Elliptic Curve Cryptography
for
Lightweight Applications

by

Yvonne Roslyn Hitchcock

Bachelor of Applied Science (Mathematics)
Bachelor of Information Technology

Thesis submitted in accordance with the regulations for
Degree of Doctor of Philosophy

Information Security Research Centre
Faculty of Information Technology
Queensland University of Technology

December 2003
Keywords

Elliptic curve (EC), elliptic curve cryptosystem (ECC), discrete logarithm problem (DLP), elliptic curve discrete logarithm problem (ECDLP), speed, code size, memory usage, RAM usage, fixed curve, random curve, elliptic curve digital signature algorithm (ECDSA), Pollard’s rho method, Shanks’s baby-step giant-step method (BSGS), Canetti-Krawczyk proof model, key exchange protocols, security proof, tripartite key exchange, pairings, password-based authentication, point addition, projective coordinates, Jacobian coordinates, Chudnovsky Jacobian coordinates, modified Jacobian coordinates, binary method, simultaneous multiple exponentiation, two-in-one scalar multiplication, coprocessor, smart card, lightweight device, simple power analysis (SPA), side channel attack, equivalent security.
Abstract

Elliptic curves were first proposed as a basis for public key cryptography in the mid 1980’s. They provide public key cryptosystems based on the difficulty of the elliptic curve discrete logarithm problem (ECDLP), which is so called because of its similarity to the discrete logarithm problem (DLP) over the integers modulo a large prime. One benefit of elliptic curve cryptosystems (ECCs) is that they can use a much shorter key length than other public key cryptosystems to provide an equivalent level of security. For example, 160 bit ECCs are believed to provide about the same level of security as 1024 bit RSA. Also, the level of security provided by an ECC increases faster with key size than for integer based discrete logarithm (DL) or RSA cryptosystems. ECCs can also provide a faster implementation than RSA or DL systems, and use less bandwidth and power. These issues can be crucial in lightweight applications such as smart cards. In the last few years, ECCs have been included or proposed for inclusion in internationally recognized standards. Thus elliptic curve cryptography is set to become an integral part of lightweight applications in the immediate future.

This thesis presents an analysis of several important issues for ECCs on lightweight devices. It begins with an introduction to elliptic curves and the algorithms required to implement an ECC. It then gives an analysis of the speed, code size and memory usage of various possible implementation options. Enough details are presented to enable an implementor to choose for implementation those algorithms which give the greatest speed whilst conforming to the code size and RAM restrictions of a particular lightweight device. Recommendations are made for new functions to be included on coprocessors for lightweight devices to support ECC implementations.

Another issue of concern for implementors is the side-channel attacks that have recently been proposed. They obtain information about the cryptosystem by measuring side-channel information such as power consumption and process-
ing time and the information is then used to break implementations that have not incorporated appropriate defences. A new method of defence to protect an implementation from the simple power analysis (SPA) method of attack is presented in this thesis. It requires 44% fewer additions and 11% more doublings than the commonly recommended defence of performing a point addition in every loop of the binary scalar multiplication algorithm. The algorithm forms a contribution to the current range of possible SPA defences which has a good speed but low memory usage.

Another topic of paramount importance to ECCs for lightweight applications is whether the security of fixed curves is equivalent to that of random curves. Because of the inability of lightweight devices to generate secure random curves, fixed curves are used in such devices. These curves provide the additional advantage of requiring less bandwidth, code size and processing time. However, it is intuitively obvious that a large precomputation to aid in the breaking of the elliptic curve discrete logarithm problem (ECDLP) can be made for a fixed curve which would be unavailable for a random curve. Therefore, it would appear that fixed curves are less secure than random curves, but quantifying the loss of security is much more difficult. The thesis performs an examination of fixed curve security taking this observation into account, and includes a definition of equivalent security and an analysis of a variation of Pollard’s rho method where computations from solutions of previous ECDLPS can be used to solve subsequent ECDLPS on the same curve. A lower bound on the expected time to solve such ECDLPS using this method is presented, as well as an approximation of the expected time remaining to solve an ECDLP when a given size of precomputation is available. It is concluded that adding a total of 11 bits to the size of a fixed curve provides an equivalent level of security compared to random curves.

The final part of the thesis deals with proofs of security of key exchange protocols in the Canetti-Krawczyk proof model. This model has been used since it offers the advantage of a modular proof with reusable components. Firstly a password-based authentication mechanism and its security proof are discussed, followed by an analysis of the use of the authentication mechanism in key exchange protocols. The Canetti-Krawczyk model is then used to examine secure tripartite (three party) key exchange protocols. Tripartite key exchange protocols are particularly suited to ECCs because of the availability of bilinear mappings on elliptic curves, which allow more efficient tripartite key exchange protocols.
# Table of Contents

Keywords iii  
Abstract v  
Declaration xxi  
Previously Published Material xxiii  
Acknowledgements xxv  

## 1 Introduction  
1.1 Aims and Outcomes ........................................... 2  
1.2 Overview of Thesis ........................................... 4  

## 2 Elliptic Curve Overview  
2.1 Definition and Fundamental Concepts .......................... 6  
2.2 Curve Operations ............................................. 8  
2.2.1 Field Arithmetic ........................................... 8  
2.2.1.1 Modular Reduction for Addition and Subtraction 9  
2.2.1.2 Modular Reduction for Squaring and Multiplication 10  
2.2.1.3 Modular Inversion ....................................... 14  
2.2.1.4 Modular Square Root Algorithm ......................... 16  
2.2.2 Point Addition, Doubling and Representation ............ 16  
2.2.3 Scalar Multiplication ...................................... 19  
2.3 Security Aspects ............................................. 22  
2.4 Cryptographic Algorithms and Protocols .................... 25  
2.4.1 Diffie-Hellman Key Exchange ................................ 27  
2.4.2 ElGamal Encryption ....................................... 27  
2.4.3 Elliptic Curve Digital Signature Algorithm ............ 28
# Smart Card Implementation of an Elliptic Curve Cryptosystem

## Field Arithmetic
- **3.1.1 Selection of the Modular Reduction Algorithm**
- **3.1.2 Modular Inversion**

## Point Coordinates
- **3.2.1 Affine Coordinates**
- **3.2.2 Projective Coordinates**
- **3.2.3 Jacobian Coordinates and Variants**
- **3.2.4 Mixed Coordinates**
- **3.2.5 Addition and Doubling Algorithms**
- **3.2.6 Point Addition and Doubling Efficiency Comparison**

## Scalar Multiplication

## Comparison of RSA with ECDSA

## Recommendations for Future Coprocessors
- **3.5.1 Reduction of $f$ modulo $r$**
- **3.5.2 Inversion of $z$ modulo $p$**
- **3.5.3 Division by 2 Modulo $p$ and/or Right Shift**
- **3.5.4 Increment and Decrement**
- **3.5.5 Modular Arithmetic Using a Special Modulus**
- **3.5.6 Addition, Doubling and Scalar Multiplication of Points in Hardware**
- **3.5.7 Conversion to and from NAF**
- **3.5.8 Uncompression of Points or Square Root Algorithm**

## Conclusion
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2</td>
<td>Montgomery Ladder</td>
<td>81</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Möller’s Windowing Method</td>
<td>83</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Universal Exponentiation Algorithm</td>
<td>83</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Using the Same Algorithm for EC Addition and Doubling</td>
<td>84</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Splitting EC Point Addition</td>
<td>84</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Randomized Addition-Subtraction Chain</td>
<td>85</td>
</tr>
<tr>
<td>4.2.8</td>
<td>Width-(w) NAF Method</td>
<td>85</td>
</tr>
<tr>
<td>4.2.9</td>
<td>Non-deterministic Right-to-Left Method with Precomputation</td>
<td>85</td>
</tr>
<tr>
<td>4.3</td>
<td>New SPA Defence and Efficiency Analysis</td>
<td>87</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Description of New Defence</td>
<td>88</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Stopping Information Leakage</td>
<td>91</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Efficiency Analysis</td>
<td>98</td>
</tr>
<tr>
<td>4.4</td>
<td>Conclusion</td>
<td>104</td>
</tr>
<tr>
<td>5</td>
<td>The Security of Fixed versus Random Elliptic Curves</td>
<td>107</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of Fixed Curve Benefits</td>
<td>108</td>
</tr>
<tr>
<td>5.2</td>
<td>Existing Methods of Attack</td>
<td>110</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Pohlig-Hellman Algorithm</td>
<td>110</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Index Calculus and Related Methods</td>
<td>110</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Shanks’s Baby-Step Giant-Step Method</td>
<td>110</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Pollard’s Rho Method</td>
<td>111</td>
</tr>
<tr>
<td>5.3</td>
<td>Security Comparison</td>
<td>127</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Equivalent Security I</td>
<td>127</td>
</tr>
<tr>
<td>5.3.1.1</td>
<td>BSGS and ES-I</td>
<td>128</td>
</tr>
<tr>
<td>5.3.1.2</td>
<td>Pollard’s Rho and ES-I</td>
<td>128</td>
</tr>
<tr>
<td>5.3.1.3</td>
<td>ES-I Result</td>
<td>129</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Equivalent Security II</td>
<td>129</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Curves with Special Properties</td>
<td>134</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Special Purpose Hardware</td>
<td>134</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Results of Analysis and Performance Effects</td>
<td>135</td>
</tr>
<tr>
<td>5.4</td>
<td>Conclusion</td>
<td>136</td>
</tr>
<tr>
<td>6</td>
<td>Provably Secure Elliptic Curve Protocols</td>
<td>139</td>
</tr>
<tr>
<td>6.1</td>
<td>Overview of the Canetti-Krawczyk Approach</td>
<td>140</td>
</tr>
<tr>
<td>6.2</td>
<td>Password-Based Protocols Secure in the CK Model</td>
<td>146</td>
</tr>
</tbody>
</table>
6.2.1 Password-Based Protocols and Their Constraints ........................................ 146
6.2.2 The Authenticator ..................................................................................... 148
   6.2.2.1 The Parties ....................................................................................... 149
   6.2.2.2 Initialization ...................................................................................... 150
   6.2.2.3 Protocol Description ......................................................................... 150
6.2.3 Proof of Security of Authenticator .............................................................. 151
   6.2.3.1 Variations of $\lambda_{P-ENC}$ ................................................................ 158
      6.2.3.1.1 Encryption Indistinguishable under Chosen Plaintext Attack .... 158
      6.2.3.1.2 Using a Public Password .......................................................... 159
6.2.4 Some Applications of the Authenticator ...................................................... 159
   6.2.4.1 Existing Components in the CK-model ................................................. 160
      6.2.4.1.1 Provably Secure Protocols in the AM ........................................ 160
         6.2.4.1.1.1 The Diffie-Hellman Key Exchange ...................................... 161
         6.2.4.1.1.2 The ENC Protocol .............................................................. 162
      6.2.4.1.2 An Encryption-based MT-authenticator ...................................... 162
   6.2.4.2 Key Exchanges of Halevi and Krawczyk .............................................. 163
      6.2.4.2.1 Mutual Authentication and Key Exchange ................................. 164
      6.2.4.2.2 Mutual Authentication and Diffie-Hellman Key Exchange .......... 164
   6.2.4.3 Application of $\lambda_{P-ENC}$ ............................................................... 165
      6.2.4.3.1 Application of $\lambda_{P-ENC}$ to 2DH ......................................... 165
      6.2.4.3.2 Application of $\lambda_{P-ENC}$ to ENC ......................................... 169
   6.2.4.4 Protocol Performance Comparison ...................................................... 170
      6.2.4.4.1 Computational Requirements ..................................................... 170
      6.2.4.4.2 Number of Message Flows ......................................................... 171
6.3 Tripartite Key Exchange .............................................................................. 172
   6.3.1 Definition of Secure Tripartite Key Exchange ......................................... 173
   6.3.2 Tripartite Key Exchange Protocol in the AM ......................................... 175
   6.3.3 Applying Authenticators to the Joux Protocol ........................................ 180
   6.3.4 Efficiency of Joux Based Protocols in the UM ...................................... 181
6.4 Conclusion ................................................................................................. 191

7 Conclusion ..................................................................................................... 193
   7.1 Summary of Research .............................................................................. 194
   7.2 Directions for Future Research ................................................................. 196
A  Definitions and Notational Conventions  199
   A.1  Indistinguishability of an Encryption Scheme . . . . . . . . . . . . . . . . . . . . . . 200
   A.2  Left-or-Right Indistinguishability . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 202
   A.3  Equivalence of Indistinguishability Definitions . . . . . . . . . . . . . . . . . . . . . . 203

Bibliography  205
List of Figures

2.1 Dependencies of elliptic curve operations .......................... 9
2.2 Graphical illustration of addition and doubling of elliptic curve points ...................................................... 17

3.1 Jacobian 1 point addition—Stage 1 .................................. 57
3.2 Jacobian 1 point addition—Stage 2 .................................. 58
3.3 Jacobian 1 point addition—Stage 3 .................................. 59
3.4 Jacobian 1 point addition—Stage 4 .................................. 60
3.5 Jacobian 1 point addition—Stage 5 .................................. 61
3.6 Jacobian 1 point addition—Stage 6 .................................. 62
3.7 Jacobian 1 point addition—Stage 7 .................................. 63
3.8 Number of variables saved for the different options for ECDSA ......................................................... 68
3.9 ECDSA signature and verification timings on the smart card simulator ..................................................... 69
3.10 Code size for ECDSA signature and verification for the smart card ...................................................... 70

4.1 Transition matrix $A$ ......................................................... 97

5.1 Actual iterations to solve 50 ECDLPs on a 25-bit curve averaged over 200 trials with 1 in 400 points distinguished and $s = 5$ .................................. 115
5.2 Actual iterations to solve 50 ECDLPs on a 32-bit curve, averaged over 200 trials, with $s = 21$ and 1 in 800 points distinguished. The theoretical bound from (5.4) is also shown ............................................ 125

6.1 Graphical representation of an authenticator .......................... 141
List of Tables

2.1 Timings of Montgomery multiplication and reduction for a 32 bit word size in milliseconds ........................................ 13
2.2 Timings of classical (using Knuth’s algorithm), Barrett and Montgomery modular reduction [BGV94] .......................... 13
2.3 Timings of various inversion algorithms ................................... 16
2.4 Equivalent key sizes from various sources ................................. 24
2.5 Elliptic curve notation .......................................................... 26

3.1 Point conversion complexity .................................................. 49
3.2 Addition and doubling efficiencies ....................................... 64
3.3 Estimated time for signed (NAF) and unsigned scalar multiplication on the smart card using the binary method ...................... 66
3.4 Code size, number of variables saved and timings for the smart card ........................................................................... 71
3.5 ECDSA and RSA time comparison on smart card and Pentium . 72

4.1 States of the Markov chain model .......................................... 95
4.2 Probability of each protected NAF length for a 160 bit scalar . 97
4.3 Minimum, expected and maximum values for the ratio of the total number of point additions and doublings required by one algorithm to that of another for various algorithms .................................. 101
4.4 Comparison of expected number of additions and doublings .... 103

5.1 Definitions for Pollard’s rho method ..................................... 112
5.2 Loss in security of a fixed curve according to ES-II ................. 132
5.3 Time in milliseconds of elliptic curve scalar multiplication using the MIRACL library [Sha00] on a Pentium IV 1.8 GHz .......... 136

6.1 Possible provably secure protocols in the UM using $\lambda_{P-ENC}$ ................................................................. 160
6.2 Protocol computational loads (where A is the client and B is the server) ................................. 170
6.3 Operations and messages required by tripartite UM protocols .......................... 187
6.4 Efficiency of signature schemes using pairings ................................................. 188
6.5 Efficiency of encryption schemes using pairings ............................................. 188
6.6 Operations and messages required by Al-Riyami and Paterson’s tripartite protocols compared with Protocol 6.27 ............................... 191
# List of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Modular addition [MvOV96]</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Modular subtraction [MvOV96]</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Barrett reduction</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Montgomery reduction</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Montgomery multiplication</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>Inversion based on the extended Euclidean algorithm</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>Inversion based on Fermat’s (little) theorem</td>
<td>14</td>
</tr>
<tr>
<td>2.8</td>
<td>Inversion based on the binary extended GCD algorithm</td>
<td>15</td>
</tr>
<tr>
<td>2.9</td>
<td>Modular square root algorithm [MvOV96]</td>
<td>16</td>
</tr>
<tr>
<td>2.10</td>
<td>Addition for points in affine coordinates</td>
<td>18</td>
</tr>
<tr>
<td>2.11</td>
<td>Doubling for points in affine coordinates</td>
<td>18</td>
</tr>
<tr>
<td>2.12</td>
<td>Binary scalar multiplication</td>
<td>19</td>
</tr>
<tr>
<td>2.13</td>
<td>Conversion of a scalar to NAF format</td>
<td>20</td>
</tr>
<tr>
<td>2.14</td>
<td>Signed binary scalar multiplication</td>
<td>20</td>
</tr>
<tr>
<td>2.15</td>
<td>Two-in-one scalar multiplication</td>
<td>21</td>
</tr>
<tr>
<td>2.16</td>
<td>Generation of a secure elliptic curve</td>
<td>23</td>
</tr>
<tr>
<td>2.17</td>
<td>ECDSA signature</td>
<td>31</td>
</tr>
<tr>
<td>2.18</td>
<td>ECDSA verification</td>
<td>32</td>
</tr>
<tr>
<td>3.1</td>
<td>Projective doubling</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Projective addition</td>
<td>51</td>
</tr>
<tr>
<td>3.3</td>
<td>Jacobian 1 addition</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Jacobian 3 addition</td>
<td>52</td>
</tr>
<tr>
<td>3.5</td>
<td>Jacobian 1 doubling</td>
<td>52</td>
</tr>
<tr>
<td>3.6</td>
<td>Chudnovsky Jacobian addition</td>
<td>53</td>
</tr>
<tr>
<td>3.7</td>
<td>Chudnovsky Jacobian doubling</td>
<td>53</td>
</tr>
<tr>
<td>3.8</td>
<td>Modified Jacobian and variants addition</td>
<td>54</td>
</tr>
<tr>
<td>3.9</td>
<td>Modified Jacobian and variants doubling</td>
<td>54</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.10</td>
<td>Jacobian 1 addition with non-optimal variable usage</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>Binary scalar multiplication algorithm defended against SPA (double and add always)</td>
<td>82</td>
</tr>
<tr>
<td>4.2</td>
<td>Width-$w$ NAF resistant to SPA formatting algorithm for scalars</td>
<td>86</td>
</tr>
<tr>
<td>4.3</td>
<td>Scalar multiplication algorithm using width-$w$ NAF</td>
<td>87</td>
</tr>
<tr>
<td>4.4</td>
<td>Protected NAF formatting algorithm resistant to SPA</td>
<td>90</td>
</tr>
<tr>
<td>4.5</td>
<td>New binary scalar multiplication algorithm for protected NAF scalar</td>
<td>91</td>
</tr>
<tr>
<td>6.1</td>
<td>HMAC function [BCK96]</td>
<td>163</td>
</tr>
<tr>
<td>6.2</td>
<td>Building a distinguisher for DBDH</td>
<td>178</td>
</tr>
</tbody>
</table>
List of Protocols

2.1 Diffie-Hellman key exchange ........................................... 27
2.2 ElGamal encryption ....................................................... 28
2.3 PSEC-1 encryption ....................................................... 29
6.1 Halevi and Krawczyk’s Encrypted Challenge-Response protocol . 148
6.2 Password based authenticator, $\lambda_{P-ENC}$ ..................................... 149
6.3 2DH (Diffie-Hellman) .................................................... 161
6.4 ENC (Encryption based key exchange) .................................... 162
6.5 Encryption-based MT-authenticator, $\lambda_{ENC}$ .......................... 162
6.6 HK ................................................................. 164
6.7 HKDH ............................................................... 165
6.8 Unoptimized 2DHPE .................................................... 167
6.9 Optimized 2DHPE using a separate nonce .............................. 167
6.10 Optimized 2DHPE using Diffie-Hellman value as nonce ........... 168
6.11 Optimized 2DHPE using a public password and separate nonce . 168
6.12 ENCP ............................................................... 169
6.13 ENCP using a public password ........................................ 169
6.14 Joux broadcast protocol ............................................... 175
6.15 Joux protocol in the AM without broadcast messages .............. 176
6.16 Variant of Joux protocol in the AM that can be used with authenti-

cators to create efficient UM protocols ................................... 176
6.17 Signature-based MT-authenticator, $\lambda_{SIG}$ ........................... 180
6.18 Joux protocol authenticated with $\lambda_{SIG}$, not optimized .......... 182
6.19 Joux protocol authenticated with $\lambda_{SIG}$ ............................ 182
6.20 Joux protocol authenticated with $\lambda_{SIG}$ using a minimal number of

signatures ........................................................................... 183
6.21 Joux protocol authenticated with $\lambda_{SIG}$, broadcast version .... 183
6.22 Variant of Joux protocol authenticated with $\lambda_{SIG}$, not optimized . 184
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.23</td>
<td>Variant of Joux protocol authenticated with $\lambda_{\text{SIG}}$</td>
<td>184</td>
</tr>
<tr>
<td>6.24</td>
<td>Joux protocol authenticated with $\lambda_{\text{ENC}}$, not optimized</td>
<td>185</td>
</tr>
<tr>
<td>6.25</td>
<td>Joux protocol authenticated with $\lambda_{\text{ENC}}$</td>
<td>186</td>
</tr>
<tr>
<td>6.26</td>
<td>Variant of Joux protocol authenticated with $\lambda_{\text{ENC}}$, not optimized</td>
<td>186</td>
</tr>
<tr>
<td>6.27</td>
<td>Variant of Joux protocol authenticated with $\lambda_{\text{ENC}}$</td>
<td>187</td>
</tr>
</tbody>
</table>
Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed:.......................... Date:......................
Previously Published Material

The following papers have been published or presented, and contain material based on the content of this thesis.


Acknowledgements

Many people have provided their assistance to me during the completion of this PhD, in one way or another making a contribution to the quality of the final product. My thanks go firstly to my principal supervisor, Professor Ed Dawson. He has provided support, encouragement and advice throughout the last three and a half years, as well guiding me through the University reporting procedures and ensuring the research remained on track. Although a busy man, he always found the time to proofread and suggest improvements to many documents, often for several different revisions and within a very short space of time.

My thanks also go to my associate supervisor, Dr. Andrew Clark, particularly for his support throughout the early stages of my PhD. He was always happy to answer my numerous questions on the best programming techniques and algorithms to use, and his friendly and helpful attitude has made my time of study more enjoyable. Likewise, I would like to thank Dr. Paul Montague, also an associate supervisor, for his assistance and support throughout the last three and a half years. His knowledge of the smart card used in the first part of the research proved invaluable, and he answered the many questions I had whilst programming for it. He also provided a valuable contribution to those parts of the thesis involving mathematics, particularly the new efficiency results for Pollard’s rho method and the security proof of the password-based authenticator; his work forms a significant contribution to the quality of this thesis.

My thanks also go to Dr. Gary Carter who worked closely with me on the comparison of the security of fixed and random curves. I greatly value his input and support during this stage of my research, as well as his willingness to proofread many revisions of the work and his high-quality suggestions for improvements.

I would also like to thank Associate Professor Colin Boyd, Dr. Juan Manuel González Nieto and Mr. Yiu Shing Terry Tin for their contribution to the work in Chapter 6. Whilst I carried out the definition and security proof of the password-
based authenticator, the basic analysis of the use and efficiency of the authenticator was mainly performed by Terry. In addition, Colin and Juanma provided numerous helpful comments and suggestions on the work, as well as guidance for that part of the research.

The many other members of the Information Security Research Centre, both staff and students, have all helped to make my time here more interesting and enjoyable, through their friendliness, helpfulness and participation in stimulating discussions on research related topics. There are too many to name individually, so I thank them all collectively. I also gratefully acknowledge the Australian Research Council, Motorola and Queensland University of Technology who provided funding through an ARC SPIRIT project (number C10024103).

Finally, my greatest thanks go to my parents, without whom this thesis would never have been written. As well as providing their love and support, both of them have made and continue to make numerous and priceless sacrifices spanning many years to ensure that each of their seven children receives the education my father never had. This manuscript is a witness to their unfailing faith, hope and love.
Chapter 1

Introduction

Elliptic curves were first proposed as a basis for public key cryptography in the mid 1980's independently by Koblitz [Kob87] and Miller [Mil86]. Elliptic curves provide discrete log based public key cryptosystems and can use a much shorter key length than other public key cryptosystems to provide an equivalent level of security. For example, 160 bit elliptic curve cryptosystems (ECCs) provide about the same level of security as 1024 bit RSA [CMO98, p.51], [RSA78]. Also, the rate at which key size increases in order to obtain increased security is much slower than the rate at which discrete log (DL) [DH76, ElG85] and RSA key sizes must be increased for the same increase in security. ECCs can also provide a faster implementation than RSA or DL systems, and also use less bandwidth and power [HNM98]. These issues can be crucial in lightweight applications such as smart cards.

