the item on top of $S_p$.

   (c) If $p_y > p_z$, create $e(y, z)$.

   (d) Else Pop($S_p, y$), and repeat from Step 7(b).

   (e) Push($S_p, z$).

Figure 5.5. Algorithm to reconstruct the structure of a tree given the set of structural positions.
5.3.6 Illustration

Consider the tree in our running example (Figure 3.3) and suppose that the tree has been assigned post and pre-order numbers (Figure 5.6(a)) and their randomized counterparts (Figure 5.6(b)). Since the IVs and the hash values are large bit strings (e.g., 256 bits for SHA2), we do not show their actual values. The document $T(V, E)$, its signature $\sigma_{T(V, E)}$, the structural position of each node $\eta_x$, and the salt $\omega_T$, if any, are then stored at the database server (distributor).

The cashier has access to subtree $T_{a_1}$. The database server $D$ sends two nodes - $a_2$ and $a_3$, their structural positions, the tree-signature $\sigma_T$, and $VO$. $VO$ is computed by $D$ from the IVs ($\xi_x$) of the remaining nodes. The cashier receives two nodes $a_2$ and
a_3. She applies the authentication procedure which includes computing the IV of \( a_2 \) by computing the hash of the (66.2, 69.1) and \( c_2 \) (and similarly that of \( a_3 \)). If the authentication is successful, then the integrity verification of structural relationships and orderings is carried out. Then because \( a_2 \) and \( a_3 \) were sent as siblings, the cashier verifies whether \( p_2 \) (=66.2), the RPON of \( a_2 \) is smaller than \( p_3 \) (=69.5), the RPON of \( a_3 \); if so, then \( a_3 \) is an ancestor or a right sibling of \( a_2 \). However since \( r_2 \) (=69.1) < \( r_3 \) (=78.2), \( a_3 \) is not an ancestor of \( a_2 \). Thus their relationship is correctly verified.

In the second scenario, the nurse is authorized to access \( T_{b2} \); however suppose he receives a tampered \( T_{b2} \), such that in the tampered tree, \( a_{10} \) is the child of \( a_5 \) and a left sibling of \( a_8 \). Such a violation of structural integrity can be detected by comparing the structural positions of the nodes as discussed in Section 6.2.3. The RRON of node \( a_{10} \) is greater than that of \( a_8 \), which means that \( a_{10} \) cannot be a left sibling of \( a_8 \). If \( a_{10} \) is received as the right sibling of \( a_8 \), the structural integrity is violated. Such violation is detected, because the RPON of \( a_{10} \) is smaller than that of \( a_8 \), which means that \( a_{10} \) cannot be a right sibling of \( a_8 \).

### 5.4 Security Analysis

This section analyzes the security of the structural signature scheme in terms of its authenticity and confidentiality guarantees with respect to information leakage defined earlier.

**Lemma 5.4.1 (Name-transparency)** Under the assumption that Boldyreva et al.’s order-preserving encryption scheme OPE [26] is POPF-CCA secure, structural positions as secure names are name-transparent.

**Proof** Follows from [26].
Lemma 5.4.2 (Authenticity) Under the random oracle hypothesis, and the assumption that the RSA problem is hard, $r\Pi = (rGen, rSign, rRedact, rVrfy)$ is existentially unforgeable under chosen-message attacks over subset operation over trees/graphs/forests.

**Proof** [Sketch] The signature $\sigma_{T(V,E)}$ of a tree $T(V,E)$ is unforgeable under adaptive chosen message attack (CRSA under strong RSA assumption [105], or BGLS under the assumption that computational Computational Diffie-Hellman problem is hard [28]). Thus any violation to the authenticity of contents and structural positions would be detected by the structural signature scheme. Consider that an adversary $A$ can compromise the content $c_x$ or the structural position $\eta_x$ of a node $x$ in $T$ without invalidating the structural signature of the tree, then $A$ has successfully found a collision in $H$, which not feasible under the random oracle hypothesis, or or $A$ has been able to solve the RSA problem or the computation Diffie-Hellman problem (BGLS: [28]), efficiently, which contradicts our assumption.

*Structural integrity.* Any unauthorized re-ordering of two or more nodes (violation of structural integrity) can be detected using the RPON’s and RRON’s (Lemma 5.2.2). By the Random Oracle Hypothesis [16], it is not feasible one to find a collision in $H$ function, which is why one cannot replace a node by an unauthorized one (or else the integrity verifier would be invalidated).

However, anyone can compute the signature $\sigma'_{T_{\delta}(V_\delta,E_\delta)}$ of a subtree $T_\delta$ of tree $T(V,E)$, without the knowledge of the secret key. Therefore, the signature scheme is existentially unforgeable against chosen message attacks over the $\subset$ operation.

Lemma 5.4.3 (Transparency) Under the assumption that the order-preserving encryption scheme is POPF-CCA-secure, then $r\Pi$ preserves transparency.

**Proof** [Sketch] Consider the experiment for transparency: $\text{Sig-Transp}_{A,r\Pi}^{\text{transp}}(\lambda)$ Experiment 4.4.2. The adversary $A$ invokes the signing oracle, with two trees: $T(V,E)$: that $x$ as a root of $w \prec y \prec z$; and $T_{\delta}(V_\delta,E_\delta)$: that $x$ as a root of $w \prec z$; i.e., $y$ is not
present in $T_\delta(V_\delta, E_\delta) \subset T(V, E)$. Let $\Upsilon_0(V_0, E_0)$, and $\Upsilon_0(V_0, E_0)$ refer to $T_\delta(V_\delta, E_\delta)$ and $\Upsilon_0(V_0, E_0)$ refer to $T(V, E)$.

1. $rGen(1^\lambda)$ is run to obtain keys $(pk, sk)$.

2. Probabilistic polynomial-time adversary $A$ is given $pk$ and oracle access to $rSign_{sk}(\cdot)$, $rRedact(\cdot)$ and $rVrfy_{pk}(\cdot)$. Let $Q$ be the set of all trees/graphs/forests for which $A$ queries $rSign_{sk}(\cdot)$ for their signatures.

3. $A$ outputs two trees/graphs/forests $\Upsilon_0(V_0, E_0)$, and $\Upsilon_1(V_1, E_1)$, such that $\Upsilon_0(V_0, E_0) \subset \Upsilon_1(V_1, E_1)$.

4. Draw a random $b$ uniformly from $\{0, 1\}$. $(\sigma_{\Upsilon_0}, \Upsilon_b(V_b, E_b)) \leftarrow rSign_{sk}(\Upsilon_b(V_b, E_b))$. If $b = 1$, then the signature for $\Upsilon_0(V_0, E_0)$ is computed from $\sigma_{\Upsilon_1}$:

$(\sigma_{\Upsilon_0}, \Upsilon_0(V_0, E_0)) \leftarrow rRedact(pk, \sigma_{\Upsilon_1}, \Upsilon_1(V_1, E_1), V_1 \setminus V_0, E_1 \setminus E_0)$. Then the challenge $(\sigma_{\Upsilon_0}, \Upsilon_0(V_0, E_0))$ is given to $A$.

5. The adversary $A$ continues to have oracle access to $rSign_{sk}(\cdot)$, $rRedact_{pk}(\cdot, \cdot)$, and $rVrfy_{pk}(\cdot)$. Eventually, $A$ outputs a bit $b'$ (for the challenge $(\sigma_{\Upsilon_0}, \Upsilon_0)$).

6. The output of the experiment is 1 if $b' = b$, and 0 otherwise.

Consider that $\text{Sig-Transp}^\text{transp}_{A, \Pi}(\lambda)$ returns 1. For $\text{Sig-Transp}^\text{transp}_{A, \Pi}(\lambda)$ to return 1, $A$ determined the correct value of $b$, which in turn implies that a node (i.e., $y$) is missing from the set of children of $x$, and that $w$ is left of this missing node, which is left of $z$. In other words, $A$ has managed to break the POPF-CCA security of Bondyreva et al.'s symmetric order-preserving encryption, which is against our assumption. So the lemma is proven that $r\Pi$ is transparent.

5.5 Complexity and Performance Analysis

In this section, we present the complexity and performance analysis of the structural signatures for trees.
Figure 5.7. CRSA: Average time to sign versus number of IVs.

5.5.1 Complexity Analysis

Trees: The cost of signing a tree $T(V, E)$ is $O(|V|)$. The number of signatures that is computed is $|V|+1$. The number of structural positions that a user receives is the number of nodes in $T_δ$. The number of signatures that need to be sent to a user is 1 irrespective of the size of the subtree $T_δ(V_δ, E_δ)$ that a user receives. The time to compute this aggregate signature of the subtree at the distributor is $O(|V_δ|)$. The user receives 1 signature for the subtree $T_δ(V_δ, E_δ)$, 1 signature for the complete tree, and 1 verification object VO. The verification cost for content/structural integrity is linear in terms of the size of the received subtree, that is, $O(|V_δ|)$. The cost of comparison of RPON’s and RRON’s (and RIONs for binary trees) is constant.

5.5.2 Performance

We implemented the structural signature scheme and the Merkle hash technique in Java 1.6 and JCA 6.0 (Java Cryptography Architecture) APIs. The experiments were carried out on an IBM Thinkpad T61 with the following specification: 32-bit Linux (Ubuntu 10.04) on Intel Core 2 Duo CPU with 3GB RAM (2560MB as the maximum heap size for Java). Performance analysis of our BGLS-based authentication scheme
is based on a C++ implementation. We used the implementation of pairing-based cryptography (PBC) from Stanford, OPENSSL and GMP\(^1\).

*Experiments* Our experiments are on complete trees. The trees are 2-ary with the height from 1 to 16. The number of nodes in the largest tree is 65535. The subtrees used for authentication are complete left-most subtrees in a tree with the height varying from 1 to 16. We carried two experiments: one for the time taken to

\(^1\)PBC and GMP are available at http://crypto.stanford.edu/pbc and http://gmplib.org, respectively.
sign using CRSA-based structural signature and another using BGLS-based structural signatures. The size of RSA keys is 1024-bits and SHA1 as the hash function. We have also implemented the BGLS scheme.

**CRSA Vs. BGLS** The BGLS authentication scheme is much more expensive for trees than the CRSA-based scheme. This is expected [105] as the BGLS aggregate signature scheme is based on elliptic curves and bilinear maps. We proposed the use of the BGLS scheme in the paper, because cryptosystems developed on elliptic
curves have already been in use as an alternative to the RSA cryptosystems. In case of BGLS however, signing a tree of \( n \) nodes and verification of a tree of the same number nodes \( n \) take the same amount of time. This can be explained by considering the cost of the bilinear operation, which is much higher than any other cost such as verification of structural relationships.

**Signing:** Figure 5.7 presents the performance of computing the signature of a tree using CRSA, and Figure 5.8 presents signing using BGLS based structural signatures. As mentioned above, it takes more time to compute the BGLS signatures than the CRSA signatures. Each node has one integrity verifier in a Ltree. In comparison, the less secure Merkle Hash Technique incurs much less cost (about two seconds to sign 65535 integrity verifiers using RSA) for signing a tree (the common tradeoff of cost Vs. security) as it computes a single signature. However, the cost of signing a tree with structural signatures is not an issue especially when the structural signature is computed once and re-used many times for any subtree.

**Distribution:** Figure 5.9 presents the performance of distributing subtrees with height from 1 to 16 using CRSA, and Figure 5.10 presents distribution using BGLS based structural signatures. The larger the subtree of a tree with 65535 nodes being distributed by a server, the less the number of nodes over which the verification object

![Figure 5.12. BGLS: Average time to verify versus number of nodes.](image-url)
VO being computed, and more the number of nodes over which the condensed (aggregate) signature is being computed. Therefore, as the size of the subtree increases (number of nodes), the cost of distribution decreases. The cost of distribution for CRSA is slightly less than that for BGLS. Pre-computed integrity verifiers of nodes are stored at the distributor, and are used in distribution.

Verification: Figure 5.11 presents the performance of verifying subtrees (that are distributed) with height from 1 to 16 using CRSA, and Figure 5.12 presents performance of verification using BGLS based structural signatures. The amount of time taken for verification using our structural signature scheme using CRSA is much less than the time required by the BGLS. BGLS takes more time than CRSA because two bilinear maps are to be applied and \( m \) number of elliptic curve computations are needed to be carried out, while for CRSA only \( m \) modular multiplications and two modular exponentiations are needed to be carried out. MHT takes more time (about 2.25 seconds for 65535 nodes) than the CRSA based structural signatures, but less than BGLS. Even small overhead in verification is significant because: (1) the time taken for user-side authentication affects the end-to-end response time at the user side, and (2) since authentication would be carried out by many users, the collective overhead would be very high, (3) for energy-constrained devices, the processing overhead of even 2.25 seconds should be avoided.

5.6 Dynamic Trees

Updates to a tree include insertions, deletions or a combination of insertions and deletions. An update to the tree is reflected by a new signature such that replay attacks cannot be carried out. If the signature does not reflect the updates, a replay attack on the authentication of dynamic trees can be carried out easily. When the owner Alice updates the tree \( T(V, E) \) to \( T'(V', E') \), a malicious distributor may continue distributing or processing queries on an older version; this can be prevented if
some extra authentication information is supplied to the user alongwith the signature. Certification validation and revocation are used in this context.

In our solution, whenever Alice signs an updated tree, she includes the hash of the current timestamp in the computation of the signature. In order to make all the previous signatures invalid, Alice publishes the latest timestamp when the current signature was computed. A signature is valid, if it contains the latest timestamp.

**Insertion of nodes:** Upon insertion of a node $z$, its structural position $\eta_z$ is computed such that the RPON $p_z$ and RRON $r_z$ preserve correct relationships with the parent and siblings of the node. Insertion of a subtree is carried out as a sequence of insertions.

**Leaf-level updates:** Let $z$ be the new leaf node in $T'$ added as the child of $x$, a leaf in $T$. Let the lowest RPON in $T$ be $p_u$, and the largest RRON in $T$ be $r_v$. RPON $p_z$ should satisfy $p_u < p_z < p_x$, and RRON $r_z$ should satisfy $r_x < r_z < r_v$.

**Root-level updates:** Let $x$ be the root node and $z$ be the new root node added as the parent of $x$. RPON $p_z$ should satisfy $p_x < p_z$, and RRON $r_z$ should satisfy $r_z < r_x$.

**Updates in-between two siblings:** Let $z$ be inserted as a new node between two siblings $x$ and $y$ such that $x$ is to the left of $y$. RPON $p_z$ should satisfy $p_x < p_z < p_y$, and RRON $r_z$ should satisfy $r_x < r_z < r_y$.

**Updates in-between parent and child:** Let $z$ be inserted as a new node between parent $x$ and child $y$ such that $x$ is to the left of $y$. RPON $p_z$ should satisfy $p_y < p_z < p_x$, and RRON $r_z$ should satisfy $r_x < r_z < r_y$.

Let $(L_p, R_p)$, and $(L_r, R_r)$, respectively be the intervals in which the RPON $p_z$ and RRON $r_z$ of $z$ should be in (as specified above). Computation of the RPON and RRON can be carried out as follows: repetitively select a secure random number $\eta$ until $L_p < \eta < R_p$; $\eta$ is assigned to $p_z$. 
**Deletion of nodes:** Upon the deletion of a node or a subtree, only the signature needs to be recomputed on the remaining set of nodes \((V')\). The structural positions of nodes that are remaining from the old tree are not modified.

**Insertion and deletion of edges:** Deletion of an edge is generally followed by an insertion of a new edge and vice versa, in order to maintain the structural properties typical of trees to remain connected and to have \(n - 1\) edges for \(n\) nodes. If \(e(x, y)\) is removed and \(e(z, y)\) is inserted, then the structural position \(\eta_y\) of \(y\) (and all its children, if any) needs to be re-computed with respect to \(z\). It is similar to deletion of \(y\) and then insertion of \(y\) as a child of \(z\). The subtree rooted at \(y\) remains rooted at \(y\). Such a process leads to re-computation of RPON and RRON of \(y\), and the nodes in its subtree in the same manner as it is done for insertion of nodes.

**Incremental computation of the signatures:** Alice may store the current \(\sigma_x\) for each node \(x \in V'\), so that it can be re-used during the next computation of the signature. As seen earlier, it might be necessary to re-compute RPONs and RRONs after a large number of insertions (in the order of millions for 512-bit RPONs and RRONs) between nodes. In practice, such re-computations would not occur frequently and thus are not a major concern, because in large trees (with millions of nodes), insertions get distributed over various parts of the tree. For small trees, signing (includes computation of RPONs and RRONs) is not expensive (corroborated by our experimental results). If a node is dropped, the multiplicative inverse (CRSA) or additive inverse (BGLS) of its signature (signatures in case of cyclic graphs) is computed and is used to cancel out the corresponding signature(s) of the node in the signature of the tree/graph.

**Complexity:** Insertion or update of a node and a subtree that has \(m\) nodes incurs \(O(1)\) and \(O(m)\) cost, respectively including the cost of computing one signature. Deletion of a node or a subtree incurs only the cost of computation of the signature. In con-
trast, in the case of Merkle Hash technique, an insertion or deletion of a single node 
z in a tree of n nodes incurs $O(\log(n))$ cost in average. This is because, the Merkle 
hashes of all the ancestors of z are affected due to that operation (even though only 
one signature has to be computed). Insertion or deletion of a subtree of m nodes 
incurs $O(m\log(n))$ cost in the MHT.

5.7 Applications

The structural authentication scheme facilitates structural recovery of trees, which 
is of importance in data recovery and in digital forensics.

5.7.1 Automatic Recovery from Structural Errors

Consider the case in which the nodes (contents and structural positions) in a tree 
are not compromised, but their structural integrity is compromised by compromising 
the edge relationships and sibling orderings (as specified in the adjacency list or other 
means). From the structural positions (RPONs and RRONs), the correct structure 
of the tree can be easily re-constructed. Such a capability helps automatic correction 
of data without any interaction with the distributor. The cost of such re-construction 
is $O(|V|)$ for a tree $T(V,E)$.

For example, suppose that two nodes $x$ and $y$ are received and are authenticated 
against the signature. However, the structural integrity is found to be invalid. Suppose 
that the ordering that $y$ is left of $x$ is the received (but incorrect) order for these 
nodes in a subtree. The user tests using the structural positions if $x$ is left of $y$. If 
the test succeeds, the correct order is recovered. If there are many nodes with their 
structural integrity found to be invalid, the reconstruction algorithm is applied to 
re-construct the structure of the subtree.

Structure-based Routing: Secure Publish/Subscribe of XML: The structural signa-
ture scheme can be used for secure dissemination of XML documents (and of data
objects organized as trees, in general) in a publish-subscribe model. Chapter 8 describes this scheme of structure-based routing in detail.

5.8 Summary

In this chapter, we proposed the notion of structural signatures in order to assure authenticity of subtrees and subgraphs without leaking. The notion of structural signatures for trees is based on the simple notion of tree traversals and the fact that a combination of two tree traversals - post-order and pre-order can be used to uniquely re-construct a tree and any of its subtrees. The number of signatures computed is linear in the number of nodes. However, the number of signatures that are sent to the user by the distributor is constant: 2. In terms of complexity, the scheme for trees incurs $O(n)$ cost for time and storage ($n$ is the number of nodes in a tree).

In this chapter, we have also discussed two important applications of the proposed structural signatures scheme for trees. One in the area of automatic data recovery from errors - our scheme is capable of automatic correction of structural errors in the data with no extra overhead. The other application is the secure publish/subscribe of XML documents using structure-based routing. This technique is an extension to our earlier work, and is more secure and efficient.
In the previous chapter, we have described a scheme for authentication of a single subtree that is based on the notion of depth first traversal numbers - pre-order numbers and post-order numbers. However, this scheme cannot be directly applied to graphs, because graphs are structurally different from trees, and graphs with cycles cannot be topologically sorted.

In this chapter, we propose two schemes on how to authenticate DAGs and directed cyclic graphs without leaking, which are the first such schemes in the literature. It is based on the structure of the graph as defined by depth-first graph traversals and aggregate signatures. Graphs are structurally different from trees in that they have four types of edges: tree, forward, cross, and back-edges in a depth-first traversal. The fact that an edge is a forward, cross or a back-edge conveys information that is sensitive in several contexts. Moreover, back-edges pose a more difficult problem than the one posed by forward, and cross-edges primarily because back-edges add bidirectional properties to graphs. We prove that the proposed technique is both authenticity-preserving and non-leaking. While providing such strong security properties, our scheme is also efficient, as supported by the performance results. The common notations used in this chapter are given in Table 6.

6.1 Background

Graphs contain edges that can be organized into four different types based on a specific depth first traversal of the graph: tree-edges, forward-edges, cross-edges and back-edges (related to cycles), while trees have only one type of edges: tree-
Table 6.1
Acronyms and notations

<table>
<thead>
<tr>
<th>Notation/Acro.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Related to forward, the direction of forward-, cross- and tree-edges.</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Related to backward, the direction of back-edges. Used in the context of $\beta$-nodes and $\beta$-reachable nodes.</td>
</tr>
<tr>
<td>$f, \chi, \tau$</td>
<td>For forward-, cross- &amp; tree-edges.</td>
</tr>
<tr>
<td>$q_\chi^x, r_\chi^x$</td>
<td>$\chi$-RON of $x$, $\chi$-RRON of $x$.</td>
</tr>
<tr>
<td>$\alpha_x^{u\rightarrow v}, \beta_x^{u\rightarrow v}$</td>
<td>$\beta$-PON, $\beta$-RPON, $\beta$-RON, $\beta$-RRON, resp. of $x$ with respect to a not-$\beta$-covered back-edge $e(u,v)$.</td>
</tr>
<tr>
<td>$\eta_x^\tau, \eta_x^\chi$</td>
<td>Structural position of $\tau$- or $\chi$-node $x$.</td>
</tr>
<tr>
<td>$\eta_x^{\beta;u\rightarrow v}, \sigma_x^{\beta;u\rightarrow v}$</td>
<td>Structural position/signature resp., of a $\beta$-node or $\beta$-reachable node $x$, with respect to a not-$\beta$-covered back-edge $e(u,v)$.</td>
</tr>
<tr>
<td>$\sigma_x^\alpha$</td>
<td>Forward structural signature of $x$. (same as $\alpha$-signature of $x$).</td>
</tr>
<tr>
<td>$\sigma_G^\alpha$</td>
<td>Structural signature of DAG $G$. Also referred to as $\alpha$-signature of $G$.</td>
</tr>
<tr>
<td>$\sigma_G$</td>
<td>Structural signature of graph $G$.</td>
</tr>
<tr>
<td>$\cup$</td>
<td>Union of the left set with the right set followed by the assignment of the result to the left set.</td>
</tr>
</tbody>
</table>

depth-first traversal. Numbers are assigned to a node in the order in which edges [77]. Depth-first traversal numbers are used to determine the type of an edge for that specific traversal. Such numbers are assigned to a node in the order in which
they are visited in a specific traversal; for example, post-order and pre-order numbers are assigned to the nodes in post-order and pre-order traversals, respectively. (For more details, the reader is referred to [77]). The various types of edges in a graph are defined below using the notion of traversal numbers.

**Definition 6.1.1** Let $\tau$ be the depth-first tree (DFT) of a directed graph $G(V,E)$. Let $x, y \in V$, and $e(x,y) \in E$. Let $o_x$ and $q_x$ refer to post-order number and pre-order number of node $x$, respectively. With respect to the DFT $\tau$, $e(x,y)$ is a (1) tree-edge, iff $o_x > o_y$, and $q_x < q_y$; (2) forward-edge, iff there is a path from $x$ to $y$ consisting of more than one tree-edges, $o_x > o_y$, and $q_x < q_y$; (3) cross-edge, iff $o_x > o_y$, and $q_x > q_y$; (4) back-edge, iff $o_x < o_y$, and $q_x > q_y$.