In the last few years, confidence in the security of ECCs has also risen, to the point where they have now been included or proposed for inclusion in internationally recognized standards (specifically IEEE Std 1363, WAP (Wireless Application Protocol), ANSI X9.62, ANSI X9.63 and ISO CD 14888-3). Thus elliptic curve cryptography is set to become an integral part of lightweight applications in the immediate future.
1.1 Aims and Outcomes

The aim of this research is to develop an efficient elliptic curve infrastructure and investigate applications of elliptic curves, particularly those suitable for lightweight devices. This is achieved using a three-way approach. Firstly, a detailed description of an implementation of an ECC for a smart card is presented, followed by a method of resisting simple power analysis on a smart card. Secondly, the security of fixed versus random elliptic curves is investigated, since this issue is crucial for lightweight applications. Thirdly, the security of elliptic curve protocols is investigated using a formal proof model.

The main contributions of this thesis are as follows. Firstly, precise details of the efficiency of various elliptic curve point coordinate systems as well as the RAM usage, code size and speed of an elliptic curve digital signature and verification algorithm are provided. It is observed that a higher speed of ECC requires higher RAM usage and code size, and on some lightweight devices, RAM and code size restrictions may preclude faster implementations. The research considers different implementation options, and presents the RAM usage, code size and speed of each implementation option. This allows an implementor to choose the fastest implementation option that will still meet memory and code size restrictions on a lightweight device. Also provided are detailed elliptic curve addition and doubling algorithms with low memory usage and suitable for implementation. Although the mathematical formulae upon which such algorithms are based are available in the literature, detailed algorithms with low memory usage suitable for implementation (such as those provided here) have not previously been published for some methods of point representation, or have required the use of more memory than those presented here. In addition, recommendations of functions for future coprocessors to enable efficient elliptic curve implementations on smart cards are provided, as well as an innovative elliptic curve scalar multiplication algorithm with low memory usage and high speed, resistant to simple power analysis attacks. This work resulted in two publications, [1] and [2] (details of publications are provided on page xxiii).

Secondly, an analysis of the security of fixed versus random elliptic curves is provided, which is an important issue for constrained devices, e.g. smart cards, mobile telephones, personal digital assistants (PDAs), etc. It is intuitively obvious that a large precomputation to aid in the breaking of the elliptic curve discrete logarithm problem (ECDLP) can be made for a fixed curve. However, in the case
of a random curve, it is likely that a much smaller amount of computing power is available. Therefore, it would appear that fixed curves are less secure than random curves, but quantifying the loss of security is much more difficult. On the other hand, implementations using fixed curves can have many advantages over those using random curves, such as using less bandwidth, code size and processing time. Since fixed curves are so attractive from an implementation point of view and have been included in various standards, their security compared to that of random curves is examined in detail. This comparison required a close study of Pollard’s rho algorithm for solving the ECDLP, and some new results are presented. A lower bound on the expected number of iterations to solve multiple ECDLPs using Pollard’s rho method is proven, followed by an approximation of the number of iterations expected to be required to solve an ECDLP given the existence of a precomputation of a certain size. These results are used in conjunction with a definition of “equivalent security” proposed by this research to conclude that increasing the curve order of a fixed curve by approximately 11 bits is sufficient to provide equivalent security to a random curve. This work forms the first study of the security of fixed versus random elliptic curves in the literature [3].

Finally, some new proofs of security of protocol components in a formal proof model are presented. The Canetti-Krawczyk proof model [CK01a] was chosen for this work since it offers the advantage of a modular proof with reusable components. The first part of this work is the definition and proof of security of a password-based authentication mechanism, followed by an analysis of the use and efficiency of the authentication mechanism in provably secure key exchange protocols. This part of the work has been published as [4]. The second part of this work is the use of the Canetti-Krawczyk model to examine secure tripartite (three party) key exchange protocols. Tripartite key exchange protocols are particularly suited to ECCs because of the availability of bilinear mappings on elliptic curves, which allow more efficient tripartite key exchange protocols. A new definition of security of tripartite key exchange is provided, accompanied by a proof of security of an existing tripartite key exchange protocol. This is followed by an analysis of the efficiency of the protocol in conjunction with various authentication mechanisms.
1.2 Overview of Thesis

The remainder of this dissertation is structured as follows. Chapter 2 provides an overview of elliptic curves, including their definition, a description of fundamental concepts, curve operations and security aspects and an overview of the existing cryptographic algorithms and protocols used in other parts of the dissertation.

Chapter 3 analyses the implementation options available for an elliptic curve cryptosystem on a smart card in terms of efficiency, code size and memory usage, and includes recommendations for the functionality of future coprocessors and elliptic curve point addition and doubling algorithms with low memory usage.

Chapter 4 provides a description and analysis of the security of a new method for defending an elliptic curve cryptosystem against simple power analysis.

Chapter 5 provides the analysis of the security of fixed versus random elliptic curves, including new results on the efficiency of Pollard’s rho method.

Chapter 6 provides an overview of the Canetti-Krawczyk proof model, followed by the definition and proof of security of a password-based authenticator and an analysis of its efficiency in provably secure protocols. The chapter also contains the definition and security proof of a tripartite key exchange protocol and an analysis of its efficiency when combined with various authentication mechanisms.

Chapter 7 concludes the dissertation by summarizing the main achievements and outlining areas of possible future research. An appendix is provided containing additional background information. The reader is referred to the information contained therein at appropriate points throughout the thesis.
Chapter 2

Elliptic Curve Overview

In this dissertation various aspects of ECCs over the Galois field with $p$ elements ($GF(p)$ [LN86]) (where $p$ is prime) are studied. In the past, much research has focused on curves over the Galois field with $2^m$ elements, $GF(2^m)$, because it is possible to create efficient hardware implementations [Cer98]. However, because of the speed advantages of elliptic curves over the field $GF(p)$ compared to $GF(2^m)$ when a crypto coprocessor for modular arithmetic is available [DeWMPW98], and because of patent issues associated with curves over $GF(2^m)$, this research has investigated curves over $GF(p)$. The scope of this research is restricted to general elliptic curves over $GF(p)$. That is, curves such as Montgomery or Hessian elliptic curves [OKS00, XB01] which have a special form are not considered by this research.

The remainder of this chapter provides some introductory material for elliptic curve cryptosystems and is organized as follows. Firstly, the definition of an elliptic curve over $GF(p)$ is presented, followed by a description of some fundamental concepts of an ECC. Next, the basic curve operations are described, and some security aspects of an ECC discussed. Finally, details are provided of some basic cryptographic algorithms and protocols used throughout this dissertation and applicable to elliptic curves.
2.1 Definition and Fundamental Concepts

An elliptic curve (EC) over $GF(p)$ where $p$ is prime and $p > 3$ may be defined [BSS99] as the points $(x, y)$ satisfying the curve equation

$$E : y^2 \equiv x^3 + ax + b \pmod{p},$$

(2.1)

where $a$ and $b$ are constants satisfying $4a^3 + 27b^2 \equiv 0 \pmod{p}$. Points satisfying this equation are known as affine points. Other kinds of points are discussed in Section 2.2. In addition to the points satisfying the curve equation $E$, a point at infinity, $\infty$, is also defined.

A suitable definition of addition of points (see Section 2.2) enables the points of an elliptic curve to form a finite abelian group with addition and doubling of points being the group operation, and the point at infinity being the identity element. Various protocols to provide authenticity and confidentiality through the use of a finite abelian group (such as the one provided by an elliptic curve) are available, some of which are described in Section 2.4. However, such protocols require the use of a higher level operation than point addition, which (in the case of elliptic curves) is called scalar multiplication and defined in Definition 2.1.

**Definition 2.1 (Scalar multiplication).** Scalar multiplication of a point $P$ by a scalar $k$ is denoted $[k]P$ and defined as:

$$[k]P = P + P + \ldots + P \ (k \text{ summands}).$$

(2.2)

This operation can be likened to the exponentiation of a DL-based cryptosystem. In fact, most algorithms originally proposed for DL-based cryptosystems, such as Diffie-Hellman key exchange [DH76] or the Digital Signature Algorithm (DSA) [NIS00] can be converted to a form suitable for use by ECCs.

The security of ECCs rests on the infeasibility of the elliptic curve discrete logarithm problem described in Definition 2.2.

**Definition 2.2 (Elliptic curve discrete logarithm problem).** Given a prime modulus $p$, curve constants $a$ and $b$ and two points $P$ and $Q$ on the corresponding curve, the elliptic curve discrete logarithm problem (ECDLP) is to find a scalar $k$ such that $Q = [k]P$. The value $k$ is known as the elliptic curve discrete logarithm (ECDL).

Secure elliptic curves are those for which the ECDLP is infeasible and this infeasi-
bility enables scalar multiplication to be the basic cryptographic operation of an elliptic curve.

For the ECDLP to be infeasible on a particular EC, it is necessary to ensure that several conditions are met (see Section 2.3 for a list of requirements). One of these is to ensure that the number of points on the curve (also known as the order of the curve and denoted $\#E(GF(p))$ where the curve is over $GF(p)$) has a large prime factor. One method to meet this requirement is to use the complex multiplication method to generate the curve. Although the method is fast, the curves produced have a special property and there is concern that although no attack is currently known which exploits the property, such an attack may be proposed in the future [LZ94, BSS99].

Another method is to use a subfield curve (also known as a curve of Koblitz type) [Kob98, BSS99]. Such a curve is generated by first choosing a curve over a fairly small Galois field, $GF(q)$ (where $q$ is not necessarily prime), and counting the number of points on it. This is fairly easy because of the small size of the curve. The next step is to use a formula to find the number of points on the same curve over the extension field, $GF(q^m)$. Obviously, this method is not suitable for generating curves over $GF(p)$ where $p$ is prime. In addition, curves over composite extension fields (fields over $GF(p^m)$ where $m$ is not prime) can be less secure than curves over the fields $GF(p)$ and $GF(2^r)$ (where $p$ and $r$ are prime) due to the method of Weil descent for solving the ECDLP [Sma00].

The third method is to choose a prime $p$ and values $a$ and $b$ (where $a$ and $p$ may be either random or chosen to increase efficiency) and use a point counting algorithm to find the order of the curve. The most efficient algorithm currently known for this purpose is the Schoof-Elkies-Atkin (SEA) algorithm [Sch85, Sch95, Elk98, Ler97, M"{u}1995, BSS99]. Whilst it is believed that a more secure curve will be generated using this method, it also requires a much greater amount of time. For example, one implementation takes 2-3 minutes on a 180 MHz Pentium Pro to count the points on a 160 bit curve and 3.5 - 5.5 minutes for a 192 bit curve [Sco99]. On a smart card platform, it would take much longer—a 10 MHz smart card could be expected to take at least 36 minutes to count the points on a 160 bit curve based on processor speed. In addition, even if users were prepared to accept the slow speed of the method, the code size and RAM requirements render it infeasible to implement on a smart card. The above difficulties associated with point counting have led to the inclusion in various standards documents of “fixed
Chapter 2. Elliptic Curve Overview

The issues associated with fixed curves are discussed further in Section 2.3 and Chapter 5.

2.2 Curve Operations

Once a secure curve has been chosen as the basis for an elliptic curve cryptosystem, the curve operations must be implemented. There are three main levels of arithmetic. The lowest level is the field or modular arithmetic. This is then used in order to implement the EC point addition and doubling routines. The addition and doubling are then used as basic components to implement scalar multiplication. Finally, scalar multiplication can be used in cryptographic protocols such as key exchange or digital signatures. These dependencies are shown graphically in Figure 2.1. The following subsections discuss each level of arithmetic in turn.

2.2.1 Field Arithmetic

The most basic level of arithmetic is the field arithmetic, namely arithmetic in the field $GF(p)$, which is the same as arithmetic modulo $p$. Two basic concepts of modular arithmetic are the notions of modular reduction and modular equivalence. Reducing a number $x \pmod{p}$ can be defined as finding the remainder $r$ when $x$ is divided by $p$. We then write $x \equiv r \pmod{p}$ and say that $x$ is equivalent to $r$ modulo $p$. In general, if the difference between $x$ and $z$ is a multiple of $p$ (i.e. $x - z = kp$ where $k$ is a constant integer) then $z \equiv x \pmod{p}$. Thus every integer less than 0 or greater than $p - 1$ is equivalent to some other integer between 0 and $p - 1$.

After an addition or multiplication, e.g. $z = x * y$, the result $z$ is generally reduced modulo $p$; i.e. the remainder $r$ is found when $z$ is divided by $p$. This reduction is performed to keep the size of the numbers being used as small as possible. Otherwise, the size of the numbers will become substantially larger, and thus addition and multiplication will take larger amounts of time and require the use of more memory. Although the definition of modular reduction is in terms of division by the modulus, performing such a division in practice would be very slow. Therefore, other ways of achieving the same result have been devised and are discussed below.

Another operation which is required by an ECC is modular inversion. The inverse of a number $x$ can be defined as the value $x^{-1}$ such that $x^{-1}x \equiv 1$
2.2. Curve Operations

Fig. 2.1: Dependencies of elliptic curve operations

(mod p). A modular square root algorithm is also likely to be required by an ECC in order to uncompress EC points which have been stored in a compressed format. These algorithms are also discussed below.

2.2.1.1 Modular Reduction for Addition and Subtraction

Modular reduction after an addition is quite easy because the inputs are in the range 0 to \( p - 1 \), and so the result of the addition must be less than twice the modulus. Therefore, if the result is between \( p \) and \( 2p \), it can be reduced by simply subtracting \( p \), as described in [MvOV96] and shown in Algorithm 2.1. In the case of subtraction, if \( x - y \pmod{p} \) is to be found, then if \( x \geq y \) the subtraction can be done normally and the result will be between 0 and \( p - 1 \) because the inputs
were also in this range. Otherwise, $p$ should be added to $x$ before the subtraction begins, and then the result will be between 0 and $p-1$. The details (as described in [MvOV96]) are shown in Algorithm 2.2.

<table>
<thead>
<tr>
<th>Algorithm 2.1: Modular addition [MvOV96]</th>
</tr>
</thead>
<tbody>
<tr>
<td>To find $x + y \pmod p$ where $x, y \in [0, p - 1]$:</td>
</tr>
<tr>
<td>- Find $z = x + y$.</td>
</tr>
<tr>
<td>- If $z \geq p$ then let $z = z - p$.</td>
</tr>
<tr>
<td>- Return $z$. (Note $z \in [0, p - 1]$.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 2.2: Modular subtraction [MvOV96]</th>
</tr>
</thead>
<tbody>
<tr>
<td>To find $x - y \pmod p$ where $x, y \in [0, p - 1]$:</td>
</tr>
<tr>
<td>- If $x &lt; y$ let $x = x + p$.</td>
</tr>
<tr>
<td>- Find $z = x - y$.</td>
</tr>
<tr>
<td>- Return $z$. (Note $z \in [0, p - 1]$.)</td>
</tr>
</tbody>
</table>

### 2.2.1.2 Modular Reduction for Squaring and Multiplication

It is possible to perform a modular reduction after a squaring or multiplication by finding the remainder after division by the modulus. A straightforward implementation of the classical division algorithm (which computes the quotient bit-by-bit through division by the modulus of a number one bit larger than the modulus) would lead to quite a slow modular reduction. However, Knuth [Knu81, Algorithm 4.3.1A] has published a more efficient version of the classical algorithm which estimates the quotient before it is computed. Although the algorithm by Knuth is fairly efficient, there are some other options available. In the case of ECCs, because the field arithmetic uses only one modulus, it is possible to perform a precomputation to allow faster modular reduction. Two efficient methods available to do this are Barrett reduction and Montgomery reduction. They are compared below in terms of speed. It is possible to implement each of the above
three algorithms with reasonable memory usage and code size on a smart card, making speed a major factor in the final choice of algorithm.

Barrett reduction [Bar87] computes \( r \equiv x \pmod{p} \) given \( x \) and \( p \). The precomputation is the value \( \mu = \left\lfloor \frac{b^{k+1}}{p} \right\rfloor \) where \( b \) is the radix (usually the word size of the processor) and \( k \) is the number of digits in \( p \) (when using the base \( b \)). Barrett reduction is shown (as described in [MvOV96, pp.603–604]) in Algorithm 2.3.

\[
\begin{align*}
\text{Input:} & \quad \text{Positive integers } x = (x_{2k-1} \cdots x_1 x_0)_b, \\
& \quad p = (p_{k-1} \cdots p_1 p_0)_b \text{ (with } p_{k-1} \neq 0 \text{ and } p > 3) \text{ and } \\
& \quad \mu = \left\lfloor \frac{b^{k+1}}{p} \right\rfloor. \\
\text{Output:} & \quad r \equiv x \pmod{p} \\
\text{Algorithm:} & \quad q_1 = \left\lfloor \frac{x}{b^{k+1}} \right\rfloor, \\
& \quad q_2 = q_1 \cdot \mu, \\
& \quad q_3 = \left\lfloor \frac{q_2}{b^{k+1}} \right\rfloor, \\
& \quad r_1 = x \pmod{b^{k+1}}, \\
& \quad r_2 = q_3 \cdot p \pmod{b^{k+1}}, \\
& \quad r = r_1 - r_2, \\
& \quad \text{If } r < 0 \text{ then } r = r + b^{k+1}. \\
& \quad \text{While } r \geq p \text{ do: } r = r - p \\
& \quad \text{Return } r.
\end{align*}
\]

\textbf{Algorithm 2.3:} Barrett reduction

Since the divisions in the Barrett reduction algorithm can be implemented as right shifts, they are actually quite fast. It is also possible to increase the efficiency of Barrett reduction by not computing the \( k+1 \) least significant digits of \( q_2 \). This is discussed in [MvOV96] and [BGV94].

Montgomery reduction [Mon85] can be used instead of Barrett reduction, and is also slightly faster [BGV94]. It also requires a precomputation, consisting of the values \( R = b^k \) (where \( \gcd(p, R) = 1 \)) and \( p' = -p^{-1} \pmod{b} \) where \( b \) is the radix and \( k \) is the number of digits required to represent \( p \). Algorithm 2.4 describes Montgomery reduction (which performs a modular reduction on any number up to twice the size of the modulus) and Algorithm 2.5 shows Montgomery multiplication (which multiplies two numbers less than the modulus and reduces the result in the one algorithm) as described in [MvOV96]. Timings on a Pentium III 450 mHz of Montgomery multiplication and reduction for bit sizes commonly used by eccs are shown in Table 2.1. It can be seen that if a squaring is required, it is faster to square and then use Montgomery reduction than it is to perform a
Montgomery multiplication.

**Algorithm 2.4: Montgomery reduction**

Input: \(x = (x_{2k-1} \cdots x_1 x_0)_b,\)
\(y = (y_{k-1} \cdots y_1 y_0)_b,\)
\(p = (p_{k-1} \cdots p_1 p_0)_b\) (with \(0 \leq x, y < p\)),
\(R = b^k\) (with \(\gcd(p, b) = 1\)) and
\(p' = -p^{-1} \mod b).\)

Output: \(xR^{-1} \mod p)\)

Algorithm:
\begin{align*}
\text{a} &= x \quad (\text{The notation used is } a = (a_{2k-1} \cdots a_1 a_0)_b.) \\
\text{For } i \text{ from } 0 \text{ to } (k - 1) \text{ do the following:} \\
\{ & u_i = a_i \cdot p' \mod b \} \\
\{ & a = a + u_i \cdot p \cdot b^i \} \\
\text{a} &= a/b^n \\
\text{If } a \geq p \text{ then } a = a - p \\
\text{Return } a.
\end{align*}

**Algorithm 2.5: Montgomery multiplication**

Since Montgomery reduction finds \(xR^{-1} \mod p)\) instead of \(x \mod p)\), it is necessary to put the numbers into Montgomery form before the calculations are performed. The Montgomery form of a number \(x) is the number \(xR \mod p)\). When two numbers in Montgomery form are multiplied together (using the Montgomery multiplication algorithm) or a number is squared and reduced (using the Montgomery reduction algorithm), the resulting number is still in Montgomery form. The numbers can then be converted to the normal format after all of the
2.2. Curve Operations

Table 2.1: Timings of Montgomery multiplication and reduction for a 32 bit word size in milliseconds

<table>
<thead>
<tr>
<th>Length of modulus (bits)</th>
<th>Montgomery Multiplication</th>
<th>Squaring and Mont. Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>0.0039</td>
<td>0.0034</td>
</tr>
<tr>
<td>192</td>
<td>0.0056</td>
<td>0.0047</td>
</tr>
<tr>
<td>224</td>
<td>0.073</td>
<td>0.062</td>
</tr>
</tbody>
</table>

Table 2.2: Timings of classical (using Knuth’s algorithm), Barrett and Montgomery modular reduction [BGV94]

<table>
<thead>
<tr>
<th>Words in modulus</th>
<th>Length of modulus (bits)</th>
<th>Times in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Classical</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>0.278</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>0.870</td>
</tr>
<tr>
<td>32</td>
<td>512</td>
<td>3.05</td>
</tr>
</tbody>
</table>

modular computations have taken place. Of course, if there are a large number of inputs or outputs and only a comparatively low number of modular operations to perform, the time to convert the inputs to and from Montgomery form may outweigh any saving due to the faster arithmetic. This is not the case when performing elliptic curve scalar multiplication and Montgomery modular arithmetic is efficient to use in this case.

Bosselaers, Govaerts and Vandewalle provide a detailed comparison of Barrett, Montgomery and classical reduction [BGV94]. They provide some timings of modular reduction of a number twice the size of the modulus for the classical algorithm (Knuth’s version), Barrett reduction and Montgomery reduction. The timings were performed on a 33 MHz 80386 based PC, and those for a modulus up to 512 bits are provided in Table 2.2. These timings obviously support their main conclusion that while the classical algorithm is best when only a single modular reduction is required, either Barrett or Montgomery reduction is best when many modular reductions involving the same modulus are required. Whether Barrett or Montgomery reduction should be chosen depends on whether the speed gained by using Montgomery reduction outweighs the time lost by converting the numbers to and from Montgomery format. In the case of elliptic curve cryptosystems, Montgomery arithmetic can be expected to offer the best performance.
2.2.1.3 Modular Inversion

Modular inversion is quite a time consuming operation compared to modular multiplication, and takes the time of about 50 Montgomery multiplications on a Pentium. Therefore, this operation is usually avoided as much as possible. Nevertheless, it is still required on some occasions. Algorithm 2.6 is an inversion algorithm based on the extended Euclidean algorithm (EEA) given in [MvOV96, Algorithm 2.107], and Algorithm 2.7 is an inversion algorithm based on Fermat’s (little) theorem [MvOV96, p.69]. The latter inversion algorithm is also known as the exponentiation method. In addition, Algorithm 2.8 is an inversion algorithm based on the binary extended GCD (BEGCD) algorithm which has been derived from Algorithm 14.61 and Note 14.64 in [MvOV96]. The EEA requires multiple-precision divisions, which are very time consuming. The BEGCD requires more iterations, but removes the requirement for multiple-precision divisions.

<table>
<thead>
<tr>
<th>Input:</th>
<th>Modulus $a$, and number to invert, $b &gt; 0$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$y \equiv b^{-1} \pmod{x}$.</td>
</tr>
<tr>
<td>Algorithm:</td>
<td>$y_2 = 0$</td>
</tr>
<tr>
<td></td>
<td>$y_1 = 1$</td>
</tr>
<tr>
<td></td>
<td>While $b &gt; 0$ do the following:</td>
</tr>
<tr>
<td></td>
<td>$q = \lfloor a/b \rfloor$</td>
</tr>
<tr>
<td></td>
<td>$r = a - qb$</td>
</tr>
<tr>
<td></td>
<td>$y = y_2 - qy_1$</td>
</tr>
<tr>
<td></td>
<td>$a = b$</td>
</tr>
<tr>
<td></td>
<td>$b = r$</td>
</tr>
<tr>
<td></td>
<td>$y_2 = y_1$</td>
</tr>
<tr>
<td></td>
<td>$y_1 = y$</td>
</tr>
<tr>
<td>If $a \neq 1$ return “no inverse”</td>
<td></td>
</tr>
<tr>
<td>$y = y_2$</td>
<td></td>
</tr>
<tr>
<td>Return $y$</td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm 2.6:** Inversion based on the extended Euclidean algorithm

<table>
<thead>
<tr>
<th>Input:</th>
<th>Modulus $p$, and number to invert, $y$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$y^{-1} \pmod{x}$.</td>
</tr>
<tr>
<td>Algorithm:</td>
<td>$a = y^{p-2} \pmod{p}$</td>
</tr>
<tr>
<td>Return $a$</td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm 2.7:** Inversion based on Fermat’s (little) theorem

Table 2.3 shows that on a Pentium, both the EEA and BEGCD algorithms require almost the same amount of time, but the BEGCD algorithm is slightly
### Algorithm 2.8: Inversion based on the binary extended GCD algorithm

<table>
<thead>
<tr>
<th>Input:</th>
<th>Modulus $x$, and number to invert, $y$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$y^{-1} \pmod{x}$.</td>
</tr>
</tbody>
</table>

$v = y$

\[
u = x \quad B = 0 \quad D = 1 \quad \beta = \text{TRUE} \quad (\beta \text{ indicates whether } B \geq 0) \quad \delta = \text{TRUE} \quad (\delta \text{ indicates whether } D \geq 0)\\|
\] do:

\[
\begin{align*}
\text{while } (u \text{ is even}) & : \\
& \quad u = u/2 \\
& \quad \begin{cases} 
B = B/2 & \text{if } (B \text{ is even}) \\
\beta = -\beta & \text{else}
\end{cases} \\
\text{while } (v \text{ is even}) & : \\
& \quad v = v/2 \\
& \quad \begin{cases} 
D = D/2 & \text{if } (D \text{ is even}) \\
\delta = -\delta & \text{else}
\end{cases} \\
& \quad \begin{cases} 
B = (x - B)/2 & \text{if } (\beta = \delta) \\
\beta = -\beta & \text{if } (\beta >\beta) \\
\delta = -\delta & \text{else}
\end{cases} \\
& \quad \begin{cases} 
B = B + D & \text{if } (\delta >\beta) \\
D = D + B & \text{else}
\end{cases}
\end{align*}
\]

\[
\text{while } (u \neq 0).\\
\text{if } (\delta) \text{ then return } D \\
\text{else return } (x - D)
\]
Table 2.3: Timings of various inversion algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pentium III 450 MHz Actual Timings</th>
<th>Smart Card Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEA</td>
<td>0.214</td>
<td>Exponentiation 68%</td>
</tr>
<tr>
<td>BEGCD</td>
<td>0.207</td>
<td>BEGCD between 68% and 100%</td>
</tr>
</tbody>
</table>

faster. Although exponentiation is generally quite slow, Table 2.3 shows that on a smart card (details of which are provided in Chapter 3), the inversion using exponentiation is estimated to be comparable to the time of the BEGCD algorithm. This paradox occurs because the exponentiation operation is available in hardware on the smart card under consideration, whereas the BEGCD algorithm would have to be implemented in software.

2.2.1.4 Modular Square Root Algorithm

In order to facilitate low bandwidth and storage requirements, ECCs should be able to handle points that have been stored in a compressed format (see Section 2.2.2 for details of the format). This requires the availability of a modular square root algorithm in order to uncompress the points for later use. If the modulus \( p \) is equivalent to 3 (mod 4), then a short and fast square root algorithm is available which is given in [MvOV96, Algorithm 3.36] and shown here as Algorithm 2.9. Otherwise, a longer and more expensive algorithm must be used, such as that in [BSS99, p.18] or [MvOV96, Algorithm 3.34].

To find \( \sqrt{x} \) (mod \( p \)) where \( p \equiv 3 \) (mod 4):

- Find \( r = x^{(p+1)/4} \) (mod \( p \)).
- If \( r^2 \neq x \) (mod \( p \)) return “No square root exists.”
- Else return \( (r, p - r) \).

**Algorithm 2.9:** Modular square root algorithm [MvOV96]

2.2.2 Point Addition, Doubling and Representation

The second level of arithmetic consists of the curve operations of addition and doubling. Addition and doubling of points is shown graphically for a curve over
2.2. Curve Operations

Fig. 2.2: Graphical illustration of addition and doubling of elliptic curve points

the real numbers in Figure 2.2. To double a point $P$, a tangent to the curve at the point $P$ is drawn, which intersects the curve in one other place, $-R$. A vertical line is drawn at $-R$ to intersect the curve at another point, $R$, which is taken to be the result of the doubling operation. To add two points $P$ and $Q$, a line is drawn through the two points and the line intersects the curve in one other place, $-R$. A vertical line is drawn through $-R$ which also intersects the curve at the point $R$. $R$ is taken to be the result of the addition operation. The result of adding two points which are vertically aligned is defined to be the point at infinity. This graphical representation can be converted to a mathematical calculation or algorithm, and these calculations also apply to curves over the field $GF(p)$. Algorithms 2.10 and 2.11 describe point addition and doubling corresponding to the graphical representation for a curve over $GF(p)$ where $p > 3$ [BSS99].

Algorithms 2.10 and 2.11 each require a modular inversion, which is very slow. In order to avoid the inefficiency of the inversion, other methods of representing elliptic curve points have been created. These representations do not just use $x$ and $y$ coordinates, but also a $z$ coordinate, and sometimes a fourth or fifth coordinate also. The coordinate systems studied in this dissertation are projective, Jacobian, modified Jacobian and Chudnovsky Jacobian coordinates. Details of these coordinate systems are given in Chapter 3.

It is also possible to represent points in a compressed format in order to reduce the amount of storage space or bandwidth required to store or transmit a point. This format stores the $x$ coordinate of the point, as well as one extra bit to indicate the value of the $y$ coordinate. To uncompress a point, the value
• Let $P$ and $Q$ be the two points to be added together.

• If $P$ is $\phi$ then return $Q$ as the result.

• Else if $Q$ is $\phi$ then return $P$ as the result.

• Else:
  - Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.
  - Let $T_1 = (y_2 - y_1)$
  - Let $T_2 = (x_2 - x_1)$
  - If $T_2$ is zero then
    * If $T_1$ is zero then return Double($Q$) as the result.
    * Else return $\phi$ as the result.
  - Let $\lambda = T_1 \cdot T_2^{-1}$
  - Let $x_3 = \lambda^2 - x_1 - x_2$
  - Let $y_3 = \lambda (x_1 - x_3) - y_1$
  - Return $(x_3, y_3)$.

**Algorithm 2.10**: Addition for points in affine coordinates

• Let the point to be doubled be $P$.

• If $P$ is $\phi$, return $\phi$ as the result.

• Else
  - Let $P = (x_1, y_1)$.
  - Let $\lambda = (3x_1^2 + a) \cdot (2y_1)^{-1}$
  - Let $x_3 = \lambda^2 - 2x_1$
  - Let $y_3 = \lambda (x_1 - x_3) - y_1$
  - Return $(x_3, y_3)$.

**Algorithm 2.11**: Doubling for points in affine coordinates
2.2. Curve Operations

\[ x^3 + ax + b \pmod{p} \] must be found and a modular square root taken. Of the two square roots, the extra bit is used to choose the correct root (one method is to choose the root whose least significant bit is the same as the extra bit) [IEE00].

With the above definitions of addition and doubling, the points on the elliptic curve form a group, with the point at infinity being the identity element of the group and the negative of a point \( P = (x, y) \) being the point \( Q = (x', y') \) such that \( P + Q = \phi \). The point \( Q \) satisfying this condition is one such that \( x' = x \) and \( y' = p - y \), and so point negation is equal to the cost of a modular subtraction. This enables point subtraction to be performed in about the same time as point addition, since \( P - R = P + (-R) \).

### 2.2.3 Scalar Multiplication

The highest level of EC arithmetic is scalar multiplication, which is the addition of a point to itself several times, as defined in Definition 2.1. Much effort has been given to optimizing scalar multiplication algorithms, since the efficiency of scalar multiplication is directly related to the efficiency of the cryptographic operation being performed such as a digital signature or key exchange. One of the simplest algorithms is the binary scalar multiplication algorithm [BSS99] described in Algorithm 2.12.

```
Input: \( P \) (the point to multiply),
\( k \) (the scalar) such that \( k = \sum_{i=0}^{m-1} k_{m-i-1}2^i \),
\( m \) (length of \( k \))
Output: \( Q \) such that \( Q = [k]P \)
Algorithm: \( Q = \phi \)
For \( i = 0 \) to \( m - 1 \)
\( \{ \begin{array}{l}
Q = [2]Q \\
\text{If } (k_i == 1) \\
\{ Q = Q + P \}
\end{array} \)\nReturn \( Q \)
```

Algorithm 2.12: Binary scalar multiplication

Conversion of the scalar to a signed format can increase the efficiency of the algorithm by decreasing the number of non-zero values in the scalar, thus reducing the number of point additions required by the scalar multiplication. This method of optimization is only possible because point negation (described in Section 2.2.2) is trivial (requiring only a modular subtraction) and so point subtraction takes
about the same amount of time as point addition. One method of converting an unsigned scalar to a signed scalar is to use the non-adjacent form (NAF) method, which decreases the number of non-zero digits expected in a random scalar from one-half of the digits to one-third of the digits. An algorithm to convert a scalar to the NAF format can be found in [BSS99]. However, a version that is more easily understood is provided by Algorithm 2.13 [IEE00]. This necessitates a revision of the binary algorithm to account for the possibility of a negative digit, as shown in Algorithm 2.14.

To find the NAF representation of $k$:

- Let $h_{m-1}h_{m-2}\ldots h_0$ be the binary representation of $3k$ and let $k_{m-1}k_{m-2}\ldots k_0$ be the binary representation of $k$.
- For $i$ from 1 to $m - 1$ do:
  - Set $g_{i-1} = h_i - k_i$.
- Return $g = g_{m-2}g_{m-3}\ldots g_0$.

**Algorithm 2.13:** Conversion of a scalar to NAF format

Another method of increasing the efficiency of some ECCs is to use a two-in-one variant of the scalar multiplication algorithm to compute $[h]P + [k]Q$ where $h$ and $k$ are scalars and $P$ and $Q$ are points on the curve. Such a scalar multiplication would be beneficial for ECCs required to perform Elliptic Curve Digital Signature Algorithm (ECDSA, see Section 2.4.3) verifications, since such
a computation is required in this case. An appropriate algorithm can be based on the simultaneous multiple exponentiation algorithm in [MvOV96, p.618] and is shown as Algorithm 2.15.

<table>
<thead>
<tr>
<th>Algorithm 2.15: Two-in-one scalar multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>To find $R = [h]P + [k]Q$:</td>
</tr>
<tr>
<td>• Let $h_{m-1}h_{m-2}\ldots h_0$ be the binary representation of $h$ and let $k_{m-1}k_{m-2}\ldots k_0$ be the binary representation of $k$.</td>
</tr>
<tr>
<td>• Set $S_0 = P$</td>
</tr>
<tr>
<td>• Set $S_1 = Q$</td>
</tr>
<tr>
<td>• Set $S_2 = P + Q$</td>
</tr>
<tr>
<td>• Set $S_3 = P - Q$</td>
</tr>
<tr>
<td>• Set $R = \emptyset$</td>
</tr>
<tr>
<td>• For $i$ from $m - 1$ to 0 do:</td>
</tr>
<tr>
<td>• Set $s = h_i$.</td>
</tr>
<tr>
<td>• If $h_i$ is the same as $k_i$ then set $j = 2$.</td>
</tr>
<tr>
<td>• Else if $h_i$ is zero then set $j = 1$ and $s = k_i$.</td>
</tr>
<tr>
<td>• Else if $k_i$ is zero then set $j = 0$.</td>
</tr>
<tr>
<td>• Else set $j = 3$.</td>
</tr>
<tr>
<td>• If $s &lt; 0$ then set $R = R - S_j$.</td>
</tr>
<tr>
<td>• Else if $s &gt; 0$ then set $R = R + S_j$.</td>
</tr>
<tr>
<td>• Return $R$.</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
As written, the algorithm requires extra memory to store the two points $S_2$ and $S_3$. However, the requirement for this extra memory can be eliminated by computing $S_2$ and $S_3$ each time they are required, rather than relying on the precomputed values. This means that there is a decrease in the speed of the algorithm since two point additions will be required in the place of one when these values are needed. However, this algorithm is still faster than using the signed binary method twice, once for each scalar multiplication. Details are given in Chapter 3 of the exact efficiency of this method and the other scalar
2.3 Security Aspects

In order to ensure the security of the overall elliptic curve cryptosystem, it is necessary to ensure that the underlying elliptic curve meets certain security requirements. These are listed below [BSS99]:

• The group of elliptic curve points should have a subgroup of large prime order $n$. This avoids the Pohlig-Hellman attack [PH78] which reduces the ECDLP to a series of ECDLPs in the subgroups of prime power order, as well as the baby-step giant-step (BSGS) method of Shanks [Sha71] and Pollard’s rho and lambda methods [Pol78] for solving a general DLP. Further details of these attacks are provided in Chapter 5. The size of $n$ is usually at least 160 bits, which provides a security level approximately equivalent to 1024 bit DL or RSA systems. This is discussed further below.

• The curve should not be anomalous (the number of points on the curve, $n$, should not be equal to $p$) in order to avoid the attack on anomalous curves proposed by Smart [Sma99] and Satoh and Araki [SA98]. The attack is able to succeed by solving the ECDLP on an elliptic curve over the $p$-adic numbers and using this solution to solve the ECDLP over the field $GF(p)$. The attack succeeds in linear time.

• The smallest value of $l$ such that $p^l \equiv 1 \pmod{\text{curve order}}$ should be large. This condition ensures that the curve does not have a trace of zero (i.e. the curve is not supersingular) or two. The condition is necessary because it is possible to reduce the ECDLP on a curve over $GF(p)$ to an ordinary DLP in $GF(p^l)$ where $l$ is defined as above. Ensuring $l$ is large also ensures the DLP in $GF(p^l)$ is hard. This attack was proposed by Menezes, Okamoto and Vanstone [MOV93] and generalized by Frey and Rück [FR94]. It is commonly known as the MOV attack, with the condition ensuring its prevention known as the MOV condition. Standardized minimum values for $l$ are given in [IEE00] for various sizes of $p$. As an example, 160 bit curves should have $l$ greater than 7 and 320 bit curves should have $l$ greater than 16.

These requirements can be used to create Algorithm 2.16 [BSS99] to generate a secure elliptic curve.

- Take as input the prime $p$ defining the Galois field over which the curve will be defined, as well as a small positive integer $\bar{c}$, which is an upper limit on the value of the curve order cofactor.
- Do (until a secure curve is found and returned):
  - Choose the curve parameters $a$ and $b$ modulo $p$.
  - Find the order of the curve, $#E(GF(p)) = \eta$.
  - Check the MOV condition (the smallest value of $l$ such that $p^l \equiv 1 \pmod{\eta}$ is “large”) and anomalous condition ($\eta \neq p$). If either of these fail, go back to the beginning of the loop (choose a new curve).
  - If $\eta$ is prime, proceed to the next step. Otherwise, attempt to factor $\eta$. If the attempt is successful, proceed to the next step. Otherwise, if the attempt has not succeeded within a “reasonable” time, conclude that $\eta$ does not have a large prime factor and the curve is hence insecure. Go to the beginning of the loop.
  - If $\eta = c \cdot n$ and $c \leq \bar{c}$ and $n$ is prime, return the values defining the (secure) curve, $p$, $a$ and $b$, as well as the curve order and its factorization, $\eta = c \cdot n$.

**Algorithm 2.16:** Generation of a secure elliptic curve

The only known method of attack on a curve satisfying the above requirements is to use a general algorithm to break the ECDLP (such as Shanks’s BSGS method or Pollard’s rho algorithm) in the subgroup of large prime order [Odl00]. There is currently no algorithm which is sub-exponential in $\log_2(n)$ (the size of the order of the curve) available to break the ECDLP [BSS99, Odl00]. Therefore, an ECC requires a much smaller key size than other cryptosystems for which sub-exponential attacks exist. In addition, the security of an ECC increases faster with key size than other public key cryptosystems because of the existence of sub-exponential attacks on those cryptosystems. This is reflected in Table 2.4 which shows equivalent security levels between EC, RSA and DL ciphers [LV01, BSS99, Odl00]. Lenstra and Verheul [LV01] provide a thorough description of the derivation of their figures and the assumptions used in that process. The figures from [BSS99] are only approximate due to the neglect of various constants and
Table 2.4: Equivalent key sizes from various sources

<table>
<thead>
<tr>
<th></th>
<th>According to [LV01]</th>
<th>According to [BSS99]</th>
<th>According to various others</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC</td>
<td>EC*</td>
<td>DL / RSA</td>
</tr>
<tr>
<td>105</td>
<td>85</td>
<td></td>
<td>417</td>
</tr>
<tr>
<td>111</td>
<td>96</td>
<td></td>
<td>513</td>
</tr>
<tr>
<td>117</td>
<td>106</td>
<td></td>
<td>622</td>
</tr>
<tr>
<td>124</td>
<td>121</td>
<td></td>
<td>777</td>
</tr>
<tr>
<td>129</td>
<td>129</td>
<td></td>
<td>879</td>
</tr>
<tr>
<td>135</td>
<td>139</td>
<td></td>
<td>1028</td>
</tr>
<tr>
<td>141</td>
<td>148</td>
<td></td>
<td>1191</td>
</tr>
<tr>
<td>146</td>
<td>160</td>
<td></td>
<td>1369</td>
</tr>
<tr>
<td>152</td>
<td>172</td>
<td></td>
<td>1562</td>
</tr>
<tr>
<td>160</td>
<td>185</td>
<td></td>
<td>1825</td>
</tr>
<tr>
<td>164</td>
<td>193</td>
<td></td>
<td>1995</td>
</tr>
<tr>
<td>166</td>
<td>197</td>
<td></td>
<td>2054</td>
</tr>
<tr>
<td>172</td>
<td>207</td>
<td></td>
<td>2299</td>
</tr>
<tr>
<td>179</td>
<td>222</td>
<td></td>
<td>2629</td>
</tr>
<tr>
<td>188</td>
<td>239</td>
<td></td>
<td>3061</td>
</tr>
<tr>
<td>191</td>
<td>244</td>
<td></td>
<td>3214</td>
</tr>
<tr>
<td>197</td>
<td>255</td>
<td></td>
<td>3533</td>
</tr>
<tr>
<td>206</td>
<td>272</td>
<td></td>
<td>4047</td>
</tr>
</tbody>
</table>

* Assuming cryptanalytic progress on the ECDLP.

the use of approximations. In contrast to the figures from [LV01] and [BSS99], many works state that 160 bit ECDs provide the same level of security as 1024 bit RSA or DL systems [Odl00], [CMO98, p.51]. A similar estimate can be derived from figures given in [KMV00], namely that a curve with the size of \( n \) equal to 157 bits would provide equivalent security to 1024 bit RSA.

The Certicom Challenge [Cer97] provides a practical gauge of the difficulty of the ECDLP. The object of the challenge is to break a given ECDLP on a given curve, but the challenge has a number of curves of different sizes and associated ECDLPs from which to choose. The largest challenge ECDLP solved so far was on a 109 bit curve and the solution required the use of 10,000 computers running for 549 days, with the solution being reported in November 2002. According to Certicom, the curves used in actual cryptosystems are generally at least 163 bits, and it would take about 100,000,000 times as long to solve an ECDLP on such a curve [Cer02]. Therefore, the infeasibility of the ECDLP is supported by practical
achievements as well as theoretical results.

As discussed in Section 2.1, generating a random but secure elliptic curve is very time consuming due to the time required to find the curve order to ensure it has a large prime factor etc. Because of the time required to generate a secure curve and the infeasibility of generating secure curves on constrained devices such as smart cards, fixed elliptic curves have been included in standards documents for use in ECC implementations. Whilst this method may be efficient, there are concerns that many users having the same fixed curve could present an easier target to an attacker than many users each using a different curve.

In addition, the fixed curves provided in standards often use a special prime modulus \( p \) or curve parameter \( a \) to allow a more efficient implementation of the curve arithmetic. Unfortunately, this also has the effect of speeding up any attack on the ECC. Another concern is that it might be possible to create special purpose hardware to attack a fixed curve much more quickly than existing software methods.

Although these security issues are well known, a study of the exact impact of these issues on the level of fixed curve security had not been carried out prior to this research. Because a quantization of any security loss due to the use of fixed curves is crucial to provide confidence in smart card ECC implementations, these issues are studied in detail in Chapter 5.

## 2.4 Cryptographic Algorithms and Protocols

Various cryptographic algorithms and protocols suitable for elliptic curves are mentioned or used throughout the dissertation. This section provides a description of three protocols: Diffie-Hellman key exchange, ElGamal encryption and the Elliptic Curve Digital Signature Algorithm. This is preceded by some introductory information.

Table 2.5 provides a list of the notation used below. With the exception of the public and private keys and some notational conventions used in the protocols below, the symbols in the table have already been introduced earlier in this chapter. Key generation is a basic ECC operation and is described next.

To generate a private and public key pair [IEE00], a private key is first chosen. It consists of an integer \( d \) modulo the prime \( n \) (where \( n \) is the prime order of the group or subgroup of EC points). It should be randomly chosen, in the range
Chapter 2. Elliptic Curve Overview

Table 2.5: Elliptic curve notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>The elliptic curve under consideration, which is defined over the field $GF(p)$ where $p$ is a large prime and consisting of the point at infinity, $\phi$, and the points $(x, y)$ satisfying the equation $E : y^2 \equiv x^3 + ax + b \pmod{p}$ where $a$ and $b$ are constants and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.</td>
</tr>
<tr>
<td>$p$</td>
<td>A large prime which specifies the field over which the elliptic curve is defined, $GF(p)$.</td>
</tr>
<tr>
<td>$a$ and $b$</td>
<td>Constant curve parameters, as described above.</td>
</tr>
<tr>
<td>$x$ and $y$</td>
<td>The $x$ and $y$ coordinates of an affine point on the curve.</td>
</tr>
<tr>
<td>$G$</td>
<td>A point on the curve with order $n$, referred to as the base point and forming part of the domain parameters.</td>
</tr>
<tr>
<td>$P$, $Q$ and $R$</td>
<td>Points on the curve.</td>
</tr>
<tr>
<td>$#E(GF(p))$ or $\eta$</td>
<td>The number of points on the curve, also known as the order of the curve.</td>
</tr>
<tr>
<td>$n$</td>
<td>The large prime order of the group of elliptic curve points, or the large prime order of a subgroup of that group.</td>
</tr>
<tr>
<td>$c$</td>
<td>A value such that $\eta = #E(GF(p)) = c \cdot n$.</td>
</tr>
<tr>
<td>$d$</td>
<td>The private key of a user of the curve such that $d \in [1, n - 1]$.</td>
</tr>
<tr>
<td>$W$</td>
<td>The public key of a user of the curve. $W$ is found using the equation $W = [d]G$.</td>
</tr>
<tr>
<td>$r</td>
<td></td>
</tr>
<tr>
<td>$r \in_R S$</td>
<td>$r$ is randomly chosen from the set $S$.</td>
</tr>
<tr>
<td>$\oplus$</td>
<td>Bitwise exclusive or.</td>
</tr>
<tr>
<td>$x_Q$</td>
<td>The $x$ coordinate of the point $Q$.</td>
</tr>
</tbody>
</table>
and stored in such a manner that any adversary can not obtain any information concerning the key. The corresponding public key is the EC point \( W = [d]G \) where \( G \) is a point on the curve with order \( n \) and forms part of the domain parameters of the curve. \( G \) is often referred to as the base point.

### 2.4.1 Diffie-Hellman Key Exchange

A basic EC algorithm is Diffie-Hellman key exchange [DH76]. It enables two parties who do not possess any shared secret information before the start of the protocol to exchange a secret key. An unauthenticated version is shown as Protocol 2.1 [CK01a]. Before the algorithm could be used in a practical situation, a method to allow the two parties to authenticate each other’s messages would need to be added. It is assumed that both parties participating in the protocol know a unique session identifier, \( sid \). The notation \( x \in_R \mathbb{Z}_n \) means the value \( x \) is randomly chosen from the set \( \mathbb{Z}_n \), in this case the integers modulo \( n \). \( K’ \) and \( K \) are the keys computed by the parties \( A \) and \( B \) respectively. If the protocol is carried out correctly, they will be the same. Section 6.2.4 discusses Diffie-Hellman key exchange and the use of authentication mechanisms in more detail.

![Protocol 2.1: Diffie-Hellman key exchange](image)

### 2.4.2 ElGamal Encryption

ElGamal [ElG85] published an encryption scheme based on the difficulty of the DLP in 1985. Its purpose is to transmit a message from one party to another with whom no secret information is shared, whilst preserving the confidentiality of the message. The scheme can easily be converted to an EC encryption scheme, as shown in Protocol 2.2, provided the message is represented as a point on the elliptic curve. It works by setting up a secret value in the same manner as Diffie-Hellman key exchange and then adding the message to this secret value and transmitting the result. Therefore, only the sender and the receiver have
any knowledge of the message, since they are the only ones who know the shared secret value.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known:</td>
<td>Known:</td>
</tr>
<tr>
<td>Curve parameters</td>
<td>Curve parameters</td>
</tr>
<tr>
<td>Base point $G$</td>
<td>Base point $G$</td>
</tr>
<tr>
<td>$B$’s public key, $W_B$</td>
<td>Private key $d_B$</td>
</tr>
<tr>
<td>Message $M$ (is a point on curve)</td>
<td>Public key $W_B = [d_B]G$</td>
</tr>
<tr>
<td>$k \in R \mathbb{Z}_n$</td>
<td></td>
</tr>
</tbody>
</table>

Encrypt:

$C_1 = [k]G$

$C_2 = [k]W_B + M$

$C_1, C_2$

Decrypt:

$M = C_2 - [d_B]C_1$

$= [k][d_B]G + M - [d_B][k]G$

**Protocol 2.2: ElGamal encryption**

Since messages are usually expressed as integers, rather than points on an elliptic curve, a conversion mechanism from integers to points is necessary in order to use the encryption scheme. Alternatively, Okamoto, Fujisaki and Morita [OFM99] have proposed an encryption scheme called PSEC-1 (which stands for Provably Secure Encryption Scheme) based on the ElGamal encryption scheme but which overcomes the above problem. Protocol 2.3 describes the scheme. The main differences between this protocol and Protocol 2.2 are the use of exclusive or instead of the group operation to combine the message with the shared secret, and the concatenation a random string to the message before using it in the encryption, enabling an increased level of security through the use of random padding.