An example of depth-first tree, types of edges are given in Figure 1.1 with the post- and pre-order numbers for each node being given in the table in the figure. An authenticity-preserving and confidentiality-preserving scheme for graph data must not convey the knowledge of whether a given edge is a forward-edge (edge $e(g_3,g_6)$), a cross-edge (edge $e(g_5,g_6)$) or a back-edge (edge $e(g_6,g_2)$), unless the user is authorized to access a corresponding tree-edge(s) or the associated cycle, respectively. The information leakages due to the knowledge about the type of the edge are listed in Table 6.1. Such leakages are described in detail in the context of health information in Section 3.3. Whether an edge is a back or a cross-edge can be determined using both post-order and pre-order numbers [77], as the above definition specifies.
Table 6.2
Information leakages via edge-types.

<table>
<thead>
<tr>
<th>Type of $e(x, y)$</th>
<th>Associated information leakages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward-edge</td>
<td>(i) in-degree of $y \geq 2$.</td>
</tr>
<tr>
<td></td>
<td>(ii) $\geq 2$ edges $e'$ incident on $y$.</td>
</tr>
<tr>
<td></td>
<td>(iii) $e'$ is a tree-edge.</td>
</tr>
<tr>
<td></td>
<td>(iv) $\geq 2$ nodes $w$, such that $x \ldots w \ldots y$ is a simple path.</td>
</tr>
<tr>
<td></td>
<td>(v) $G$ is larger than the $G_δ$.</td>
</tr>
<tr>
<td>Cross-edge</td>
<td>(i), (ii), (v).</td>
</tr>
<tr>
<td>Back-edge</td>
<td>(a) $\geq 1$ simple path from $y$ to $x$.</td>
</tr>
<tr>
<td></td>
<td>(b) $\geq 1$ cycle in the graph,</td>
</tr>
<tr>
<td></td>
<td>(c) one cycle between $x$, $y$; (d) (v).</td>
</tr>
</tbody>
</table>

In the remaining sections of this chapter, we first propose the structural signatures for DAGs, and then the structural signatures for graphs with cycles.

6.2 DAGs

In this section, we develop structural signatures for DAGs. Table 6.1 summarizes the common acronyms and notations that are used for DAGs and graphs. Consider a user authorized to access one or more cross-edges incident upon a node $x$, but not the associated tree-edge(s). With reference to Figure 3.4, the user has access only to $e(g_{11}, g_{12})$, which is a cross-edge. One way to share a forward or cross-edge with the user, without leaking it to the user the type of the edge (such as the fact that “$e(g_{11}, g_{12})$ is a cross-edge”) is to convey to the user that it is a tree-edge. By concealing the original type of an edge (such as forward or cross-edge) and conveying to the user that any edge it receives is of type tree-edge, unless the user also receives the associated tree-edge(s), we can prevent leakages associated with cross-edges. Note
that the structural signature makes use of the notion of traversal numbers, and that post-order and pre-order numbers allows one to detect whether an edge is a cross-edge or not (Definition 6.1.1). Moreover, such numbers cannot be used to differentiate a forward-edge from a tree-edge.

We thus need to define a different notion of traversal numbers so that a cross-edge would be verified by Bob as a tree-edge. In that context, we refer to the end-point of cross-edge(s) as a $\chi$-node. By Definition 6.1.1, it is the pre-order number (and not the post-order number) of a $\chi$-node that violates the behavior of pre-order numbers that is exhibited in case of tree-edges. We first define a variant of the pre-order numbers denoted by $\chi$-pre-order numbers (in short, $\chi$-RONs) (Definition 6.2.2) specifically for $\chi$-nodes. Using a randomized notion of such variants of pre-order numbers, denoted by $\chi$-randomized pre-order numbers (in short, $\chi$-RRONs), we define a structural position of a $\chi$-node that satisfies the constraints of a tree-edge in terms of traversal numbers (Definition 6.1.1). Such a notion of structural position is then used to compute the aggregate signature for the DAG. Since the definitions of $\chi$-RONs and $\chi$-RRONs are specific to cross-edges and $\chi$-nodes only, tree-edges are always conveyed as they are.

Given that the tree-edges, forward-edges, and cross-edges have the same direction, we refer to the signatures for DAGs as forward-signatures (denoted by $\alpha$-signatures). A node that is not a $\chi$-node such as $g_3$ in Figure 3.4, is referred to as a tree-node (in short, $\tau$-node).

**Definition 6.2.1 ($\chi$-node)** A node $x$ in a connected directed acyclic graph $G(V, E)$ is a $\chi$-node, iff there exists an edge $e(w, x)$ in $G$ such that $e(w, x)$ is a cross-edge.

**Definition 6.2.2 ($\chi$-RONs)** Let $x$ be a $\chi$-node in a connected directed acyclic graph $G(V, E)$. Let $e(w, x)$ and $e(x, y)$ be two edges in $G$. The $\chi$-pre-order numbers of $x$ and $w$ denoted by $q^x_x$ and $q^w_w$ are defined such that they satisfy the following conditions: (1) $q^x_x > q^x_w$; (2) if $w$ is a $\chi$-node, $q^x_w > q^w_w$, else $q^x_w > q^w_w$; (3) $q^x_v < q^w_v$; (4) $q^x_u < q^x_x < q^x_v$, where $u$, $x$, and $v$ are siblings and $u < x < v$. 


Since a DAG can be topologically-ordered, the properties of a $\chi$-pre-order number can be satisfied. For a $\chi$-node $x$, its $\chi$-randomized pre-order number ($\chi$-RRON) is the randomized version of $q_x^\chi$ and is denoted by $r_x^\chi$. It is defined in the same way RPON is defined for PON and RRON for RON.

In our running example (Figure 3.4), $g_{10}$ and $g_{12}$ are $\chi$-nodes; $r_{g_{10}}^\chi$ and $r_{g_{12}}^\chi$ are the $\chi$-RRONs of $g_{10}$ and $g_{12}$. The notions of $\tau$-structural and $\chi$-structural position ($\tau$-position and $\chi$-position in short, respectively) of a $\tau$-node and a $\chi$-node, respectively are defined by the following definitions.

**Definition 6.2.3 ($\tau$-structural position)** Let $x$ be a node in a connected DAG $G(V, E)$. Its $\tau$-structural position, denoted by $\eta_x^\tau$, is defined as: $\eta_x^\tau = (p_x^\tau, r_x^\tau)$.

**Definition 6.2.4 ($\chi$-structural position)** Let $x$ be a $\chi$-node in a connected DAG $G(V, E)$. Its $\chi$-structural position, denoted by $\eta_x^\chi$, is defined as: $\eta_x^\chi = (p_x^\chi, r_x^\chi)$.

In our running example (Figure 3.4), the $\tau$-structural position of node $g_3$ is $\eta_3^\tau = (p_3^\tau, r_3^\tau)$; for $\chi$-node $g_{10}$, its $\chi$-structural position is $\eta_{g_{10}}^\chi = (p_{g_{10}}^\chi, r_{g_{10}}^\chi)$.

### 6.2.1 Signing a DAG

The algorithm to sign a DAG is described in Figure 6.2.

**Definition 6.2.5 (Integrity Verifier of a Node)** Let $x$ be a node in a connected DAG $G(V, E)$. Let $\mathcal{H}$ denote a one-way cryptographic hash function. The integrity verifier $\xi_x$ of $x$ is defined as follows: $\xi_x = \mathcal{H}(\eta_x \| c_x)$, where: if $x$ is a $\chi$-node, $\eta_x$ is equal to $\eta_x^\chi$, else it is equal to $\eta_x^\tau$.

**Definition 6.2.6 (Signature of a DAG using CRSA)** Let $\mathcal{H}$ denote a random oracle, and $\omega_G$ be a random. The signature $\sigma_x$ of a node $x$ is defined as: $\sigma_x \leftarrow \xi_x^d \mod \bar{n}$, where $\xi_x$ is the IV of $x$. The structural signature of a graph $G(V, E)$, denoted by $\sigma_{G(V,E)}$, is defined as follows:

$$\sigma_{G(V,E)} \leftarrow (\omega_G \prod_{x \in V} \xi_x)^d \mod \bar{n}. \quad (6.1)$$
Definition 6.2.7 (Signature of a DAG using BGLS) Let \( H \) denote a random oracle, and \( \omega_G \) be a random. Let \( sk \) be the private key and \( P \) be defined as in an aggregate signature framework. Let \( \omega_G \) be a random. The signature \( \sigma_x \) of a node \( x \) is defined as: \( \sigma_x \leftarrow sk \xi_x \). The structural signature of a graph \( G(V,E) \), denoted by \( \sigma_{G(V,E)} \), is defined as follows:

\[
\sigma_{G(V,E)} \leftarrow e(P, sk(\omega_G + \sum_{x \in V} \xi_x)).
\] (6.2)

6.2.2 Distribution

Let \( G_\delta(V_\delta, E_\delta) \) be an arbitrary weakly connected subgraph of the weakly connected DAG \( G(V, E) \). A weakly connected (directed) graph is one in which if all the edges are changed from being directed to undirected, the graph turns into a connected one [45].

D sends the following items to Bob: (1) \( G_\delta(V_\delta, E_\delta) \), and (2) \( \sigma'_{G_\delta} = (\{\eta_x, \sigma_x | x \in V_\delta\}, \sigma_G, \sigma_{G_\delta}, VO) \) the set of verification units \( VO_{G_\delta(V_\delta, E_\delta)} \) consisting of the signature of the DAG \( \sigma_G \), the integrity verifier of the subgraph \( \xi_{G_\delta} \), and aggregate signature of the subgraph \( \sigma_{G_\delta} \). \( \xi_{G_\delta} \) and \( \sigma_{G_\delta} \) are computed as follows. The computation of one bilinear map in the BGLS step is to balance the work done between the distributor and the user(s). If the distributor does not carry out the bilinear mapping, then the user has to carry out two mappings.

6.2.3 Authentication

Figure 6.4 summarizes the interactions between Alice, D and Bob. A user Bob receives (1) a subgraph \( G_\delta(V_\delta, E_\delta) \), (2) the signature of the DAG \( \sigma_G \), and (3) the verification object \( VO \) of the subgraph from the distributor D. The verification process goes as follows.

Verification of Contents: If (2) (or 4) is invalid and (1) (resp., 3) is valid, then some nodes have been dropped from the subgraph (i.e., \( G_\delta \) does not contain some
rSign:

1. Execute a depth-first traversal of \( G(V,E) \).
2. For each node \( x \in V \), compute its post-order number \( \sigma_x^\tau \) and pre-order number \( q_x^\tau \).
3. If \( e(w,x) \in E \) is such that \( \sigma_w^\tau > \sigma_x^\tau \) and \( q_w^\tau > q_x^\tau \), then mark \( e(w,x) \) as a \( \chi \)-edge and \( x \) as a \( \chi \)-node.
4. For each \( \chi \)-node, compute its \( \chi \)-RON \( q_\chi^\chi \).
5. For all nodes in \( V \), transform the traversal numbers into traversal numbers, that preserve the order.
6. For a \( \chi \)-node \( x \), \( \eta_x^\chi \leftarrow (p_x^\tau, r_x^\chi) \).
7. For each \( x \) that is not a \( \chi \)-node, assign \( (p_x^\tau, r_x^\tau) \) to \( x \) as its structural position \( \eta_x^\tau \).
8. For each node \( x \) in \( V \), compute its integrity verifier \( \xi_x \):
   
   (a) If \( x \) is a cross-node, \( \eta_x \leftarrow \eta_x^\chi \). Else \( \eta_x \leftarrow \eta_x^\tau \).
   
   (b) \( \xi_x \leftarrow \mathcal{H}(\eta_x \| c_x) \).
9. Choose a random \( \omega_G \).
10. Compute the signature of the DAG either using CRSA (equation 6.1) or using BGLS (equation 6.2).

Figure 6.2. Algorithm to sign a DAG.
Verifying Structural Relations

Verifying structural relations in a DAG involves traversing the DAG and comparing the RPON (RRON) of each node with the RPON (RRON) of its parent or its sibling. The steps are given in Figure 6.2.3.

Example: Suppose that the back-edge $e(g_{14}, g_9)$ did not exist in our running example (Figure 3.4), thus turning this graph into a DAG. In such a DAG, the cross-edges are $e(g_{11}, g_{12})$, $e(g_8, g_{10})$, and $e(g_3, g_{10})$. Consider the cross-edge $e(g_{11}, g_{12})$ and the cross-node $g_{12}$. The $\alpha$-structural signatures for this DAG are computed such that $r_{g_{12}}^x$ is larger than $r_{g_{11}}^x$ and $p_{g_{12}}^\tau$ is smaller than $p_{g_{11}}^\tau$, which conveys that $e(g_{11}, g_{12})$ is a tree-edge and conceals the fact that it is a cross-edge.
Authentication of contents:

1. CRSA: \( \prod_{y \in V_{\delta}} \xi_y \mod \bar{n} \stackrel{?}{=} \sigma_{G_{\delta}} \).

2. CRSA: \( \forall \xi_y \prod_{y \in V_{\delta}} \xi_y \mod \bar{n} \stackrel{?}{=} \sigma_{G} \).

3. BGLS: \( e(Q, \sum_{y \in V_{\delta}} \xi_y) \stackrel{?}{=} \sigma_{G_{\delta}} \).

4. BGLS: \( e(Q, \Omega + \forall \xi) \stackrel{?}{=} \sigma_{G} \), where \( \Omega \leftarrow \sum_{y \in V_{\delta}} \mathcal{H}(\eta_y \parallel c_y) \).

Verification of structural relations:

1. Execute a depth-first traversal of \( G_{\delta} \).

2. Let \( x \) be an immediate ancestor of \( z \); if \( p_x \leq p_z \) or \( r_x \geq r_z \), then this relationship between \( x \) and \( z \) is incorrect.

3. For ordered-DAGs, let \( y \) be the right sibling of \( z \); if \( p_z \geq p_y \) or \( r_z \geq r_y \), then the left-right order among the siblings \( y \) and \( z \) is incorrect.

Figure 6.5. Algorithm to authenticate a DAG.

6.3 Graphs with Cycles

In this section, we build on the solution for DAGs and present the general solution for graphs with cycles that handles all the four types of edges. Like in the case of DAGs, in our scheme, a user verifies a back-edge as a tree-edge, in order to conceal the fact that it is a back-edge and thus it prevents the leakages associated with it.

A back-edge (such as \( e(g_{14}, g_9) \) in the graph in Figure 3.4) should be presented to the user as a tree-edge unless the user has access to an cycle associated with the back-edge also. Following Definition 6.1.1, both the post-order and pre-order numbers of a node (e.g., \( g_{14} \)), which is the origin of a back-edge (denoted by \( \beta \)-nodes), violate
the behavior of the respective numbers that is exhibited in case of tree-edges. In
order to handle back-edges, we define a notion of $\beta$-post-order number ($\beta$-PON) and
$\beta$-pre-order number ($\beta$-RON) for each node that either is a $\beta$-node or is reachable
from a $\beta$-node over a simple path. Nodes that are reachable from a $\beta$-node $x$ over a
simple path over a $\beta$-edge $e(x, w)$, also need to be considered for the following reason.
The integrity verifier of such nodes must be such that when they are presented to an
authenticity prover along-with the related back-edge $e(x, w)$, they should not leak the
fact that $e(x, w)$ is a back-edge; rather they should be consistent with the information
presented to the prover that $e(x, w)$ is a tree-edge. Examples of $\beta$-reachable nodes
from the $\beta$-node $g_{14}$ in Figure 3.4 are $g_9$, $g_{11}$, $g_{12}$, $g_{15}$ and $g_{16}$.

We define the notion of $\beta$-nodes like the notion of $\chi$-nodes. $\beta$-nodes are nodes that
are origins of back-edges (Definition 6.3.1). We then define the notion of $\beta$-reachable
nodes (Definition 6.3.2). Our goal is to be able to provide the prover with a set of
verification items about back-edges and $\beta$-reachable nodes such that the prover would
learn that all the edges it received are tree-edges (since she does not have any right
to learn otherwise). In order to define such signatures, we define a variant of PON
and RON for such nodes (Definition 6.3.4). Such variants satisfy the property of
tree-edges in terms of traversal numbers.

**Definition 6.3.1 ($\beta$-node)** A node $x$ in a graph $G(V, E)$ is a $\beta$-node, iff there exists
an edge $e(x, w)$ in $G$ such that $e(x, w)$ is a back-edge.

**Definition 6.3.2 ($\beta$-reachable)** Let node $x$ be a $\beta$-node in a graph $G(V, E)$ and let
$e(x, w)$ be a back-edge. A node $y$ is said to be $\beta$-reachable from $x$ (over $e(x, w)$) in
$G$ iff either $y$ is $w$ or there exists a simple path $sp(x, y)$ from $x$ to $y$ in $G$ such that
$sp(x, y) = x \rightarrow w \ldots \rightarrow y$.

Unlike the case of cross-edges (Section 6.2), in the case of back-edges, a $\beta$-node
or a $\beta$-reachable node has the following position(s) and integrity verifier(s): (1) $\beta$-
structural; and (2) $\chi$-structural, if it is a $\chi$-node, $\tau$-structural, otherwise. A $\beta$-node,
or a node that is $\beta$-reachable, may have multiple $\beta$-structural positions ($\beta$-positions
in short) and integrity verifiers. A given node may be reachable from multiple back-edges over a simple-path. We only need to consider a minimal set of such edges, which are not covered by other back-edges in a simple-path. Such back-edges are called not-\(\beta\)-covered (Definition 6.3.3). The number of such positions and integrity verifiers for a node is the same as the number of not-\(\beta\)-covered back-edges (Definition 6.3.3) from which it is reachable over a simple path. However, a user receives one position per node. If a user does not have access to a cycle, but to the associated back-edge, the \(\beta\)-structural position and signature are sent to the user. However, if she has access to the cycle, conveying the fact that one of the edges in the cycle is a back-edge does not leak any information. In what follows, \(x \rightarrow y\) denotes the edge \(e(x, y)\) and \(y \rightarrow z\) denotes the simple path \(sp(y, z)\) from \(y\) to \(z\).

In a graph, a node maybe \(\beta\)-reachable from many back-edges. The question is do we need to consider only one of them or “some” of them in order to minimize the number of \(\beta\)-positions. Consider the graph in Figure 6.6 with \(a\) as the root of the depth-first tree; edges \(e(a, b)\) and \(e(a, c)\) are the tree-edges. Edges \(e(b, a)\) and \(e(c, a)\) are back-edges. Both \(b\) and \(c\) are \(\beta\)-nodes. \(a\) is \(\beta\)-reachable from both \(b\) and \(c\). The question is would there be one or two \(\beta\)-positions for \(a\). If a user has access to all nodes and only the back-edges, then how would both of them be proven as tree-edges. To this end, let us first define \(\beta\)-covered and not-\(\beta\)-covered edges.

**Definition 6.3.3 (\(\beta\)-covered, not-\(\beta\)-covered)** A back-edge \(e(u, v)\) is said to be “\(\beta\)-covered” by another back-edge \(e(x, y)\) in graph \(G(V, E)\) iff there is a simple path \(sp(x, v)\) in \(G\) such that \(sp(x, v) = x \rightarrow y \ldots \rightarrow u \rightarrow v\). A back-edge \(e(u, v)\) is said to be “not-\(\beta\)-covered” iff there exists no such back-edge \(e(x, y)\) in \(G\).
\( \beta \)-structural positions are assigned to nodes as follows: for each node \( x \) that is \( \beta \)-reachable from a not-\( \beta \)-covered back-edge \( e(u, v) \), a \( \beta \)-position is assigned to \( x \) for each such \( e(u, v) \). In order to define the constituents of a \( \beta \)-position, we define \( \beta \)-PONs and \( \beta \)-RONs next.

Definition 6.3.4 (\( \beta \)-PONs, \( \beta \)-RONs) Let \( e(x, w) \) and \( e(y, z) \) be back-edges in graph \( G(V, E) \). Let \( e(x, w) \) be not-\( \beta \)-covered and \( e(y, z) \) be \( \beta \)-covered by \( e(x, w) \). The \( \beta \)-post-order (\( \beta \)-pre-order) numbers of \( x, w, y \) and \( z \) with respect to \( e(x, w) \), denoted by \( \beta^p_{x\rightarrow w}, \beta^p_{w\rightarrow x}, \beta^p_{y\rightarrow z}, \text{ and } \beta^p_{z\rightarrow y} \), respectively \( (q^p_{x\rightarrow w}, q^p_{w\rightarrow x}, q^p_{y\rightarrow z}, \text{ and } q^p_{z\rightarrow y}) \), respectively) satisfy the following conditions:

1. \( \beta^p_{x\rightarrow w} = q^p_{x} \), if \( x \) is a \( \chi \)-node \( q^p_{x\rightarrow w} = q^p_{x} \), else \( q^p_{x\rightarrow w} = q^p_{z} \);
2. \( \beta^p_{w\rightarrow x} < \beta^p_{x\rightarrow w} ; q^p_{w\rightarrow x} > \beta^p_{x\rightarrow w} \);
3. \( \beta^p_{z\rightarrow y} < \beta^p_{y\rightarrow z} < \beta^p_{w\rightarrow y} \).

For a \( \beta \)-node \( x \), the randomized versions of its \( \beta \)-post-order number with respect to \( e(x, w) \) \( \beta^p_{x\rightarrow w} \) and \( \beta \)-pre-order number \( q^p_{x\rightarrow w} \) are denoted by \( p^p_{x\rightarrow w} \) and \( q^p_{x\rightarrow w} \), respectively, and are defined in the same manner as RPON is defined for PON and RRON for RON. In order to define \( \alpha \)-signatures for non-acyclic graphs, the \( \chi \)-pre-order-numbers for \( \chi \)-nodes are defined below with an additional constraint related to back-edges, but without changing their properties in Definition 6.2.2. Difference from Definition 6.2.2 is “such that neither of them is a back-edge”.

Definition 6.3.5 (\( \chi \)-RONs) Let \( x \) be a \( \chi \)-node in a connected directed graph \( G(V, E) \). Let \( e(w, x) \) and \( e(x, y) \) be edges in \( G \) “such that neither of them is a back-edge”. The \( \chi \)-pre-order number of \( x \) and \( w \) denoted by \( q^\chi_{x} \) and \( q^\chi_{w} \), respectively, satisfy the following conditions:

1. \( q^\chi_{x} > q^\chi_{z} \);
2. \( q^\chi_{y} > q^\chi_{w} \), else \( q^\chi_{z} > q^\chi_{w} \); and
3. \( q^\chi_{x} < q^\chi_{y} \).

Definition 6.3.6 (\( \beta \)-structural position) Let \( x \) be a node, which is \( \beta \)-reachable from \( y \) over the not-\( \beta \)-covered back-edge \( e(y, z) \) in a graph \( G(V, E) \). The \( \beta \)-structural
position of $x$ with respect to $e(y, z)$ denoted by $\eta^{\beta:y\rightarrow z}_x$, is defined as the pair of its $\beta$-RPON $p^{\beta:y\rightarrow z}_x$ and $\beta$-RRON $r^{\beta:y\rightarrow z}_x$, that is, $\eta^{\beta:y\rightarrow z}_x = (p^{\beta:y\rightarrow z}_x, r^{\beta:y\rightarrow z}_x)$. Similarly, the $\beta$-structural position of $y$ with respect to $e(y, z)$ denoted by $\eta^{\beta:y\rightarrow z}_y$:

The integrity verifiers associated with $\beta$-nodes, $\beta$-reachable nodes and back-edges are called backward signatures ($\beta$-integrity verifier in short). Each node has an $\alpha$-integrity verifier. Moreover, if a node $x$ is $\beta$-reachable from a not-$\beta$-covered back-edge $e(y, z)$ in $G$, it has a $\beta$-integrity verifier $\sigma^{\beta:y\rightarrow z}_x$.