### 2.4.3 Elliptic Curve Digital Signature Algorithm

The purpose of a digital signature [JMV01] is to allow a recipient to be certain that the specified sender did indeed send the specified message (data origin authentication), without alteration (data integrity). A signature can also be used to provide a method of ensuring that the signing entity can not subsequently deny messages or commitments (non-repudiation). In particular, it should be infeasible for someone other than the specified signer to forge a signature on any
2.4. Cryptographic Algorithms and Protocols

A

Known:
- Curve parameters $a$, $b$, $p$
- $L_p$ (length of $p$)
- $L_m$ (length of message)
- $L_r$ (length of random string)
  such that $L_m + L_r \leq L_p$
- Base point $G$
- Hash function $h$
- $B$’s public key, $W_B$
- Message $m \in \{0, 1\}^{L_m}$
- $r \in \{0, 1\}^{L_r}$

Encrypt:

$$t = h(m||r)$$
$$C_1 = [t]G$$
$$Q = [t]W_B$$
$$c_2 = x_Q \oplus (m||r)$$

$B$

Known:
- Curve parameters $a$, $b$, $p$
- $L_p$ (length of $p$)
- $L_m$ (length of message)
- $L_r$ (length of random string)
  such that $L_m + L_r \leq L_p$
- Base point $G$
- Hash function $h$
- Private key $d_B$
- Public key $W_B = [d_B]G$

Decrypt:

$$D = [d_B]C_1$$
$$u = c_2 \oplus x_D$$
If $C_1$ equals $[h(u)]G$ return $u$
Else return null

Protocol 2.3: PSEC-1 encryption
message. A more formal definition of security for signature schemes can be found in [GB01].

ElGamal [ElG85] proposed a digital signature scheme based on the difficulty of the DLP in 1985. The U.S. government’s Digital Signature Algorithm (DSA) is based on ElGamal’s scheme, but has some computational advantages compared to ElGamal’s original scheme [KMV00, BSS99]. The scheme has also been converted for use in ECCs, and is known as the Elliptic Curve Digital Signature Algorithm (ECDSA). It has been included in various standards, including IEEE Std 1363 [IEE00] and ANSI X9.62 [ANS98]. Algorithms 2.17 and 2.18 [IEE00] describe the signature and verification algorithms respectively.

The signature algorithm begins by using a hash function to compress the message to a value \( f \) of a length that can be handled by the algorithm. The hash function must be such that it is infeasible to find a different message that hashes to the same value, in order to ensure the security of the signature scheme. Next, the one-time key pair \((u, V)\) is generated. The one-time private key \( u \) must be kept secret and should be erased at the end of the signature generation. The algorithm then finds a value \( \delta \) such that \( f \equiv u \cdot \delta - d \cdot x_V \) (where \( d \) is the long-term private key) and returns \((x_V, \delta)\) as the signature. To verify the signature, the truth of the equation is verified by ensuring \( x_V = x_Q \) where \( Q = [\delta^{-1}(f + x_V \cdot d)]G = [f \cdot \delta^{-1}]G + [x_V \cdot \delta^{-1}]W \) (where \( W \) is the long-term public key).

### 2.5 Conclusion

This chapter gave an introduction to elliptic curves and provides a foundation for later chapters. It first defined an elliptic curve over the field \( GF(p) \) and outlined the reasons for which this research chose to study elliptic curves over this field rather than elliptic curves over other finite fields. The elliptic curve discrete logarithm problem was defined, and secure elliptic curves were defined as those for which the ECDLP is infeasible. This infeasibility enables scalar multiplication to be the basic cryptographic operation of an elliptic curve.

The chapter then described the three levels of arithmetic required to implement scalar multiplication and presented relevant algorithms for each of the three levels. This material provides a basic foundation upon which the description of an efficient implementation of an ECC on a smart card in Chapter 3 will build. The
• Input:
  - Domain parameters $p$, $a$, $b$, $n$ and $G$ as defined in Table 2.5.
  - Signer’s private key $d \in [1, n - 1]$.
  - Message $m$.
  - Hash function $H$ returning a non-negative integer of the same length as $n$.

• Output:
  - The signature, which is a pair of integers $(c, \delta)$ where $1 \leq c, \delta < n$.

• Algorithm:
  - Find $f = H(m)$.
  - Generate a one-time key pair $(u, V)$ with the same set of domain parameters as the private key $d$. This means that $u \in [1, n - 1]$ and $V = [u]G$ and this implies $V \neq \phi$. Let $V = (x_V, y_V)$.
  - Find $c = x_V \pmod{n}$. If $c$ is zero, go back to the beginning of the algorithm (choose a new one-time key pair).
  - Compute an integer $\delta = u^{-1}(f + dc) \pmod{n}$. If $\delta$ is zero, go back to the beginning of the algorithm.
  - Output the pair $(c, \delta)$ as the signature.

**Algorithm 2.17: ECDSA signature**
• Input:
  – Domain parameters $p, a, b$ and $n$ as defined in Section 2.1. Also the
    base point $G$ with order $n$ associated with the public key $W$.
  – Signer’s public key $W$ which is a point on the curve.
  – Message $m$ and purported signature $(c, \delta)$.
  – Hash function $H$ returning a non-negative integer of the same length
    as $n$.

• Output:
  – “valid” if $m$ and $(c, \delta)$ are consistent given the key and domain pa-
    rameters; “invalid” otherwise.

• Algorithm:
  – Find $f = H(m)$.
  – If either $c$ or $\delta$ is not in the interval $[1, n - 1]$, output “invalid” and
    stop.
  – Compute the integers $h = \delta^{-1} \pmod n$, $h_1 = f \cdot h \pmod n$ and
    $h_2 = c \cdot h \pmod n$.
  – Compute an elliptic curve point $P = [h_1]G + [h_2]W$. If $P$ is $\phi$, output
    “invalid” and stop. Otherwise, let $P = (x_P, y_P)$.
  – Compute $c' = x_P \pmod n$
  – If $c$ and $c'$ are equal, output “valid”, else output “invalid.”

Algorithm 2.18: ECDSA verification
material also provides a basis for the scalar multiplication algorithm resistant to simple power analysis presented in Chapter 4.

The chapter also listed the security requirements of an elliptic curve, and showed an algorithm to generate a random but secure elliptic curve. The necessity and efficiency of point counting algorithms was also discussed, leading to the introduction of the idea of fixed curves. The chapter briefly mentioned some security concerns regarding fixed curves which are explored in much greater detail in Chapter 5.

Finally, algorithms for Diffie-Hellman key exchange, ElGamal encryption and the Elliptic Curve Digital Signature Algorithm and were provided. Whilst these algorithms are not studied in detail in this dissertation, later chapters assume the reader is familiar with them.
Chapter 3

Smart Card Implementation of an Elliptic Curve Cryptosystem

Implementation of an ECC on a smart card requires careful consideration of issues such as the maximum code size, the memory available and the resulting efficiency of the implementation. Resolution of these issues can be difficult because of the tradeoffs involved. In addition, any possible threat due to side channel attacks (described in Chapter 4) must be assessed and defences put in place if necessary. These issues were examined in detail by completing an implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA) [ANS98, IEE00] (a variant of ElGamal’s digital signature scheme [ElG85], see Section 2.4) as specified by IEEE Std 1363 [IEE00] on a smart card. This chapter focuses on the tradeoffs possible between code size, RAM usage and speed, while a discussion of appropriate countermeasures to defeat side channel attacks is left until Chapter 4.

The smart card targeted for the project was the Motorola M-Smart Jupiter™ smart card [Mot00] based on Java Card™ 2.1 technology and an ARM processor [Atm99] with a word size of 32 bits, 64 KB of ROM, 32 KB of EEPROM, 3 KB RAM and a modular arithmetic accelerator. All of the ECC operations were implemented in the C programming language, and testing was performed on a simulation of the smart card utilizing the ARM Software Development Toolkit.

The field $GF(p)$ (where $p$ is prime) was chosen as the field over which to implement the ECC due to the availability of an arithmetic coprocessor on the smart card to perform field arithmetic efficiently. This choice is discussed in
Chapter 3. Smart Card Implementation of an Elliptic Curve Cryptosystem

more detail in Chapter 2. The size of \( p \) chosen for the implementation was 160 bits, since this size is a multiple of 32 bits, a common word size, and is also considered to provide approximately the same security as 1024 bit RSA [RSA78] (see Section 2.3).

In order to achieve an efficient implementation, firstly efficient field arithmetic (modular addition, subtraction, multiplication and inversion) must be available. These operations are then used in the algorithms for addition and doubling of points. In turn, the addition and doubling operations must be efficient, in order for the scalar multiplication which uses them to be efficient. It is possible to add and double points in various coordinate systems and the choice of coordinate system can have a considerable impact on the final speed of the scalar multiplication operation. The dependencies of the various operations are shown graphically in Figure 2.1.

In the following subsections, an overview of efficient algorithms for modular arithmetic is provided, followed by a detailed description and efficiency analysis of the various point coordinate systems available for addition and doubling on an elliptic curve. This includes algorithms for addition and doubling that have been optimized to reduce the number of temporary variables required. The efficiency of scalar multiplication is then investigated, and the RAM usage, code size and speed of ECDSA [IEE00] using various scalar multiplication and point coordinate options is provided. The relative times for ECDSA and RSA signature and verification operations of equivalent security are also given. Finally, recommendations are made for additional operations that could be included on future coprocessors to facilitate efficient implementations of elliptic curve cryptosystems. All PC timings given in this chapter were performed on a Pentium III 450 MHz.

### 3.1 Field Arithmetic

In order to achieve an efficient implementation of an ECC, it is crucial to have an efficient implementation of the underlying field arithmetic, which in this case is modular addition, subtraction, multiplication and inversion. These operations are the most basic operations of the ECC and are used directly by the point addition and doubling routines. Modular addition and subtraction are relatively fast and easily implemented (see Section 2.2.1.1 and [MvOV96] for suitable algorithms). However, modular multiplication (which requires a modular reduction)
3.1. Field Arithmetic

and modular inversion are much more time consuming. Various methods of either speeding up or avoiding these operations have been published. Although the coprocessor on the smart card provided most of the required modular arithmetic operations, the available modular reduction and inversion algorithms were investigated in some detail to ensure an optimal implementation. This is discussed in the following subsections.

3.1.1 Selection of the Modular Reduction Algorithm

Two efficient methods of modular reduction that are often considered for implementation and may be used with any modulus are Barrett [Bar87] reduction and Montgomery [Mon85] reduction [BGV94], [MvOV96, pp.599–604]. Each of these methods requires a precomputation that depends on the modulus. The efficiency of both methods is due to the fact that the only divisions performed can be implemented as right shifts which are quite fast. However, Montgomery reduction also requires the operands to be converted to a special Montgomery form. If the precomputation and conversion time is ignored, Montgomery reduction is slightly faster than Barrett reduction, and both are faster than the classical algorithm [BGV94]. Barrett and Montgomery reduction were not implemented in this project since a modular reduction algorithm (for a modulus without a special form) was already available from the smart card coprocessor. However, further details of Barrett and Montgomery reduction, including algorithms, can be found in Section 2.2.1.2.

Another modular reduction method which has previously been successfully adopted in a software only implementation by Brown et al. in [BHLM01] in order to increase the speed of the ECC is to use a modulus with a special form, such as the NIST primes [NIS00]. These primes are the result of adding or subtracting a small number of powers of two from each other (where the powers of two are generally of the form $2^{32i}$ or $2^{64i}$ where $i$ is a small integer, e.g. $p = 2^{192} - 2^{64} - 1$), enabling a very fast but specialized reduction algorithm. In fact, Brown et al. achieved reduction timings that were between 6% and 33% of the time required for Barrett reduction, depending on the prime used and whether assembly language was used. ECCs using the NIST primes were considered in this research, but they did not give favourable timings because the coprocessor could not be effectively utilized in such an implementation. For example, for a 224-bit modulus, multiplication without reduction in software (which is necessary before the
Chapter 3. Smart Card Implementation of an Elliptic Curve Cryptosystem

reduction takes place) took 4.9 times as long as a hardware modular multiplication. Also, the 224-bit modular reduction in software took 1.5 times as long as a hardware modular multiplication. Multiplication and modular reduction using a 160-bit modulus of this form were also slower than the equivalent hardware operation.

A pseudo-Mersenne prime can also be used to speed up the reduction algorithm. A pseudo-Mersenne prime is a prime that is close to a power of two and is defined in [BP98] as $2^n \pm c$ for some $\log c \leq \frac{1}{3}n$. A fast reduction algorithm for primes of this form is given in [MvOV96, p.605]. However, as for the NIST primes, in order to perform a modular multiplication, a multiplication without reduction is first required which takes much longer than a hardware modular multiplication on the smart card.

Because the algorithms for special primes did not give any speed advantages on the smart card, the coprocessor was used to perform the modular arithmetic and random primes were used to define the elliptic curves that were used. Although the use of special reduction algorithms available for the NIST primes and pseudo-Mersenne primes was inefficient on this smart card since the algorithms had to be implemented in software, if hardware was available to perform the algorithms then their use could speed up the ECC even further.

3.1.2 Modular Inversion

Finding multiplicative inverses in the field $GF(p)$ (required by ECCs over $GF(p)$) is extremely slow and generally avoided as much as possible. For example, this research found that on a PC, inversion using either the extended Euclidean algorithm (EEA) [MvOV96, p.67] or binary extended GCD (BEGCD) algorithm [MvOV96, pp.608–610] takes about 50 times as long as a Montgomery multiplication for a 160-bit modulus and 47 times as long for a 192-bit modulus. These results are similar to those published in [BHLMO1], where inversion on a Pentium II 400 MHz took about 40 to 65 times as long as Barrett reduction. In other papers different values have been reported, probably due to the different platform and/or different type of prime used. For example, in [CMO98] on an UltraSPARC 143 MHz running Solaris 2.4 using a prime of special form, modular inversion took 19 times as long as modular multiplication and in [GBKP01] on the TI MSP 430x33x family of devices, using a prime of special from and a special inversion algorithm (because of the special type of the prime) inversion took 133
times as long as modular multiplication.

The use of coordinate systems other than the affine coordinate system for elliptic curve point addition and doubling greatly reduces the number of inversions required in the operations of the ECC (see Section 3.2). However, an efficient inversion algorithm is still needed for those times when inversion is required, such as during the creation of an ECDSA digital signature by one party, verification of the validity of that signature by another party, or at the end of a scalar multiplication to convert the point coordinates back to affine coordinates.

Since the smart card coprocessor did not provide an inversion algorithm, three inversion algorithms were considered for use in this project. These were the BEGCD algorithm, the EEA and the exponentiation method (from Fermat’s little theorem [MvOV96, p.69]), $a^{-1} \pmod{p} \equiv a^{p-2} \pmod{p}$. The EEA involves multi-precision divisions, which are quite slow. In order to avoid such divisions, the BEGCD algorithm uses right shifts (which are fast), but requires more iterations. The speed in software of these methods was estimated for the smart card and compared to that of the exponentiation method. Because the exponentiation method was available in hardware, it required minimal code space and did not decrease performance compared to the EEA and BEGCD algorithms which were only available in software. For these reasons, the exponentiation method was chosen as the inversion algorithm. Further details of the methods of modular inversion which were considered, including algorithms, can be found in Section 2.2.1.3.

3.2 Point Coordinates

One of the crucial decisions when implementing an efficient elliptic curve cryptosystem over $GF(p)$ is deciding which point coordinate system to use. The point coordinate system used for addition and doubling of points on the elliptic curve determines the efficiency of these routines, and hence the efficiency of the basic cryptographic operation, scalar multiplication.

The different point coordinate systems have arisen due to the large amount of time required to complete a modular inversion. Because the most basic coordinates, affine coordinates, require an inversion in both point addition and point doubling, these coordinates are not used for most point operations. Rather, a different coordinate system which requires a greater number of modular multiplications but no inversions is used. Because there are several different coordinate
systems available and each system has different memory usage and speed, the
decision of which coordinate system to use can be difficult. Therefore, this sec-
tion analyses the efficiency and memory usage of the different coordinate systems
available. There is some variation in the literature regarding the names of some
coordinate systems. The names used by Cohen et al. in [CMO98] are adopted
here.

3.2.1 Affine Coordinates

Before studying all of the various point coordinate systems available, it is neces-
sary to understand the well known affine coordinate system [BSS99, Sil86].
Although this coordinate system is used for communication between parties in
an elliptic curve cryptosystem over $GF(p)$ due to its compact nature and forms
the basis for all of the other coordinate systems, it is seldom used in low level
functions due to its inefficiency. An affine point is represented using two coor-
dinates as $(x_A, y_A)$ satisfying the equation $y_A^2 = x_A^3 + ax_A + b \pmod{p}$ where $a$
and $b$ are curve parameters, as described in Chapter 2. In an implementation
of an ecc, a convention is necessary to indicate whether a point is the point at
infinity, $\phi$. This can be done by using a special value of the $x_A$ and $y_A$
coordinates to represent $\phi$ (for example $(0,0)$ when that is not a valid point on the
curve [IEE00]), or by using a boolean value to flag the point at infinity.

The negative of an affine point $P = (x_A, y_A)$ is $-P = (x_A, p - y_A)$. Two affine
points $P$ and $Q$ can be added using the following laws:

- If $Q = \phi$ then $P + Q = P$
- If $P = \phi$ then $P + Q = Q$
- If $Q = -P$ then $P + Q = \phi$
- Otherwise, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. Then $P + Q = R = (x_3, y_3)$
  where (all operations performed modulo $p$):
  - If $P \neq Q$, $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$. Otherwise, $\lambda = \frac{3x_1^2 + a}{2y_1}$.
  - In both cases, $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = (x_1 - x_3)\lambda - y_1$

Algorithms 2.10 and 2.11 show how these laws can be implemented in practice.
3.2.2 Projective Coordinates

Projective coordinates are also well known, although sometimes called conventional or homogeneous projective coordinates [BSS99, Sil86]. They represent a point as \((X, Y, Z)\) where \(X\), \(Y\), and \(Z\) satisfy the equation:

\[
Y^2Z = X^3 + aXZ^2 + bZ^3 \quad \text{(mod } p)\]

and \(a\) and \(b\) are the same curve parameters as for affine coordinates. When \(Z = 0\), the projective point corresponds to the point at infinity, \(\phi\). Otherwise, an affine point \((x_A, y_A)\) can be converted to a projective point \((X, Y, Z)\) by generating a random value \(r\) and setting:

\[
\begin{align*}
X &= x_Ar \pmod p \\
Y &= y_Ar \pmod p \\
Z &= r \pmod p
\end{align*}
\]

A projective point can be converted to an affine point using the equations:

\[
\begin{align*}
x_A &= XZ^{-1} \pmod p \\
y_A &= YZ^{-1} \pmod p
\end{align*}
\]

It is obvious from these equations that one affine point can be converted to many different projective representations and vice versa. In fact, two projective points \((X_1, Y_1, Z_1)\) and \((X_2, Y_2, Z_2)\) are considered to be the same point if \(X_1Z_2 = X_2Z_1\) and \(Y_1Z_2 = Y_2Z_1\). It also follows from the definition of the negative of an affine point and the above equations that the negative of a projective point \(P = (X, Y, Z)\) is the point \(-P = (X, p - Y, Z)\). However, other points also exist which are the negative of \(P\), namely points of the form \((Xr, (p - Y)r, Zr)\) for any \(r\).

Projective coordinates enable addition and doubling to be performed without the use of inversions, but at the expense of extra modular multiplications and squarings. However, time saved by not performing any inversions outweighs the time lost by performing extra multiplications and squarings. Two projective points \(P = (X_1, Y_1, Z_1)\) and \(Q = (X_2, Y_2, Z_2)\) can be added using the following laws (where all operations are modulo \(p\)) [KT93]:
Chapter 3. Smart Card Implementation of an Elliptic Curve Cryptosystem

- If $Z_2 = 0$ then $P + Q = (X_1, Y_1, Z_1)$
- If $Z_1 = 0$ then $P + Q = (X_2, Y_2, Z_2)$
- Otherwise, if $X_1Z_2 \neq X_2Z_1$ and $Y_1Z_2 \neq Y_2Z_1$ then $P + Q = (X_3, Y_3, Z_3)$

where:

\[
\begin{align*}
U &= Y_2Z_1 - Y_1Z_2 \\
V &= X_2Z_1 - X_1Z_2 \\
A &= U^2Z_1Z_2 - V^2T \\
T &= X_2Z_1 + X_1Z_2 \\
X_3 &= VA \\
Y_3 &= U(V^2X_1Z_2 - A) - V^3Y_1Z_2 \\
Z_3 &= V^3Z_1Z_2
\end{align*}
\]

- Otherwise, $P + Q = [2]P = (X_3, Y_3, Z_3)$ where:

\[
\begin{align*}
S &= Y_1Z_1 \\
W &= 3X_1^2 + aZ_1^2 \\
E &= Y_1S \\
F &= X_1E \\
H &= W^2 - 8F \\
X_3 &= 2SH \\
Y_3 &= W(4F - H) - 8E^2 \\
Z_3 &= 8S^3
\end{align*}
\]

Algorithms 3.1 and 3.2 show how these laws can be implemented in practice.

3.2.3 Jacobian Coordinates and Variants

Jacobian coordinates [CC86, CMO98] represent a point as $(X, Y, Z)$, where $X$, $Y$ and $Z$ satisfy the equation:

\[Y^2 = X^3 + aXZ^4 + bZ^6 \pmod{p}\]
where \( a \) and \( b \) are the same curve parameters as for affine coordinates. In the literature, Jacobian coordinates are sometimes called projective or weighted projective coordinates \([BSS99, IEE00]\). Any Jacobian point with \( Z = 0 \) corresponds to the point at infinity. Otherwise, an affine point \((x_A, y_A)\) can be converted to a Jacobian point \((X, Y, Z)\) by generating a random value \( r \) and setting:

\[
\begin{align*}
X &= x_A r^2 \pmod{p} \\
Y &= y_A r^3 \pmod{p} \\
Z &= r \pmod{p}.
\end{align*}
\]

A Jacobian point can be converted to an affine point using the equations:

\[
\begin{align*}
x_A &= XZ^{-2} \pmod{p} \\
y_A &= YZ^{-3} \pmod{p}.
\end{align*}
\]

It is obvious from the above equations that as for projective coordinates, one affine point can be represented by many different Jacobian points. In fact, two Jacobian coordinates are considered to be the same if \( X_1 Z_2^2 = X_2 Z_1^2 \) and \( Y_1 Z_2^3 = Y_2 Z_1^3 \). It also follows from the definition of the negative of an affine point and the above equations that the negative of a Jacobian point \( P = (X, Y, Z) \) is the point \(-P = (X, p - Y, Z)\). However, other points also exist which are the negative of \( P \), namely points of the form \((Xr^2, (p - Y)r^3, Zr)\) for any \( r \).

It is possible to add and double points represented in Jacobian coordinates without using any inversions, again at the cost of extra multiplications and squarings. Two Jacobian points \( P = (X_1, Y_1, Z_1) \) and \( Q = (X_2, Y_2, Z_2) \) can be added using the following laws (where all operations are modulo \( p \)) \([CMO98]\):

- If \( Z_2 = 0 \) then \( P + Q = (X_1, Y_1, Z_1) \)
- If \( Z_1 = 0 \) then \( P + Q = (X_2, Y_2, Z_2) \)
- Otherwise, if \( X_1 Z_2^2 \neq X_2 Z_1^2 \) and \( Y_1 Z_2^3 \neq Y_2 Z_1^3 \) then \( P + Q = (X_3, Y_3, Z_3) \)
where:

\[ U_1 = X_1Z_2^2 \]
\[ U_2 = X_2Z_1^2 \]
\[ S_1 = Y_1Z_3^2 \]
\[ S_2 = Y_2Z_1^5 \]
\[ H = U_2 - U_1 \]
\[ T = S_2 - S_1 \]
\[ X_3 = -H^3 - 2U_1H^2 + T^2 \]
\[ Y_3 = -S_1H^3 + T(U_1H^2 - X_3) \]
\[ Z_3 = Z_1Z_2H \]

- Otherwise, \( P + Q = [2]P = (X_3, Y_3, Z_3) \) where:

\[ S = 4X_1Y_1^2 \]
\[ M = 3X_1^4 + aZ_1^4 \]
\[ T = -2S + M^2 \]
\[ X_3 = T \]
\[ Y_3 = -8Y_1^4 + M(S - T) \]
\[ Z_3 = 2Y_1Z_1 \]

Algorithms 3.3, 3.4 and 3.5 show how these laws can be implemented in practice. Counting the number of operations required to add and double in affine and Jacobian coordinates shows that Jacobian addition is slower than projective addition, but Jacobian doubling is faster than projective doubling. This is discussed further in Section 3.2.6.

There are some variants of Jacobian coordinates available. The first of these is Chudnovsky Jacobian coordinates [CC86], which represent a point as \((X, Y, Z, Z^2, Z^3)\). Apart from the two extra coordinates, Chudnovsky Jacobian coordinates are the same as Jacobian coordinates. However, because \(Z^2\) and \(Z^3\) must be calculated as part of a Jacobian addition but are already provided as input for Chudnovsky Jacobian addition, these two extra coordinates allow a faster addition algorithm than Jacobian coordinates. On the other hand, Chudnovsky
3.2. Point Coordinates

doubling is slower than Jacobian doubling because the extra two coordinates must be calculated at the end of the doubling operation. Although the same calculation is required at the end of Chudnovsky addition, the time lost at the end of the addition is less than the time gained at the beginning of the addition. Two Chudnovsky Jacobian points \( P = (X_1, Y_1, Z_1, Z_1^2, Z_1^3) \) and \( Q = (X_2, Y_2, Z_2, Z_2^2, Z_2^3) \) can be added using the following laws (where all operations are modulo \( p \)) [CMO98]:

- If \( Z_2 = 0 \) then \( P + Q = (X_1, Y_1, Z_1, Z_1^2, Z_1^3) \)
- If \( Z_1 = 0 \) then \( P + Q = (X_2, Y_2, Z_2, Z_2^2, Z_2^3) \)
- Otherwise, if \( X_1Z_2^2 \neq X_2Z_1^2 \) and \( Y_1Z_2^3 \neq Y_2Z_1^3 \) then \( P + Q = (X_3, Y_3, Z_3, Z_3^2, Z_3^3) \) where:

\[
\begin{align*}
U_1 &= X_1(Z_2^2) \\
U_2 &= X_2(Z_1^2) \\
S_1 &= Y_1(Z_2^3) \\
S_2 &= Y_2(Z_1^3) \\
H &= U_2 - U_1 \\
T &= S_2 - S_1 \\
X_3 &= -H^3 - 2U_1H^2 + T^2 \\
Y_3 &= -S_1H^3 + T(U_1H^2 - X_3) \\
Z_3 &= Z_1Z_2H \\
Z_3^2 &= (Z_3)^2 \\
Z_3^3 &= (Z_3)(Z_3^2)
\end{align*}
\]
• Otherwise, \( P + Q = [2]P = (X_3, Y_3, Z_3, Z_3^2, Z_3^3) \) where:

\[
\begin{align*}
S &= 4X_1Y_1^2 \\
M &= 3X_1^2 + a(Z_1^2)^2 \\
T &= -2S + M^2
\end{align*}
\]

\[
\begin{align*}
X_3 &= T \\
Y_3 &= -8Y_1^4 + M(S - T) \\
Z_3 &= 2Y_1Z_1 \\
Z_3^2 &= (Z_3)^2 \\
Z_3^3 &= (Z_3)(Z_3^2)
\end{align*}
\]

Algorithms 3.6 and 3.7 show how the Chudnovsky Jacobian addition and doubling laws can be implemented in practice.

The second variant of Jacobian coordinates is modified Jacobian Coordinates [CMO98], which represent a point as \((X, Y, Z, aZ^4)\), where \(a\) is the elliptic curve parameter. Apart from the extra coordinate, modified Jacobian coordinates are the same as Jacobian coordinates. However, because \(aZ^4\) must be calculated as part of a Jacobian doubling but is already provided as input for a modified Jacobian doubling, the extra coordinate allows a faster doubling algorithm than Jacobian doubling. On the other hand, modified Jacobian coordinates require a slower addition algorithm since the extra coordinate must be calculated at the end of the addition. Two modified Jacobian points \( P = (X_1, Y_1, Z_1, aZ_1^4) \) and \( Q = (X_2, Y_2, Z_2, aZ_2^4) \) can be added using the following laws (where all operations are modulo \(p\)) [CMO98]:

• If \(Z_2 = 0\) then \( P + Q = (X_1, Y_1, Z_1, aZ_1^4) \)

• If \(Z_1 = 0\) then \( P + Q = (X_2, Y_2, Z_2, aZ_2^4) \)

• Otherwise, if \(X_1Z_2^2 \neq X_2Z_1^2\) and \(Y_1Z_2^3 \neq Y_2Z_1^3\) then \( P + Q = (X_3, Y_3, Z_3) \)
where:

\begin{align*}
U_1 &= X_1Z_2^2 \\
U_2 &= X_2Z_1^2 \\
S_1 &= Y_1Z_2^3 \\
S_2 &= Y_2Z_1^3 \\
H &= U_2 - U_1 \\
T &= S_2 - S_1 \\
X_3 &= -H^3 - 2U_1H^2 + T^2 \\
Y_3 &= -S_1H^3 + T(U_1H^2 - X_3) \\
Z_3 &= Z_1Z_2H \\
aZ_3^4 &= a(Z_3)^4
\end{align*}

- Otherwise, \( P + Q = [2]P = (X_3, Y_3, Z_3) \) where:

\begin{align*}
S &= 4X_1Y_1^2 \\
U &= 8Y_1^4 \\
M &= 3X_1^2 + (aZ_1^4) \\
T &= -2S + M^2 \\
X_3 &= T \\
Y_3 &= M(S - T) - U \\
Z_3 &= 2Y_1Z_1 \\
aZ_3 &= 2U(aZ_1^4)
\end{align*}

Algorithms 3.8 and 3.9 show how these laws can be implemented in practice.

### 3.2.4 Mixed Coordinates

Cohen et al. [CMO98] have recommended the idea of mixed coordinates, where the inputs and outputs to point additions and doublings may be in different coordinates. This can be very efficient when scalar multiplication is implemented with the base point stored in affine coordinates.

In order to use mixed coordinates it is sometimes necessary to convert a
point representation from one coordinate system to another to have the input in the required format for the addition or doubling algorithm. A point in affine coordinates, \((x_A, y_A)\), can easily be converted to a point \(Q\) in any of the other coordinate systems using the following equations:

**Projective:** \(Q = (x_A, y_A, 1)\)

**Jacobian:** \(Q = (x_A, y_A, 1)\)

**Chudnovsky Jacobian:** \(Q = (x_A, y_A, 1, 1, 1)\)

**Modified Jacobian:** \(Q = (x_A, y_A, 1, a)\)

None of these conversions require modular squarings, multiplications or inversions, making them quite fast. On the other hand, a conversion from a projective point \((X, Y, Z)\) to a point \(Q\) in any other coordinate system is more expensive, requiring several multiplications or squarings in each case as shown:

**Affine:** \(Q = (X Z^{-1}, Y Z^{-1})\)

**Jacobian:** \(Q = (X Z, Y Z^2, Z)\)

**Chudnovsky Jacobian:** \(Q = (X Z, Y Z^2, Z, Z^2, Z^3)\)

**Modified Jacobian:** \(Q = (X Z, Y Z^2, Z, a Z^4)\)

Conversion from a point in Jacobian coordinates to any of the other coordinates is generally more efficient than the projective case as shown below:

**Affine:** \(Q = (X Z^{-2}, Y Z^{-3})\)

**Projective:** \(Q = (X Z, Y, Z^3)\)

**Jacobian:** \(Q = (X, Y, Z)\)

**Chudnovsky Jacobian:** \(Q = (X, Y, Z, Z^2, Z^3)\)

**Modified Jacobian:** \(Q = (X, Y, Z, a Z^4)\)

The equations for conversion from Chudnovsky Jacobian and modified Jacobian coordinates to any of the other coordinate systems are the same as those for Jacobian coordinates. The number of operations required for each of these conversions between coordinate systems is shown in Table 3.1. It shows that conversion from affine coordinates to any of the other coordinate systems is very efficient because the conversions only consist of setting all of the \(Z\), \(Z^2\) and \(Z^3\) coordinates to one
3.2. Point Coordinates

Table 3.1: Point conversion complexity

<table>
<thead>
<tr>
<th>From \ To</th>
<th>Affine</th>
<th>Projective</th>
<th>Jacobian</th>
<th>Chudnovsky</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affine</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Projective</td>
<td>2M + I</td>
<td>-</td>
<td>2M + S</td>
<td>3M + S</td>
<td>3M* + 2S</td>
</tr>
<tr>
<td>Jacobian</td>
<td>3M + S + I</td>
<td>2M + S</td>
<td>-</td>
<td>M + S</td>
<td>1M* + 2S</td>
</tr>
<tr>
<td>Chudnovsky</td>
<td>3M + S + I</td>
<td>1M</td>
<td>-</td>
<td>-</td>
<td>1M* + 1S</td>
</tr>
<tr>
<td>Modified</td>
<td>3M + S + I</td>
<td>2M + S</td>
<td>-</td>
<td>M + S</td>
<td>-</td>
</tr>
</tbody>
</table>

M Multiplication  S Squaring  I Inversion  * May be reduced by one if \( a = p - 3 \)

and the \( aZ^4 \) coordinate to \( a \) (the elliptic curve parameter). On the other hand, conversion to Affine coordinates from any other coordinate system is inefficient because of the inversion involved. Conversion to or from projective coordinates is mostly less efficient than converting between the Jacobian variant coordinate systems. Conversions among the three Jacobian variants are the most efficient, and the use of these is therefore the most likely to provide the fastest mixed coordinate scalar multiplication algorithm. The data in Section 3.2.6 shows that this is indeed the case.

3.2.5 Addition and Doubling Algorithms

Although [CMO98] provides general formulae for addition and doubling in the various coordinate systems, it does not provide the detailed algorithms needed to produce an efficient implementation. Considerable effort is required to produce such algorithms, mainly due to the necessity of ensuring a low number of temporary variables are used by each algorithm. This optimization is particularly important for smart card implementations where memory to store such temporary variables is at a premium. Algorithms 3.1 to 3.7 describe addition and doubling in projective and Jacobian coordinates and variants (including mixed coordinates). For addition, the first two letters indicate the coordinates of the two input points. The third letter indicates the output coordinates. For example, \( \text{AJM} \) is an addition algorithm with input points in affine and Jacobian coordinates and an output point in modified Jacobian coordinates. For doubling, the first letter indicates the input point coordinates and the second letter indicates the output point coordinates. For example, \( \text{MJ} \) is a doubling algorithm with an input point in modified Jacobian coordinates and an output point in
Jacobian coordinates. Because three different Jacobian addition algorithms have been used, these are distinguished with a number at the end of the acronym. The two different Jacobian doubling algorithms are distinguished in the same way.

The JJ3, AJJ3, JJ1 and JJ1–3 algorithms have been taken from [HNM98] and are included for completeness. However, checks have been added to ensure that the point at infinity is not one of the points being added and that the points being added are not identical or each other’s negative. The other algorithms are derived from the formulae given in [CMO98] but have been optimized to reduce the number of temporary variables. Each algorithm assumes that the output will overwrite an input point.

Although Algorithms 3.3 and 3.4 are both Jacobian addition algorithms, they have both been included because JJ1 and AJJ1 are faster, but JJ3 and AJJ3 require one less variable. Although additional algorithms for Jacobian addition and doubling are available in [IEE00], these have not been included since they are less efficient than the most efficient Jacobian algorithms presented here. The inclusion of the Jacobian addition algorithm AJJ2 (Algorithm 3.8) is explained in Section 3.3.

Figures 3.1 to 3.7 show how the optimization process took place for the JJ1 algorithm. First, a version which was not optimized for variable usage was derived from the formulae in [CMO98]. This is shown as Algorithm 3.10. A diagram of this algorithm was then drawn, showing the dependencies of the calculations, as depicted in Figure 3.1. Of course, at this stage, the step numbers and variable names shown for each operation in the figure were not known.

The order of the operations was then determined with the intent to minimize the number of temporary variables necessary. The process started by analysing which operations could be completed first and of these, which operation would allow the lowest variable usage if actually completed first. That operation was then chosen to be completed next, followed by any operations that could be completed without requiring the use of additional variables. The whole process was then repeated at the next stage. The text accompanying Figures 3.1 to 3.7 gives a detailed description of how the order of the operations was chosen. Once the order of the operations had been determined, a check to ensure the points to be added were not the same was inserted, and the algorithm was then modified to accommodate other input points (for example, one affine and one Jacobian input). The end result is shown by Algorithm 3.3. The other algorithms (from
### 3.2. Point Coordinates

**Algorithm 3.1**  
PP Doubling  
\[ Q = Q + Q \]
where \( Q = (X, Y, Z) \)

```
if (Z == 0) return Q
T_1 = Y \cdot Z
T_3 = Z \cdot Z
T_4 = X \cdot X
T_3 = 3 \cdot T_4
T_3 = T_3 + T_4
Z = T_1 \cdot T_1
Z = 8 \cdot Z
Z = Z \cdot T_1
T_4 = T_3 \cdot T_3
T_5 = 2 \cdot T_1
T_1 = T_1 \cdot Y
T_2 = T_1 \cdot T_1
T_1 = T_1 \cdot X
Y = 4 \cdot T_1
T_1 = 8 \cdot T_1
T_4 = T_4 - T_1
X = T_4 \cdot T_3
Y = Y - T_4
Y = Y \cdot T_3
T_2 = 8 \cdot T_2
Y = Y - T_2
```

**Algorithm 3.2**  
APP and PPP Addition  
\[ Q = Q + P \]
where \( Q = (X, Y, Z) \)  
and \( P = (X_2, Y_2, Z_2) \)

```
if (P == \phi)  
{  
return Q  
}  
if (Z == 0)  
{  
Q = P  
{  
return Q  
}  
if doing APP  
{  
T_1 = Y_2 \cdot Z  
T_2 = X_2 \cdot Z  
}  
else  
{  
X = X \cdot Z_2  
Y = Y \cdot Z_2  
T_1 = Y_2 \cdot Z  
T_2 = X_2 \cdot Z  
Z = Z \cdot Z_2  
T_1 = T_1 - Y  
T_2 = T_2 - X  
if (T_2 == 0)  
{  
if (T_1 == 0)  
{  
Q = P  
{  
Double(Q)  
{  
return Q  
}  
else  
{  
Z = 0  
return Q  
}  
T_4 = T_2 \cdot T_2  
T_3 = T_4 \cdot X  
T_3 = T_1 \cdot T_1  
X = Z \cdot T_3  
T_3 = T_4 \cdot T_2  
Z = T_3 \cdot Z  
X = X - T_3  
Y = T_3 \cdot Y  
T_3 = 2 \cdot T_3  
X = X - T_3  
T_5 = T_5 - X  
T_5 = T_5 \cdot T_1  
Y = T_5 - Y  
X = T_2 \cdot X  
```

**Algorithms 3.1 and 3.2:** Projective doubling and addition
Algorithm 3.3
JJJ1 and AJJ1

Addition

\[ Q = Q + P, \text{ where} \]
\[ Q = (X, Y, Z) \text{ and} \]
\[ P = (X_2, Y_2) \text{ or} (X_2, Y_2, Z_2) \]

\[
\begin{align*}
\text{if } (P == \phi) \text{ return } Q \\
\text{if } (Z == 0) \\
\{ \quad Q = P \\
\} \text{ return } Q \\
\text{if } (P \text{ is not affine and } Z_2 \neq 1) \\
\{ T_1 = Z^2 \\
X = X \ast T_1 \\
Y = Y \ast T_1 \\
\} \text{ return } Q \\
\text{else} \\
\{ Z = 0 \\
\text{ return } Q \\
\} \text{ if } (P \text{ is not affine and } Z_2 \neq 1) \\
\{ Z = Z \ast Z_2 \\
\}
\end{align*}
\]

Algorithm 3.4 [HNM98]

JJJ3 and AJJ3

Addition

\[ Q = Q + P, \text{ where} \]
\[ Q = (X, Y, Z) \text{ and} \]
\[ P = (X_2, Y_2) \text{ or} (X_2, Y_2, Z_2) \]

\[
\begin{align*}
\text{if } (P == \phi) \text{ return } Q \\
\text{if } Z == 0 \\
\{ \quad Q = P \\
\} \text{ return } Q \\
\text{if } (P \text{ is not affine and } Z_2 \neq 1) \\
\{ T_1 = Z^2 \\
X = X \ast T_1 \\
Y = Y \ast T_1 \\
\} \text{ return } Q \\
\text{else} \\
\{ Z = 0 \\
\text{ return } Q \\
\} \text{ if } (P \text{ is not affine and } Z_2 \neq 1) \\
\{ Z = Z \ast Z_2 \\
\}
\end{align*}
\]

Algorithm 3.5 [HNM98]

JJJ1 and AJJ1 – 3

Doubling

\[ Q = Q + Q, \text{ where} \]
\[ Q = (X, Y, Z) \]

\[
\begin{align*}
\text{if } (Z == 0) \text{ return } Q \\
T_1 = Z^2 \\
X = Y \ast Z \\
Z = 2Z \\
\text{if } (a == p - 3) \\
\{ T_2 = X - T_1 \\
T_3 = T_2 \ast T_2 \\
T_1 = T_1 + T_2 \\
\} \text{ else} \\
\{ T_1 = T_1^2 \\
T_2 = a \ast T_1 \\
T_3 = X^2 \\
T_1 = T_2 + T_1 \\
T_2 = 2T_2 \\
T_1 = T_2 + T_1 \\
Y = 2Y \\
Y = Y^2 \\
T_2 = Y^2 \\
T_3 = Y \ast X \\
T_2 = T_2^2 \\
X = T_2^2 \\
X = X - Y \\
Y = Y - X \\
Y = Y \ast T_1 \\
Y = Y - T_2 \\
\}
\end{align*}
\]

Algorithms 3.3 to 3.5: Jacobian 1 and 3 addition and Jacobian 1 doubling
### 3.2. Point Coordinates

#### Algorithm 3.6

**ACC and CCC Addition**

\[
Q = Q + P, \quad \text{where} \\
Q = (X, Y, Z, Z^2, Z^3) \\
P = (X_2, Y_2) \text{ or } (X_2, Y_2, Z_2, Z_2^2, Z_2^3)
\]

if \(P == \phi\)
  \{ return \(Q\) \}

if \(Z == 0\)
  \{ \(Q = P\) \}
  \{ return \(Q\) \}

if \(P\) is not affine and \(Z_2 \neq 1\)
  \{ \(X = X * (Z_2^2)\) \}
  \(Z_2 = X_2 * (Z_2^2)\)
  \(Z = (Z_2^2) - X\)
  \(T_1 = (Z_2^2)^2\)

if \(P\) is not affine and \(Z_2 \neq 1\)
  \{ \(Z = Z * Z_2\) \}
  \(Z = (Z_2^3) + Z\)
  \(Z_2 = (Z_2^3) * T_1\)
  \(T_1 = X * T_1\)

if \(P\) is not affine and \(Z_2 \neq 1\)
  \{ \(Y = Y * (Z_2^2)\) \}
  \(Z_2 = Y_2 * (Z_2^3)\)
  \(Z_2 = (Z_2^3) - Y\)

if \((Z_2^2) == 0\)
  \{ if \((Z_2^3) == 0\)
    \{ \(Q = P\) \}
    \{ Double(Q) \}
    \{ return \(Q\) \} \}
  \{ else \}
    \{ \(Z = 0\) \}
    \{ \(Z_2 = 0\) \}
    \{ \(Z_3 = 0\) \}
    \{ return \(Q\) \} \}

\(X = (Z_2^3)^2\)
\(X = X - (Z_2^3)\)
\(X = X - T_1\)
\(T_1 = T_1 - X\)
\(Z_3 = (Z_3^3) * T_1\)
\(Y = Y * (Z_2^3)\)
\(Y = (Z_3^3) - Y\)
\(Z_2 = (Z_2^3)^2\)
\(Z_3 = Z * (Z_2^3)\)

#### Algorithm 3.7

**CC Doubling**

\[
Q = Q + Q, \quad \text{where} \\
Q = (X, Y, Z, Z^2, Z^3)
\]

if \(Z == 0\)
  \{ \(Q = Q + Q\) \}

\(Z = Y * Z\)
\(Z = 2Z\)
\(Y = Y^2\)
\(Z^3 = X * Y\)
\(Z^3 = 2 \times (Z^3)\)
\(Z^3 = 2 \times (Z^3)\)
\(X = X^2\)
\(Z^2 = (Z^2)^2\)
\(Z^2 = a * (Z^2)\)
\(Z^2 = (Z^2)^2 + X\)
\(Z^2 = (Z^2)^2 + X\)
\(X = (Z^2)^3\)
\(X = X - (Z^3)\)
\(X = X - (Z^3)\)
\(Z^3 = (Z^3)^3 - X\)
\(Z^3 = (Z^3)^3 - X\)
\(Y = Y^2\)
\(Y = Y^2\)
\(Y = (Z^3)^3 - Y\)
\(Z^2 = (Z^2)^3\)
\(Z^3 = Z * (Z^2)\)

**Algorithms 3.6 and 3.7:** Chudnovsky Jacobian addition and doubling
**Algorithm 3.8**

AJM, AJM - 3, JMM, AND - 3 Addition

\[ Q_{\text{out}} = Q_{\text{in}} + P, \]

where

\[ Q_{\text{in}} = (X, Y, Z) \text{ or } (X, Y, Z, aZ^4) \]

\[ P = (X_2, Y_2, Z_2) \text{ or } (X_2, Y_2, Z_2, aZ_2^4) \]

if AJJ2

\[ \{ \text{Q_{out} = (X, Y, Z) and } aZ^4 \text{ below is a temporary variable } \} \]

else

\[ \{ \text{Q_{out} = (X, Y, Z, aZ^4) } \} \]

if \((P == 0) \)

\[ \{ aZ^4 = a * Z^4 \text{ if necessary } \} \]

return \(Q_{\text{out}}\)

if \((Z == 0) \)

\[ \{ Q_{\text{out}} = P \} \]

return \(Q_{\text{out}}\)

if \((P \text{ is not affine and } Z_2 \neq 1) \)

\[ aZ^4 = Z_2^2 \]

\[ X = X \times aZ^4 \]

\[ aZ^4 = Z_2 \times aZ^4 \]

\[ Y = Y \times aZ^4 \]

\[ aZ^4 = Z_2 \]

\[ T_1 = X_2 \times aZ^4 \]

\[ T_1 = T_1 - X \]

\[ aZ^4 = Z \times aZ^4 \]

\[ aZ^4 = Y_2 \times aZ^4 \]

\[ aZ^4 = aZ^4 - Y \]

if \((P \text{ is not affine and } Z_2 \neq 1) \)

\[ \{ Z = Z \times Z_2 \} \]

\[ Z = Z \times T_1 \]

if \((T_1 == 0) \)

\[ \{ \text{Q_{out} = P} \} \]

\[ \{ \text{Double}(Q_{\text{out}}) \} \]

return \(Q_{\text{out}}\)

else

\[ \{ Z = 0 \} \]

\[ aZ^4 = 0 \]

return \(Q_{\text{out}}\)

\[ T_2 = T_2 \]

\[ T_1 = T_1 + T_2 \]

\[ Y = T_1 \times Y \]

\[ T_2 = X \times T_2 \]

\[ X = (aZ^4)^2 \]

\[ X = X - T_1 \]

\[ X = X - T_2 \]

\[ X = X - T_2 \]

\[ T_2 = T_2 - X \]

\[ T_2 = aZ^4 \times T_2 \]

\[ Y = T_2 - Y \]

if not doing AJJ2

\[ \{ aZ^4 = Z_2^2 \} \]

\[ aZ^4 = (aZ^4)^2 \]

\[ \text{if } a == p - 3 \]

\[ \{ aZ^4 = 0 - 3aZ^4 \} \]

else

\[ \{ aZ^4 = a \times aZ^4 \} \]

**Algorithm 3.9**

MM, MJ and JJ2 - 3

Doubling

\[ Q_{\text{out}} = Q_{\text{in}} + Q_{\text{in}} \]

where

\[ Q_{\text{in}} = (X, Y, Z) \text{ or } (X, Y, Z, aZ^4) \]

if AJJ2

\[ \{ Q_{\text{in}} = (X, Y, Z) \} \]

else

\[ \{ Q_{\text{in}} = (X, Y, Z, aZ^4) \} \]

if \((Z == 0) \)

return \(Q_{\text{out}}\)

if doing JJ2 - 3

\[ \{ T_2 = Z^2 \} \]

\[ T_1 = 2Y \]

\[ Z = T_1 \times Z \]

\[ T_1 = 2X \]

\[ T_1 = 2T_1 \]

\[ T_1 = T_1 \times Y \]

if doing JJ2 - 3

\[ \{ T_2 = (X - T_2) \times (X + T_2) \} \]

\[ X = 2T_2 \]

\[ T_3 = T_2 + X \]

else

\[ \{ T_2 = X^2 \} \]

\[ X = 2T_2 \]

\[ T_3 = T_2 + aZ^4 \]

\[ X = T_2^2 \]

\[ X = X - T_1 \]

\[ X = X - T_1 \]

\[ T_1 = T_1 \times X \]

\[ T_2 = T_2 \times T_1 \]

\[ Y = 2Y \]

\[ Y = Y^2 \]

\[ Y = 2Y \]

if doing MM

\[ \{ T_1 = 2Y \} \]

\[ aZ^4 = T_1 \times aZ^4 \]

\[ Y = T_2 - Y \]

# This line may be calculated as:

\[ T_3 = X - T_2 \]

\[ X = X + T_2 \]

\[ T_2 = X \times T_3 \]

In the smart card implementation, it was possible to use a coprocessor register in place of \(T_1\).

By using an extra addition, it is also possible to compute \(T_2\) without using the additional variable:

\[ X = X - T_2 \]

\[ T_2 = T_2 + T_2 \]

\[ T_2 = X + T_2 \]

\[ T_2 = T_2 \times X \]

**Algorithms 3.8 and 3.9:** Modified Jacobian and variants addition and doubling
3.2. Point Coordinates

Input: \((X, Y, Z)\) and \((X_2, Y_2, Z_2)\).