**Definition 6.3.7 (\beta-integrity verifier of a Node)** Let $x$ be a node and $e(y, z)$ be a not-$\beta$-covered edge in a weakly connected directed graph $G(V, E)$ such that $x$ is $\beta$-reachable from $y$ over $e(y, z)$. The $\beta$-structural integrity verifier of $x$, denoted by $\xi^{\beta:y\rightarrow z}_x$, is defined as $\xi^{\beta:y\rightarrow z}_x = H(\eta^{\beta:y\rightarrow z}_x \parallel c_x)$. The $\beta$-structural signature of the $\beta$-node $y$, denoted by $\xi^{\beta:y\rightarrow z}_y$, is defined as $\xi^{\beta:y\rightarrow z}_y = H(\eta^{\beta:y\rightarrow z}_y \parallel c_y)$.

**Definition 6.3.8 (Signature of a Graph using CRSA)** Let $H$ denote a random oracle, and $\omega_G$ be a random. The signature $\sigma_x$ of a node $x$ is defined as: $\sigma_x \leftarrow \xi^d_x \mod \bar{n}$, where $\xi_x$ is the IV of $x$. The structural signature of a graph $G(V, E)$, denoted by $\sigma_{G(V,E)}$, is defined as follows:

$$\sigma_{G(V,E)} \leftarrow (\omega_G \prod_{x \in V} \xi^\alpha_x \prod_{x \in V} \prod_{e(y, z) \in S_x} \xi^{\beta:y\rightarrow z}_x)^d \mod \bar{n}. \quad (6.3)$$

**Definition 6.3.9 (Signature of a Graph using BGLS)** Let $\omega_G$ be a random. The signature $\sigma_x$ of each node $x$ is defined as: $\sigma_x \leftarrow sk(\xi^\alpha_x + \sum_{e(y, z) \in S_x} \xi^{\beta:y\rightarrow z}_x)$. Let $S_x$ be the set of not-$\beta$-covered edges from each of which $x$ is $\beta$-reachable. The structural signature of a graph $G(V, E)$, denoted by $\sigma_{G(V,E)}$, is defined as follows:

$$\sigma_{G(V,E)} \leftarrow e(P, sk(\omega_0 + \sum_{x \in V} \xi^\alpha_x + \sum_{x \in V} \sum_{e(y, z) \in S_x} \xi^{\beta:y\rightarrow z}_x)). \quad (6.4)$$
6.3.1 Signing a Graph

The steps that the trusted owner Alice uses to sign and share a graph $G(V,E)$ with Bob are presented in Figure 6.7. How the set of not-$\beta$-covered back-edges are determined is stated by Lemma 6.3.1. Answer freshness can be achieved by using a timestamp in the computation of the signature, which would deter replay attacks.

**Lemma 6.3.1** In a graph $G$, the $\beta$-node $u$ that has the lowest $\tau^\sigma_u$ among all $\beta$-nodes is such that a back-edge $e(u,v)$ in $G$ is not-$\beta$-covered by any other back-edge.

**Proof** Let $w$ be a node and $e(w,x)$ be a back-edge in $G$ such that $e(u,v)$ is $\beta$-covered by $e(w,x)$. In such a case, by the property of post-order numbers, the depth-first traversal would assign a post-order number $\tau^\sigma_w$ to $w$ such that $\tau^\sigma_w < \tau^\sigma_u$, which is a contradiction. ■

6.3.2 Distribution

In this section, we show how to provide an optimal distribution technique that sends only $O(1)$ integrity verifiers to the user.

$D$ sends the following items to Bob: $G_\delta(V_\delta,E_\delta)$, the signature of the graph $\sigma_G$, and a verification object of the subgraph $\mathcal{V}\mathcal{O}$. The set of verification units $\sigma_{G_\delta(V_\delta,E_\delta)} = \{\sigma_G, \mathcal{V}\mathcal{O}, \sigma_{G_\delta}\}$. $\mathcal{V}\mathcal{O}$ is computed as shows in Figure 6.9, which is continued in Figure 6.10.

6.3.3 Authentication

Figure 6.11 summarizes the interactions between Alice, $D$ and Bob. Bob receives (1) a subgraph $G_\delta(V_\delta,E_\delta)$, (2) the signature of the graph $\sigma_G$, and (3) $\sigma'_{G_\delta}$. The user verifies the authenticity of the contents as well as the structural position of the nodes in $G_\delta$ by using the aggregate signature $\sigma_{G_\delta(V_\delta,E_\delta)}$. The process for verification
of contents is same as the process described in Section 6.2.3. The algorithm given in Figure 6.12 are used to verify the contents authenticity of structural relations.

Using the depth-first traversal carried out during authentication of structural relations, it is easy to determine if \( x \) is a \( \beta \)-node or \( e(x, z) \) is a \( \beta \)-edge in \( G_\delta(V_\delta, E_\delta) \) (Definition 6.1.1). We just need to initialize a pair of new pre-order and post-order numbers of each node to \(-1\). These numbers are different than the ones that have been received as signature items. For each edge \( e(x, z) \), if the PON of \( z \) is already computed (i.e., its value is larger than \(-1\)), then \( e(x, z) \) is not a back-edge, which implies that \( x \) is not a \( \beta \)-node.

**Example:** Consider our running example in Figure 3.4. Edge \( e(g_{14}, g_9) \) is a back-edge; \( g_{14} \) is a \( \beta \)-node but is not \( \beta \)-reachable (and is the one with minimum \( p_{g_{14}}^{\beta} \)) (Lemma 6.3.1). Therefore the computation of \( \beta \)-signatures, i.e. step 2(\( a \)) starts from \( g_{14} \). The \( \beta \)-reachable nodes are \( g_9, g_{11}, g_{12}, g_{15} \) and \( g_{16} \). Notice that nodes \( g_{11} \) and \( g_{12} \) are also \( \chi \)-nodes. \( \beta \)-structural signatures are defined for each of these nodes. \( \chi \)-structural signatures are defined for \( g_{11} \) and \( g_{12} \) and \( \tau \)-structural signatures are defined for \( g_{14}, g_9, g_{15} \) and \( g_{16} \). The information that (1) \( p_{g_{14}}^{\beta g_{14} \rightarrow g_9} \) is larger than \( p_{g_9}^{\beta g_{14} \rightarrow g_9} \) and (2) \( r_{g_{14}}^{\beta g_{14} \rightarrow g_9} \) is smaller than \( r_{g_9}^{\beta g_{14} \rightarrow g_9} \), convey the prover that \( e(g_{14}, g_9) \) is a tree-edge. Similarly consider that a prover has access to the edges \( e(g_{14}, g_9) \) and \( e(g_9, g_{11}) \). The signatures of \( g_9 \) and \( g_{11} \) are computed such that (a) \( p_{g_9}^{\beta g_{14} \rightarrow g_9} \) is larger than \( p_{g_{11}}^{\beta g_{14} \rightarrow g_9} \) and (b) \( r_{g_9}^{\beta g_{14} \rightarrow g_9} \) is smaller than \( r_{g_{11}}^{\beta g_{14} \rightarrow g_9} \); (a) and (b) convey that \( e(g_9, g_{11}) \) is a tree-edge. The same can be verified for other structures.
**rSign**: Sign a graph \( G(V, E) \).

1. Forward pass on graph \( G(V, E) \).

   (a) Execute a depth-first manner traversal on \( G(V, E) \).

   (b) For each node \( x \in V \), compute \( o_x^r \) and \( q_x^r \).

   (c) If \( e(w, x) \in E \), and

      i. If \((o_w^r > o_x^r)\) and \((q_w^r > q_x^r)\), then mark \( e(w, x) \) as a cross-edge and \( x \) as a \( \chi \)-node.

      ii. If \((o_w^r < o_x^r)\) and \((q_w^r > q_x^r)\), then mark \( e(w, x) \) as a back-edge and \( w \) as a \( \beta \)-node.

   (d) For each \( \chi \)-node, compute its \( q^\chi \)

2. Backward pass on graph \( G(V, E) \).

   (a) Let \( V^\beta \) be the set of all back-nodes in \( G \).

   (b) Let \( y \in V^\beta \) such that \( o_y^r < o_u^r \), \( \forall u \in (V^\beta - \{y\}) \)

   (c) For each back-edge \( e(y, z) \) from \( y \), execute a depth-first traversal of the graph from \( z \).

   (d) For each \( w \) \( \beta \)-reachable from \( y \) over \( e(y, z) \),

      i. If \( w \) is not visited earlier from \( y \), compute \( o_w^{\beta:y\rightarrow z} \) and \( q_w^{\beta:y\rightarrow z} \).

      ii. Else compute \( o_w^{\beta:y\rightarrow z} \) and \( q_w^{\beta:y\rightarrow z} \), such that \( o_w^{\beta:y\rightarrow z} \) is less than and \( q_w^{\beta:y\rightarrow z} \) is larger than the respective values computed in the previous visit.

      iii. if \( w \) is a back-node, \( V^\beta \leftarrow V^\beta - \{w\} \).

   (e) Goto step (a) until \( V^\beta \) is empty.

---

Figure 6.7. Algorithm to sign a graph (continued to Figure 6.8).
rSign (Continued):

3. For each node $x \in V$, transform the traversal numbers into traversal numbers, so that they preserve the order.

4. For each node $x \in V$ that is a $\chi$-node, assign $(p_x^\tau, r_x^\chi)$ to $x$ as its structural position $\eta_x^\chi$.

5. For each node $x \in V$ that is not a $\chi$-node, assign $(p_x^\tau, r_x^\tau)$ to $x$ as its structural position $\eta_x^\tau$.

6. For each node $x \in V$, and for each not-$\beta$-covered back-edge $e(y, z)$ such that $x$ is $\beta$-reachable from $y$ over $e(y, z)$, $\eta_x^{\beta:y \rightarrow z} \leftarrow (p_x^{\beta:y \rightarrow z}, r_x^{\beta:y \rightarrow z})$.

7. For each node $y \in V$ such that $e(y, z)$ is a not-$\beta$-covered back-edge, $\eta_y^{\beta:y \rightarrow z} \leftarrow (p_y^{\beta:y \rightarrow z}, r_y^{\beta:y \rightarrow z})$.

8. For each node $x \in V$:
   
   (a) If $x$ is a $\chi$-node, $\eta_x \leftarrow \eta_x^\chi$ else $\eta_x \leftarrow \eta_x^\tau$.
   
   (b) Compute the $\alpha$-signature $\xi_x^\alpha \leftarrow \mathcal{H}(\eta_x || c_x)$.
   
   (c) If $x$ is $\beta$-reachable from a not-$\beta$-covered back-edge $e(y, z)$, compute the $\beta$-signature $\xi_x^{\beta:y \rightarrow z} \leftarrow \mathcal{H}(\eta_x^{\beta:y \rightarrow z} || c_x)$.
   
   (d) If $x$ is such that $e(x, w)$ is a not-$\beta$-covered back-edge, compute the $\beta$-signature $\xi_x^{\beta:x \rightarrow w} \leftarrow \mathcal{H}(\eta_x^{\beta:x \rightarrow w} || c_x)$.

9. Choose a secure random $\omega$; Let $\omega_G \leftarrow \mathcal{H}(\omega)$.

10. Compute the signature of the graph $G(V, E)$ either using CRSA (equation 6.3) or using BGLS (equation 6.4).

Figure 6.8. Algorithm to sign a graph (continued from Figure 6.7).
1. \( \emptyset \leftarrow \emptyset \).

2. Let \( \psi \) be a set: \( \psi \leftarrow \{ \langle x, \eta_x, \xi_x \rangle \mid \forall x \in V, \eta_x \text{ is a structural position of } x, \text{ and } \xi_x \text{ refers to the authentication unit of } x \text{ associated with } \eta_x \} \).

3. If no edge in \( G_\delta \) is a back-edge in \( G \), For each node \( x \) in \( G_\delta \),
   (a) If \( x \) is a cross-node, \( \eta_x \leftarrow \eta_x^\chi \). Else \( \eta_x \leftarrow \eta_x^\tau \).
   (b) \( \emptyset \cup (x, \eta_x, \xi_x^\alpha) \).

Else proceed to the next step.

4. For each \( e(x, y) \) in \( G_\delta \) that is a not-\( \beta \)-covered back-edge in \( G \),
   (a) For each node \( z \) in \( G_\delta \) such that \( z \) is either \( x \) or is \( \beta \)-reachable from \( x \) over \( e(x, y) \) in \( G_\delta \),
   \( \emptyset \cup (z, \eta_z^\beta:x\rightarrow y, \xi_z^\beta:x\rightarrow y) \). Flag \( z \) as visited.
   Flag \( e(x, y) \) as visited.

5. Let \( E_0^\beta \leftarrow \{ e(x, y) \mid e(x, y) \text{ is in } G_\delta \text{ and is not visited, } e(x, y) \text{ is a back-edge in } G \text{ and is not-\( \beta \)-covered in } G_\delta \} \).

6. For each \( e(x, y) \in E_0^\beta \),
   (a) Let be the back-edge \( e(u, v) \) in \( G \) but not in \( G_\delta \) such that \( e(x, y) \) is \( \beta \)-covered by \( e(u, v) \) in \( G \).
   (b) For each node \( z \) such that \( z \) is in \( G_\delta \), \( z \) is not visited, \( z \) is either \( x \) or \( \beta \)-reachable from \( x \) over \( e(x, y) \) in \( G_\delta \),
   \( \emptyset \cup (z, \eta_z^\beta:u\rightarrow v, \xi_z^\beta:u\rightarrow v) \). Flag \( z \) as visited.

Figure 6.9. Algorithm to distribute a subgraph (continued to Figure 6.10).
7. For each node $w$ that is not visited and has a simple path $w \ldots \rightarrow x$ in $G_{\delta}$, if $w$ is $\beta$-reachable from $u$ over $e(u,v)$, and $e(u,v)$ is in $G$, but not in $G_{\delta}$, flag $w$ as visited.

8. For each node $w$ in $G_{\delta}$ that is not visited, if $w$ is a $\chi$-node, flag $w$ as visited. Else flag $w$ as visited.

9. Compute $\xi_{G_{\delta}(V_{\delta}, E_{\delta})}$ and $\sigma_{G_{\delta}(V_{\delta}, E_{\delta})}$ as follows:

10. CRSA:
    (a) $\xi_{G_{\delta}} \leftarrow \prod_{(x, \eta_{x}, \xi_{x}) \in \psi \cup \theta} \xi_{x} \bmod \bar{n}$.
    (b) $\sigma_{G_{\delta}} \leftarrow \prod_{(x, \eta_{x}, \xi_{x}) \in \psi \cup \theta} \sigma_{x} \bmod \bar{n}$.

11. BGLS:
    (a) $\xi_{G_{\delta}} \leftarrow \sum_{(x, \eta_{x}, \xi_{x}) \in (\psi \cup \theta)} \xi_{x}$.
    (b) $\sigma_{G_{\delta}} \leftarrow e(P; \sum_{(x, \eta_{x}, \xi_{x}) \in (\psi \cup \theta)} \sigma_{x})$.

Figure 6.10. Algorithm to distribute a subgraph (continued from Figure 6.9).

Figure 6.11. Protocol for graphs.
Authentication of contents:

1. CRSA: \( \prod_{y \in V_\delta} \xi_y \mod \bar{n} = \sigma_{G_\delta} \).

2. CRSA: \( \mathcal{V} \mathcal{O} \prod_{y \in V_\delta} \xi_y \mod \bar{n} = \sigma_G \).

3. BGLS: \( e(Q, \sum_{y \in V_\delta} \xi_y) = \sigma_{G_\delta} \).

4. BGLS: \( e(Q, \Omega + \mathcal{V} \mathcal{O}) = \sigma_G \), where \( \Omega \leftarrow \sum_{y \in V_\delta} \mathcal{H}(\eta_y \parallel c_y) \).

Verification of structural relations:

1. Execute a depth-first traversal of \( G_\delta \). Let the structural position of each node \( x \) be denoted by \( \eta_x = (p_x, r_x) \).

2. For edge \( e(x, z) \), if \( p_x < p_z \) and \( r_x > r_z \), then if \( x \) is not a \( \beta \)-node in \( G_\delta \), then \( e(x, z) \) is not a back-edge (there is no cycle in \( G_\delta \) involving \( x \) and \( z \)), so the parent-child relationship between \( x \) and \( z \) is incorrect.

3. Else if \( p_x \leq p_z \) or \( r_x \geq r_z \), then parent-child relationship between \( x \) and \( z \) is incorrect.

Figure 6.12. Algorithm to verify the structural integrity of a graph.
6.4 Security Analysis

The proposed schemes for DAGs/graphs with cycles support non-repudiation primarily because both the CRSA and BGLS schemes support non-repudiation. Moreover, the proposed schemes for DAGs and graphs with cycles are “existentially unforgeable under adaptive chosen-message attack”. In what follows, we provide proof sketches for the security of the proposed schemes.

**Lemma 6.4.1 (Authentication (DAGs))** Under the random oracle model, any authenticity violation of a graph can be detected by using the structural signatures for DAGs.

**Proof** [Sketch] Let $\xi_{G_\delta}^{V', E'}$ and $\sigma_{G_\delta}^{V', E'}$ be the authentication units received by a Bob, who receives the $G_\delta^{V', E'}$, subgraph of $G^{V, E}$. Suppose that the authenticity verification scheme in Section 6.2.3 authenticates a graph, $G''_\delta^{V'', E''}$ different from $G_\delta$, to $G$, using $\xi_{G_\delta}^{V', E'}$ and $\sigma_{G_\delta}^{V', E'}$. By Definitions of $\chi$-RONs, $\tau$-PONs and $\tau$-RONs, the relationship between the nodes in $G''_\delta$ must be identical to that of $G'_\delta$ (Section 6.2.3), otherwise the randomized traversal numbers are not secure, which is a contradiction. Contents of $G''_\delta$ would be authenticated iff Verification of $G''_\delta$ is satisfied, which implies that the random oracle $H$ incurred a collision, and the CRSA/aggregate signature scheme and $H$ are not secure, which is a contradiction, under the random oracle hypothesis [76] and hardness of computational Diffie-Hellman problem [28]. Thus the lemma is proven.

**Lemma 6.4.2 (Non-leakage (DAGs))** Under the random oracle model, the structural signature of a DAG do not lead to any leakage of any extraneous information.

**Proof** [Sketch] A user receiving a sub-DAG $G'_\delta^{V', E'}$ of DAG $G$ should not be able to infer any extraneous information of $G$ from (1) the signature $\sigma_G$, (2) set of verification units $\mathcal{V}_{G_\delta}$, and (3) the $\tau$- and $\chi$-structural positions of the nodes in $G'_\delta$. Due to the properties of Bilinear maps and aggregate signatures, (1) and (2) do not leak any information (i.e., the probability of leakage is negligible) [28]. We now would prove that (3) does not leak.

(Forward-edges:) The $\tau$-structural positions of nodes in a forward-edge are identical to the rules of tree-edge (Definition 6.1.1); therefore an edge $e(x, y)$ that is a forward-edge in $G$
but not a forward-edge in $G'_δ$ would be verified by the user as a tree-edge. (Cross-edges:)

By Definitions 6.2.2 and 6.2.4, a $χ$-node $x$ has a structural position such that $r^χ_x$ is larger than the $r^τ_w$ for each $w$ that has a cross-edge incident on $x$. By Definition 6.1.1, the user would only know that $e(w, x)$ is a tree-edge and thus cannot learn whether it is a cross-edge (Definition 6.1.1). Thus the lemma is proved. If the signature of the DAG and the integrity verifier of the subgraph leak extraneous information, then $H$ incurred a collision, and the aggregate signature scheme and $H$ are not secure, which is a contradiction, under the random oracle hypothesis [76] and hardness of computational Diffie-Hellman problem [28]. Thus the lemma is proven.

Lemma 6.4.3 (Authentication (Graphs with Cycles)) Under the random oracle model, any authenticity violation of a graph can be detected by using the structural signature scheme for connected directed graphs.

Proof [Sketch] Let $G(V, E)$ be a directed connected graph and $x$ be a node in it.

Content authenticity: Any compromise of the content $c_x$ or the structural position (either $τ$, $χ$, or $β$) of a node $x$ in $G$ would invalidate the structural integrity verifier $ξ_x$, which is a hash of a message that contains $c_x$ and structural position of $x$ as defined by Definitions 6.2.4, and 6.3.6, unless the hash function $H$ encounters a collision, which contradicts our assumption and the random oracle hypothesis. Moreover, if a node/edge is added to or dropped from the received subgraph $G'_δ(V', E')$, it would invalidate the received integrity verifier of the subgraph $ξ_{G'_δ(V', E')}$. If $ξ_{G'_δ(V', E')}$ does not get invalidated, then either $H$ is not a random oracle or the CRSA/BGLS scheme is not secure – both are contradictions.

Structural authenticity: The signature of a node in a graph is a forward-signature, if it is not reachable from a back-edge. Such a signature is with respect to the DFT obtained from a forward traversal of the graph. Any unauthorized re-ordering between two or more nodes (violation of structural integrity) in such a DFT can be detected using the randomized traversal numbers. If a node is a $β$-node or is reachable from a back-edge, then such a node also belongs to the DFT obtained from a depth-first traversal carried out from $β$-node(s) over back-edges. Any unauthorized re-ordering would be detected here as well. Suppose $x$ belongs to another tree $G'$, but claimed to belong to $G$. By the arguments similar to the one in the proof of Lemma 6.4.1, it is not possible.
Lemma 6.4.4 (Non-leakage (Graphs with Cycles)) Under the random oracle model, the structural signatures for connected directed graphs do not lead to any leakage of any extraneous information.

Proof [Sketch] A user receiving a subgraph $G'_δ(V', E')$ of graph $G$ should not be able to infer any extraneous information of $G$ from (1) the signature $σ_G$, (2) set of verification units $V_C G'_δ$, and (3) the $τ$, $χ$, and $β$-structural positions of the nodes in $G'_δ$. Due to the properties of Bilinear maps and aggregate signatures, (1) and (2) do not leak any information (i.e., the probability of leakage is negligible) [28]. We now would prove that there is no leakage due to (3).

In order to be confidentiality-preserving, an authentication scheme must not leak any extraneous information about (a) cross-edges, (b) forward-edges, (c) back-edges. Following Lemmas 6.4.2, the authentication scheme for connected directed graphs does not leak any information about (a) and (b). The scheme does not leak any information via the back-edges or $β$-structural positions proven as follows.

Let $e(x, w)$ be a not-$β$-covered back-edge in graph $G(V, E)$ (e.g., $e(g_{14}, g_{9})$ in Figure 3.4). By Definitions 6.3.4 and 6.2.4, the $β$-node $x$ has a $β$-structural position such that $p^β_x$ and $r^β_x$ are larger and smaller than the $p^β_w x → w$ and $r^β_w x → w$, respectively. By Definition 6.1.1, the user would only verify that $e(x, w)$ is a tree-edge and cannot learn from it whether $e(x, w)$ is a back-edge.

The following cases arise when a user is authorized to access a connected subgraph that includes a back-edge $e(x, w)$, which is not-$β$-covered in the subgraph; the subgraph may also include: (I) nodes reachable from $w$, (II) nodes reachable from $x$ and (III) nodes $x$ is reachable from.

Case I: Nodes reachable from $w$: The user is authorized to access the back-edge $e(x, w)$ and the edge $e(w, y)$ (e.g., $e(g_{9}, g_{11})$ in Figure 3.4). By Definition 6.3.2, $y$ is $β$-reachable from $x$. Further by Definitions 6.3.4 and 6.2.4, $p^β_x x → w$ and $r^β_x x → w$ are larger and smaller than the $p^β_y x → w$ and $r^β_y x → w$, respectively. By Definition 6.1.1, the user would only verify that an edge $e(w, y)$ is a tree-edge and cannot learn from it whether $e(x, w)$ is a back-edge.