Output: \((X, Y, Z)\).

\[
egin{align*}
U_1 &= Z \cdot Z \\
U_2 &= Z_2 \cdot Z_2 \\
U_3 &= X \cdot U_2 \\
U_4 &= X_2 \cdot U_1 \\
U_5 &= Z \cdot U_1 \\
U_6 &= Z_2 \cdot U_2 \\
U_7 &= Y \cdot U_6 \\
U_8 &= Y_2 \cdot U_5 \\
U_9 &= U_4 - U_3 \\
U_{10} &= U_8 - U_7 \\
U_{11} &= U_9 \cdot U_9 \\
U_{12} &= U_9 \cdot U_{11} \\
U_{13} &= U_{10} \cdot U_{10} \\
U_{14} &= U_{11} \cdot U_3 \\
U_{15} &= U_{13} - U_{12} \\
U_{20} &= 2 \cdot U_{14} \\
X &= U_{18} - U_{20} \\
U_{15} &= U_{12} \cdot U_7 \\
U_{16} &= U_{14} - X \\
U_{19} &= U_{10} \cdot U_{16} \\
Y &= U_{19} - U_{15} \\
U_{17} &= Z \cdot Z_2 \\
Z &= U_{17} \cdot U_9
\end{align*}
\]

Algorithm 3.10: Jacobian 1 addition with non-optimal variable usage
Algorithm 3.1 to Algorithm 3.7 with the exception of Algorithms 3.4 and 3.5) were obtained in a similar manner. This method of optimization gives quite favourable results. For example, the Jacobian addition algorithm presented in the IEEE Std 1363 [IEE00] (where it is called projective addition) requires the use of four temporary variables, whereas this optimization method has been used to ensure only three variables are necessary.

### 3.2.6 Point Addition and Doubling Efficiency Comparison

Table 3.2 contains the times for addition and doubling in various coordinate systems. All calculations were performed for curves over a 160 bit prime. The first column specifies the coordinates used in the algorithm, where the naming convention used is described in Section 3.2.5.

In order to increase efficiency, the $a$ parameter of the elliptic curve is sometimes set to be $p - 3$ [BSS99, pp.59–60]. This enables a faster Jacobian doubling algorithm or a faster modified Jacobian addition algorithm. However, not all curves can be represented in this way. A “$-3$” at the end of an acronym indicates an algorithm with $a = p - 3$.

Next, the table gives the number of additions, subtractions, multiplications, squarings, inversions and shifts required for each addition and doubling algorithm. The total number of multiplications and squarings is then given, which can be used to make a rough estimate of the time it would take to run the algorithm. Actual timings on a PC are then given, as well as an estimated time for the smart card (based on estimated times for individual smart card operations). When the actual PC timings and the smart card estimates are sorted according to speed, they are mostly in the same order, indicating that the estimations are reasonable.

The table also gives the minimum number of variables the same size as the modulus $p$ that are required for each algorithm. This value includes input and output point coordinates as well as temporary variables, and assumes that the output point will overwrite an input point. The value also includes the elliptic curve $a$ parameter for those algorithms requiring its use; such algorithms are indicated with an asterisk. In some instances, the minimum number of variables required was not calculated because the algorithm could not be used in an efficient scalar multiplication algorithm. This is discussed further below.

Lastly, the table gives times for converting points from one coordinate system to another. These operations are sometimes needed when using mixed coordi-
Stage 1 It is possible to find either $U_1$, $U_2$ or $U_{17}$ as the first computation. In order to choose which one to do first, it was noted that when $U_3$ and $U_7$ are calculated, these can overwrite the values $X$ and $Y$ (since the output point overwrites the input point and $X$ and $Y$ are not used in any other calculations). Although a temporary variable ($T_1$) is needed to calculate $U_3$ and $U_7$, once they have both been calculated, no temporary variables are needed to store intermediate results for use in other calculations. Therefore, the initial strategy was chosen to be the calculation of $U_3$ and $U_7$. This can be accomplished by first finding $U_2$ and storing it in $T_1$, then calculating $U_3$ and storing it in $X$. Then $U_6$ can be calculated and can overwrite $T_1$ since $U_2$ is not required in any other calculations. Finally, $U_7$ can be found and stored in $Y$, which frees $T_1$.

Fig. 3.1: Jacobian 1 point addition—Stage 1
**Stage 2** It is now possible to calculate either \( U_1 \) or \( U_{17} \). Since \( U_{17} \) is used only for the calculation of the output value \( Z \), and since this output must overwrite the input value \( Z \), calculation of \( U_{17} \) is left until it can overwrite the input value \( Z \), which will in turn allow the output value \( Z \) to overwrite \( U_{17} \). Therefore, \( U_1 \) is calculated next and stored in \( T_1 \). Either \( U_4 \) or \( U_5 \) can then be calculated. No matter which one is found first, an extra temporary variable is needed to store intermediate results for later use. Therefore, \( U_4 \) is found next and stored in \( T_2 \). Now \( U_1 \) is only needed by \( U_5 \), so \( U_5 \) is calculated next and overwrites \( T_1 \). Similarly, \( U_5 \) is only needed by \( U_9 \) and again overwrites \( T_1 \). \( U_9 \) is only needed by \( U_{10} \), so \( U_{10} \) is calculated and overwrites \( T_1 \). Going back to \( U_4 \) which is stored in \( T_2 \), \( U_4 \) is only needed by \( U_9 \), so \( U_9 \) is found next and overwrites \( T_2 \).

Fig. 3.2: Jacobian 1 point addition—Stage 2
Stage 3  It is now possible to calculate $U_{17}$, $U_{13}$ or $U_{11}$. Both $U_{13}$ and $U_{11}$ require the use of an extra variable at this point, but $U_{17}$ does not, so it is calculated next and overwrites $Z$. The output value $Z$ is then calculated and overwrites the $Z$ storage location.

Fig. 3.3: Jacobian 1 point addition—Stage 3
Stage 4 Either $U_{13}$ or $U_{11}$ can be calculated at this stage. Both require the use of an extra temporary variable, $T_3$, but finding $U_{13}$ provides no immediate gain since it is only possible to calculate $U_{11}$ afterwards and that would then require the use of a fourth temporary variable. Therefore $U_{11}$ is calculated next and stored in $T_3$. $U_9$ is now only needed by $U_{12}$, so $U_{12}$ is calculated next and overwrites $T_2$. Now $U_{11}$ is only needed by $U_{14}$, so $U_{14}$ is found next and it overwrites $T_3$. This frees the value $X$ which had been used to store $U_3$.

Fig. 3.4: Jacobian 1 point addition—Stage 4
Stage 5 It is now possible to find $U_{20}$, $U_{15}$ or $U_{13}$. If $U_{20}$ is found next, it uses up an extra variable and then $U_{13}$ or $U_{15}$ must be calculated in any case. Therefore, either $U_{13}$ or $U_{15}$ should be found next; it does not matter which one is found first. Therefore, $U_{13}$ is found next and put in the $X$ memory location. $U_{15}$ is then found and overwrites the $Y$ memory location.

Fig. 3.5: Jacobian 1 point addition—Stage 5
Stage 6 Now that $U_{15}$ has been calculated, $U_{12}$ is only needed by $U_{18}$, so $U_{18}$ is found next and it overwrites $T_2$. This also frees the $X$ memory location. It is then only possible to find $U_{20}$ next, which overwrites $X$. Once $U_{20}$ has been found, there is again only one possible value to find next, which is $X$ and it again overwrites memory location $X$.

Fig. 3.6: Jacobian 1 point addition—Stage 6
3.2. Point Coordinates

Stage 7  Calculation of the remaining three values is fairly straightforward since there is only one possible order of calculation. Care should be taken to ensure that the output value $Y$ overwrites memory location $Y$ in the final step.

Fig. 3.7: Jacobian 1 point addition—Stage 7
### Table 3.2: Addition and doubling efficiencies

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Add</th>
<th>Subtract</th>
<th>Multiply</th>
<th>Square</th>
<th>Other</th>
<th>Mul. +</th>
<th>Square</th>
<th>Pentium Timings (ms)</th>
<th>Smart card Estimate</th>
<th>Number of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point Addition</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAC</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0.027</td>
<td>46%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AAM</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>0.035</td>
<td>54%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AJJ2</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>-</td>
<td>68.05%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>11</td>
<td>-</td>
<td>68.17%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AJJ1</td>
<td>1</td>
<td>6</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>-</td>
<td>68.17%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>-</td>
<td>68.84%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AJM−3</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td>-</td>
<td>77.67%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AMM−3</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td>-</td>
<td>77.67%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AJM</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>14</td>
<td>-</td>
<td>78.62%  *</td>
<td>9*</td>
<td></td>
</tr>
<tr>
<td>AMM</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>14</td>
<td>-</td>
<td>78.62%  *</td>
<td>9*</td>
<td></td>
</tr>
<tr>
<td>AJJ3</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>3</td>
<td>Shift</td>
<td>11</td>
<td>-</td>
<td>83.29%</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>CCC</td>
<td>0</td>
<td>7</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td>0.059</td>
<td>84.48%</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>PPP</td>
<td>1</td>
<td>6</td>
<td>12</td>
<td>2</td>
<td>0</td>
<td>14</td>
<td>0.059</td>
<td>86.93%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>JJ1</td>
<td>1</td>
<td>6</td>
<td>12</td>
<td>4</td>
<td>0</td>
<td>16</td>
<td>0.067</td>
<td>91.90%</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>JJM</td>
<td>0</td>
<td>7</td>
<td>13</td>
<td>6</td>
<td>0</td>
<td>19</td>
<td>0.077</td>
<td>100.00%</td>
<td>10*</td>
<td></td>
</tr>
<tr>
<td>MMM</td>
<td>0</td>
<td>7</td>
<td>13</td>
<td>6</td>
<td>0</td>
<td>19</td>
<td>0.078</td>
<td>100.00%</td>
<td>11*</td>
<td></td>
</tr>
<tr>
<td>AAA</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>Inv.</td>
<td>3</td>
<td>0.237</td>
<td>64.04%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td><strong>Point Doubling</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AJ</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>0.030</td>
<td>UC</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>MJ</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td>0.033</td>
<td>50.03%</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>9</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>0.038</td>
<td>58.00%</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>JJ2−3</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>-</td>
<td>58.67%</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>JJ1−3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>Shift</td>
<td>8</td>
<td>-</td>
<td>59.75%</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>0.038</td>
<td>UC</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>11</td>
<td>0.049</td>
<td>65.84%</td>
<td>6*</td>
<td></td>
</tr>
<tr>
<td>JJ1</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>Shift</td>
<td>10</td>
<td>0.044</td>
<td>67.91%</td>
<td>6*</td>
<td></td>
</tr>
<tr>
<td>PP</td>
<td>14</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>12</td>
<td>0.055</td>
<td>70.13%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>Inv.</td>
<td>4</td>
<td>0.240</td>
<td>653.71%</td>
<td>UC</td>
<td></td>
</tr>
<tr>
<td><strong>Point Conversion</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C to P</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>6.00%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J,M to C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.008</td>
<td>8.87%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C to M</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>-</td>
<td>9.25%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J to M</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0.011</td>
<td>12.51%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J,M to P</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>13.69%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P to J</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>14.86%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P to C</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>-</td>
<td>20.86%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P to M</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>24.90%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A  Affine
P  Projective
J  Jacobian
C  Chudnovsky Jacobian
M  Modified Jacobian

−3 Optimized for \( a = p − 3 \) [BSS99, pp.59-60]
* Including the \( a \) parameter
1, 2 or 3 Different versions for the same coordinate system
UC Uncalculated because inefficient
nates, as discussed in Section 3.2.4.

Although the AAC, AAM, AJ and AM operations are very fast, these methods are not very useful in scalar multiplication routines. Because a scalar multiplication consists of repeated additions and doublings and the output of an addition or doubling is never in affine coordinates for an efficient algorithm, in order to use the AAC, AAM, AJ or AM operations in scalar multiplication, the input point must be converted to affine coordinates. This requires an inversion and makes any scalar multiplication algorithm that uses these addition and doubling algorithms computationally intensive.

### 3.3 Scalar Multiplication

Scalar multiplication is the basic cryptographic operation of an ECC, and consists of a series of point additions and doublings. The scalar multiplication algorithm chosen for the smart card implementation was the binary method [BSS99, p.63], because it does not require a precomputation and therefore uses less memory, unlike other more efficient methods. When implementing the binary method, there are a number of options available. Because point subtraction takes about the same time as point addition (it only requires one extra field subtraction), it is possible to use a signed digit representation of the scalar in order to increase efficiency. A commonly used representation is the non-adjacent form (NAF) [BSS99, pp.67-68]. The NAF increases the length of the scalar by at most one bit and has no adjacent non-zero digits. Because the NAF represents a scalar with a smaller number of non-zero digits, fewer point additions are required in a binary scalar multiplication using this representation since each non-zero digit corresponds to one point addition. Algorithms for conversion of a scalar to NAF and subsequent use of the converted scalar in scalar multiplication are shown in Section 2.2.3.

The estimated scalar multiplication figures in Table 3.3 indicate that using a NAF scalar should make the scalar multiplication about 10% faster than when using an unsigned scalar. The values in Table 3.4 show an increase in efficiency of about 6% for the time required for a person to digitally sign a value using ECDSA (see Section 2.4) and 4% to 17% (depending on the settings used) for the time required for another person to verify the validity of the digital signature.

Another option is to use the two-in-one variant of binary scalar multiplication shown in Algorithm 2.15 that computes \([k_1]P + [k_2]Q\), where \(k_1\) and \(k_2\) are scalars.
Table 3.3: Estimated time for signed (NAF) and unsigned scalar multiplication on the smart card using the binary method

<table>
<thead>
<tr>
<th>Addition Algorithm</th>
<th>Doubling Algorithm</th>
<th>NAF $a = p - 3$</th>
<th>NAF $a \neq p - 3$</th>
<th>Unsigned $a = p - 3$</th>
<th>Unsigned $a \neq p - 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJJ2–3</td>
<td>MJ/MM</td>
<td>77.16%</td>
<td></td>
<td>87.91%</td>
<td></td>
</tr>
<tr>
<td>AJJ2</td>
<td>JJ2–3</td>
<td>77.29%</td>
<td></td>
<td>87.77%</td>
<td></td>
</tr>
<tr>
<td>AJM</td>
<td>MJ/MM</td>
<td>77.46%</td>
<td></td>
<td></td>
<td>88.36%</td>
</tr>
<tr>
<td>AJJ1</td>
<td>JJ1–3</td>
<td>78.36%</td>
<td></td>
<td>88.85%</td>
<td></td>
</tr>
<tr>
<td>AMM–3</td>
<td>MM</td>
<td>79.66%</td>
<td></td>
<td>91.67%</td>
<td></td>
</tr>
<tr>
<td>AMM</td>
<td>MM</td>
<td>79.95%</td>
<td></td>
<td>92.12%</td>
<td></td>
</tr>
<tr>
<td>AMM</td>
<td>MM</td>
<td>80.60%</td>
<td></td>
<td>93.09%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>JJ1–3</td>
<td>81.33%</td>
<td></td>
<td>93.34%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>MM</td>
<td>82.56%</td>
<td></td>
<td>96.05%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>JJ1–3</td>
<td>83.08%</td>
<td></td>
<td>95.98%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>JJ1–3</td>
<td>83.08%</td>
<td></td>
<td>95.98%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>MM</td>
<td>84.14%</td>
<td></td>
<td>98.44%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>MM</td>
<td>84.38%</td>
<td></td>
<td>94.94%</td>
<td></td>
</tr>
<tr>
<td>AMM–3</td>
<td>JJ1–3</td>
<td>85.23%</td>
<td></td>
<td>99.22%</td>
<td></td>
</tr>
<tr>
<td>AJJ1</td>
<td>JJ1</td>
<td>86.15%</td>
<td></td>
<td>96.60%</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>MJ/MM</td>
<td>86.25%</td>
<td></td>
<td>101.63%</td>
<td></td>
</tr>
<tr>
<td>MMM</td>
<td>MM</td>
<td>86.63%</td>
<td></td>
<td>102.20%</td>
<td></td>
</tr>
<tr>
<td>AJJ1</td>
<td>CC</td>
<td>86.94%</td>
<td></td>
<td>98.81%</td>
<td></td>
</tr>
<tr>
<td>CCC</td>
<td>MM</td>
<td>87.44%</td>
<td></td>
<td>103.43%</td>
<td></td>
</tr>
<tr>
<td>JJJ</td>
<td>MM</td>
<td>88.01%</td>
<td></td>
<td>104.28%</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>PP</td>
<td>88.26%</td>
<td></td>
<td>98.70%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>JJ1</td>
<td>89.13%</td>
<td></td>
<td>101.09%</td>
<td></td>
</tr>
<tr>
<td>CCC</td>
<td>CC</td>
<td>89.27%</td>
<td></td>
<td>102.32%</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>JJ1</td>
<td>90.87%</td>
<td></td>
<td>103.72%</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>CC</td>
<td>92.56%</td>
<td></td>
<td>107.29%</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>CC</td>
<td>92.56%</td>
<td></td>
<td>107.29%</td>
<td></td>
</tr>
<tr>
<td>AMM</td>
<td>CC</td>
<td>93.09%</td>
<td></td>
<td>108.10%</td>
<td></td>
</tr>
<tr>
<td>AMM</td>
<td>JJ1</td>
<td>93.32%</td>
<td></td>
<td>107.42%</td>
<td></td>
</tr>
<tr>
<td>JJJ</td>
<td>JJ1</td>
<td>93.56%</td>
<td></td>
<td>107.78%</td>
<td></td>
</tr>
<tr>
<td>CCC</td>
<td>JJ1</td>
<td>94.02%</td>
<td></td>
<td>108.47%</td>
<td></td>
</tr>
<tr>
<td>PPP</td>
<td>PP</td>
<td>94.12%</td>
<td></td>
<td>107.54%</td>
<td></td>
</tr>
<tr>
<td>JJJ</td>
<td>CC</td>
<td>94.35%</td>
<td></td>
<td>110.00%</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>JJ1</td>
<td>95.07%</td>
<td></td>
<td>110.06%</td>
<td></td>
</tr>
<tr>
<td>MMM</td>
<td>CC</td>
<td>99.77%</td>
<td></td>
<td>118.18%</td>
<td></td>
</tr>
<tr>
<td>MMM</td>
<td>JJ1</td>
<td>100.00%</td>
<td></td>
<td>117.50%</td>
<td></td>
</tr>
</tbody>
</table>
and \( P \) and \( Q \) are points on the curve [MvOV96, p. 618], [WHB98]. If there is insufficient memory to store \((P + Q)\) or \((P - Q)\), these points need not be stored, but may be computed each time they are needed. However, this can cause the algorithm to be slower, depending on the coordinates in which the temporary points \((P + Q)\) and \((P - Q)\) are stored and the time taken to convert the points to these coordinates. Estimates based on the data in Table 3.2 indicate that a two-in-one scalar multiplication takes about 60\% to 70\% of the time that two separate scalar multiplications take, depending on the options chosen. The data in Table 3.4 indicates that ECDSA verification using this method actually takes 65\% to 70\% of the time taken when not using the two-in-one scalar multiplication.

The basic coordinate system chosen for the smart card implementation was the mixed Jacobian and modified Jacobian coordinate system, with one input to the addition in affine coordinates \((AJM/MJ/MM)\). This coordinate system was chosen because the estimations in Table 3.3 indicate that it is the most efficient coordinate system to use if \( a \neq p - 3 \). One of the inputs to the addition was chosen to be affine because of the faster implementation available and because fewer variables were required in this case.

In order to see how much efficiency could be gained by setting \( a = p - 3 \), the \( AJM \) addition was modified slightly to create the \( AJM-3/MJ/MM \) coordinates. Because Jacobian coordinates allow a further speedup from setting \( a = p - 3 \) which is not available when using modified Jacobian coordinates, the \( AJM-3/MJ/MM \) algorithms were further modified to allow this speedup and to use only Jacobian coordinates, resulting in the \( AJJ2/JJ2-3 \) coordinates.

Because of the limited amount of memory available, Jacobian coordinates were also implemented in order to see how much speed needed to be sacrificed in order to use fewer variables. Two different addition algorithms were available—one that used three temporary variables but was faster \((AJJ1)\), and one that used two temporary variables but was slower \((AJJ3)\). These algorithms were implemented for \( a \neq p - 3 \) and also optimized for \( a = p - 3 \), giving the four sets of coordinates \( AJJ1/JJ1, AJJ1/JJ1-3, AJJ3/JJ1 \) and \( AJJ3/JJ1-3 \). Figure 3.8 shows the number of variables that can be saved for each coordinate system and scalar multiplication setting.

In order to decrease code size and RAM usage as much as possible, inline functions were not used, as many variables as possible were made global variables to avoid pushing them onto the stack multiple times when they were passed to
multiple procedures, and generality was removed from some procedures when the number of different inputs actually passed to the procedure was smaller than the number of different possible inputs the procedure allowed. These optimizations saved about 20% of the code size of an earlier version of the code in which these optimizations had not been made.

Figure 3.9 displays the running time (as a percentage of the longest running time) of the ECDSA signature and verification for each of the options that was implemented. The settings that were used were signed or unsigned scalars (NAF or no NAF), two separate multiplications or a two-in-one multiplication for the verification, and when a two-in-one multiplication was performed, whether there were two temporary points calculated at the beginning of the scalar multiplication \((P + Q)\) and \((P - Q)\), one point \((P + Q)\) for no NAF and \((P - Q)\) for NAF) or no points. The temporary points were stored in affine format in order to be able to guarantee one affine input to the addition algorithm, however, the time to calculate the points can outweigh any time saved in some instances. Although storing the points in Jacobian coordinates may have given a faster implementation by avoiding the inversion per point needed to convert the points to affine, this option was not implemented because of the increased code size for the addition and increased number of variables that would have been required (one extra
3.3. Scalar Multiplication

Fig. 3.9: ECDSA signature and verification timings on the smart card simulator

variable per point). In any case, the time taken to calculate each point is only about 2% of the total verification time (this is 1% on the graph where 100% is the slowest verification speed and 50% is the speed of the verification in question), and thus storing the points in Jacobian format would not greatly increase efficiency, bearing in mind that even less than the 2% of verification time per point would be saved because of the slower addition algorithm being used.

Figure 3.9 shows that the AJM/MJ/MM coordinates are best when \( a \neq p - 3 \). When \( a = p - 3 \) and the NAF form of the scalar is used, the AJM–3/MJ/MM coordinates are fastest. Using a NAF scalar always gives a faster result than an unsigned scalar and the two-in-one multiplication algorithm enables a faster verification.

Figure 3.10 gives the code size for each of the different implementations. Because the interface assumes that all points are passed to it in compressed form (i.e. an affine \( x \)-coordinate and one byte to specify the sign of the \( y \)-coordinate and point format), a point uncompression procedure was implemented. The main part of this procedure is the square root algorithm, which is long when \( p \equiv 1 \pmod{4} \). It is possible to save a further 528 bytes of code space from any of the implementations by omitting the point uncompression procedure for when \( p \equiv 1 \pmod{4} \) and not using these curves.
The data used to generate Figures 3.8, 3.9 and 3.10 is displayed in Table 3.4. It shows the ECDSA verification and signature timings, code size and number of variables saved (not used) for each different efficiency option.

The optimal choice of coordinate system and scalar multiplication algorithm depends on how important speed is compared to code size and minimal RAM usage. If speed is considered the most important, the best compromise may be to choose a signed scalar with two-in-one multiplication and no temporary points stored and using either the AJM/MJ/MM or AJJ2/JJ2–3 coordinates, whichever is appropriate. This gives good signature and verification speeds, and saves a medium amount of code space and variables.