Case II: Nodes reachable from $x$: The user is authorized to access the back-edge $e(x, w)$ and the edge $e(x, y)$ (e.g., $e(g_{14}, g_{18})$ in Figure 3.4).
• **y is not \(\beta\)-reachable and x is not \(\beta\)-reachable:** By sending \(\tau\)-position and \(\tau\)-signatures for both \(x\) and \(y\) and \(\beta\)-position and \(\beta\)-signatures for \(w\), the edges would be verified as tree-edges. This is because, \(p^\tau_x = p^\beta_{x \rightarrow w}\) and \(r^\tau_x = r^\beta_{x \rightarrow w}\) (Definition 6.3.4) and thus the \(\tau\)-signature for \(x\) are same as its \(\beta\)-signature.

• **y is not \(\beta\)-reachable and x is \(\beta\)-reachable:** Not possible by the definition of the notion of \(\beta\)-reachable.

• **y is \(\beta\)-reachable and x is not \(\beta\)-reachable:** By sending \(\tau\)-position and \(\tau\)-signatures for \(y\) and \(\beta\)-position and \(\beta\)-signature for \(x\) and \(w\), both the edges \(e(y, x)\) and \(e(x, w)\) are verified by the user as tree-edges, thus hiding the original type of \(e(x, w)\). This is because, by Definition 6.3.4, \(p^\tau_x > p^\beta_{x \rightarrow w}\) and \(r^\tau_x < r^\beta_{x \rightarrow w}\). Thus and by the properties of post-order and pre-order numbers, since \(e(y, x)\) is a tree-edge in \(G\), \(p^\tau_y < p^\beta_{x \rightarrow w}\) and \(r^\tau_y > r^\beta_{x \rightarrow w}\).

• **y is \(\beta\)-reachable and x is \(\beta\)-reachable:** By sending \(\beta\)-positions and \(\beta\)-signatures for both \(y\) and \(x\) and \(w\), the edges would be verified as tree-edges. This is because, by Definition 6.3.4, \(p^\beta_{y \rightarrow w} > p^\beta_{x \rightarrow w} > p^\beta_{w \rightarrow w}\) and \(r^\beta_y < r^\beta_{x \rightarrow w} < r^\beta_{w \rightarrow w}\).

**Case III:** Nodes from which \(x\) is reachable: The user is authorized to access the back-edge \(e(x, w)\) and the edge \(e(y, x)\) (e.g., \(e(g_{11}, g_{14})\) in Figure 3.4). Four cases arise:

• **y is not \(\beta\)-reachable and x is not \(\beta\)-reachable:** Proof is identical to this condition in Case-II.

• **y is not \(\beta\)-reachable and x is \(\beta\)-reachable:** By sending \(\tau\)-positions and \(\tau\)-signatures for y and \(\beta\)-positions and \(\beta\)-signatures for \(x\) and \(w\), the edges would be verified as tree-edges. This is because, for an \(x\) that is \(\beta\)-reachable and a \(\beta\)-node, \(p^\beta_{x \rightarrow w} < p^\tau_x\) and \(r^\beta_{x \rightarrow w} > r^\tau_x\). Since \(e(y, x)\) is an edge in the graph, \(p^\tau_x < p^\tau_y\) and \(r^\tau_x > r^\tau_y\).

• **y is \(\beta\)-reachable and x is not \(\beta\)-reachable:** Proof is identical to this condition in Case-II.

• **y is \(\beta\)-reachable and x is \(\beta\)-reachable:** Proof is identical to this condition in Case-II.

The above arguments can be easily extended to multiple **not-\(\beta\)-covered** back-edges in the graph. Consideration of one **not-\(\beta\)-covered** back-edge takes care of all other back-edges.
that are it covers. So the argument also extends to multiple $\beta$-covered and not-$\beta$-covered back-edges.

Suppose that a user Bob has access to $G'_\delta$, a subgraph in $G$. Bob receives the structural signature of $G$, the node signatures of $G'_\delta$ and their structural positions. Any leakage would be a direct leakage through these information or an inference from them.

**Direct leakage:** Clearly (as per Definitions 6.2.7, 6.2.5, 6.3.9 and 6.3.7, and the protocols specified in Sections 6.2.2 and 6.3.2) Bob does not need the integrity verifier of any node $u$ that is in $G$ but not in $G'_\delta$. He therefore does not need to know any of the structural relationships and ordering that exist in $G$, but not in $G'_\delta$. Therefore none of (3), (4), (5) and (6) is directly leaked to Bob; he does not learn any extra information from the authentication process.

**Indirect leakage through the signature of the graph and integrity verifier for $G'_\delta$:** Under the Random Oracle Hypothesis and the hardness of Computational Diffie-Hellman problem, the structural signature of the tree reveals neither (3) the signature of $u$, nor (4) the existence of $u$. Similarly, the structural signature of a node leaks neither (3) nor (4). Therefore (5) – the structural relations (edges or paths) and (6) – the structural order among nodes in $G'_\delta$ and $u$ are also not revealed by the signatures.

In addition, the structural positions of the nodes in $G'_\delta$ do not reveal any information [84]: because the probability of inference (and leakage) about (2) – the existence of node $u$ between two immediate siblings from such randoms is negligible. Therefore structural positions of nodes cannot be used to determine the structural signature of $u$. Since (3) and (4) cannot be inferred from the RPON’s and RRON’s, (5) and (6) also cannot be inferred from the structural position of a node.

6.5 Complexity and Performance Results

In the following sections, we analyze the complexity and performance of the proposed schemes.
6.5.1 Complexity

Signature Generation Complexity. The pre-order and post-order numbers can be generated by a single traversal of the graph $G(V, E)$. The traversal complexity is thus $O(|V| + |E|)$. In the signing scheme of graphs (Section 6.3.1), a graph is traversed once forward, a number of times backward, and once for computing the signatures in the end. The number of backward traversals is the number of not-$\beta$-covered back-edges $d$, which is in $[0, |V| - 1]$ (0 in case of DAGs and $(|V| - 1)$ in case of complete graphs). In case of signing DAGs, the scheme requires only two traversals (no backward traversal is required here). Therefore, the complexity for signing a DAG is $O(|V| + |E|)$. In case of graphs with cycles, the scheme requires $d$ number of backward traversals, where $d$ is the number of not-$\beta$-covered back-edges.

The number of authentication items that need to be computed, and stored is $(|V|*(2 + 2d))$ as explained below by assuming that the sizes of the secure random number and the cryptographic hash are identical. In the worst case (when the graph is a complete graph), each node is a $\beta$-node; so each node has one forward and $d$ backward positions. Each structural position involves two secure random numbers. Therefore, the storage complexity of structural signatures is $O(|V|*d)$.

Distribution Complexity. Let the signed subgraph that needs to be shared with a user be $G_\delta'(V', E')$. In the worst case ($G_\delta'(V', E')$ is a complete graph), the distributor has to send $(2 * |V'|)$ number of signature items as follows. Each node in such a subgraph is a $\beta$-node; thus the $\beta$-structural position (two authentication items) of each node is sent to the user. Therefore the distribution complexity of structural signatures is $O(|V'|)$.

Integrity Verification Complexity. The procedure for the verification of content integrity incurs a cost linear in terms of the size of the subgraph $G_\delta'(V', E')$ received, that is, $O(|V'| + |E'|)$. It accounts for one hashing for each node. The cost of verification of structural integrity is also linear: $O(|V'| + |E'|)$, as the cost of comparison of randomized traversal numbers is constant.
6.5.2 Performance Results

We implemented the two schemes - DAG and graphs in Java 1.6 and JCA 6.0 (Java Cryptography Architecture) APIs. The experiments were carried out on a desktop with the following specification: 64-bit Linux (Ubuntu 8.10) on Intel Core 2 Duo CPU with 4GB RAM. For signing using aggregate signatures and bilinear maps, we used the C implementation of bilinear maps [3]. The performance metrics measured are: time to compute the $\chi$-, $\tau$-structural positions and $\beta$-structural positions (for graphs) versus the number of cross-edges, back-edges in the DAG and the graphs. The other performance metric that we have evaluated the schemes against is time taken for signing and verification.

Time to Compute Structural Positions: The process of computing appropriate structural positions of each node in either a DAG or a graph is part of the one-time process of signing. Figure 6.13 shows that the time for computation of $\chi$- and $\tau$-structural positions for DAGs is very efficient (linear). For DAGs of size about 2 million nodes, it takes only 14 seconds. Figure 6.14 shows that the time to compute all the structural positions for all the nodes in a graph with respect to the number of
back-edges in the graph is also highly efficient - for more than 0.5 million back-edges, it takes 6.5 seconds.

The performance of signing a graph, redaction and verification of a subgraph is same as that of the trees. This is primarily because computation of various structural positions in case of a graph is what is the major difference between trees and graphs. However such a cost as shown in the previous paragraph is almost negligible in comparison to the cost of signing, which is in thousands of seconds. Such high cost of signing and verification is due to involvement of modular exponentiations. Our performance results are in terms of the number of integrity verifiers; in case of trees, the number of integrity verifiers per node is one, whereas in a graph, it is more especially for nodes involved in cycles.

6.6 Summary

In this chapter, we proposed two schemes in order to address the problem of leakage-free authentication - one for DAGs and another for graphs with cycles. The proposed schemes are based on structure of graphs and aggregate signatures.
A graph contains nodes and tree-edges, forward-edges, cross-edges and back-edges. The fact that an edge is a forward-, cross- or a back-edge in a depth-first traversal conveys information that is sensitive in several contexts. Moreover, back-edges pose a more difficult problem than the one posed by forward- and cross-edges (i.e. DAGs) primarily because back-edges add a bidirectional nature to the graph, which turns graphs into complex structures, which in turn makes the problem of authentication with confidentiality of such structures a hard one.

Our schemes prevent leakage of information while facilitating authentication of content as well as the structure of trees and graphs. The security of such schemes are based on the security of cryptographic hash functions (random oracles) and aggregate signatures (Computational Diffie-Hellman problem). In terms of complexity, the schemes for DAGs and graphs with cycles incur $O(n)$ and $O(n \times d)$, respectively, for time and storage ($n$ is the number of nodes in the graph). Performance of our schemes on large DAGs and on graphs as large as 0.5-million back-edges nodes show that our scheme is efficient. Further, experimental results show that our technique for graphs with cycles performs linearly – better than the complexity analysis.

**Limitations of Structural Signatures** Structural signature schemes for trees and graphs are however not secure when the user receives more than one subtrees/sub-graphs of a tree/graph. A user can infer about the existence of path relationship
between nodes that are different subtrees. In Figure 6.15, we show that a user receives first and second subtrees as the results of query-1 and query-2, respectively. As part of this process, the user also receives the randomized post-order and pre-order numbers of these nodes. Since post-order (pre-order) number of an ancestor is larger (resp., smaller) than that of the descendant. In this example, the post-order (pre-order) number of $g_2$ is greater (resp., smaller) than that of $g_6$, which why, the user can infer that there exists a path between between these two nodes in the original trees. This maybe a sensitive information in many scenarios, such as, healthcare (disease is denoted by $g_2$ and patient is referred to by $g_6$), in finance (a specific loan is denoted by $g_2$ and an individual is referred to by $g_6$).

In the next chapter, we have presented a scheme that is leakage-free even when a user has access to multiple subtrees/subgraphs.
The structural signature scheme described in the previous chapters can authenticate only “single” subtrees, and will leak when “multiple” subtrees are results of one or more queries. Moreover, structural signatures compute a linear number of signatures for the trees and at least a linear number of signatures for graphs. In this chapter, we present a scheme that computes only one signature for a tree/graph/forest, is generic for trees, graphs, and forests, and can be used to authenticate multiple subtrees/subgraphs without leaking.

7.1 Trees

In this section, we propose a leakage-free signature scheme $r\Pi$ for trees. It relies on the notion of “secure names” that are assigned to the nodes in a tree.

The purpose of secure names is to convey the order of siblings (which node is to the left of which other node) without leaking anything else (e.g., whether they are adjacent siblings, how many other siblings are between them, etc). For example, in Figure 1.1(a), $a, b,$ and $c$ are siblings such that $a \prec b \prec c$. Secure names $\eta_a, \eta_b,$ and $\eta_c$ are assigned to $a, b,$ and $c$, respectively. Given $\eta_a$ and $\eta_c$, along with $a$ and $b$, a user can establish the fact that $a \prec c$. But it cannot learn anything about $b$, or its existence (extraneous information).

The signing procedure traverses a tree $T(V, E)$ bottom-up, and assigns an $N$-bit secure name $\eta_x$ to each node $x$ in the tree, and then computes the signature $\sigma_T$ of the tree using these secure names. Using the secure names of the nodes in a tree,
an “integrity verifier” for each node is computed, which in turn is used to define the signature of the tree $\sigma_{T(V,E)}$. A user that receives a subtree $T_\delta$ also receives the signature of the tree, and a verification object ($\mathcal{VO}$) in order authenticate the integrity and the origin of this subtree. $\mathcal{VO}$ is computed using the integrity verifiers of those nodes that are not in the subtree $T_\delta$. The user verifies the signature of the tree using the $\mathcal{VO}$ and the received subtree together. The structural relationships between the nodes and the order between the siblings in $T_\delta$ are verified using their secure names.

In the following sections, we have shown how to compute the signature of a tree, and distribute, and how to authenticate a subtree. To ease the exposition, we first introduce a preliminary approach for naming the nodes, which is easier to understand and is secure but is not efficient - it has an exponential complexity. We then present the efficient solution that is both secure and efficient for trees with large branching factor.

7.1.1 Preliminary Scheme (Scheme-1)

Our approach for generating secure names follows a bottom-up strategy. Let $v_1, \ldots, v_k$ be a list of siblings listed in left to right order. Let $\text{lsb}(s)$ denote the least significant bit of the bit-string $s$. The secure names of siblings $v_i$ and $v_{i+1}$ are computed such that the least significant bits of the hash of $\eta_{v_i} \parallel \eta_{v_{i+1}}$ and the hash of $\eta_{v_{i+1}} \parallel \eta_{v_i}$ are 1 and 0, respectively. We call this as the ordering property of secure names. This scheme is given in Figure 7.1.

Example: In the tree in Figure 1.1(a), $N$-bit secure names $\eta_a$, and $\eta_b$, are assigned to $a$, and $b$, respectively. $\eta_a$ is assigned as a random. $\eta_b$ is computed such that $\text{lsb}(\mathcal{H}(\eta_a \parallel \eta_b)) = 1$ and $\text{lsb}(\mathcal{H}(\eta_b \parallel \eta_a)) = 0$. This process is repeated for each set of siblings.
**rNameGen**: Compute the secure names for siblings \(v_1, v_2, \ldots, v_k\), children of node \(x\), where \(v_i \prec v_j, i < j\).

1. For the root node \(\text{root}\) of \(T\), assign a random to \(\eta_{\text{root}}\).

2. Repeat the following statements for each \(x \in V\). Let \(v_1, v_2, \ldots, v_k\) be the set of the children of \(x\).

3. Generate a random permutation \(\pi\) of the integers \(\{1, \ldots, k\}\).

4. Set \(\eta_{v_{\pi(1)}}\) to be any random.

5. For \(i = 2, \ldots, k\), compute \(\eta_{v_{\pi(i)}}\) as follows.

   (a) Choose a random \(r\).

   (b) For \(j = 1, \ldots, i - 1\), do the following:

      i. \(\lambda_\prec \leftarrow \mathcal{H}(\eta_{v_{\pi(j)}} \parallel r)\).

      ii. \(\lambda_\succ \leftarrow \mathcal{H}(r \parallel \eta_{v_{\pi(j)}})\).

      iii. If \(v_{\pi(i)}\) is to the left (resp., right) of \(v_{\pi(j)}\), then check whether \(\text{lsb}(\lambda_\prec)\) is 1 (resp., 0) and \(\text{lsb}(\lambda_\succ)\) is 0 (resp., 1).

      iv. If the answer is “yes” for all \(j\), then \(\eta_{v_{\pi(i)}} \leftarrow r\).

      v. Else go back to Sub-step 5(a).

**rNameVrfy**: Verify the order between two nodes \(v_i \prec v_j\), using their secure names \(\eta_{v_i}\) and \(\eta_{v_j}\).

1. \(v_i \prec v_j \iff \text{lsb}(\mathcal{H}(\eta_{v_i} \parallel \eta_{v_j})) = 1 \land \text{lsb}(\mathcal{H}(\eta_{v_j} \parallel \eta_{v_i})) = 0\).

Figure 7.1. Algorithm to compute secure names for \(T(V, E)\) (Scheme-1).
Figure 7.2. Secure names $\eta V_i$ and $\eta_x$ of siblings $V_i$ and $x$ in the context of the efficient naming scheme.

Complexity

Scheme-1 takes $O(n)$ time, where $n$ is the number of nodes in the tree. The probability that a particular choice of $r$ is found suitable for $\eta_{\pi(i)}$ is $4^{-i+1}$, and the average number of $r$ values generated for the selection of such an $\eta_{\pi(i)}$ is $4^{i-1}$. The expected time to compute the secure names all $k$ siblings is therefore: $\sum_{i=1}^{k} 4^{i-1} = (4^k - 1)/3$, and the average time to compute all secure names is $(n - \ell) * (4^k - 1)/3$. Note that, although it is quite unlikely to happen, it is nevertheless possible that two non-sibling nodes receive the same secure name. In such a case, step 5(a) should be repeated.

7.1.2 Efficient Scheme (Scheme-2)

The main drawback of the Scheme-1 is the fact that the worst-case time to compute an $\eta_x$, when $x$ is the $(j + 1)$'th leftmost child of its parent, is exponential in $j$ (Step 3 in Section 7.1.1). This section describes an improved scheme that does not suffer from this drawback. As earlier, a non-leaf node in a tree has $k$ number of children.
7.1.3 Secure Names

The idea is, as before, to compute the $\eta_x$'s (secure names) bottom-up and, within a set of siblings, in left-to-right order. The main difference is how a secure name $\eta_x$ is computed. This scheme (Scheme-2) is given in Figure 7.3.

In this approach, we split the $N$-bit long secure name $\eta_x$ of a node $x$ into two disjoint parts: $\eta^l_x$ and $\eta^r_x$ of sizes $L$ and $R$ refer to the left and right parts of $\eta_x$, respectively (Figure 7.2). If $x$ is the leftmost child (i.e., the first child) of its parent then $\eta_x$ is selected randomly. If $x$ is $(m+1)$’th leftmost child of its parent, then $\eta_x$ depends on the secure names of its left siblings. Let $w$ be a left sibling of $x$: $w < x$. Two bits ($b_w$ and $b'_w$) in $\eta^l_w$ and two bits ($b_x$ and $b'_x$) in $\eta^l_x$ are selected and their values are set such that $b_w \oplus b_x < b'_w \oplus b'_x$. $b_w$ and $b_x$ are the $j$’th leftmost bit in $w$ and $x$, respectively, where $j$ is computed using $\eta^r_w$ and $\eta^r_x$ in this order (because $w < x$). Similarly $b'_w$ and $b'_x$ are the $(j')$’th leftmost bit in $w$ and $x$, respectively, where $j'$ is computed using $\eta^r_w$ and $\eta^r_x$ in the reverse order. This is the ordering property of secure names computed in this fashion. Alongwith $N$, $(L - 2 \times k)$, and $R$ are sufficiently large as security parameters.

**Example:** For $N = 512$, choose $L=360$ for $k \leq 100$, and $R=152$ in the context of current computational power. In the tree in Figure 1.1(a), secure names $\eta_a$, and $\eta_b$, are assigned to $a$, and $b$, respectively. $\eta_a$ is an $N$-bit random and each bit of $\eta^l_a$ is marked as not-used. $\eta_b$ is computed as follows. $\eta^r_b$ is an $R$-bit random and $\eta^l_b$ is initialized to 0. Each bit of $\eta^l_b$ is marked as not-used. $j$ is computed as $(1 + \mathcal{H}(\eta^r_a \parallel \eta^r_b) \mod L)$. Since $j$’th leftmost bits of $\eta^l_a$ and $\eta^l_b$ referred to as $b_i$ and $b$, respectively, are marked as not-used, $j'$ is computed as $(1 + \mathcal{H}(\eta^r_b \parallel \eta^r_a) \mod L)$. If $(j \neq j')$ and the $(j')$’th leftmost bits of $\eta^l_a$ and $\eta^l_b$ referred to as $b'_i$ and $b'$, respectively, are marked as not-used, then proceed as follows. Assign either $(0,0)$ or $(1,1)$ to $(b_i,b'_i)$ in $\eta^l_a$, and $(0,1)$ or $(1,0)$ to $(b,b')$ in $\eta^l_b$. Such an assignment assures that $b_i \oplus b(= 0) < b'_i \oplus b'(= 1)$. The $j$’th and $(j')$’th bits of $\eta_a$ and $\eta_b$ are marked as used. $\eta_c$ depends on both $\eta_a$ and $\eta_b$. This process is repeated for each set of siblings.
**rNameGen**: Compute the secure name for \( x \) such that \( v_1, v_2, \ldots, v_k, x \) are siblings, where \( v_i \prec v_j \prec x, i < j \).

1. Choose a sufficiently large \( N \). Choose \( L \) and \( R \) are such that (a) \( N = L + R \), (b) \( R \geq \log(L) \), and (c) \( L - 2 \ast k \) and \( R \) are sufficiently large as security parameters. Let \( \eta^r_x \) and \( \eta^l_x \) refer to the right-part consisting of \( R \) and left-part consisting of \( L \) bits of the secure name \( \eta_x \) of \( x \).

2. Assign random values to \( \eta^r_x \), and zero values to \( \eta^l_x \). Associate with each bit of \( \eta^l_x \) a status that is initially set to *not-used*.

3. Compute secure names of siblings in the left-to-right order of the siblings. Let \( v_1, \ldots, v_k \) be the siblings to the left of \( x \), where \( v_i \) is the \( i \)th leftmost one. (Each of the \( \eta_{v_i} \)'s of these \( k \) siblings of \( x \) have already been computed.)

4. For \( i = 1 \) to \( k \) do the following.
   
   (a) \( \lambda_\prec \leftarrow H(\eta^r_{v_i} \parallel \eta^r_x); \ j \leftarrow 1 + (\lambda_\prec \mod L) \). Let \( b_i \) (resp., \( b \)) denote the \( j \)th leftmost bit of \( \eta^l_{v_i} \) (resp., \( \eta^l_x \)). If the status of \( b \) is *not-used* then continue with the next step, else go back to step (2).

   (b) \( \lambda_\succ \leftarrow H(\eta^r_x \parallel \eta^r_{v_i}); \ j' \leftarrow 1 + (\lambda_\succ \mod L) \). Let \( b'_i \) (resp., \( b' \)) denote the \( (j') \)th leftmost bit of \( \eta^l_{v_i} \) (resp., \( \eta^l_x \)).

   (c) If \( j \neq j' \) and the status of \( b' \) is *not-used* then proceed to the next step, else go back to step (2).

   (d) Set \( b \) and \( b' \) such that \( b_i \oplus b < b'_i \oplus b' \).

   (e) Change the status of \( b \) and \( b' \) from *not-used* to *used*.

Figure 7.3. Efficient algorithm to compute secure names for tree \( T(V, E) \) (Scheme-2) (Continued to Figure 7.4).
**rNameVrfy**: Verify whether $y \prec z$, using their secure names $\eta_y$ and $\eta_z$.

1. $j \leftarrow 1 + (\mathcal{H}(\eta_y \parallel \eta_z) \mod L)$
2. $j' \leftarrow 1 + (\mathcal{H}(\eta_z \parallel \eta_y) \mod L)$
3. Let $b_y$ and $b_z$ be the $j$’th, and $b'_y$ and $b'_z$ are the $(j')$’th bits in $\eta_y$ and $\eta_z$, respectively.
4. Check the following: $y \prec z \iff b_y \oplus b_z < b'_y \oplus b'_z$.

Figure 7.4. Efficient algorithm to compute secure names for tree $T(V, E)$ (Scheme-2) (Continued from Figure 7.3).
Complexity

The above algorithm translates into a simple and constant-time test of which of two given siblings is to the left of the other. But we need to analyze the expected number of re-starts. Suppose that the size $L$ of the left part ($\eta^L$) of a secure name is 500. The probability of a “collision” and re-start at Step (1) is the probability that $2^k$ numbers drawn randomly from the 500 choices $[1,500]$ are not all distinct, i.e., that at least 2 of them are equal. This is the classic birthday problem, and the probability of a re-start is (assuming $2^k \leq 500$): 
$$1 - \prod_{j=1}^{2^k-1} \left\{ (1 - (j/500)) \right\} \approx 1 - e^{-\left(\frac{(2^k)(2^k-1)}{1000}\right)}.$$ For $2^k = 50$ this probability is 0.91, hence the expected number of re-starts is $(1/(1 - 0.91)) = 11$, which is much better than the preliminary scheme where the expected number of re-starts would have been proportional to $4^{25}$. Scheme-2 incurs linear cost $O(n)$ in terms of the number of nodes in the tree.

7.1.4 Leakage-Free Signatures of Trees (rSign)

In this section, we describe the signature, distribution and verification protocols for trees. Prior to computing the signatures, a dummy node is inserted by splitting an edge: if $e(x,y)$ is an edge in the original tree, add a node $w$ such that $e(x,w)$ and $e(w,y)$ are the new edges in the modified tree. Secure name $\eta_w$ of each inserted node $w$ is a random. Such node $w$ when given to a user only when the user has access to both $x$ and $y$. The ordering between them is not needed to be verified by secure names.

Integrity Verifiers

An integrity verifier ($IV$) of a node is the hash of the secure name of its parent, its secure name and its contents. In case of inserted nodes, no contents is used in $IV$. Using the $IV$s, we define a signature $\sigma_{T(V,E)}$ (also referred to as $\sigma_T$) for $T(V,E)$. In cases when “the received subtree (sent to the user) is the same as the original tree” is
a sensitive information, the signature of a tree may be salted using a random value in order to protect this fact. The (salted) tree signature is publicly available or passed to the user along with the subtree that the user has access to. $\sigma_{T(V,E)}$ is an aggregate signature, computed over the IVs of its nodes. We define two types of signatures for trees: one based on the condensed-RSA signatures [105] and the other based on bilinear maps [28].

**Definition 7.1.1 (Integrity Verifier)** Let $x$ be a node in tree $T(V, E)$, and $c_x$ be the content of node $x$. Its integrity verifier (IV) denoted by $\xi_x$, is defined as: $\xi_x \leftarrow H(\eta_x \| \eta_x \| c_x)$.

In this section, we define the signature of a tree based on Condensed-RSA signature scheme [105] and aggregate signatures [28].

**Definition 7.1.2 (Signature of Trees using CRSA)** Let $T(V, E)$ be a tree. Let $H$ denote a random oracle. Let the salt be $\omega_T$. The signature $\sigma_x$ of each node $x$ is defined as $\sigma_x \leftarrow \xi_x^d \mod \bar{n}$, where $\xi_x$ is the IV of $x$. Let the salt be $\omega_T$ be a random, and let $\Omega_T \leftarrow \omega_T^d \mod \bar{n}$. The signature of $T$, denoted by $\sigma_T$, is defined as

$$\sigma_T = \Omega_T \prod_{x \in V} \sigma_x \mod \bar{n}. \quad (7.1)$$

**Definition 7.1.3 (Signature of Trees using BGLS)** Let $T(V, E)$ be a tree. Let $H$ denote a random oracle. Let the salt be $\omega_T$. The signature $\sigma_x$ of each node $x$ is defined as $\sigma_x \leftarrow sk \xi_x$. The signature of $T$, denoted by $\sigma_T$, is defined as

$$\sigma_T \leftarrow (P, sk(\omega_T + \sum_{x \in V} \xi_x)). \quad (7.2)$$

7.1.5 Distribution (rRedact)

The distributor $D$ sends the following items to Bob, who has access to $T_\delta(V_\delta, E_\delta)$, a subtree of tree $T(V, E)$: $(T_\delta(V_\delta, E_\delta), \mathcal{VO}_{T_\delta}, \sigma_T)$, where $\mathcal{VO}_{T_\delta(V_\delta, E_\delta)}$ (also referred to as $\mathcal{VO}_{T_\delta}$) is the verification object of $T_\delta$, and $\sigma_T$ the signature of the $T(V, E)$.
Figure 7.5. Algorithm to sign a tree.

The following steps show how to compute $\mathcal{VO}_{T_δ}$. $D$ computes two collective integrity verifiers $σ_{T_δ}$ and $Δ_{T_δ}$ as part of $\mathcal{VO}_{T_δ}$ over the integrity verifiers of all the nodes that are not in the subtree and also includes the salt.

$\mathcal{VO}$ is used to verify the signature of the tree, and is used to detect if any node(s) has been dropped from $T_δ$ in an unauthorized manner. $σ_{T_δ}$ is used to verify the signature of all the nodes in the subtree in an aggregate manner, and is used to detect if any node(s) has been injected from $T_δ$ in an unauthorized manner. $η_x$ is the secure name of $x$.

7.1.6 Authentication ($rVrfy$)

Bob receives the subtree $T_δ(V_δ, E_δ)$, the signature of the tree $σ_T$, and the verification object $\mathcal{VO}$. As part of the content authentication process, Bob computes the
**rRedact**: Computed signature of the redacted subtree $T_\delta(V_\delta, E_\delta) \subseteq T(V, E)$.

1. $\sigma'_{T_\delta} \leftarrow \langle \sigma_{T_\delta}, \mathcal{V}\mathcal{O}, \Theta_{T_\delta} \rangle$, computed as follows.

2. $\Theta_{T_\delta}$ is the set of all secure names of the nodes and their respective parents in $T_\delta$: $\Theta_{T_\delta} \leftarrow \{ (\eta_x, \hat{\eta}_x) | x \in V_\delta \}$.

3. Compute the collective integrity verifier $\mathcal{V}\mathcal{O}$ as follows.

   (a) CRSA: $\mathcal{V}\mathcal{O} \leftarrow \omega_T \prod_{x \in (V - V_\delta)} \xi_x \ mod \ \bar{n}$;
       $\sigma_{T_\delta} \leftarrow \prod_{x \in V_\delta} \sigma_x \ mod \ \bar{n}$.

   (b) BGLS: $\mathcal{V}\mathcal{O} \leftarrow \omega_T + \sum_{x \in (V - V_\delta)} \xi_x$;
       $\sigma_{T_\delta} \leftarrow (P, \sum_{x \in V_\delta} \sigma_x)$.

Figure 7.6. Algorithm to redact a subtree.

integrity verifiers of the nodes in $V_\delta$ and combines them with the received collective integrity verifier $\mathcal{V}\mathcal{O}$. If the contents of the nodes are valid, the structural integrity is verified with the help of secure names: the parent-child relationship, and the order among the siblings. Authentication of contents and structural positions of the subtree received includes (1) verification of integrity and, (2) verification of the source of the subtree. The integrity verification of structural relations in a tree involves traversing the tree and using the secure-name of two siblings of its parent or its sibling. The user can carry out verification of integrity of a $n'$-node subtree in $O(n')$-time. The verification procedure is given in Figure 7.7.
rVrfy: Verify authenticity of subtree $T_\delta(V_\delta, E_\delta)$.

**Authentication of nodes:**

1. For each node $y \in V_\delta$, compute $\xi_y \leftarrow H(\eta_{\hat{p}} \| \eta_y \| c_y)$.
2. CRSA: Verify (a) and (b):
   - (a) $(\sigma_{T_\delta})^e \equiv \prod_{y \in V_\delta} \xi_y \pmod{\bar{n}}$, and
   - (b) $(\sigma_T)^e \equiv \nuO \prod_{y \in V_\delta} \xi_y \pmod{\bar{n}}$.
3. BGLS: Verify (a) and (b):
   - (a) $(\sigma_{T_\delta}) \equiv (Q, \sum_{y \in V_\delta} \xi_y)$, and
   - (b) $(\sigma_T) \equiv (Q, (\nuO + \sum_{y \in V_\delta} \xi_y))$.
4. If (a) and (b) are valid, then the contents and secure names of $T_\delta$ are authenticated. Otherwise, if (b) is invalid and (a) is valid, then the received nodes are authenticated, but either some nodes have been dropped, or $\nuO$ and/or $\sigma_T$ have been tampered with.

**Verification of edges and ordering among siblings:**

1. Carry out a depth-first traversal on $T_\delta$.
2. **Parent-child relationship:** Let $x$ be the parent of $y$ in $T_\delta$; if $(\eta_x \neq \eta_{\hat{p}})$, then this relationship is incorrect.
3. **Order among siblings:** For ordered trees, in $T_\delta$, let $y$ and $z$ are children of $x$, and let $y < z$.
   - (a) For scheme-1 (Section 7.1.1): $y < z \iff (\text{lsb}(H(\eta_y \| \eta_z)) = 1) \land (\text{lsb}(H(\eta_z \| \eta_y)) = 0)$.
   - (b) For scheme-2 (Section 7.1.2):
     - i. $j \leftarrow 1 + (H(\eta^+_y \| \eta^+_z) \mod L)$
     - ii. $j' \leftarrow 1 + (H(\eta^+_y \| \eta^+_z) \mod L)$
     - iii. $b_y$ and $b_z$ are the $j$’th, and $b'_y$ and $b'_z$ are the $(j')$’th bits in $\eta_y$ and $\eta_z$, respectively.
     - iv. $y < z \iff b_y \oplus b_z < b'_y \oplus b'_z$.

Figure 7.7. Algorithm to verify a subtree.
7.2 Graphs

Our proposed authentication scheme for graphs is a general one. It can be used for trees, DAGs, graphs with cycles as well as forests. However, for trees, it is recommended to use the scheme specifically developed for trees (in Section 7.1), especially when the fact that the data is organized as trees need not be kept as a secret from users. The scheme for trees computes only one signature and is more efficient than that for the graphs (described below). Moreover, as we will see later (Section 7.6.1), the scheme for graphs can be used for signing forests of trees and graphs as well as authenticate such forests in a leakage-free manner.

Consider a simple graph $G$ shown in Figure 1.1(b). It is a directed acyclic graph (DAG) with node $c$ having two immediate ancestors - $d$ and $h$ (this DAG can be turned into a cyclic graph, by adding a back-edge such as one from $f$ to $h$). Our solution for trees described earlier, does not work for graphs. In case of graphs, a node may have multiple incoming edges (i.e., multiple immediate ancestors such as $c$), whereas in case of trees, a node has only one parent (immediate ancestor) except for the root, which does not have any parent. Therefore, in the context of graphs, we cannot use the notion of integrity verifiers that is used for trees (Definition 7.1.1). The challenge in designing leakage-free signatures for graphs arises from the fact that the set $\alpha_{\delta}(x)$ of immediate ancestors of a node $x$ in a subgraph $G_{\delta}$ is a (possibly empty) subset of the set $\alpha(x)$ of immediate ancestors of $x$ in $G$. The question is how to verify the authenticity of $\alpha_{\delta}(x)$ without leaking any information about $(\alpha(x) \setminus \alpha_{\delta}(x))$: whether it is empty or non-empty, what is its size, etc? For example, $c$ has only $d$ as its immediate ancestor $G_{\delta}$, whereas it has $d$ and $h$ as the immediate ancestors in $G$. How to authenticate the fact that $d$ in fact is a correct immediate ancestor of $c$ in $G_{\delta}$, without leaking any information about $h$.

Ordering among siblings makes sense for graphs with cycles if and only if, the back-edge (or the edge which can be removed to break a given cycle among nodes) need to be of different semantics.
7.2.1 Leakage-Free Signatures for Graphs (\textit{rSign})

Our proposed scheme computes the integrity verifier of a node independent of the secure name of the parent, and integrity verifiers for edges. It computes secure names for nodes that have specific ordering with their siblings. If in a graph, the ordering between some siblings is not possible, then the secure names of such nodes are just randoms.

**Definition 7.2.1 (Integrity Verifier: Node)** Let $x$ be a node in graph $G(V, E)$, and $c_x$ be the content of node $x$. Its integrity verifier (IV) denoted by $\xi_x$, is defined as: $\xi_x \leftarrow H(\eta_x \| c_x)$.

**Definition 7.2.2 (Integrity Verifier: Edge)** Let $e(x, y)$ be an edge in graph $G(V, E)$. Its integrity verifier (IV) denoted by $\xi_{(x,y)}$, is defined as: $\xi_x \leftarrow H(\eta_x \| \eta_y)$.

Signature of a graph is then computed as the aggregate signature of the integrity verifiers of nodes and edges. Distribution is similar in the case of trees: if a user has access to a specific set of nodes and edges, signatures of the integrity verifiers of the edges and nodes as well as the secure names of the nodes are given to the user along with the nodes and edges. Also the aggregate signature of these IVs are given to the user along with the signature of the source graph and the verification object.

The signature scheme has a complexity of $O(|V| + |E|)$. It computes as many signatures as the number of nodes and edges in the graph (as in the case of trees) and another signature for the whole graph.

7.2.2 Distribution of Graphs (\textit{rRedact})

The distributor $D$ sends the following items to Bob, who has access to $G_\delta(V_\delta, E_\delta)$, a subgraph of graph $G(V, E)$: $(G_\delta, \mathcal{VO}_{G_\delta}, \sigma_G)$, where $\mathcal{VO}_{G_\delta}$ (also referred to as $\mathcal{VO}_{G_\delta}$) is a verification object, and $\sigma_G$ is the signature of $G$. 
**rSign**: Sign a graph \( G(V, E) \).

1. For each node \( x \in V \),
   (a) For each node \( x \), compute its secure name \( \eta_x \).
   (b) For each node \( x \), compute its integrity verifier
       \( \xi_x \leftarrow \mathcal{H}(\eta_x \| c_x) \); For each edge \( e(x, y) \), compute its integrity verifier
       \( \xi_{(x,y)} \leftarrow \mathcal{H}(\eta_x \| \eta_y) \).

2. Assign a salt \( \omega_G \) to \( G \).

3. If CRSA is used, compute the signature \( \sigma_G \):
   (a) For each \( x \in V \), \( \sigma_x \leftarrow (\xi_x)^d \mod \bar{n} \); For each edge
       \( e(x, y) \in E \), \( \sigma_{(x,y)} \leftarrow (\xi_{(x,y)})^d \mod \bar{n} \).
   (b) \( \Omega_G \leftarrow \omega_G^d \mod \bar{n} \). Compute
       \( \sigma_{G(V,E)} \leftarrow \Omega_G \prod_{x \in V} \sigma_x \prod_{e(x,y) \in E} \sigma_{(x,y)} \mod \bar{n} \).

4. If BGLS is used, compute \( \sigma_G \) as follows:
   \( \sigma_{G(V,E)} \leftarrow (P, sk(\omega_G + \sum_{x \in V} \xi_x + \sum_{e(x,y) \in E} \xi_{(x,y)})) \).

Figure 7.8. Algorithm to sign a graph.

**rRedact**: Computation of the redacted signature of \( G_\delta(V_\delta, E_\delta) \):

1. Compute \( \sigma_{G_\delta} \) and \( \Delta_{G_\delta} \) as follows.
   (a) CRSA: (a) \( \sigma_{G_\delta} \leftarrow \prod_{y \in V_\delta} \sigma_y \prod_{e(x,y) \in E_\delta} \sigma_{(x,y)} \mod \bar{n} \).
       (b) \( \Delta_{G_\delta} \leftarrow \omega_G \prod_{y \in V - V_\delta} \xi_y \prod_{e(x,y) \in E - E_\delta} \xi_{(x,y)} \mod \bar{n} \).
   (b) BGLS: (a) \( \sigma_{G_\delta} \leftarrow (P, \sum_{y \in V_\delta} \sigma_y + \sum_{e(x,y) \in E_\delta} \sigma_{(x,y)}) \).
       (b) \( \Delta_{G_\delta} \leftarrow \omega_G + \sum_{y \in V - V_\delta} \xi_y + \sum_{e(x,y) \in E - E_\delta} \xi_{(x,y)} \).

2. \( \Theta_{G_\delta} \leftarrow \{ \eta_x \mid x \in \} \); \( \mathcal{V} \mathcal{O}_{G_\delta} \leftarrow \{ \sigma_{G_\delta}, \Delta_{G_\delta}, \Theta_{G_\delta} \} \).

Figure 7.9. Algorithm to redact a subgraph.
**rVrfy**: Verification of authenticity of subgraph $G_δ(V_δ, E_δ)$

**Authentication of contents:**

1. For each node $x \in V_δ$ in a subgraph $G_δ(V_δ, E_δ)$, compute its integrity verifier: $ξ_x \leftarrow \mathcal{H}(η_x \parallel c_x)$.
2. For each edge $e(x, y)$, compute its integrity verifier: $ξ(x, y) \leftarrow \mathcal{H}(η_x \parallel η_y)$.
3. CRSA: Compute (a) $((\sigma_{G_δ})^e)^{2^p} = \prod_{x \in V_δ} ξ_x \pmod{\bar{n}}$ and, (b) $((\sigma_G)^e)^{2^p} = Δ_{G_δ} \prod_{x \in V_δ} ξ_x \pmod{\bar{n}}$.
4. BGLS: (a) $(\sigma_{G_δ})^{2^p} = (q, \sum_{x \in V'} ξ_x)$ and, (b) $(\sigma_G)^{2^p} = (q, Δ_{G_δ} + \sum_{x \in V'} ξ_x)$.
5. If (a) and (b) are valid, then the contents and secure names of $G_δ$ are authenticated. Otherwise, if (b) is invalid and (a) is valid, then the received nodes are authenticated, but either some nodes have been dropped, $Δ_{G_δ}$ and/or $σ_G$ have been tampered with. Parent-child relationship is verified during this process.

**Verification of ordering among siblings:**

1. Carry out a depth-first traversal on $G_δ$.
2. Order among siblings: In $G_δ$, let $y$ and $z$ are children of $x$, and let $y \prec z$.
   - (a) For scheme-1 (Section 7.1.1): $y \prec z \Leftrightarrow \text{lsb}(\mathcal{H}(η_y \parallel η_z)) = 1) \land (\text{lsb}(\mathcal{H}(η_y \parallel η_z)) = 0)$.
   - (b) For scheme-2 (Section 7.1.2):
     i. $j \leftarrow 1 + (\mathcal{H}(η_y \parallel η_z) \pmod{L})$
     ii. $j' \leftarrow 1 + (\mathcal{H}(η_y \parallel η_z) \pmod{L})$
     iii. $b_y$ and $b_z$ are the $j$’th, and $b'_y$ and $b'_z$ are the $(j')$’th bits in $η_y$ and $η_z$ respectively.
     iv. $y \prec z \Leftrightarrow b_y \oplus b_z < b'_y \oplus b'_z$.

Figure 7.10. Algorithm to verify a subgraph.
\( \Delta_{G_\delta} \) is used to verify the signature of the graph, and is used to detect if any node(s) has been dropped from \( G_\delta \) in an unauthorized manner. \( \sigma_{G_\delta} \) is used to verify the signature of all the nodes in the subgraph in an aggregate manner, and is used to detect if any node(s) has been injected from \( G_\delta \) in an unauthorized manner. \( \eta_x \) is the secure name of \( x \).

**Example:** \( D \) has to send \( G_\delta \) in our example to Bob. \( \sigma_{G_\delta} \) is a CRSA-signature. \( D \) computes the \( \Delta_{G_\delta} \) as a modular multiplication of the salt \( \omega_G \), and the integrity verifiers of \( f, g, \) and \( h \), because \( f, g, \) and \( h \) are not in \( G_\delta \). Now \( VO_{G_\delta} \) is the tuple consisting of \( \sigma_{G_\delta}, \Delta_{G_\delta} \) and a set consisting of an element for each node in \( G_\delta \). \( D \) then sends the signature of the graph \( \sigma_G \) and \( VO_{G_\delta} \) along with \( G_\delta \), to the user.

### 7.2.3 Authentication (rVrfy)

Bob receives the subgraph \( G_\delta(V_\delta, E_\delta) \), the secure name \( \eta_x \) of each node \( x \), verification object \( VO_{G_\delta} \), and the signature of the graph \( \sigma_G \). It verifies the authenticity of the contents; if they are authentic then the structural integrity is verified.

#### Authentication of the contents of subgraph \( G_\delta \)

By contents, we mean the contents of each node \( x \) as well as \( \eta_x \). In order to authenticate contents of \( G_\delta(V_\delta, E_\delta) \), Bob first computes the integrity verifiers \( \xi_x \) for each node, and then combines them appropriately with \( \Delta_{G_\delta} \) in order to verify the signature \( \sigma_G \). If the signature verifies, the edges and ordering among siblings are also verified. Authentication of contents of \( G_\delta(V_\delta, E_\delta) \) has a complexity of \( O(|V_\delta| + |E_\delta|) \).

**Example:** Bob computes the integrity verifiers of \( a, b, c \) and \( d \) in \( G_\delta \) in our example. Consider CRSA signatures. Bob computes a modular multiplication of these integrity verifiers together with \( \Delta_{G_\delta} \) received as part of \( VO_{G_\delta} \). Then Bob applies the signature verification process of CRSA on the result of this multiplication and the received signature \( \sigma_G \) of the graph. If the verification turns out to be valid, the contents are authenticated.
Authentication of the structural relationships

In order to verify the integrity of the ordering among siblings (in ordered DAGs), Step 4(a) or 4(b) in Section 6.2.3 can be used for Scheme-1 or Scheme-2, respectively. Verification of immediate ancestors and the structural order between siblings in $G_\delta(V_\delta, E_\delta)$ has a complexity of $O(|V_\delta| + |E_\delta|)$.

7.3 Single Signature Scheme

In this section, we propose a construction of leakage-free redactable signatures for trees that is transparent as well as highly efficient (computes only one signature). The LFR signature scheme is based on the notion of secure names developed earlier in the dissertation, and the redactable set signatures developed by Johnson, Molnar, Song and Wagner (JMSW) [74].

Review of JMSW Redactable Set Signatures: Johnson et al. [74] developed a redactable set signature scheme based on RSA primitives and random oracle. The signature secure (EU-CMA over $\subset$ and $\cup$ operation). Since it is redactable, given the signature of a set, and the elements that are to be removed from the set (that results in the subset), anyone who knows the public key, can efficiently compute the signature of the subset. The signature is history-independent, and thus can be shown to be a leakage-free redactable signature for sets. Let the public key be $\bar{e}$ and the RSA modulus be $\bar{n}$. Some constraints on RSA are that $\bar{n} = \bar{p}\bar{q}$, where $\bar{p}$ and $\bar{q}$ are “safe primes”.

Given a set $S = \{s_1, s_2, \ldots, s_n\}$, its signature $\sigma_S$ is computed as follows: compute $H(S) = \prod_{1 \leq i \leq n} H(S_i) \mod \bar{n}$, and $I(S) = H^{-1}(S) \mod \phi(\bar{n})$, where $\phi(\bar{n}) = (\bar{p} - 1)(\bar{q} - 1)$. Signature $\sigma_S$ is computed as $(\bar{e})^{I(S)}$. Verification proceeds as follows: Given a set $S'$, and a signature $\sigma$, one computes the $H(S') = \prod_{1 \leq i \leq |S'|} H(S'_i)$, where $S'_i$ is the i’th element in $S'$; And then it is checked if $(\sigma)^H(S')$ is equal to $\bar{e}$. If the equality holds, $\sigma$ is a valid signature of the set $S'$. In order to compute the signature of a subset
**rSign**: Sign tree \( T(V, E) \). Let \( p(x) \) be the parent of node \( x \).