### 3.4 Comparison of RSA with ECDSA

Table 3.5 compares the speed of ECDSA signing and verifying (using the two-in-one scalar multiplication algorithm with two precomputed points, a signed scalar and AJM/MJ/MM coordinates) to the speed of RSA signing and verifying on the smart card and the publicly available MIRACL library [Sha00] on a Pentium. It should
### 3.4. Comparison of RSA with ECDSA

Table 3.4: Code size, number of variables saved and timings for the smart card

<table>
<thead>
<tr>
<th>Subfield Migration</th>
<th>Number of Temporary Polisher (NAF)</th>
<th>Signed Scalar Used</th>
<th>Doubling Algorithm(s)</th>
<th>ECDSA Signature Time (Simulated)</th>
<th>ECDSA Verification Time (Simulated)</th>
<th>Code Size (bytes)</th>
<th>Variables Saved in Signature</th>
<th>Variables Saved in Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>AJM-3</td>
<td>42.9%</td>
<td>84.3%</td>
<td>5321</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>AJJ2</td>
<td>42.3%</td>
<td>86.1%</td>
<td>5060</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>AJM</td>
<td>42.7%</td>
<td>85.9%</td>
<td>5280</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>AJJ1</td>
<td>44.4%</td>
<td>90.0%</td>
<td>5060</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>J1</td>
<td>47.7%</td>
<td>93.7%</td>
<td>5120</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>AJJ3</td>
<td>46.9%</td>
<td>96.1%</td>
<td>5156</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>no</td>
<td>J1</td>
<td>51.2%</td>
<td>100.0%</td>
<td>5204</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>yes</td>
<td>AJM-3</td>
<td>40.1%</td>
<td>77.1%</td>
<td>5321</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>yes</td>
<td>AJJ2</td>
<td>40.7%</td>
<td>78.3%</td>
<td>5380</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>yes</td>
<td>AJM</td>
<td>40.3%</td>
<td>76.9%</td>
<td>5592</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>yes</td>
<td>AJJ1</td>
<td>42.6%</td>
<td>82.3%</td>
<td>5388</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>yes</td>
<td>J1</td>
<td>45.0%</td>
<td>86.6%</td>
<td>5436</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>no</td>
<td>0</td>
<td>yes</td>
<td>AJJ3</td>
<td>44.6%</td>
<td>86.3%</td>
<td>5472</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>AJM-3</td>
<td>42.9%</td>
<td>59.0%</td>
<td>5332</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>AJJ2</td>
<td>42.4%</td>
<td>60.0%</td>
<td>5068</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>AJM</td>
<td>42.7%</td>
<td>60.9%</td>
<td>5288</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>AJJ1</td>
<td>44.4%</td>
<td>62.1%</td>
<td>5076</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>J1</td>
<td>47.8%</td>
<td>63.5%</td>
<td>5124</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>AJJ3</td>
<td>46.9%</td>
<td>68.2%</td>
<td>5164</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>no</td>
<td>J1</td>
<td>51.3%</td>
<td>69.4%</td>
<td>5212</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>AJM-3</td>
<td>40.2%</td>
<td>59.6%</td>
<td>5112</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>AJJ2</td>
<td>40.8%</td>
<td>59.9%</td>
<td>5456</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>AJM</td>
<td>40.2%</td>
<td>59.6%</td>
<td>5664</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>AJJ1</td>
<td>42.7%</td>
<td>53.0%</td>
<td>5464</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>J1</td>
<td>45.1%</td>
<td>56.1%</td>
<td>5512</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>AJJ3</td>
<td>44.7%</td>
<td>57.7%</td>
<td>5548</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>J1</td>
<td>47.3%</td>
<td>60.1%</td>
<td>5596</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>AJM-3</td>
<td>42.9%</td>
<td>52.3%</td>
<td>5580</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>AJJ2</td>
<td>42.4%</td>
<td>52.4%</td>
<td>5112</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>AJM</td>
<td>42.7%</td>
<td>52.5%</td>
<td>5332</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>AJJ1</td>
<td>44.4%</td>
<td>55.1%</td>
<td>5120</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>J1</td>
<td>47.8%</td>
<td>57.7%</td>
<td>5172</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>AJJ3</td>
<td>46.9%</td>
<td>59.9%</td>
<td>5208</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>J1</td>
<td>51.2%</td>
<td>62.2%</td>
<td>5256</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>AJM-3</td>
<td>40.1%</td>
<td>49.4%</td>
<td>5921</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>AJJ2</td>
<td>40.8%</td>
<td>50.0%</td>
<td>5652</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>AJM</td>
<td>40.3%</td>
<td>49.3%</td>
<td>5876</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>AJJ1</td>
<td>42.7%</td>
<td>52.0%</td>
<td>5660</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>J1</td>
<td>45.1%</td>
<td>54.6%</td>
<td>5704</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>AJJ3</td>
<td>44.7%</td>
<td>56.9%</td>
<td>5740</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>yes</td>
<td>J1</td>
<td>47.2%</td>
<td>58.4%</td>
<td>5792</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>AJM-3</td>
<td>40.0%</td>
<td>49.5%</td>
<td>5828</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>AJJ2</td>
<td>40.7%</td>
<td>50.2%</td>
<td>5548</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>AJM</td>
<td>40.3%</td>
<td>49.2%</td>
<td>5780</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>AJJ1</td>
<td>42.6%</td>
<td>52.2%</td>
<td>5556</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>J1</td>
<td>45.1%</td>
<td>55.0%</td>
<td>5612</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>AJJ3</td>
<td>44.6%</td>
<td>56.5%</td>
<td>5640</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>yes</td>
<td>J1</td>
<td>47.2%</td>
<td>58.4%</td>
<td>5606</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
be noted that the RSA verification times are faster because a small exponent has been used. The table gives the times as a percentage of the EC signature time on that platform. These results demonstrate that the ratio of the time taken for the EC operations to the time taken for the RSA signature is about the same for both the smart card and the Pentium. However, on the smart card, the RSA signature was mostly performed in hardware, whereas a large number of the EC computations had to be performed in software, which would slow down the EC times. The fact that the RSA signature time on the smart card is longer than the time for the Pentium shows that we have quite an efficient implementation of ECDSA verification and signing on the smart card.

### 3.5 Recommendations for Future Coprocessors

The implementation work described in this chapter was on a smart card designed for RSA. Although the coprocessor originally designed for RSA was able to be effectively utilized in the implementation of EC arithmetic, improvement in the quality of an ECC on a smart card can be obtained by using smart cards designed with coprocessor support specifically for elliptic curves. The above implementation has lead to some recommendations for the design of such future coprocessors. The recommendations consist of a list of functions that could be included on future coprocessors to increase the efficiency of elliptic curve cryptosystems over the field $GF(p)$.

#### 3.5.1 Reduction of $f$ modulo $r$

One operation which needs to be performed in elliptic curve cryptosystems (in particular in DSA signatures and verifications) is the reduction of $f$ modulo $r$ when $f > r$. This can not be done on the current coprocessor since a reduc-
tion can only performed in conjunction with an addition, subtraction or scalar multiplication and these operations all require the operands to be less than the modulus.

### 3.5.2 Inversion of $z$ modulo $p$

Another operation also required in elliptic curve cryptosystems (in particular, when converting a point to the affine coordinate system at the end of a scalar multiplication) is inversion of $z$ modulo $p$. This operation is currently done using an exponentiation (as discussed in Section 3.1.2). However, other algorithms such as the BEGCD and EEA can provide a faster implementation. It is recommended that an inversion operation be included in future coprocessors to reduce code size and increase efficiency.

### 3.5.3 Division by 2 Modulo $p$ and/or Right Shift

Some point addition and doubling algorithms require a division by two. This can be implemented by checking whether the number to be shifted is even or odd. If it is even, right shift it by one place. Otherwise if it is odd, add the modulus and then right shift the result. Alternatively, if the number is odd it may be necessary to right shift the number and modulus by one, add the two results and then increment by one to avoid storing numbers bigger than the modulus. Unfortunately, neither the right shift operation nor division by two is currently available on the coprocessor. It is recommended that at least a right shift operation be made available, if not a division by 2 (mod $p$).

### 3.5.4 Increment and Decrement

Currently there are no increment or decrement operations available on the coprocessor. Decrement is currently used in inversion modulo $p$ and increment is used in division by two and point uncompression. It is expected that using the ordinary coprocessor addition and subtraction routines does not give optimal performance. It is therefore recommended that increment and decrement operations be made available.
3.5.5 Modular Arithmetic Using a Special Modulus

Brown et al. [BHLM01] have demonstrated that it is possible to achieve a significant speed up of modular reduction by using a modulus with a special form such as the NIST primes [NIS00]. Alternatively, pseudo-Mersenne primes can also provide a significant speedup in modular reduction. Both of types of primes are described in Section 3.1.1. Such primes require a specialized modular reduction algorithm to realize the efficiency gain which is possible, but the coprocessor can not currently be utilized in such a modular reduction. In fact, it has been found that multiplication and reduction for these primes is slower than using the modular multiplication provided by the coprocessor. If the coprocessor was able to perform reductions for such specialized moduli, the speed of elliptic curve cryptosystems using these moduli would be greatly increased.

3.5.6 Addition, Doubling and Scalar Multiplication of Points in Hardware

In order to increase the efficiency of the elliptic curve cryptosystem, addition and doubling of points could be implemented in hardware using the optimal coordinate system. Scalar multiplication could also be implemented in the hardware. However, it should be noted that access to addition and doubling routines may still be required even if scalar multiplication is available in the hardware. Also, if the scalar used in the scalar multiplication algorithm is a short-term or long-term private key, then the algorithm should be resistant to side channel attacks, as discussed in Chapter 4.

3.5.7 Conversion to and from NAF

Implementing the conversion to NAF for scalars in hardware may marginally increase efficiency and decrease code size. Currently conversion from NAF to binary format is not used. However, if it was available, two variables (typically 20 bytes each) could be saved in the ECDSA signature operation. However, consideration must be given to the use of scalars converted to this form, and if the scalar is a short-term or long-term private key, resistance to side channel attacks must be ensured, as discussed in Chapter 4.
3.6 Conclusion

The efficiency of various coordinate systems and scalar multiplication algorithms available when implementing an elliptic curve cryptosystem over the field \( GF(p) \) on a smart card with a coprocessor for support of modular arithmetic operations has been investigated. Firstly, the choice of the prime \( p \) and modular reduction and inversion algorithms were discussed. This was followed by an analysis of the efficiency of addition and doubling in the coordinate systems available for elliptic curve point operations (including variable usage, estimated smart card speed and the number and type of field operations required). Scalar multiplication speed using various combinations of addition and doubling algorithms and signed and unsigned scalars was then estimated. Several coordinate systems which were estimated to provide either a fast implementation or a low level of memory usage were then chosen for implementation. Also implemented were scalar multiplication algorithms using signed and unsigned scalars to find \([k_1]P\) and \([k_1]P + [k_2]Q\) (where \(k_1\) and \(k_2\) are scalars and \(P\) and \(Q\) are points on the curve). The code size, speed and variable usage for each different choice of addition, doubling and scalar multiplication algorithm were presented. Based on this data, a fast coordinate system and scalar multiplication algorithm with medium code size and variable usage were recommended. However, the data in this chapter is also sufficient to make an informed choice of algorithm for other requirements. Algorithms for addition and doubling in Jacobian, Chudnovsky Jacobian and modified Jacobian coordinates with a low number of temporary variables have been presented in Section 3.2.5, and a description of the optimization process used to produce these algorithms was provided. Finally, recommendations for future coprocessors have been made which, if implemented, will increase the efficiency of elliptic curve cryptosystems on these coprocessors. The information contained in this chapter
enables implementers of ECCs on constrained devices to make an informed choice of which algorithms to implement to achieve the best possible speed whilst conforming to code size and memory usage constraints. Such information is generic enough to apply to constrained devices in general, not just the Motorola smart card on which this implementation was performed.
Chapter 4

Countermeasures for Simple Power Analysis on a Smart Card

Side channel attacks on smart card implementations of cryptosystems have been proposed by Kocher et al. [KJJ99]. These attacks can obtain information about the cryptosystem by measuring side-channel information such as power consumption and processing time. This information can then be used to break implementations that have not incorporated defences against these attacks. An overview of various methods of attack is provided in this chapter, as well as a detailed summary of countermeasures available for simple power analysis and a description of a new countermeasure. This innovative countermeasure is based on the NAF (non-adjacent form) representation of a scalar and requires 44% fewer additions and 11% more doublings than the commonly recommended double and add always algorithm shown by Algorithm 4.1. The new method also requires at most two points in a precomputation.

Firstly, an overview of the possible side channel attacks on a smart card is presented, followed by a more detailed description of the various countermeasures currently available for resisting simple power analysis. The description and security analysis of the new countermeasure proposed in this chapter is then provided. Finally, an analysis of the speed and memory usage of the various countermeasures is performed. It is concluded that in terms of speed and memory usage, the new countermeasure compares quite favourably to other countermeasures which were available at the time that it was published. However, two countermeasures
which are slightly more efficient have been published since that time.

4.1 Overview of Side Channel Attacks on Smart Cards

There are several methods available for obtaining side channel information in order to find the secret key used in an ECC. This chapter deals only with simple power analysis (SPA), which is described below. However, several other methods of attack are also briefly summarized here for completeness.

4.1.1 Simple Power Analysis (SPA)

In a simple power analysis (SPA) attack, the power consumption of the device is measured for a single execution of a cryptographic operation [Cor99]. Because different operations performed by the device require differing amounts of power, the power consumption can be used to determine which operations were performed in what order. In the case of elliptic curves, it may be possible to determine which parts of the power trace were generated by a point doubling, and which parts were generated by a point addition. From this information, the secret key used as the scalar in the elliptic curve scalar multiplication can be recovered if the scalar multiplication algorithm was not protected against SPA.

A more powerful use of SPA has been proposed by Messerges, Dabbish and Sloan [MDS02], where the attacker can observe the Hamming weight of bytes manipulated by the smart card through measuring the height of the pulse in the power trace at the point where the byte is manipulated. By observing the Hamming weight of key bytes and shifted key bytes for DES, they were able to determine the DES key being used. Defences for SPA which have been proposed in the literature are summarized in Section 4.2.

4.1.2 Enhanced Simple Power Analysis

Oswald [Osw02] has proposed an enhanced method of SPA which uses Markov models to attack addition-subtraction chain scalar multiplication methods. This method appears to be based on a similar idea to that employed by Okeya and Sakurai [OS02a] to break the addition-subtraction scheme of Oswald and Aigner [OA01]. An appropriate Markov chain is used to decide which bits are
most likely to have caused particular patterns of additions and doublings. The results of the Markov modelling process are used to significantly reduce the number of possibilities for the secret scalar and to guess the most likely value of the scalar first. Although she does not provide details due to lack of space, Oswald states that the method can also be applied to randomized addition-subtraction chain models.

### 4.1.3 Differential Power Analysis (DPA)

Differential power analysis (DPA) was first proposed by Kocher et al. [KJJ99] and various methods of differential power analysis (DPA) applicable to ECCs have been proposed in the literature since that time. All of the methods use a statistical analysis of power signals from many scalar multiplications to find similarities and differences between the different power signals. These similarities and differences are due to either data [MDS99] or register address [IIT02] similarities and differences. These similarities or differences can then be used to reveal the secret scalar. Usually, when defences are recommended for DPA it is assumed that the scalar multiplication algorithm under consideration is already resistant to SPA, since SPA is a simpler attack. Therefore, it is important to have both an SPA and a DPA defence in an ECC implementation.

Second and higher order DPA [OS02b, Wal01] are more powerful (but also more difficult) forms of DPA. In this case, a statistical analysis is made which allows comparison of parts of power signals corresponding to different times with each other, rather than comparing different power signals from the same time with each other. However, this method requires the attacker to know the scalar multiplication algorithm in use and to be able to match different parts of the power signal with different parts of the algorithm. These extra requirements for the attacker mean that second order DPA is harder for an attacker to implement. However, second order DPA may succeed where ordinary (first order) DPA fails.

Various countermeasures for DPA have been published in the literature. Many are written for the defence of modular exponentiation, but such defences can generally be applied to EC scalar multiplication also, because the scalar of the scalar multiplication is used in a similar way to the exponent of an exponentiation. The countermeasures include:

- Randomization of the private scalar:
– Using an addition mask [Cor99, IIT02, MDS99]
– Using exponent splitting [CJ01, IIT02]
– Using exponent splitting with a multiplicative mask [TB02]

• Window Methods:
  – Overlapping window method [IYTT02]
  – Randomized table window method [IYTT02]
  – Hybrid overlapping and randomized table window method [IYTT02]

• Randomization of the data point:
  – Perturbation point [Cor99, MDS99]
  – Point of small order as a perturbation point [TB02]
  – Randomized projective or Jacobian coordinates [Cor99]
  – Randomized isomorphic curve or field [JT01]

• Scalar multiplication algorithm randomization
  – Using both binary methods [MDS99]
  – Non-deterministic right-to-left method with precomputation [TB02]

• Register addresses with the same Hamming weight [IIT02]

4.1.4 Electromagnetic Analysis

Electromagnetic analysis [GMO01] is quite similar to power analysis, but uses a different method of data collection. Instead of analysing power consumption, electromagnetic radiation is monitored. This involves the construction and positioning of an electromagnetic probe to collect the data. The data can then be analysed in the same manner as described for power analysis.

4.1.5 Other Attacks

Other side channel attacks on smart cards include fault and timing attacks. Fault attacks [BMM00] work by either feeding the smart card invalid input or else causing a fault during a computation by the smart card involving the secret
key. The output of the smart card in such circumstances can reveal information about the secret key. Timing attacks [Koc96] work when an implementation of an operation using the secret key does not run in constant time and the time taken is correlated with the value of the secret key. Execution times can then be used to predict the most likely value of the secret key and these guesses tested and either confirmed or rejected. It is important that countermeasures for these attacks be implemented in an ECC.

4.2 Overview of Existing SPA Countermeasures

This subsection provides a review of the existing methods of defending against SPA in the literature. An analysis of how these methods compare with the new method proposed later in this chapter is provided in Section 4.3.3.

4.2.1 Double and Add Always

In order to ensure the resistance of an implementation to SPA, a simple defence based on the binary scalar multiplication algorithm is often recommended [Cor99, OA01]. The binary scalar multiplication algorithm and the defended version are given by Algorithms 2.12 and 4.1. Instead of only performing a point addition when the next bit of the scalar \( k \) is 1, the defended algorithm performs an addition in each iteration of the loop and discards the results of those additions which are irrelevant. This prevents SPA because the pattern of additions and doublings is the same for all scalar multiplications and does not reveal the scalar used.

4.2.2 Montgomery Ladder

A Montgomery ladder can also be used to resist SPA because it performs a double and add in every loop, in a similar fashion to the double and add always countermeasure. However, the computations are slightly different because unlike the double and add always countermeasure, a Montgomery ladder does not use any dummy operations. The method was originally proposed for use with Montgomery curves, since it allowed a fast point addition algorithm which did not require the use of the \( y \)-coordinate on these curves [BJ02]. However, it can also be used as a method of resisting SPA attacks when ordinary addition and
Chapter 4. Countermeasures for Simple Power Analysis on a Smart Card

Input:  
- \( P \) (the point to multiply),
- \( k \) (the scalar) such that \( k = \sum_{i=0}^{n-1} k_{n-i-1}2^i \),
- \( n \) (length of \( k \))

Output:  
- \( Q[0] \) such that \( Q[0] = [k]P \)

Algorithm:  
- \( Q[0] = \phi \)
  - For \( i = 0 \) to \( (n - 1) \)
    - \( Q[0] = [2] (Q[0]) \)
    - \( Q[1] = P + Q[0] \)
    - \( Q[0] = Q[k_i] \)
  - Return \( Q[0] \)

**Algorithm 4.1:** Binary scalar multiplication algorithm defended against SPA (double and add always)

doubling algorithms are used on any curve. A version of the Montgomery ladder which was described in [TB02] is shown below:

- Let \( s = s_{u-1} \ldots s_1 s_0 \) be the secret scalar and suppose it is wished to find \([s]P\).
- Set \( P_1 = P \)
- \( P_0 = [2]P \)
- For \( i \) from \( u - 2 \) down to \( 0 \) do:
  - \( d = 1 - s_i \)
  - \( P_d = P_0 + P_1 \)
  - \( P_{s_i} = [2]P_{s_i} \)
- Return \( P_1 \)

A method for finding the addition and doubling required by Montgomery’s ladder without using the \( y \)-coordinates of the points for any elliptic curve has been proposed by Brier and Joye in [BJ02]. When using projective coordinates, the formulae proposed require only 7 multiplications plus 3 multiplications by a constant for addition and 7 multiplications plus 2 multiplications by a constant for doubling. Hence 14 multiplications and 5 multiplications by a constant are required for each loop in the Montgomery ladder algorithm. Actually, in order to use the proposed formulae, the difference of the two points to be added must
be known, so the method can only be used in conjunction with Montgomery’s ladder.

Fischer, Giraud, Knudsen and Seifert [FGKS02] provide an algorithm for computing \((P + Q)\) and \([2]Q\) in parallel (not using the \(y\)-coordinates of the points). When parallel processors are available, this method can be used with the Montgomery ladder to provide a significant speed increase compared to the ordinary Montgomery method. The method requires 19 multiplications and 14 additions, but when performed in parallel, it only requires the time of 10 multiplications and 8 additions.

### 4.2.3 Möller’s Windowing Method

Another method of defence against SPA proposed by Bodo Möller [Möl01] involves using a windowing method. The digit 0 is not allowed as a value for a window, and thus each iteration of the scalar multiplication algorithm requires a point addition. However, because of the amount of memory needed to store the precomputation that is required, windowing methods are not generally used in smart cards.

### 4.2.4 Universal Exponentiation Algorithm

Clavier and Joye [CJ01] have recommended a “universal exponentiation algorithm” to avoid SPA attacks. It is based on the assumption that one EC point addition is indistinguishable from another EC point addition using SPA. Given this assumption, an addition chain is used to compute \([s]P\) using a fixed addition chain. Since all additions are indistinguishable from one another, the algorithm is SPA resistant. However, if addition and doubling algorithms are fundamentally different, this method of scalar multiplication assumes that during use of the addition chain, no two identical points are added together (since this would be a doubling and distinguishable from an addition). The efficiency of the method is given by the length of the addition chain which is expected to be \(1.25 \log_2(s)\) on average (this is the same as the number of EC additions required). However, the number of points which must be stored whilst computing the addition chain is unspecified. Therefore, the exact memory usage is unknown, although Trichina and Bellezza state that it is large [TB02].
4.2.5 Using the Same Algorithm for EC Addition and Doubling

Using the one formula for EC addition and doubling has been recommended in the past, but usually the formula proposed is suitable only for either the Jacobi or the Hessian form of an elliptic curve [BJ02]. However, not all elliptic curves have a Jacobi or Hessian form, including most that are recommended for use in various standards. Quite recently, Brier and Joye [BJ02] have proposed a method for using the one set of formulae for both EC addition and doubling for any curve over $GF(p)$ without using the Montgomery ladder. Formulae for addition in both affine and projective coordinates are provided. The affine algorithm requires one field inversion and five field multiplications. For projective coordinates, the unified formulae require 17 multiplications and one multiplication by a constant. However, when $a = -1$, the number of multiplications is only 16.

4.2.6 Splitting EC Point Addition

Trichina and Bellezza [TB02] have proposed and implemented using a split EC point addition to defeat SPA for curves over $GF(2^n)$. The addition algorithm is split into two parts, both requiring about the same amount of computation as the double routine, and then dummy operations added where required (to the addition and doubling operations) to make the execution sequence of each half of the addition identical to that of a double. This countermeasure can then be used with any unprotected scalar multiplication algorithm and has a much lower performance cost than other methods such as those in Sections 4.2.1 and 4.2.2.

At the same conference, Gebotys and Gebotys [GG02] proposed and implemented the same idea for curves over $GF(p)$. They provide a comprehensive analysis of their method by mounting an actual SPA attack against their implementation. After analysing their algorithm using SPA, they modified their original algorithm in order to improve its resistance to SPA. A new measure, the implementation security index (ISI) was also defined in order to determine which parts of the algorithm were leaking information detectable with SPA, given a power trace of the algorithm.
4.2.7 Randomized Addition-Subtraction Chain

Oswald and Aigner [OA01] have proposed using a randomized addition-subtraction chain for the scalar multiplication algorithm to resist SPA. It was also stated that this algorithm would resist DPA attacks if doublings and additions were not able to be distinguished. However, this algorithm has been successfully attacked using power analysis and timing attacks by Okeya and Sakurai [OS02a]. Their attack was made possible because although randomization of the algorithm was present, only a limited number of patterns of addition and doubling were possible for each run of exponent bits. When enough actual scalar multiplications had been performed and the corresponding patterns of additions and doublings collected, the number of scalars able to produce all of the observed patterns can be reduced to only one. The attack method proposed by Oswald [Osw02] (see Section 4.1.2) appears to be based on a similar idea.

4.2.8 Width-$w$ NAF Method

Okeya and Takagi [OT03] have proposed a windowing method based on the NAF method to resist SPA. Because their method of construction of the windows ensures that all window values are odd, the size of the precomputation table which must be stored is smaller. The method is somewhat similar to Möller’s method, but requires less memory. In order to prevent second-order differential power attacks (See Section 4.1.3), they recommend randomizing an item in the table of precomputed points each time it is used in a scalar multiplication.

The algorithm for converting a scalar to the format suitable for the SPA resistant width-$w$ NAF method is described by Algorithm 4.2, and the corresponding scalar multiplication procedure is described by Algorithm 4.3. These algorithms describe the version that does not use dummy operations. The version that does use dummy operations is less efficient in terms of speed and also memory usage, therefore its description is not included here.

4.2.9 Non-deterministic Right-to-Left Method with Pre-computation

The non-deterministic right-to-left method with precomputation proposed by Trichina and Bellezza [TB02] is useful for defence against both SPA and DPA.
Chapter 4. Countermeasures for Simple Power Analysis on a Smart Card

Input:  
- \( s \) (the scalar)  
- \( m \) (length in bits of \( s \))  
- \( w \) (window size, where \( w \geq 2 \))

Output: \( f \) such that \( f \) is the SPA resistant width-\( w \) NAF.

Algorithm:  
Let \( \text{sign}(u) \) be 1 if \( u > 0 \) and -1 if \( u < 0 \) 
If \( s \) is even then \( s = s + 1 \) and \( f_{-1} = 1 \)  
Else \( s = s + 2 \) and \( f_{-1} = 2 \). End If  
\( v = \lceil m/w \rceil \)  
\( u_0 = s \pmod{2^w} \)  
\( s = (s - u_0)/2^w \)

For \( i = 1 \) to \( v \):
\[
\begin{align*}
  u_i &= s \pmod{2^w} \\
  \text{If } u_i &\text{ is even then} \\
  g &= \text{sign}(u_{i-1}) \\
  u_i &= u_i + g \\
  u_{i-1} &= u_{i-1} - g2^w \\
  f_{(i-1)w} &= u_{i-1} \\
  f_{(i-1)w+1} &= 0 \\
  \vdots \\
  f_{(i-1)w+w-1} &= 0 \\
  s &= s - u_i \\
  s &= s/2^w \\
  f_{vw} &= u_v \\
  f_{vw+1} &= 0 \\
  \vdots \\
  f_{vw+w-1} &= 0
\end{align*}
\]

Return \( f = f_m, f_{m-1}, \ldots, f_0, f_{-1} \)

Note: \( f_i \in \{0, \pm1, \pm3, \pm5, \ldots, \pm(2^w - 1)\} \) for \( i \geq 0, f_{-1} \in \{1, 2\} \).