1. For each node \( w \) with \( m \) children, let \( x_i \) be the \( i \)'th child, \( 1 \leq i \leq m \), and \( x_j < x_{j+1}, 1 \leq j \leq m - 1 \).
2. Let \( \eta_x \) refer to the secure name assigned to node \( x \).
3. Add a dummy node \( d_{xy} \) to each edge \( e(x, y) \), thereby splitting the edge to two edges \( e(x, d_{xy}) \) and \( e(d_{xy}, y) \). Assign a random \( \eta_{d_{xy}} \) to each dummy node \( d_{xy} \).
4. For each node \( x \) in \( V' \), compute \( \theta_x \leftarrow H(\eta_{p(x)} \parallel \eta_x \parallel c_x) \).
5. \( H(V) \leftarrow \prod_{x \in V} \theta_x \mod \bar{n} \).
6. \( \sigma_T \leftarrow (\bar{e})^{I(V)} \mod \bar{n} \), where \( I(V) \leftarrow H_{\bar{V}}^{-1} \mod \phi(\bar{n}), \phi(\bar{n}) = (\bar{p} - 1)(\bar{q} - 1) \).

**Figure 7.11.** Algorithm to sign a tree.

**rRedact**: Compute signature of subtree(s) \( T_\delta(V_\delta, E_\delta) \subset T(V, E) \).

1. \( \sigma_{T_\delta} \leftarrow (\sigma_T)^{H(V \setminus V_\delta)} \mod \bar{n} \), where \( H(V \setminus V_\delta) \leftarrow \prod_{x \in V \setminus V_\delta} \theta_x \mod \bar{n} \).

**Figure 7.12.** Algorithm to redact a tree.

\( S'' \subset S \), one redacts the hashes of the elements \( S \setminus S'' \) from the signature \( \sigma_S \) as follows: compute \( H(S \setminus S'') \) and \( \sigma_{S''} \) is computed as \( \sigma_S^{H(S \setminus S'')} \).

### 7.3.1 LFR Signature: \( r\Pi \)

In this section, we present an LFR signature scheme \( r\Pi \) for trees that can be easily extended to graphs/forests. Ordered graphs and forests can be signed using the above scheme. The number of signatures computed is optimal: 1, and the number
Verification (rVrfy): Verify \((\sigma, T_\delta(V_\delta, E_\delta))\); verifier receives \(\theta_x\) for each node \(x\) in \(V_\delta\), and \(\theta_{dxy}\) for each edge \((x,y)\) in \(V_\delta\).

**Verification of Contents:**
1. For each node \(x\) in \(V'\), compute \(\theta_x \leftarrow H(\eta_{p(x)} \parallel \eta_x \parallel c_x)\).
2. \(H(V_\delta) \leftarrow \prod_{x \in V_\delta} \theta_x \mod \bar{n}\), \(\theta_x = H(\eta_{p(x)} \parallel \eta_x \parallel c_x)\).
3. If \(\sigma_{T_\delta} H(V_\delta) \mod \bar{n} = \bar{e}\), all nodes are authenticated.

**Verification of Edges and Ordering:**
1. Carry out a depth-first traversal on \(T_\delta\).
2. **Parent-child relationship:** Let \(x\) be the parent of \(y\) in \(T_\delta\);
   if \(\eta_x \neq \eta_{\hat{p}y}\), then this relationship is incorrect.
3. **Order among siblings:** For ordered trees, in \(T_\delta\), let \(y\) and \(z\) are children of \(x\), and let \(y < z\).
   (a) For Scheme-1 (Section 7.1.1): \(y < z \iff (\text{lsb}(H(\eta_y \parallel \eta_z)) = 1) \land (\text{lsb}(H(\eta_z \parallel \eta_y)) = 0)\).
   (b) For Scheme-2 (Section 7.1.2):
      i. \(j \leftarrow 1 + (H(\eta_y^r \parallel \eta_z^r) \mod L)\)
      ii. \(j' \leftarrow 1 + (H(\eta_z^r \parallel \eta_y^r) \mod L)\)
      iii. \(b_y\) and \(b_z\) are the \(j\)'th, and \(b'_y\) and \(b'_z\) are the \((j')\)'th
      bits in \(\eta_y\) and \(\eta_z\) respectively.
      iv. Check: \(y < z \iff b_y \oplus b_z < b'_y \oplus b'_z\).
4. If contents, the edges, and the orderings among all siblings are verified to be authentic, return 1, else return 0.

**Figure 7.13. Algorithm to verify a tree.**

of hashings carried out is \(O(|V| + |E|)\). The signing, distribution, and verification schemes are given in Figures 7.11, 7.13, and 7.12, respectively.
7.3.2 Complexity

*Single signature scheme:* Our single signature scheme takes a single traversal on a tree/graph/forest, and incurs cost of $O(n + m)$, where $n$ and $m$ are number of nodes and edges, respectively. The number of signatures computed is 1, and the number of order-preserving encryptions that are carried out is $n/k$, as that many groups of siblings are there. The cost of computing a subtree/subgraph/sub-forest of $n'$ nodes and $m'$ edges is $O(n - n' + m - m')$, because $n - n'$ nodes and $m - m'$ edges have to be redacted from the signature. The cost of verifying the integrity of a subtree/subgraph/sub-forest is $O(n' + m')$, but needs to verify only 1 signature. There is no decryption of the encrypted integers.

*Brzuska et al’s scheme:* For trees, it computes $n$ signatures for the tree, and quadratic number of signatures for each group of siblings: $O(k^2)$ (and there are $n/k$ such groups of siblings), $k$ is the arity of the tree. More precisely, it computes $O(n+nk)$ signatures, and incurs cost of $O(n+nk)$. The cost of computing a subtree of $n'$ nodes is $O(n'+nk)$, because $n'k$ orderings and their signatures have to be selected. The cost of verifying the integrity of a subtree/subgraph/sub-forest is $O(n' + n'k)$: these many signatures are verified.

*Comparison:* In comparison to our scheme, signing by Brzuska et al’s is $n + nk$ times more expensive. For computation of signature of redacted signatures, our scheme requires one modular exponentiation, whereas the other scheme does not need any such operations; however, their scheme incurs $O(nk)$ more traversal cost. Verification of the integrity of a subtree using Brzuska et al’s is $(n' + n'k)$ times more expensive than our scheme.

7.4 Security Analysis

We prove the security of the signature schemes by proving the name-transparency of the secure names, unforgeability and transparency of the signature schemes, for both trees and graphs.
7.4.1 Secure Names

Lemma 7.4.1 (Name-transparency) Under the random oracle hypothesis, secure names computed by Scheme-1 are name-transparent.

Proof [Sketch] Consider that an adversary $A$ can determine with non-negligible probability whether a given set of secure names for a subset of siblings $V_δ$ have been computed as part of the computation of secure names of $V$ ($V_δ \subset V$), or have been computed afresh. Consider the sets of nodes as output from $A$: $V_0 = \{x, z\}$ and $V_1 = \{x, y, z\}$, $x \prec y \prec z$. $b$ is drawn uniformly and randomly from $\{0, 1\}$. The adversary then receives the challenge ($V_0, \Theta_0$). Consider that $b = 1$, and $A$ outputs $b' = 1$. $A$ determines that there are one or more siblings in between $x$ and $z$. It implies that the $j'$th and ($j'$)th bit positions for $(x, y)$ pair and/or for $(y, z)$ pair are known to $A$, which implies that $A$ has been able to carry out second-preimage attack on $H$. Therefore, $H$ is not a random oracle, which contradicts our assumption.

Let all secure names be in the interval $[1 : U]$. Let $y$ be a node with rank $i$ among its siblings and be referred to as $v_i$ (to remain consistent with the terminology used in the scheme). To prove that a secure name $\eta_{v_i}$ reveals nothing about $i$, it suffices to prove that the proposed process for secure-name computation is such that the probability of an $\eta_{v_i}$ being equal to any $u \in [1 : U]$ is independent of $i$. We write the event $\{\eta_{v_i} = u\}$ as the union of $k$ disjoint events $E_1, \ldots, E_k$ where $E_j = \{\pi(i) = j, \eta_{v_i} = u\}$. We thus have: $Pr(\eta_{v_i} = u) = \sum_{j=1}^{k} Pr(\pi(i) = j, \eta_{v_i} = u) = \sum_{j=1}^{k} (1/k)(1/U)(1/4^{j-1})$ where we used the facts that: (i) $Pr(\pi(i) = j) = 1/k$; (ii) $\eta_{v_i}$ is $u$ iff Sub-step 3(a) selects $u$ out of the $U$ choices (with probability $1/U$) and that choice is not discarded in Sub-step 3(b), i.e., the choice is admissible relative to the $j - 1$ other already assigned secure names (probability of non-discard is $4^{-j+1}$). Because $\sum_{j=1}^{k} 1/4^{j-1} = (4/3)(1 - 4^{-k})$ we obtain: $Pr(\eta_{v_i} = u) = (4/3kU)(1 - 4^{-k})$, which is independent of $i$, as required.

The secure names do not leak information on $k$ or $m$ (the number of nodes in the tree) either, because each secure name is drawn uniformly from a subset of $[1 : U]$
that is both large (of size approximately $4U/3k$) and uniform over all such subsets of $[1 : U]$, hence indistinguishable from a random choice over $[1 : U]$. 

**Lemma 7.4.2 (Name-transparency)** Under the random oracle hypothesis, secure names computed by Scheme-2 are name-transparent.

**Proof** [Sketch] Consider that an adversary $A$ can determine with non-negligible probability whether a given set of secure names for a subset of siblings $V_δ$ have been computed as part of the computation of secure names of $V$ ($V_δ \subset V$), or have been computed afresh. Consider the sets of nodes as output from $A$: $V_0 = \{x,z\}$ and $V_1 = \{x,y,z\}$, $x < y < z$. $b$ is drawn uniformly and randomly from $\{0,1\}$. The adversary then receives the challenge $(V_0,Θ_0)$. Consider that $b = 1$, and $A$ outputs $b' = 1$. $A$ determines that there are one or more siblings in between $x$ and $z$. It implies that the $j'$th and $(j')$'th bit positions for $(x,y)$ pair and/or for $(y,z)$ pair are known to $A$, which implies that $A$ has been able to carry out second-preimage attack on $H$. The bits of the secure names are assigned using $\oplus$ operation, and the probability of each bit being assigned 0 or 1 is $\frac{1}{2}$. $R$: the number of bits on the $η_w$, $w$ is either of $x,y$, or $z$, is a security parameter ($λ_2$). Therefore, $H$ is not a random oracle, which contradicts our assumption. Otherwise, $A$ has carried out a brute-force attack by enumerating all possible secure names; however, the number of bits that are never used for any pair of siblings ($L - 2 * k$) is a large value - a security parameter ($λ_1$). It implies that $A$ can carry out such brute force search over an exponential search space, a contradiction to the assumption that $A$ is a probabilistic-polynomial adversary.

As earlier, let all secure names be in the interval $[1 : U]$. To prove that a secure name $η_x$ reveals nothing about its rank $i$ among its siblings, it suffices to prove that the process for secure-name assignment is such that the probability of a bit in $η_x$ being either 0 or 1 is $\frac{1}{2}$, and it is true for all the bits in $η_x$. We give a proof by induction.

**Basis:** Case I: $x$ is the left-most child of its parent: $η_x$ is randomly chosen.

Case II: $x$ is the second left-most child of its parent: Let $v_1$ be the left sibling of
The \( R \) bits are randomly chosen. Two out of the remaining bits referred to as \( b \) and \( b' \) are chosen such that \((b_1 \oplus b) < (b'_1 \oplus b')\) (Step 2 of the scheme). However, \( b_1 \) and \( b'_1 \) are bits in the random \( v_1 \), i.e. the probability that the value of \( b_1 \) (or \( b'_1 \)) is either 0 or 1 is \( \frac{1}{2} \). Result of the XOR (\( \oplus \)) of a random number with another (possibly non-random) number is also a random number [76]. Thus \( b \) and \( b' \) are also random bits. The remaining bits of \( x \) are “not used” and randomly chosen. Thus the number \( n(x) \) is a random.

**Inductive step:**

If \( v_k \) is the \( k \)’th left-most child of its parent and \( \eta_{v_k} \) is a random number, then \( \eta_x \) is also a random number where \( x \) is the \((k+1)\)st leftmost child of its parent. \( r(v_k) + 2(k-1) \) number of bits in \( \eta_x \) are already “used”. By Step 2 in the scheme, two bits at positions \( j \) and \( j' \) that are still unused in \( \eta_{v_k} \) are chosen. The \( r(x) \) bits are randoms as well as the two bits at \( j \) and \( j' \) leftmost positions in \( \eta_x \) are also randoms. The remaining bits are “not used” and chosen randomly. Thus \( \eta_x \) is also a pseudo-random.

### 7.4.2 Trees

**Lemma 7.4.3** The signature scheme \( r\Pi \equiv (r\text{Gen}, r\text{Sign}, r\text{Redact}, r\text{Vrfy}) \) for trees using either the CRSA or BGLS scheme is existentially unforgeable under the adaptive chosen-message attack over the subset operation.

**Proof** [Sketch] Unforgeability of the signature is due to the unforgeability of CRSA or BGLS. If the signature of a tree can be forged by an \( A \), then \( A \) has managed to solve the RSA problem or the Computational Diffie-Hellman problem, which however are assumed to be hard problems.

Given the correctness of the secure names (scheme-1 or scheme-2), the order between siblings can be verified. In case the order between siblings or edge relationship in a tree has been forged, then the hash function \( H \) is not a random oracle, which contradicts our assumption.
Lemma 7.4.4 The signature scheme $r \Pi = (r \text{Gen}, r \text{Sign}, r \text{Redact}, r \text{Vrfy})$ for trees using either the CRSA or BGLS scheme is transparent.

Proof [(Sketch)] $r \Pi$ is transparent if and only if the secure naming scheme is transparent, which is proven in Lemmas 7.4.1 and 7.4.1.

7.4.3 Graphs

Lemma 7.4.5 The signature scheme $r \Pi = (r \text{Gen}, r \text{Sign}, r \text{Redact}, r \text{Vrfy})$ for graphs using either the CRSA or BGLS scheme is existentially unforgeable under the adaptive chosen-message attack over the subset operation.

Proof is similar to the proof of Lemma 7.4.3.

Lemma 7.4.6 The signature scheme $r \Pi = (r \text{Gen}, r \text{Sign}, r \text{Redact}, r \text{Vrfy})$ for graphs using either the CRSA or BGLS scheme is transparent.

Proof is similar to the proof of Lemma 7.4.4.

7.4.4 Single Signature Scheme

The following lemmas state the security of the proposed construction $r \Pi$.

Lemma 7.4.7 Under the random oracle hypothesis, and the assumption that the RSA problem is hard, and that the secure naming scheme Scheme-2 is secure, $r \Pi$ is existentially unforgeable under chosen-message attack over subset (and union) operation over trees/graphs/forests.

Proof Suppose that $r \Pi$ can be forged for a subtree $T_δ$. In one scenario, either a node $x$ can be substituted by another node $y$ in the subtree such that $θ_0 = θ_y$. It implies that $H$ has encountered a collision, contradicting the assumption that $H$ is a random oracle. Similarly, forging a wrong parent-child relationship is not feasible under the
random oracle hypothesis. If forging is carried out by forging the signature of a set (other than subset and union operations), then the JMSW signature scheme has been broken, which implies that the adversary has solved the RSA problem efficiently [74]. The order between two siblings cannot be forged under the random oracle hypothesis (because the $\mathcal{H}$ involves $\theta_x$ of each node $x$).

**Lemma 7.4.8** Under the random oracle hypothesis, and the assumption that the RSA problem is hard, then $r\Pi$ preserves transparency.

**Proof** Suppose that $r\Pi$ is not transparent for a subtree $T_\delta$ of tree $T$. In other words, according to Definition 4.4.2, if $T_\delta$ and $T$ are given to the $r\Pi$, the signature that one receives from $T_\delta$ leaks the fact whether it was computed from scratch or by redaction from the signature of $T$. If existence of a sibling in $T$, but not in $T_\delta$, is leaked by the signature of $T_\delta$, then the plaintext values of the position of a sibling among other siblings has been recovered, i.e., the secure naming scheme Scheme-1 or 2 has computed secure names that are not name-transparent, which however is not true. If a parent-child (edge) relationship between two nodes $x$ and $y$ in $T_\delta$ is leaked, then $\theta_{dxy}$ is not distinct, which is a contradiction.

7.5 Performance Results

We carried out experiments over the two schemes proposed in Sections 7.1.1 and 7.1.2. We implemented these two techniques in Java 1.6 and JCA 6.0 (Java Cryptography Architecture) APIs. The experiments were carried out on an IBM Thinkpad with the following specification: Linux (Ubuntu 8.10) on Intel Core 2 Duo CPU 2.2GHz with 2.98GB RAM. SHA-512 is used as the hash function. We have compared the two secure naming schemes with respect to: (1) the average number of attempts, and (2) the average time needed to successfully compute the secure names. We have carried out performance analysis of signing, distribution and verification.
Computing Secure Names: We considered trees of branching factor (the number of children a non-leaf node can have) ranging from 1 to 100. Considering the upper limit, a tree that has a breadth of 100 and a height as small as 3, can have as many as 1 million nodes. In the plots, the rank of a child among its siblings (other children of the same parent) is \( i \), \( 0 \leq i \leq 99 \). Figures 7.14 and 7.15 refer to the performance results with respect to (1) and (2), respectively. Scheme 1 and 2 respectively refer to the preliminary technique (Sections 7.1.1) and the better technique (Section 7.1.2).

We have also carried out the experiments for branching factor 1 to 300, which requires modification of the size of the secure names (only in the case of the second technique) in order to accommodate the breadth of 300, which could not have been possible with the size of 512-bits for the secure name. 300 is a very large Branching factor; a tree of a small height 3 and branching factor 300 has as many as 27 million nodes. The plots in Figures 7.16 and 7.17 refer to the performance results with respect to the number of attempts to compute a secure name of a node and the time to compute such a secure name, respectively with respect to the position of a node among its siblings.
Figure 7.15. Average time in micro-sec to assign a secure name to a node; branching factor $\leq 100$. 
Figure 7.16. Average number of attempts to assign a secure name to a node; branching factor $\leq 300$.

Our performance results corroborate the theoretical analysis and show that the second technique outperforms the first technique both in the number of attempts and
the time required to successfully assign a secure name to a node especially when the breadth of a tree is as high as in the order of hundreds.

7.5.1 CRSA/BGLS-based Schemes

*Signing, Distribute, Verify.* We have carried out experiments for the efficient scheme. The performances of these algorithms are similar to the performances of
the algorithms in structural signatures. The dominant cost factor in signing, and verification is the modular exponentiation (modular multiplication for distribution) for CRSA, and the bilinear map operation and computation over the elliptic curve group for BGLS (for distribution as well). Verification of structural relationships is quite fast (less expensive than the cost of computing the secure names in the efficient scheme as shown in Figure 7.15), and do not affect the verification cost in any significant way; such cost is not included in the plot for verification. These performance results are applicable for forests as well, as the schemes for them are only different from the tree in the sense of representation of the edges. The BGLS authentication scheme is much more expensive for trees than the CRSA-based scheme [105] as the BGLS aggregate signature scheme is based on elliptic curves and bilinear maps.

7.5.2 Single Signature Scheme

We have implemented a prototype of the JMSW redactable signatures of sets in Java and Java Cryptographic Architecture, and carried out the experiments on a Lenovo Thinkpad T61 with 3GB RAM, of which 2560MB was specified as the
maximum heapsize for the Java Virtual Machine. We used 2-ary trees. Experimental results corroborate our complexity analysis.

Signing a tree using our scheme (that deals with sibling ordering as well) of $2 \times 2^{20}$ (more than 2 Million) nodes requires about 70 seconds (Figure 7.21), which is in fact quite efficient. For a tree with 65535 nodes, computing the RSA signatures for Brzuska et al’s signature scheme takes more than 1100 seconds (even without sibling
ordering). It is significantly expensive than our signing scheme. The time to compute signatures of redacted subtrees decreases as the size of the redacted subtree increases (Figure 7.5.2), because the number of nodes to be redacted decreases with the increase in the size of the subtree. It takes about 5 seconds to compute the redacted signature of a 1-Million-node subtree (of a 2 Million node tree). It corroborates the fact that redaction of signatures is more efficient than re-computing them. Verification of the signature of a redacted subtree of 1 Million nodes (of a 2 Million node tree) requires about 5 seconds (Figure 7.5.2), which is significantly less expensive than Brzuska et al’s scheme: that has to verify 1 Million signatures just for parent-child relationships.

7.6 Discussion

We would now describe how the schemes presented in this chapter can be used for certain other scenarios.
7.6.1 Forests

Our scheme for graphs can be used to sign and authenticate forests, i.e., a set of dis-connected trees/graphs in a leakage-free manner. The idea is to sign the forest as a graph (though disconnected). Our scheme for graphs do not depend on connectedness.

7.6.2 Encrypted Trees, Graphs and Forests

In cloud computing, often plaintext data is not delivered to the cloud servers. If the contents of the nodes are encrypted (but not the structure), our schemes (including the structural signature schemes) can be used directly to such scenarios. The only changes that are needed are (1) Share with the server the integrity verifiers, which are hashed values. (2) use a perfect one-way hash functions [36] to compute the hashes, which can then be hashed again to be converted to full-domain hashes. Perfect one-way hash functions do not leak contents of the message being hashed, whereas standard (such as SHA1 pr SHA2) hash functions leak information.

7.6.3 Dynamic Trees, Graphs and Forests

In order to incrementally compute the signature of the updated tree, an insertion (resp., deletion) of a new node requires a new secure name, and leads to a modular multiplication (resp., division) in case of CRSA and an group addition (resp., subtraction) on the elliptic curve followed by a bilinear operation. In an updated graph, the signature of immediate ancestors has also to be updated appropriately. Unlike in the MHT, in our schemes, the updates do not get propagated up in a tree. They do not affect the secure name of other siblings or nodes in Scheme-1; however, in Scheme-2, they affect the secure name of other siblings. The single signature scheme supports both redaction and union of two signatures, which is why, it also supports dynamic updates on trees, graphs and forests.
Computation of a new secure name in the $j$'th rank among its siblings does not affect any secure names of other siblings in case of Scheme-1. However, in the case of Scheme-2, it affects the secure names of all other siblings.

*Answer Freshness and Prevention of Replay Attacks:* The proposed authentication schemes prevent replay attacks and guarantee answer freshness by incorporating timestamps in the signatures as an extra element.

### 7.6.4 Automatic Recovery from Structural Errors

If the contents of a subtree/subgraph is not compromised, but some or all structural relationship and/or order among the nodes have been compromised (maliciously or by communication errors), then the correct parent-child relationship and correct order can be easily recovered using the secure names. Such a capability helps automatic recovery from structural errors without any interaction with the distributor.

For example, suppose that Bob receives a $G_δ$ and authenticates its contents to be valid. However, Bob finds that the received order between two siblings $y$ and $z$, ($y ≺ z$) is incorrect. The correct order between these two siblings can be recovered without communicating with the distributor (server). Verify for the ($x ≺ y$) order using the secure names $η_x$ and $η_y$. If the test succeeds, the correct order is recovered. If such test also fails, then there is no ordering (that is explicitly) imposed on these two nodes in the tree. Similarly, parent-child relationships (for graphs instead of parent, it is immediate ancestor) could be recovered. In order to correct such a relationship between $x$ and $y$, it is verified whether $x$ is the parent of $y$ by $η_x ≅ η_{\hat{p}_y}$. If it fails, then the reverse is checked by evaluating $η_y ≅ η_{\hat{p}_x}$.

### 7.6.5 Path Queries

Our scheme for trees can be applied to path queries (XPath query) and leakage-free authentication of the results of path queries. Such a result is a set of nodes that form a path in the queried tree. The user should be able to verify that the received
nodes indeed lie on a path, but at the same time should not be able to learn any information extraneous to the received nodes. One important topic in which such a requirement occurs is privacy-preserving data mining. The ordering property of secure names for siblings (in the Scheme-1 or Scheme-2), is used for nodes in a path. Secure names that satisfy such property are assigned to ancestors and descendants in a path. When the user receives two or more nodes claimed to lie on a path: the secure names are used to verify whether they are indeed on a path.

Subsequences and Ordered-Set Queries

Paths or lists can also be seen as representation of subsequences and ordered-sets. The approach for path query verification can be directly applied to the problem of authenticating the subsequence queries and subsets of ordered sets.

7.7 Summary

In this chapter, we solved the problem of how to authenticate multiple sub-trees and sub-graphs without leaking. We proposed two leakage-free authentication schemes: one for trees and another for graphs. The scheme for graphs is a general one: it can be used to authenticate any form data organization structures: trees, DAGs, graphs with cycles, forests of trees and graphs. The schemes compute one signature, irrespective of the number of nodes and edges in the data structure. Our schemes are highly scalable. It is efficient than the latest scheme proposed by Brzuska et al [64].
This chapter proposes an approach to content dissemination that exploits the structural properties of XML Document Object Model in order to provide efficient dissemination by at the same time assuring content integrity and confidentiality. Our approach is based on the notion of encrypted post-order numbers that support the integrity and confidentiality requirements of XML content as well as facilitate efficient identification, extraction and distribution of selected content portions. By using such notion, we develop a structure-based routing scheme that prevents information leaks in XML-data dissemination and assures that content is delivered to users according to the access control policies, that is, policies specifying which users can receive which portions of the contents. Our proposed dissemination approach further enhances such structure-based, policy-based routing by combining it with multicast in order to achieve high efficiency in terms of bandwidth usage and speed of data delivery, thereby enhancing scalability. Our dissemination approach thus represents an efficient and secure mechanism for use in applications such as publish-subscribe systems for XML Documents. The publish-subscribe model restricts the consumer and document source information to the routers to which they register with. Our framework facilitates dissemination of contents of varying degrees of confidentiality and integrity requirements in a mix of trusted and untrusted networks, which is prevalent in current settings across enterprise networks and the web. Also it does not require the routers to be aware of any security policy in the sense that the routers do not need to implement any policy related to access control.
8.1 Some Simple Observations

In this section, we discuss the properties of XML data and post-order numbers.

![Diagram of a tree: abstract representation of an XML Document, (b) Post-order numbers associated with each node and (c) Randomized post-order numbers associated with each node.]

Figure 8.1. (a) A tree: abstract representation of an XML Document, (b) Post-order numbers associated with each node and (c) Randomized post-order numbers associated with each node.

8.2 XML Data Model

DOM is the commonly used model for representing XML-based languages [5]. DOM organizes data as a rooted tree. In what follows Document refers to such a tree and DocumentRoot refers to its root. Moreover, Element refers to an intermediate node in the tree. Content of a node includes Attr, DocumentType, DOMImplementation [5]. Each node that is one of the following - Text, CDATASection, ProcessingInstruction, Comment, is a leaf node in the tree. An Entity refers a non-root node in the tree.

Let $D$ be an XML data instance organized according to the DOM representation. Let $T(V, E)$ be a tree representing document $D$ (see Figure 8.1; $V$ and $E$ denote the set of nodes (vertices) and of edges of $D$, respectively. Let $x$ be a node in $V$. Let $D_x$ denote the subtree of $D$ rooted at $x$. Some or all the nodes in an XML-data instance contain content. Content of a node $x$ is referred to as $c_x$. $c_x$ contains only the content specific to $x$ and not of other nodes. The relation between parent-child nodes is represented as directed edges, with edges directed from parents to children. In what follows $ancestor(x)$ denotes the set of ancestors of $x$. 
Dissemination of a document exploits the following structural properties in order to meet the requirements of secure and scalable dissemination of XML-data:

- In some scenarios, XML data is order-preserving, that is, nodes $x$ and $y$ have an order among them in $D$ thus they are modeled as ordered trees. In other scenarios, XML data is modeled as unordered trees.

- The unit of data access is the sub-tree representation of a sub-document. The smallest unit is a node.

- Any element and its corresponding sub-document is accessible through by themselves or a subtree rooted at any of their ancestors.

These properties are crucial in ensuring that structure-based routing extracts the correct sub-document and routes it to the correct consumer; they are reflected by post-order numbers: We can identify and extract a specific sub-document in a document using a randomized post-order number.

8.3 Document Encoding and Encryption

Each element node in an XML document is signed using structural signatures defined in Chapter 5. The hash of structural position and content of a node $x$ used in its structural signature is referred to as $I_x$. Table 8.3 shows the encoding of each node in the XML tree in Figure 8.1. For simplicity, the integrity identifier is not enumerated as it involves a hash value. Encryption of the node contents are carried if necessary using standard encryption techniques. If contents are encrypted, then the publisher passes of the perfectly one-way hashes [36] of the nodes to the routers. Such hashes do not leak information.
Table 8.1
Encoding of XML tree in Figure 8.1.

<table>
<thead>
<tr>
<th>Node</th>
<th>$S_x : (e_x, e_{\text{lowest}}^x)$</th>
<th>Encoding: $\langle C_x, S_x \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>(11, 11)</td>
<td>$\langle ((11, 11), I_x), (43, 11) \rangle$</td>
</tr>
<tr>
<td>$y$</td>
<td>(22, 22)</td>
<td>$\langle ((22, 22), I_y), (43, 11) \rangle$</td>
</tr>
<tr>
<td>$z$</td>
<td>(43, 11)</td>
<td>$\langle ((43, 11), I_z), (107, 11) \rangle$</td>
</tr>
<tr>
<td>$t$</td>
<td>(64, 64)</td>
<td>$\langle ((64, 64), I_t), (96, 64) \rangle$</td>
</tr>
<tr>
<td>$u$</td>
<td>(75, 75)</td>
<td>$\langle ((75, 75), I_u), (96, 64) \rangle$</td>
</tr>
<tr>
<td>$v$</td>
<td>(96, 64)</td>
<td>$\langle ((96, 64), I_v), (107, 11) \rangle$</td>
</tr>
<tr>
<td>$w$</td>
<td>(107, 11)</td>
<td>$\langle ((107, 11), I_w) \rangle$</td>
</tr>
</tbody>
</table>

Figure 8.2. Three sub-trees of the content tree are shared with three consumers: consumer 1, 2 and 3.

8.4 Structure-based Routing

We propose a multicast based approach to disseminate XML-based data among the consumers. Figure 8.4 shows multiple consumer requests to access an XML tree. Consumer 1 has access to sub-tree $T_1$, consumer 2 has access to $T_2$ and consumer 3 has access to $T_3$. For dissemination of the sub-trees among various consumers, a multicast
topology based on the structure of the tree is proposed. The multicast topology is
built dynamically and asynchronously using a publish-subscribe methodology. The
publish-subscribe based multicast network uses structure-based routing.

Structure-based routing involves the following entities: the document source is
the document producer or a trusted owner of the document and has full access to
the original document and is the root of the multicast overlay network; the publisher
publishes the data to a set of subscribers; the subscriber subscribes to the data and
sends its request to a router-based publisher; the router routes the specific portion
of the data to consumers and other routers. A router is both a publisher and a
subscriber. The document source is a publisher. A consumer is a subscriber. A
consumer is said to be associated with a router for a specific document if it has
subscribed to that document through that router. For simplicity of discussion, we
assume one document source; however the proposed solution can handle multiple
document sources. A parent router of another router is one from which the latter
receives some content. A child router is defined conversely.

We assume that documents are identified by a valid Uniform Resource Identifier - URI [4] or any other naming scheme suitable for the enterprise. The owner of the
document can itself carry out publishing or can delegate the publishing functionality
to one or more other entities. The publishing routers propagate this information to
their neighbor routers.

Let \(D_z\) and \(D_q\) be sub-documents such that \(q\) is a descendant of \(z\). Thus, \(D_q \subset D_z\). Let \(R\) refer to any router that is reachable from another router \(R_z\). \(D_z\) is the
 maximal structural block at a router \(R_z\) if and only if \(R_z\) or any router \(R\) reachable
from \(R_z\) has only those consumers that have access to only \(D_z\) or \(D_q\), for any \(q\) that is
a descendant of \(z\) in \(D\). Each router is aware of the maximal structural block that it
is responsible for routing collectively to all the subscribers - consumers and routers.

Example: Let \(D\) be represented by the tree \(T\) (\(T\) is shown in Figure 8.1). Let \(R_1\)
be a router. Consumers \(u_1\) and \(u_2\) have subscribed to \(R_1\) for document \(D\). \(u_1\) and \(u_2\)
have access to the sub-document represented by $T_z$ and $T_x$, respectively. Let $R_2$ be a router reachable from $R_1$. It has a subscriber $u_3$. $u_3$ has access to the sub-document $T_x$. Therefore the maximal structural block of $R_1$ is $T_z$ and of $R_2$ is $T_x$. If a new consumer subscribes to $R_1$ with an access to $T$ itself, then the maximal structural block of $R_1$ becomes $T$.

The multicast topology with root at router $R_z$ disseminates content to a set of consumers collectively such that none of them has access or has subscribed to a subtree $D_m$ of $D$ where $D_z \subset D_m$. $D_z$ can be identified through the encrypted post-order numbers. Let $p_z$ and $p_m$ be the RPONs of $D_z$ and $D_m$, respectively; then $p_z < p_m$, by definition of RPON. Router $R_z$ routes (or publishes) only $D_z$ and its sub-trees. $R_z$ identifies its maximal structural block through its RPON $e_z$, which we call as the Publishing RPON.

A Publishing RPON (PRPON) is the encrypted post-order number of a maximal structural block being published from the corresponding router. For router $R_z$, $p_z$ is a PRPON. Routing is carried on a multicast topology which is of the form: either a tree or a directed acyclic graph (DAG). In Figure 8.3, the PRPON’s for the two routers (between the consumer and the producer) are 43 and 96.

Access permissions on the content for a consumer are expressed on a node in the document. Access permissions for a consumer $u$ on a document $d$ denoted by $L^u$ are represented as an AllowedSet defined as $\{\eta_x \mid \text{consumer has access to node } x \text{ and } \eta_x \}$ is the structural signature of $x$.

8.4.1 Content Routers

By content routers, we refer to content distributors of brokers. Such a router is an application level router that routes documents. In what follows, the notation $\{x+\}$ or $\{(x)+\}$ denotes a non-empty set of elements of type $x$. Every router $R$ is aware of the following information:
• \{⟨c-id, c-credentials, document URI, permissions, callback-address⟩+\}, where
  \(c-id\) is the id of the consumer subscribed to document accessible at \(document URI\),
  \(c-credentials\) includes parameters needed for authentication of \(c-id\), and
  permissions is the set of the nodes that a consumer has access to in the document
  (\(AllowedSet\)). The \(callback-address\) method provides a mechanism to deliver
  content to the consumer in case of asynchronous subscription.

• \{⟨parent router, \(\{S_x+\}\)⟩+\}, where \(x\) is the root of the maximal structural block
  the parent router receives and \(S_x\) is its structural position (identifier), \(p_x\) in \(S_x\)
  is the PRPON of the router.

• \{⟨child router, \(\{S_y+\}\)⟩+\}, where a child router is a router that has subscribed
  to a sub-document with PRPON \(S_y\) from \(R\). The list of child routers with their
  specific PRPONs are stored at \(R\).

**Example**  Consider Figure 8.3. The router that routes the tree with PRPON 43
has two consumers \(consumer\ 1\) and \(consumer\ 2\). It does not have any parent router
nor any child router. The information stored at this router is shown in Table II.
Table 8.2
Information at a router for PEPON 43 in Figure 8.3.

\[
\begin{array}{|c|}
\hline
\langle \text{consumer 1, credentials1, } \\
\text{URI of T, \{11, 22, 43\}, callback1} \rangle \\
\hline
\langle \text{consumer 2, credentials2, } \\
\text{URI of T, \{11, 43\}, callback2} \rangle \\
\hline
\end{array}
\]

8.4.2 Dissemination Network

A link in the document dissemination network is between two content routers and might involve intermediate network routers. In this section, we discuss the development of the dissemination network that uses the structural identifier.

Subscription

The subscription process is initiated by a consumer. Upon being successful the process returns the consumer a set of structural signatures for the nodes in the document that the consumer has access to. The set is the AllowedSet for the consumer. A consumer determines which router to join for a specific document. A router $R$ upon receiving a request for subscription to a document performs the consumer subscription, if the consumer is authorized.

Link Setup

If a router $R$ does not already have a known path to the document publisher to satisfy the request, it sends subscription requests to some or all other routers it is aware of (neighbor routers). Among many possible protocols, we propose a three-way handshake protocol to establish a subscription link between two routers. Suppose that router $R$ receives positive responses from $R_1, R_2$. Based on the document prop-
erties, and the path length from document source to each of them, $R$ then determines which one to choose and notify the router(s) accordingly. Several criteria can be used for such selection.

Outline of Link Setup Protocol

1. The consumer sends the subscription request for a document including its consumer id, credentials and callback method to a router $R$.

2. $R$ authenticates the consumer and determines the list of signatures of the content nodes that the consumer has access to.

3. The router determines the set of sub-documents (sub-trees) from the set of signatures as follows: it sorts the signatures based on the RPON in the signature; if $\eta_x$ is the signature for $x$, then the sorting parameter is $p_x$. Let the sorted set be $Q$. Let the set of sub-trees be denoted by $\Gamma$, initialized as empty. Let the signature with highest RPON in $Q$ be $\eta_z$. Remove each signature $\eta_x$ from $Q$ including $\eta_z$ such that $p_x \leq p_z$, assign this set of signature to $\gamma_x$; add $\gamma_x$ to $\Gamma$ and repeat this process until $Q$ becomes empty.

4. If the list of accessible sub-trees $\Gamma$ includes a subtree with root having RPON $p_z$ ($z$ being the document element) that is subsumed by the content tree served by this router, then the request is processed successfully.

5. Otherwise, the router $R$ sends a subscription request for the subtree rooted at $p_z$ to some or all of its neighboring routers. If the access permissions include multiple sub-trees, the subscription request includes each of these sub-trees.

6. Upon receiving a request from $R$, a router $R_i$ checks if there exists a PRPON $p_x$. If so, it returns $p_x$ with success as a response to $R$, else it recursively repeats the link setup procedure from $R_i$ for $p_z$ to all its neighbors.

7. Upon receiving the responses, the router $R$ selects a parent router; the router registers the consumer and sends the response back to the client.
8. Each router determines if the new node(s) and existing node(s) can be combined together to form a complete sub-tree of the document. If so, then it replaces all the nodes stored in the database by the lowest common ancestor of these sub-trees.

8.4.3 Content Publishing

The document publication process varies based on the recipient - router or consumer. Content is published as follows. Router $R$ receives a set of document nodes $N$ from its ancestor (topology is a tree) or ancestors (topology is a DAG). If $R$ has a non-empty set of consumers for the document, it then forwards the document to the consumers based on the permissions. If there is a non-empty set of routers that are subscribers for some nodes in this document, then $R$ forwards the document to these routers based on their requirements.

Content Delivery to Consumers

For each subscribed consumer the router determines its access permissions for the associated document.

The router identifies to which received content sub-trees, the allowed nodes (included in $AllowedSet$) belong. This is carried out by matching the RPON $p_x$ of each $\eta_x \in AllowedSet$ with the RPON of each of the roots of the received sub-trees. The sub-trees specific for the consumer in $\Gamma$ are then extracted from the identified content. The router then forwards the subtree to the consumer after encrypting it using the encryption technique in place, if any. In our running example, Figure 8.3 shows how RPONs are used for routing of sub-trees.
Content Delivery to Routers

The process of forwarding the document to a router is as follows. For each router in its subscriber set, a router determines the node(s) it is registered for. It identifies and extracts these document nodes from the respective sub-trees. They are then encrypted and sent to the subscribing router. The next section discusses the technique used for identification and extraction.

Content Identification and Extraction

Content identification and extraction is carried out at each router that has at least one subscriber. Each router has a list of the content sub-trees it receives for a given document. The list is essentially a list of signatures of the roots of these sub-trees (maximal structural blocks) that contain the encrypted post-order numbers (PRPONs) of these roots. The router also keeps track of the list of signatures of the roots of the sub-trees each of its subscribers (consumers or routers) has access to (Section 8.4.1). The identification step determines the \textit{belongs-to} relation among each of the content roots accessible to each consumer and the content sub-trees it receives. An important property of RPONs is reported here from Section 5.2.

\textit{Simple RPON Property} - any node \(y\) that belongs to the subtree rooted at node \(x\) is such that the \(p_y < p_x\), where \(p_y\) and \(p_x\) are the RPONs of \(y\) and \(x\) respectively. The identification technique uses the \textit{Simple RPON Property} while verifying the \textit{belongs-to} relation among the received content and the subscribed content. The worst case complexity of the identification step is \(O(mn)\), where \(m\) is the number of received content sub-trees and \(n\) is the number of subscribers at a given router.

During the extraction step, a depth-first traversal \[45] is carried out to determine the subscribed content root. The RPON of the root of the subscribed content root is compared to the RPON of the visited node. If these RPON’s match, the corresponding subtree is extracted. The worst case complexity of the extraction procedure is
same as that of the depth-first search - $O(v + e)$, where $v$ is the number of nodes in the received subtree and $e$ is the number of edges in the received subtree.

8.4.4 Document Verification

The security requirements for secure dissemination of XML content are two-fold (Section I): maintaining confidentiality by not sending extraneous data to a consumer (preventing information leaks) and facilitating precise verification of integrity. In this section we focus on integrity verification for the received content at the consumer side.

In order to precisely detect any integrity violations, the following verification steps must be executed at the consumer side:

- if nodes have been dropped: aggregate signatures are used to verify if any nodes have been dropped.

- if the order of the nodes has been changed: using the integrity verification mechanism of structural signatures.

- if the content of a node has been compromised: using the integrity verification mechanism of structural signatures.

- if some nodes have been added in an unauthorized manner: if signed hash is used (in other words $\mathcal{H}$ represents a signed hash), then such unauthorized additions could be verified.

- if the content of one node has been replaced with the content of another node: using the integrity verification mechanism of structural signatures.

- Non-repudiation: signed hash supports complete non-repudiation.
8.4.5 Update Management

This section discusses updates to documents - content and structure in the context of structure-based routing. The updates to the XML tree is carried as per the procedure specified in Section 5.6.

In case of changes that are structurally-invariant, only the data inside a document node changes. Thus only the local hash of the node changes. Only the updates of the changed nodes along with their signatures are forwarded to the routers.

Structural changes have to be reflected in the mapping from user credentials to accessible nodes and their signatures. Therefore the services that implement the mapping function from user credential to structural identifiers need to be notified accordingly with the new EPON’s. If it is a distributed hash table, then the document source updates the hash table. The routers are also notified of the modifications. Removal of a subtree is notified to the routers and consumers having the document. In case of addition of a new subtree, the original structure of the document is not affected. Therefore the update is propagated to all the routers that have consumers with access permission to the new subtree. In case of interchanges, the changes need to be propagated to the routers and consumers that are registered for any updated node or an ancestor of that updated node.

8.5 Discussion

In this section, we discuss the requirements for document dissemination and show that our proposed dissemination model addresses all the security requirements of a dissemination model.
Requirements Satisfaction

**Integrity:** We introduced the notion of encrypted post-order numbers in order to support all the integrity requirements. In Section 8.4.4, we developed techniques based on this notion for content verification and validation.

**Access Control and Confidentiality:** The structure based routing scheme ensures that a consumer is delivered only the portion of data that it has access to. The notion of maximal structural blocks at routers ensures that the routers have access to only that much amount of data that its consumers collectively have access to. Our post-order numbering based integrity check technique is parallel to the Merkle Hash algorithm. Such a technique requires the hash values of the subtrees that are not accessible to the consumer also to be forwarded, so that the consumer can verify document integrity by computing and matching the final hash value of the complete original tree. Our technique exploits the properties of post-order numbering for the same goal and thus avoids sending the hash values of the subtrees that are not accessible to the consumer, thereby preventing leakage of data. This is an indirect information leak that is prevented by our framework.

Efficiency of Structure-based Routing

The simple notion of post-order numbers in the context of XML-data provides powerful and sound principles for content identification and efficient extraction with linear time complexity. The framework uses an efficient content routing mechanism based on the content structure. The cost of routing is, in the worst case, linear in the document size.

The multicast topology based on structure-based routing for the dissemination model is acyclic. Multicast reduces the network usage, while the cycle-less feature ensures that the number of router hops is finite and proportional to the height of the document tree. Using Pigeonhole principle on the number of content nodes and the
number of consumers in a large dissemination network, there would be overlaps for content accessibility and subscription between consumers.

Given that each path from the document source to a consumer contains a monotonically decreasing sequence of RPON’s of the document as PRPON’s of the underlying routers and a parent router never has a less RPON as a given router’s PRPON, a cycle cannot occur. This makes the multicast topology more efficient in terms of bandwidth usage and dissemination speed. The path from the source of a document to a consumer contains a list of routers. Let the sequence of routers be $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_i \rightarrow R_{(i+1)} \rightarrow \ldots \rightarrow R_n$. Let the publishing RPON for the consumer at each $R_i$ be $\beta_i$. The following observations are crucial for the topological efficiency.

- For each $i < j, 1 \leq i, j \leq n$, $\beta_i \leq \beta_j$, that is, the PRPON’s have a monotonically decreasing order among them in such a path. This is because a router creates a link to another router during the subscription process, if and only if the router has access to the required sub-tree or a larger sub-tree from the specific document.

- Due to the monotonicity property, the sizes of the subtrees being transmitted along the path $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_n$, also decrease monotonically. In the worst case all subscribers along the path have access to the complete document; however in reality, most subscribers have access to a subset of the document. Therefore the cost of transmission of the content from the source to a consumer is less than the cost incurred in a common star/broadcast topology; or is in the worst case equivalent to such a cost in the latter.

Therefore such a model is efficient in terms of network resource usage, speed of dissemination and thus is more scalable.
8.6 Summary

We showed how the structural properties of the XML Document Object Model can be exploited in order to address issues in data security and dissemination. We used randomized post-order numbers as the content routing parameters. The structure-based routing scheme uses the notion of randomized post-order numbers to prevent information leaks in XML-data dissemination.

We proposed a dissemination model for XML content that combines multicast and structure-based routing in order to improve efficiency in terms of bandwidth usage and speed of data delivery, thereby favoring scalability. The dissemination model combined with techniques for data integrity verification and confidentiality provides a secure publish-subscribe paradigm for XML Documents. The publish-subscribe model restricts the consumer and document source information to the routers to which they register with. Such an approach to XML content dissemination satisfies the requirements of integrity, confidentiality and privacy-preservation in a holistic manner.

Structure-based routing provides a modular and flexible model for security enforcements in data distribution. Flexibility in security enforcement is known to be an important requirement in secure system design and implementation. Depending on the degree of trust on the network integrity checks may or may not be enforced. Moreover, the framework facilitates dissemination of contents with varying degrees of confidentiality and integrity in a mix of trusted and untrusted networks, which is so prevalent in current settings across enterprise networks and the web.
A widely used form for representation, storage, query, management and transmission of data is the object-oriented form. With the advent of cloud computing, web services, and JSON-type object forms to transfer data, objects are increasingly being used information units. In this paper, we define the first schemes for (leakage-free) authentication of objects and of redacted objects.

9.1 Introduction

Authentication of data especially in cloud computing paradigms is an important problem, and has been widely investigated [54, 66]. Authentication is a stronger security requirement than integrity assurance. With the advent of cloud computing, web services, and JSON-type object forms to transfer data between a browser and a server, objects are increasingly being used for representation, storage, query, management, and transmission of data. Objects are transferred across organizational and trust boundaries. In such a context, the development of authentication techniques specific to (redacted) objects is an important requirement.

An object might contain sensitive information, which need to be redacted when the object or its copy is sent to another process/service that has a different trust level or that does not need to know such information [79]. The need for authentication and redaction of objects arises in the following three scenarios: \( C' \) is a superclass of \( C \). Process/service \( P_1 \) has an object \( O \) of class \( C \). Process/service \( P_2 \) (maybe a remote or local) provides a service/functionality that \( P_1 \) requires, for which it needs
to have access to an instance of class $C'$. (1) Process $P_1$ and Process $P_2$ have the same level of privilege. (2) Process $P_2$ has a lower privilege level and thus cannot access the values of all the members in $O$. (3) $O$ has one or more members (not inherited from $C'$) that are privacy-sensitive, and disclosure of even their existence would breach privacy. Process $P_2$ has a lower privilege level than $P_1$ and thus should not learn about the existence of such members. For scenario 1, the object needs to be authenticated but not redacted. For scenario 2, the object needs to be redacted such that the values of the sensitive members are hidden/deleted, and the redacted object should be authenticated. For scenario 3, the object needs to be redacted such that the names and values of the sensitive members are hidden/deleted, and the redacted object should be authenticated. Moreover, for the latter two cases, the redacted objects should of type that is a subclass of $C'$ and defines a behavioral subtype, which is assumed to be satisfied by the process of redaction. $P_1$ might store $O$, which would be used by $P_2$ later. Multiple processes and services can also be included in such scenarios.

In this paper, we define authentication schemes using which the authenticity of an object can be verified. Moreover, we propose the notion of object redactions and redactable signatures for objects such that a public redactor (anyone) can compute the signature of an object $O'$ that is redacted from $O$. We assume that the redaction of an object is safe - i.e., the computation with $O'$ as input would always proceed in the same manner as it would proceed for $O$ as input (equivalent to behavioral subtyping). We present two schemes for authentication of objects that can be represented as trees and objects redacted from them.

9.2 Objects

**Member Fields:** An object $O$ that is an instance of class $C$ contains a set of members - that are fields representing the state of the object and methods that
represent the messaging constructs. By a member, we denote instance members and not class-specific members (such as static variables and methods in Java).

**Inheritance:** Inheritance of classes are used to define subclasses. An instance of a subclass contains all the members of all the super-classes as well as its own new members with some exceptions. The exceptions are in the context of methods. Methods can be overridden (such as virtual methods in C++ and non-static methods in Java). Let \( C \) be the class of the object \( O \). A member belongs to the class where it is declared. Therefore, if a member was defined in a superclass \( C' \) of \( C \), then it belongs to \( C' \), and is inherited by \( C \). For an instance method \( m() \) that is defined in \( C' \), that specific definition belongs to \( C' \). If \( C \) overrides the definition of the method \( m() \), then \( C \) contains this new definition of \( m() \), otherwise \( C \) inherits the definition of \( m() \). Overloading of methods lead to different method signatures, and thus are treated as different members. For simplicity, we have not explicitly handled constructors, which can be handled in a way similar to that of methods (even though they are not exactly methods).

**Example:** A class \texttt{Emp} (employees) is a superclass of class \texttt{RegEmp} (regular employees - eligible for bonuses).

```java
public class Emp {
    private int id, pay;
    public Emp(int idv, int payv) {id=idv; pay=payv;}
}

public class RegEmp extends Emp {
    private float bonus;
    public RegEmp(float bonus_pct) {bonus = bonus_pct;}
}
```

9.3 Object Trees

Objects can be represented as trees if each member field is referred to by at most one other member, and it does not have any cyclic references. Each member of such an object is represented as a tree, as specified below (See the object tree of \textit{objRegEmp} - instance of \textit{RegEmp} class in Figure 9.2(b)).
Figure 9.1. Tree representations of members of objects: (a) primitive data type, (b) array of primitive types, (c) methods, and (d) instances of user-defined types.)

**Primitive Types:** A member variable is represented as a tree (Figure 9.1(a)). Root of the tree specifies the name of the member (e.g., \textit{id}). The first (left) child specifies the class name where this member is declared - in this case it is \textit{C}. (If a superclass has declared a variable with the same name, then that there would be another entry in the object tree). Next child specifies the type, and the right child specifies the value contained in that variable.

**Arrays:** The tree for arrays (Figure 9.1(b)) contains the name as the root. The first child is the class name. Next child is the type as “type [ ]” (e.g., int [ ]), the next child of root contains the size of the array. The next child is the root of another subtree. The values in the array are represented as the leaf nodes in the subtree. It can be extended to represent arrays of user-defined types by adding references or pointers to instances in the nodes representing values.
Methods: The tree for methods (Figure 9.1(c)) contains the name of the method as the root node. The name of the class that defines the method is the first left child. The contracts (pre-condition and post-condition in this order) are specified as the next child. The return type and then the definition (method body) are defined as the next children, in this order.

Instances of User Defined Types: A user-defined type (such as class, structures and unions) contains member variables, methods (if it is a class), and class-specific variables and methods. The tree representing an instance of a user-defined type (Figure 9.1(d)) has the name as the root, the class it belongs to is the first left child, and then the nodes/subtrees with respect to each other members. If the object is not contained in another object, then the class name to which it belongs has a value “NONE”.

9.4 Redaction of Objects

Consider our example. If an object of RegEmp need to be redacted to an instance of type Emp, and should not contain the bonus information, then the bonus information, is removed from the object. Notice that if the received process learns that the original object had “bonus” as a member, then it would infer that the “original” object is an instance of RegEmp and thus the associated employee is a regular employee. In several contexts, it may breach privacy, and thus the “bonus” field should be hidden completely - even its existence. For the scenario 1 (Section 1), the object in Figure 9.2(b) is sent to the process \( P_2 \). For scenarios 2 and 3, the objects sent are in Figure 9.2(c) and (d), respectively.

How Redaction is Carried out: An access control policy \( \mathcal{P} \) is applied to the object that specifies the confidentiality levels of each member in-depth in the object, if such a policy is not already applied on a member. Each process has a specific privilege level associated with it. Whenever an object \( \mathcal{O} \) (instance of class \( \mathcal{C} \)) needs to be passed to a process of privilege level \( L \), a redaction is carried out. Suppose the process of
privilege level $L$ expects an instance of class $C'$. The secure clone can then be an operation of this sort: $O' = O.\text{redact}(\mathcal{P}, L, C')$. The formal method signature is given later, which also takes the signature of the object and auxiliary information. This operation creates $O'$ that contains all members that have a privilege level less than or equal to $L$. Moreover, the redact() operation is such that it always creates an object that conforms to the behavioral subtyping rule by ensuring that $O'$ is an instance of a class that is a subclass of $C'$. The mechanisms of redaction can be carried out by a secure clone or selective clone operation on the object. Such operations need to be supported, especially, when an $O'$ needs to be passed to a process/service (or a thread) residing in the same memory. Redaction is also be carried out during the process of serialization/marshaling. Marshalling is required when the object needs (1) to be sent to a remote process/service, or one that is in a separate memory, and/or (3) to be persistently stored.

Secure and Selective Clone: Secure and selective clone operations implement the redaction logic as well. In secure clone: the secure clone can be used as a redaction operation: $O' = O.\text{clone}(\mathcal{P}, L, C')$. This operation creates $O'$ that contains all members that have a confidentiality level less than or equal to $L$. Moreover, the clone() operation supports behavioral subtyping rules. A selective clone operation does not specify a privilege level, but specifies only the expected type of the object that is returned by the clone operation: $O' = O.\text{clone}(C')$. There are several issues such as conflicts in confidentiality levels of members and aliasing that need to be addressed in implementation of policy enforcement on objects and secure clone, which we do not address here.

Redaction During Serialization: An object can be redacted during its serialization according to its expected type and access control policies, if any. Java language supports both standard and custom serialization and deserialization mechanisms via `java.io.Serializable` and `java.io.Externalizable`. The redaction can be implemented in the `WriteObject()` and `WriteExternal()` methods. One way to specify the members that need to be redacted and whether to redact their existence as well is by populating a vector
of such members that would be referred to by these methods during serialization. If
the type of the resultant object is defined by a new class that is dynamically created,
the definition of the class has to be sent to the recipient first; by doing so, it is ensured
that the class is loaded into the memory before deserialization of the actual received
object is carried out successfully. Mechanisms for such dynamic class creation needs
to be in place and should be secure.

9.5 Authentication of Objects

In this section, we have presented our scheme for objects that can be represented
as trees. During serialization, the signature is attached and verification object is
computed for the (redacted) object. During deserialization, (1) the object is deserial-
ized first, and then (2) this object is validated against the signature and verification
object. Serialization and de-serialization of objects are necessary when the $P_1$ and
$P_2$ are either remote (do not reside in the same memory). Such operations are order-
preserving, that is, the order between the members in which they are serialized into
a stream such as file or a network socket connection, is preserved during its de-
serializaton. In case it is not, de-serialization may fail. Therefore, the authentication
mechanism should not only preserve the integrity of the contents but also preserve the
order among the members. Moreover, the straightforward scheme of signing nodes
and edges in an object tree would not work.

Cryptographic Notations: $H$ is an one-way collision resistant hash function. $(pk,
sk)$ is a pair of public and secret key respectively. $(\sigma_0, VO_0) \leftarrow rSign_{sk}(O)$ signs
an object $O$ of class $C$ using the secret key. $VO_0$ is the auxiliary information for $O$.
$VO_{O'} \leftarrow Redact(O,(\sigma_0, VO_0), P, C')$ reducts the object $O$ to $O'$ with respect to class
$C'$ and returns the redacted verification object $VO_{O'}$. The redacted signature of $O'$
is $(\sigma_{O'}, VO_{O'})$. Verification $rVrfy_{pk}((O', \sigma_0, VO_{O'})) = 1$ if the signature $(\sigma_{O'}, VO_{O'})$
is valid for $O'$, and 0 otherwise.
9.5.1 Scheme Based on Merkle Hash Technique

rSign: The Merkle hash [100] of an object is computed by by carrying out a bottom up and left to right traversal of the object tree, and computing the respective Merkle hashes (MH). The MH of a tree is computed as follows. It computes the hash of the leaves $m_i$: $h_i \leftarrow \mathcal{H}(m_i)$. It then traverses the tree bottom up and left to right (for children) and computes the MH $h_{ij}$ of node $m_{ij}$ ($j \geq i$), which is the parent of nodes $m_i, m_{i+1}, \ldots, m_j$ as follows: $h_{ij} \leftarrow \mathcal{H}(h_i || h_{i+1} \ldots || h_j)$. The MH of the tree is the MH of its root. The MH of the root of the object tree $h_O$ is signed: $\sigma_O \leftarrow \text{rSign}_{sk}(h_O)$.

Redact(;) Let the redacted object be $O'$ an instance of class $C'$. Redaction only returns the modified object $V_O$ is the set of the MH of each of the following node $x$: $x$ is an adjacent sibling of a member in $O'$ in the tree representation of $O$. For scenario 2, the member that is redacted is bonus in objRegEmp1. So $V_{\text{objRegEmp}} = \ldots$
MH of the value of bonus: 10.5. (There are hash schemes available in the literature that do not leak information (such as [35]) and can be used here).

**rVrfy**: The receiver computes the MH of the root of the tree for the received object $O'$ using the $\mathcal{V}O_{O'}$, and then verifies the signature using $\text{rVrfy}_{pk}((O', \sigma_{O'}, \mathcal{V}O_{O'}))$. For example, the MH of the root of the tree in Figure 9.2(c) is computed as earlier, but it uses the MH of the value of bonus: 10.5. The Merkle hash technique leaks information about the existence of the bonus information, and the receiver may infer that this object is for a regular employee. In order to prevent such leakages, we present the following scheme.

### 9.5.2 Leakage-free Scheme

We would use our leakage-free redactable signature scheme for trees based on JMSW homomorphic signature scheme (Section 7.3). This scheme does not leak any extraneous information that is not in the subtree/subgraph. We can use the signature schemes we have developed based on Mykletun et al’s Condensed-RSA or Boneh et al’s aggregate signature scheme. However, for efficiency, and simplicity, we would use the single-signature scheme.

**rSign**: Let $V$ be the set of nodes in the object tree of $O$. A secure name is assigned to each node $x$ or the structural position $\eta_x \leftarrow (p_x, r_x)$ is assigned to $x$ in the object tree, where $p_x$ and $r_x$ are randomized post- and pre-order numbers. Signature of the object $\sigma_O$ is computed as per the signing scheme presented in Section 7.3.

**Redact**: Let the redacted object be $O' - an instance of \mathcal{O}'$, and $V'$ be the set of nodes in the tree of $O'$. Redaction scheme in Figure 7.12 computes the modified object and the redacted signature $\sigma_{O'}$. 
rvrfy: The receiver of $O'$ follows the verification procedure outlined in Figure 7.13. It first verifies content authenticity, and then verifies the structural relationships.

**Efficiency and Security:** Distribution of the object and authenticity verification is carried out without leaking any information. It is important to note that the size of the signature items is size-oblivious ($O(1)$) independent of the sizes of the original and redacted objects. The receiver only receives information about $O'$ and constant number of verification objects, i.e., it does not learn existence/absence of the nodes in $O$ that are not in $O'$.

**Object Graphs:** Objects in general form are graphs with cycles involved. We can directly apply the leakage-free signature for graphs (Section 4.3.2) for authentication of an object or a sub-object.

### 9.6 Summary

An authentication scheme is used to verify (1) the integrity of data, and (2) the fact that whether the received data object $O$ is indeed sent by the claimed sender from the claimed data source. An authentication mechanism can help in developing data provenance purposes. A widely used form for representation, storage, query, management, and transmission of data is the object-oriented form. Object-oriented databases have been studied almost a decade ago, and such trends would only influence the revival of such databases and their much more usage.
10 CONCLUSIONS

*We can only see a short distance ahead, but we can see plenty there that needs to be done.*

Alan M. Turing [131]

In third party data distribution frameworks such as the “cloud”, storage and distribution of data is carried out by third party infrastructures and servers, which may not be trusted. Data objects stored, processed and distributed through third-party architectures are very often organized as *trees*, *graphs* or even *forests* (set of disconnected trees/graphs); for example, data organized according to XML schemas and biological graphs. In such cloud-computing paradigms, which are increasingly being employed in order to store and publish sensitive information, protection of privacy and assurance of confidentiality are as important as verifying authenticity of data. Moreover, data authenticity must be assured even when the data that a user can access is a subset of the signed data, as users maybe authorized to only access a subset of the data.

Existing solutions such as the Merkle hash technique and the redactable signature schemes lead to data leakages that can be used to infer sensitive information that is not part of the received data, which in turn would lead to privacy and confidentiality breaches. Our contributions have been summarized in the following section.

10.1 Research Contributions

In this work, we demonstrated a connection between structural leakages and their implications on privacy and confidentiality. We characterized the inference attacks that can be carried out on the widely used Merkle hash technique. We presented the
first formal security model for leakage-free redactable signatures that defines a notion of unforgeability, privacy and transparency.

In this work, a signature scheme called as “structural signatures” based on the traversal numbers has been proposed that can be used for authentication of a subtree in a leakage-free manner. For this purpose, we defined the notion of randomized traversal numbers. We have also presented the structural signature schemes for graphs – both directed acyclic graphs, and graphs with cycles, that facilitate authentication of a subgraph in a leakage-free manner. In order to support the leakage-free authentication of multiple subtrees/subgraphs, the formal notion of secure names have been developed; we have proposed two constructions of such secure names. Based on the secure names and existing redactable signature scheme for sets, we have defined a highly efficient and generic leakage-free redactable signature scheme that can be used not only for trees, but also for graphs and forests. This scheme is highly efficient – it computes only one signature per tree/graph/forest.

Our schemes for leakage-free authentication can be used for encrypted data (nodes are encrypted), which plays a significant role in enabling security in cloud computing solutions and services. We also showed how the structural signatures can be used to automatically recover from structural errors in tree-structured data. Such a scheme has several applications especially in satellite-based data transmission and in sensors. Our schemes can be used for leakage-free authentication of paths, and has applications in authentication of of XPath query results.

As an application of the structural signatures, we have developed a publish/subscribe scheme for XML distribution known as “structure-based routing”, which uses randomized traversal numbers not only for verification structural integrity, but also for efficient routing of contents. In another application scenario, leakage-free authentication of objects is an important requirement. Objects are a primary model of data representation in object-oriented programs, web services, and object-oriented databases. Serialized objects are also stored and/or transmitted in environments that involve untrusted third-party entities. We have developed the first such solution
to this problem. Moreover, in financial scenarios, organizations such as credit-card providers, banks, and tax offices manage, store and process financial information. Given the sensitive nature of such data, it is essential to facilitate authentication of financial data without leaking. Our solutions can be applied in order to address such requirements in the financial domain.

10.2 Open Problems

*Completeness verification and leakage-free authentication of query results:* The question is can we develop a scheme for trees/graphs so that query results from such databases can be authenticated without leaking, and the completeness of the query results can be verified. We think it is impossible to provide both completeness and leakage-free properties together. The question is quite interesting when a single query results in multiple subtrees/subgraphs; in such a case, what is the lower-bound on the leakage for an authentication scheme that provides verification of completeness?

*Optimality of the single-signature solution:* In Section 4.3.2, we presented a leakage-free signature that computes only one signature, and carries out $O(n)$ modular multiplications in the RSA system, where $n$ is the number of nodes and edges. The number of signatures is optimal. However, is the number of modular multiplications optimal, or can we find a scheme in which we can compute a leakage-free signature for a tree, graph and/or a forest, where it would lead to a sub-linear number of such operations?

*Leakage-free authentication of paths:* As we described earlier, the path relationship between two nodes in a tree/graph can be authenticated without leaking. However, when a user receives multiple sub-paths (or sub-sequences) from a path/sequence, the user can learn about the path relationships between the nodes that are belong to different sub-paths. A straightforward solution to this problem is to compute $O(n^2)$ number of signatures for a path consisting of $n$ nodes. There are two questions: can
we develop a scheme that is more efficient than this one, and what is the lower-bound on the number of signatures?

Undecidability of random-ness: We needed a scheme to compute randomized traversal numbers, which led us to the following question: how do we know that a given number is truly a random number or “almost” a random number. There are statistical techniques towards addressing this question, however, there are no schemes that can be used to prove whether a given number is truly random or not. We do not have such a scheme as well. The following conjecture is on this problem.

Conjecture 1. Determining whether an event or number is truly random, is undecidable.

10.3 Future Research Directions

Unforgeability of redactable signatures: Redactable signatures as as been noted by Johnson et al [74] can only support a notion of unforgeability weaker than the standard notion - EU-CMA. Johnson et al proposed a redactable signature for sets that is existentially unforgeable against chosen message attacks under the subset and union operations (i.e., one can “legally” forge signatures for a subset (or a union) from the signature of a message (or signatures of messages)). The question is can we have a redactable signature for sets that is leakage-free, and is existentially unforgeable against chosen message attacks under only the subset or the union operation, not both of them. Such a scheme has interesting applications in third-party data distribution. The following conjecture is based on this problem.

Conjecture 2. A redactable signature scheme supports subset operation iff it supports the union operation.
Authenticated databases and privacy-preserving operations: Authenticated databases are important especially in a third-party cloud context. Do the schemes proposed in this thesis support any form of authenticity requirements in third-party databases? Moreover, if there are further security and efficiency requirements in such a context, then what they are and how they can be addressed. It is essential to develop schemes, systems and standards towards various levels of security on cloud-based databases. Moreover, privacy-preserving operations on trees, graphs, and forests are gaining importance in both cloud frameworks and two-party frameworks. Such operations are useful in query evaluation on encrypted databases, caching such as of XML data, and in multi-party computation. Solutions to privacy-preserving operations on such data objects enable cloud-based databases of sensitive data such as healthcare, financial as well as biological data.

Composite software and their interactions: Composite software and their execution has three aspects: (1) data, (2) code of the components, and (3) the metadata that specifies the interaction and makes the software discoverable as a service. In the emerging models of composite web services, mobile applications, and cloud computing, each of these aspects should be assured with respect to their integrity and confidentiality. When services/applications from multiple parties are used in developing a single service, it is essential to verify their integrity and authenticity. Authenticated UDDI registries are used to discover web services (Bertino et al. [19]). Authentication of web services is essential towards addressing software-based need-to-know attacks [79]. The problem of authenticating all the three aspects in both static workflows/web services (static – no changes in any of these three aspects), and dynamic workflows and web services is important towards realizing a trusted third-party computing framework.
LIST OF REFERENCES
LIST OF REFERENCES


graph querying toolkit. In Proceedings of the VLDB Endowment, volume 1,

Physical Laboratory, published in Machine Intelligence 7, B. Meltzer and D.


towards a cloud definition. ACM SIGCOMM Computer Communication Re-

[133] Swami Vivekananda. The Complete Works of Swami Vivekananda, Volume 6,
Notes Of Class Talks And Lectures. Vedanta Press & Bookshop, 1947.

[134] H. Wang and L. V. S. Lakshmanan. Efficient secure query evaluation over
encrypted XML databases. In Proceedings of the 32nd International Conference


ographers Track at the RSA Conference (CT-RSA), volume 4377 of Lecture Notes

[137] P. Zezula, G. Amato, F. Debole, and F. Rabitti. Tree signatures for XML
querying and navigation. In Database and XML Technologies, volume 2824
of Lecture Notes in Computer Science, pages 149–163. Springer, September 8,
2003.

on Distributed Computing Systems (ICDCS), pages 427–436, Washington, DC,

[139] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in
graph data. In Proceedings of the International Workshop on Privacy, Secu-
Springer-Verlag.
VITA
VITA

Ashish Kundu was born in Mangalpur in the state of Odisha, India. During his schooling, Ashish attended the Mangalpur Primary School, the Srima Shiksha Sadan Middle English School and the Srima Shiksha Sadan High School at the Sri Aurobindo Nagar near his native place. For higher secondary education (11th and 12th) in science, he studied in the Fakir Mohan College at Balasore in Odisha. During the schooling and undergraduate, Ashish received a Primary School Scholarship of Odisha, National Rural Talent Scholarship of India, and National Merit Scholarship of India. During these years, Ashish has also received several awards for his essay writing and debating abilities.

After his higher secondary education, Ashish joined the Regional Engineering College, Rourkela (now known as the National Institute of Technology, Rourkela) in order to study for the Bachelor of Engineering in Computer Science & Engineering. In 1998, after completing the Bachelor of Engineering with Honors, he went on to study for the degree of Master of Technology in Computer Science at the Indian Institute of Technology Bombay. During the master’s program, Ashish worked with Prof. D. M. Dhamdhere, his advisor, and developed the notion of E-Paths and the scheme of E-Path-PRE for efficient partial redundancy elimination in the context of optimizing compilers.

After receiving the master’s degree, Ashish joined the IBM Research Lab in Delhi as a Research Staff Member, where his focus of research was primarily on distributed and pervasive systems. During his tenure at IBM Research (2000 – 2005), Ashish received a First Invention Achievement award, two Plateau Invention Achievement awards, and a IBM Bravo award.

In 2005, Ashish joined the Department of Computer Science at Purdue University, West Lafayette, in order to pursue a Ph.D. in Computer Science. Since then he has
been affiliated with the Center for Education and Research in Information Assurance and Security (CERIAS) as well. In spring, 2006, he started working with Prof. Elisa Bertino, his advisor, on the problem of “Authentication of Trees and Graphs without Leaking”. In his doctoral research, Ashish developed structural signatures and leakage-free redactable signatures for trees, graphs and forests, that have direct applications in the context of cloud computing. The first paper he co-authored with Prof. Bertino on this research topic was adjudged as the Best Student Paper at the 2006 IEEE Enterprise Computing conference. Ashish’s work on information leakage in the context of trees/graphs and web services, as well as on dishonest software, have also been recognized as the Best Research Posters at some of the Annual CERIAS Symposia.

In summer 2008, he interned at Google, Mountain View, California, and developed a framework for mobile ad delivery on Android-based smart-phones. In spring 2010, he interned at EMC², Santa Clara, California, where his works on dynamic performance tracing in storage kernels were incorporated in a product. While at Purdue, he served on technical program committees of the IEEE EDOC, DEXA and IDEAS. Ashish defended his doctoral dissertation in November 2010, and received the Ph.D. in December 2010.