Algorithm 4.2: Width-\( w \) NAF resistant to SPA formatting algorithm for scalars
Input: $P_1$ (the point to multiply), $P_3, P_5, \ldots, P_{2^{w-1}}$ (odd multiples of $P_1$: $[3]P_1, [5]P_1, \ldots, [2^w - 1]P_1$)

$f$ (output of Algorithm 4.2 when run with input $s, m$ and $w$)

$m$ (maximum index of $f$; i.e. $f = f_m, f_{m-1}, \ldots, f_0, f_{-1}$)

$w$ (window size)

Output: $Q$ such that $Q = [s]P$

Algorithm: $Q = P_f c$ for the largest $c$ such that $f_c \neq 0$

For $i = c - 1$ to 0:

\[
\begin{cases}
Q = [2]Q \\
\text{If } f_i > 0 \text{ then } Q = Q + P_f \\
\text{Else if } f_i < 0 \text{ then } Q = Q - P_{-f_i}
\end{cases}
\]

$Q = Q - [f_{-1}]P_1$

Return $Q$

**Algorithm 4.3:** Scalar multiplication algorithm using width-$w$ NAF

It requires no doublings (apart from those required to form the precomputation) and randomizes the required additions. It works by precomputing a table of points $[2^{u-1}]P, \ldots, [4]P, [2]P, [1]P$, where $u$ is the length in bits of the secret scalar $s$. The required scalar multiplication can then be found by simply adding values from the table. However, the very large amount of memory required for this defence renders it unsuitable for smart cards. The algorithm is given below:

- Let $s = s_{u-1} \ldots s_1 s_0$ be the secret scalar where it is wished to find $[s]P$.
- $Q = \phi$ (the point at infinity)
- $W = \text{Permute}([u - 1, \ldots, 0])$ ($W$ contains randomly permuted indices)
- For all elements $j$ from $W$ do:
  - Let $Q = Q + [s_{W_j}]A_{W_j}$
- Return $Q$

### 4.3 New SPA Defence and Efficiency Analysis

A new defence against SPA is now presented. This new defence requires only a small amount of memory, and is significantly faster than the double and add
always countermeasure. A full efficiency analysis is provided in Section 4.3.3.

4.3.1 Description of New Defence

The new defence against SPA given in this section is based on the NAF (non-adjacent form) [BSS99] representation of a scalar. The NAF is a binary signed-digit representation of the scalar, has a minimum number of non-zero digits and is at most one digit longer than the unsigned representation. It also has the property that no two adjacent digits are non-zero. An algorithm for converting a scalar to NAF format is provided in Section 2.2.3.

In order to defeat the SPA attack, it is necessary that the order in which additions, subtractions and doublings are performed does not reveal any useful information about the secret key. Here, we assume that additions are indistinguishable from subtractions using SPA. This could be ensured in practice by precomputing the negative of the point $P$, and adding this point wherever a subtraction of $P$ is required.

Because each non-zero digit of the NAF is followed by a zero, when using the signed binary algorithm, the pattern of additions and doublings is a repetition of the basic unit: (double, (optional doubles), double, (add or subtract)). We decided to insert dummy doubles and additions so that the pattern of operations carried out would always be a repetition of the basic unit: (double, double, (add or subtract)). Stated another way, we have changed the NAF (by inserting dummy additions and doublings) to have the format $0!0!0!0!...0!$ where $!$ may be either 1 or $-1$. Thus, the only information about the secret scalar that the attacker can obtain using SPA is the number of repetitions of the (double, double, (add or subtract)) unit that were performed. This knowledge of the length of the new format of the scalar can reveal some information about the scalar, and this issue is addressed in Section 4.3.2.

Algorithm 4.4 can be used to convert the scalar to the new format, and has been designed in such a way that the actual algorithm also resists SPA where possible. Note that such resistance for the conversion algorithm might not be necessary in practice. This would be the case if protection from SPA is required for scalar multiplications involving a long term private key whose new NAF representation can be computed in a secure environment at the same time as the long term private key is chosen. The new NAF version of the long term key must then be securely stored for direct use in scalar multiplication algorithms. On the
other hand, if protection from SPA is required for scalar multiplications involving a short term (i.e. single use) private key, such as those used in an ECDSA signature, then the algorithm to convert the scalar to the new NAF format must resist SPA. If the scalar is converted from an unsigned representation to NAF format beforehand, that operation must also resist SPA.

The output of Algorithm 4.4 is in groups of two bits, where (00) means perform two doublings and one dummy addition, (01) means perform two doublings and one dummy subtraction, (10) means perform one doubling and one dummy doubling and one dummy addition and (11) means perform two doublings and one subtraction. To begin, the algorithm checks whether there are any leading zeros in the scalar. If there are, it moves the counter on to the second digit of the NAF. Otherwise, the counter is set to the first digit of the NAF. The algorithm works in such a way that if the counter is on the $i^{th}$ digit of the NAF, then the $i - 1^{th}$ digit is a zero, and the $i^{th}$ and $i + 1^{th}$ digits are used to determine how to encode the $i - 1^{th}$ and $i^{th}$ digits. Once the counter has been set, the algorithm reads the next two bits of the scalar and outputs a value according to the lookup table. Finally, the algorithm moves on either one or two bits along the scalar and returns to the beginning of the loop. The algorithm uses a lookup table and outputs a value after reading only one or two bits of the scalar in order to be resistant to SPA. However, the algorithm’s output is best explained by behaviour according to the number of zeros in the next run of zeros. If this is the last run of zeros, and the last bit of the NAF is a zero, then an odd run of zeros is encoded as $(00)^{(n-1)/2}(10)$ and an even run of zeros is encoded as $(00)^{n/2}$, where $n$ is the number of zeros in the run. Otherwise, this is not the last run of zeros or the last digit of the NAF is non-zero, and an odd run of zeros followed by a non-zero digit is encoded as $(00)^{(n-1)/2}$ followed by either (01) or (11) (depending on whether an add or subtract is required), and an even run of zeros followed by a non-zero digit is encoded as $(00)^{(n/2)-1}(10)$ followed by either (01) or (11). This process is then repeated until all of the bits in the NAF have been processed, and is equivalent to Algorithm 4.4. The time required to perform the conversion to an SPA resistant scalar is negligible compared to the time required to complete the entire scalar multiplication algorithm. Algorithm 4.5 shows a variant of the left-to-right binary scalar multiplication algorithm that uses the new protected NAF format.
Chapter 4. Countermeasures for Simple Power Analysis on a Smart Card

Input: \( s \) (the scalar in NAF) such that \( s = \sum_{i=0}^{m-1} s_{m-i-1} 2^i \),
m (length of \( s \))

Output: \( k \) and \( n \) such that \( k \) is the protected NAF representation of \( s \) and has \( n \) bits in its representation

Algorithm:
\[ n = 0 \]
If \( s_0 == 0 \) Then
\[ \{ \text{ } i = 1 \} \]
Else
\[ \{ \text{ } i = 0 \} \]
End If

\[ \text{array} = \begin{bmatrix} U & 11 & U & 11 \\ 10 & 00 & 10 & 00 \\ U & 01 & U & 01 \\ 10 & 10 & 10 & 10 \end{bmatrix} \] (values specified in binary and \( U \) indicates the value is unused)

While \( (i - 1) < m \)
\[ \begin{cases} \text{If} \ (i \geq m) \\ \{ \text{digit}2 = 3 \} \end{cases} \]
Else
\[ \begin{cases} \text{digit}2 = s_i + 1 \\ \text{If} \ ((i + 1) \geq m) \\ \{ \text{digit}3 = 3 \} \end{cases} \]
Else
\[ \begin{cases} \text{digit}3 = s_{i+1} + 1 \\ k_i k_{i+1} = \text{array}_{\text{digit}2 \text{ digit}3} \\ n = n + 2 \\ i = i + (\text{array}_{\text{digit}2 \text{ digit}3} \neq 2) + 1 \end{cases} \]
(Where \( true = 1, false = 0 \))

Return \( k \) and \( n \)

Algorithm 4.4: Protected NAF formatting algorithm resistant to SPA
4.3. New SPA Defence and Efficiency Analysis

Algorithm 4.5: New binary scalar multiplication algorithm for protected NAF scalar

4.3.2 Stopping Information Leakage

As observed in Section 4.3.1, the only information about the secret scalar that the attacker can obtain using SPA is the number of repetitions of the (double, double, (add or subtract)) unit that were performed. This knowledge of the length of the new format of the scalar can reveal some information about the scalar. Indeed, by obtaining the length of the new representation, the attacker knows approximately how many runs of zeros of an even length there were in the NAF format of the scalar. This is the case because in general, two NAF digits become one symbol in the protected NAF representation, but each run of an even number of zeros requires an extra half of a protected NAF symbol to be used. The figure is not exact because there may have been no leading zero (which has the effect of using an extra half of a symbol), or the scalar may have ended in zero, in which case an even number of zeros at the end requires the usual one symbol per two NAF digits, but an odd number of zeros at the end requires an extra half of a symbol. Also, if the length of the new format is at least \( \lceil \frac{N+1}{2} \rceil \) symbols (where \( N \) is the original number of bits in the scalar), the attacker does not know whether the original NAF had a length of \( N \) or \( N + 1 \) bits. It can be seen that the attacker may gain some valuable information. We show below how this information may be non-trivial, and also propose a scheme for rendering insignificant such information leakage.
In order to stop this leakage of information, there are several options available. One is to add a new symbol to perform two dummy additions and a dummy add (or subtract), and to insert this symbol at random places throughout the protected NAF representation in order to pad the length out to the maximum of \(2\left\lceil \frac{N+1}{3} \right\rceil + (N + 1 \mod 3)\) symbols. The major problem with this option is that padding the length out to the maximum would have an adverse affect on performance. However, other problems also exist. Three instead of two bits would be required to represent each symbol. In order to avoid introducing an extra symbol, it would be possible to insert extra \((00)\) and \((10)\) symbols at the beginning of the protected NAF format to pad out the length. However, for this to be effective, additions and doublings involving the point at infinity must be indistinguishable from other additions and doublings. Because of the increase in time required for the scalar multiplication, these options are not considered further.

If we start with an \(N\) bit scalar, the minimum length of the protected NAF is \(\delta = \left\lfloor \frac{N}{2} \right\rfloor\) symbols (this is attained when all runs of zeros contain an odd number of zeros, for example the NAF \(0101\ldots0101\)), and the maximum length is \(\epsilon = 2\left\lceil \frac{N+1}{3} \right\rceil + (N + 1 \mod 3)\) symbols (this is attained by having as many runs of an even number of zeros as possible, for example the NAF \(100\ldots1001001000\ldots10010010\)). Thus there are \(\alpha = \epsilon - \delta + 1\) different protected NAF lengths. For example, for \(N = 160\), there would be 29 symbol lengths. The distribution of the \(2^N\) scalars amongst these categories is not uniform. For example, the protected NAF of the maximum length \(\epsilon\) corresponds to only of order \(2^N\) scalars. Hence, if an attacker measures the length of the protected NAF using SPA and finds it to be of such a length, then, given that there is an obvious algorithm for enumerating the scalars which correspond to a given protected NAF length, then the attacker’s task is restricted to a simple brute force search over this comparatively small set. This demonstrates that the information leakage in the new algorithm is potentially significant. However, the above analysis also provides for a solution. Our proposal is to restrict the allowed scalars to those which correspond to a particular protected NAF length, and further to choose this particular length to be that which corresponds to the maximal number of scalars. Clearly, this maximum is at least \(\gamma = \frac{2}{\alpha}\), where \(\alpha = 2^N\). For \(N = 160\), \(\gamma = \frac{2^{160}}{29}\).

The above restriction on the protected NAF length means that not all scalars can be protected using the new NAF format. Therefore, if the method is to be
used, care must be taken when choosing private keys to ensure that they have the desired length. One method of ensuring this condition is met is to randomly choose scalars until a scalar having the desired protected NAF length is found. In practice, it is not expected that this will form a significant problem, since a large number of scalars can be protected (for example, the analysis below shows that for 160 bit scalars, a length can be chosen that corresponds to 22% of all scalars). If the size of the scalar is increased slightly (by three bits in the 160 bit scalar case), any loss of security due to the length restriction can be overcome. Of course, any conversion of the scalar to the protected NAF format in order to ensure this condition is met must either be performed in a secure environment or be resistant to SPA, as noted in Section 4.3.1.

There remains the question of what particular length must be chosen in order to correspond to the maximum proportion of scalars. Clearly, for any given bit length, this is a straightforward computation. Below, we give a computationally efficient way of computing these proportions. Elementary probabilistic arguments suggest though that in general the average length of the protected NAF is of order $\frac{5N}{9}$ symbols (see Section 4.3.3), while this can also be argued to correspond to the maximum proportion of scalars. The computations below for $N = 160$ back up such arguments.

Before analysing the protected NAF, some results regarding the NAF form of a scalar are proven. Arno and Wheeler [AW93] have provided an analysis of signed digit representations, including the NAF. They have analysed the likelihood of the occurrence of the various possible digits in the NAF (and other signed digit representations) using a Markov chain. Their analysis has four states: two zero states and two non-zero states. However, it is possible to combine the states and deduce that the probability of moving from a zero state to a non-zero state is $\frac{1}{2}$, and the probability of moving from a non-zero state to a zero state is 1. These results can be used to prove the following theorems.

**Theorem 4.1.** Let $X$ be a random variable denoting the length of a run of zeros in the NAF. Then the expected length of a run of zeros, $E(X)$, is equal to two.
Chapter 4. Countermeasures for Simple Power Analysis on a Smart Card

Proof.

\[
E(X) = \sum_{i=1}^{\infty} i \cdot \Pr(X = i)
\]

\[
= \sum_{i=1}^{\infty} i \cdot \frac{1}{2^i}
\]

\[
\Rightarrow E(X) - \frac{1}{2} E(X) = \sum_{i=1}^{\infty} \frac{1}{2^i}
\]

\[
= 1 \quad \text{(using formula for sum of a geometric progression)}
\]

\[
\Rightarrow E(X) = 2
\]

\[\square\]

Theorem 4.2. Let \(X\) be a random variable denoting the length of a run of zeros in the NAF. Let \(Y\) be a random variable which is 1 if \(X\) is even, and 0 if \(X\) is odd. Then \(E(Y) = \frac{1}{3}\).

Proof.

\[
\Pr(Y = 1) = \Pr(X \text{ is even})
\]

\[
= \sum_{i=1}^{\infty} \frac{1}{2^{2i}}
\]

\[
E(Y) = \sum_{i=0}^{\infty} i \cdot \Pr(Y = i)
\]

\[
= \sum_{i=1}^{\infty} \frac{1}{2^{2i}}
\]

\[
= \frac{1}{3} \quad \text{(using formula for sum of a geometric progression)}
\]

\[\square\]

It is also possible to use a Markov chain model to find the probability of each possible number of symbols in the protected NAF for a given value of \(N\). The initial state of the model is that no NAF digits have been chosen. The model takes into account that since we start with a 160 bit scalar, the first non-zero digit of the NAF must be positive, and the NAF may have 160 or 161 bits. The states used are given in Table 4.1.
4.3. New SPA Defence and Efficiency Analysis

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Itl</td>
<td>Initial state, no digits have been chosen.</td>
</tr>
<tr>
<td>0_L</td>
<td>Only zeros have been chosen so far, and there is an even number of them.</td>
</tr>
<tr>
<td>1_L</td>
<td>Only zeros have been chosen so far, and there is an odd number of them.</td>
</tr>
<tr>
<td>i_e</td>
<td>The last digit chosen was 0, the ending run of zeros has an even number of zeros in it and ( i ) extra half-symbols have been added to the NAF (i.e. a NAF of length ( k ) has exactly ( \frac{k+i}{2} ) symbols in it).</td>
</tr>
<tr>
<td>i_o</td>
<td>The last digit chosen was 0, the ending run of zeros has an odd number of zeros in it and ( i ) extra half-symbols have been added to the NAF.</td>
</tr>
<tr>
<td>i_1</td>
<td>The last digit chosen was 1 and ( i ) extra half-symbols have been added to the NAF.</td>
</tr>
</tbody>
</table>

Note: \( i \) takes values from 0 to 2

We can then find the transition matrix, \( \bar{A} \). Let \( a_{ji} \) be the probability of moving from state \( i \) to state \( j \). The transition probabilities from the Markov chain model of the NAF that were given above are used in the construction of our transition matrix. Let the first digit chosen be the first bit of an \( N \) bit NAF or the second bit of an \( N + 1 \) bit NAF. Note that for an \( N \) bit scalar to become an \( N + 1 \) bit NAF, the first digit of the NAF must be 1, and the next non-zero digit must be -1. Now the first digit chosen can be 1 or 0, but not -1, since two non-zero digits must be separated with at least one zero, and the scalar we are forming must be positive. The following theorem allows us to deduce the probabilities of the first digit:

**Theorem 4.3.** Let the NAF of an \( N \) bit scalar \( t \) (where \( t \) may or may not have leading zeros) be \( s \). The probability that \( s \) has \( N \) bits and the first bit is 1 is \( \frac{1}{3} \).

**Proof.** Observe that Algorithm 2.13 implies that if the NAF corresponding \( t \) has \( N + 1 \) digits, then \( 3t \geq 2^{N+1} \). Also note that a NAF increases the length of a scalar by at most one bit [BSS99]. Therefore, scalars with \( N \) bit NAFs (with no leading zeros in the NAF) are those scalars \( t \) such that either:

- \( t \geq 2^{N-1} \) and \( 3t < 2^{N+1} \), or
- \( t < 2^{N-1} \) and \( 3t \geq 2^{N} \).
The total number of such scalars is therefore:

\[
\frac{2^{N+1}}{3} - \frac{2^N}{3} = \frac{2^N}{3} \cdot (2 - 1) = \frac{2^N}{3}.
\]

The probability of an \( N \) bit scalar (which may have leading zeros) falling into this category is:

\[
\frac{\left(\frac{2^N}{3}\right)}{2^N} = \frac{1}{3}.
\]

Theorem 4.3 implies that the probabilities for the first digit chosen in the Markov chain model for the protected \( \text{NAF} \) are 2/3 for choosing a 0 and 1/3 for choosing a 1. If a 1 is chosen, the \( \text{NAF} \) must have \( N \) digits. Otherwise, a 0 has been chosen, and the length of the \( \text{NAF} \) depends on the first non-zero digit. If it is a 1, the \( \text{NAF} \) is \( N \) bits with leading zeros. Otherwise, it must be an \( N + 1 \) bit \( \text{NAF} \), and the first digit (a one) must be inserted at the beginning of the \( \text{NAF} \). The transition matrix is shown in Figure 4.1. To complete the matrix, the same pattern should be followed. Note that the sum of entries in each column is 1, with the exception of the impossible state, \( 0_o \).

Once the transition matrix, \( \tilde{A} \) has been found, the probability of being in a certain category in the final state can be found by \( \tilde{A}^N \tilde{b} \), where \( \tilde{b} = [1 \ 0 \ 0 \ 0 \ \ldots ]^T \). For each \( i \), we then add the probabilities for \( i_o \), \( i_0 \) and \( i_1 \) to find the total probability of adding \( i \) half-symbols to the \( \text{NAF} \). For a 160 bit scalar, the maximum probability is 0.22718, when adding 9 symbols (18 half-symbols), or a total length of 89 symbols. The probabilities of all possible categories are given in Table 4.2.

A simulation was performed for \( N = 160 \) which supports these results. In the simulation, 1,000,000 scalars were generated using a pseudo-random number generator. The protected \( \text{NAF} \) length corresponding to the largest number of scalars was a length of 89 symbols, with 21.8% of scalars having this protected \( \text{NAF} \) length.

It should be noted that the above analysis assumes that SPA will not reveal how many leading zeros a scalar may have had. This in turn requires that additions and doublings involving the point at infinity are indistinguishable from other additions and doublings. If this is not the case, the protected \( \text{NAF} \) algorithm must be modified to skip leading zeros, which will slightly change the analysis.
4.3. New SPA Defence and Efficiency Analysis

Table 4.2: Probability of each protected NAF length for a 160 bit scalar

<table>
<thead>
<tr>
<th>Protected NAF length</th>
<th>Probability</th>
<th>Protected NAF length</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 4.49 * 10^{-11}</td>
<td>95 0.00141</td>
<td>81 2.32 * 10^{-8}</td>
<td>96 0.000187</td>
</tr>
<tr>
<td>82 1.82 * 10^{-6}</td>
<td>97 1.75 * 10^{-5}</td>
<td>83 5.19 * 10^{-5}</td>
<td>98 1.14 * 10^{-6}</td>
</tr>
<tr>
<td>84 0.000715</td>
<td>99 5.01 * 10^{-8}</td>
<td>85 0.00552</td>
<td>100 1.46 * 10^{-9}</td>
</tr>
<tr>
<td>86 0.0260</td>
<td>101 2.72 * 10^{-11}</td>
<td>87 0.0791</td>
<td>102 3.09 * 10^{-13}</td>
</tr>
<tr>
<td>88 0.161</td>
<td>103 2.00 * 10^{-15}</td>
<td>89 0.227</td>
<td>104 6.75 * 10^{-18}</td>
</tr>
<tr>
<td>90 0.225</td>
<td>105 1.03 * 10^{-20}</td>
<td>91 0.158</td>
<td>106 5.68 * 10^{-24}</td>
</tr>
<tr>
<td>92 0.0796</td>
<td>107 7.02 * 10^{-28}</td>
<td>93 0.0289</td>
<td>108 4.11 * 10^{-33}</td>
</tr>
<tr>
<td>94 0.00752</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
However, simulations indicate that the category containing the most scalars will still have about $\frac{5N}{9}$ symbols.

### 4.3.3 Efficiency Analysis

In this section we compare the efficiency of the new algorithm that we have proposed with the efficiency of other algorithms that exist. Firstly, we compare our algorithm with the unprotected NAF format of the scalar, and then compare it with Algorithms 2.12 and 4.1. A comparison with the other methods of defence in Section 4.2 is also provided, with the exception of the randomized addition-subtraction chain method since it is insecure. This analysis assumes that the method is used as presented in Section 4.3.1. That is, none of the restrictions proposed in Section 4.3.2 are used. However, if the fixed length of $5N/9$ symbols is used as suggested in Section 4.3.2, the results for the average will apply to this case. It should be noted that in this discussion, “boundary” effects are ignored (i.e. the effect of ending the NAF with a zero, or starting with a 1), since these conditions will not have much impact on the results for sufficiently large scalars. Throughout the analysis, $N$ refers to the number of bits in the original unsigned scalar.

**Theorem 4.4.** The expected length of the protected NAF corresponding to an $N$ bit scalar is $\frac{5N}{9}$ symbols.

*Proof.* Theorem 4.1 states that the expected length of a run of zeros in the NAF of a scalar, $E(X)$, is 2. This implies that $\frac{1}{3}$ of the digits of a NAF are non-zero. For the protected NAF, an extra half of a symbol (i.e. an extra double and subtract) is added for every run of zeros of even length. Theorem 4.2 shows that the expected number of half symbols that will be added to any run of zeros of even length, $E(Y)$, is $\frac{1}{3}$. Therefore, the expected number of doubles (real or dummy) in the protected NAF between a (double, add) or (double, subtract) pair (where the add or subtract is not a dummy operation) is $E(X) + E(Y) = 2\frac{1}{3}$. Thus, the number of symbols per non-zero NAF digit is expected to be $(1 + 2\frac{1}{3})/2$ and the total number of symbols per scalar is expected to be $\left(\frac{(1 + 2\frac{1}{3})}{2}\right)N/3 = \frac{10N}{9} = \frac{5N}{9}$ symbols.

**Theorem 4.5.** The smallest increase the number of point additions and doublings required when using a protected NAF compared to a NAF is an increase of 0 operations.
Proof. Dummy operations are only added to the protected NAF when the NAF has even runs of zeros. Since scalars exist with no even runs of zeros, it is possible for the protected NAF to require no extra operations compared to the NAF. □

Theorem 4.6. The greatest increase in the number of point additions and doublings required by a protected NAF compared to a NAF is an increase of 1.5 times the number of operations.

Proof. Let $l_i$ for $1 \leq i \leq r$ be the length of a non-zero digit followed by the $i$th run of an even number of zeros, and let $m_j$ for $1 \leq j \leq s$ be the length of a non-zero digit followed by the $j$th run of an odd number of zeros in the original NAF. Then the original NAF requires $(r + s)$ additions and $N$ doublings, where $\sum_{i=1}^{r} l_i + \sum_{j=1}^{s} m_i = N$, and the protected NAF requires $(N + r)$ doublings and $\frac{N + 2r}{2}$ additions. If we assume that the addition time is equal to the doubling time then the number of operations for the original NAF is $(r + s + N)$ and the number of operations for the protected NAF is $\frac{3}{2}(N + r)$. To find the upper bound of the ratio (protected NAF : unprotected NAF), we set $s = 0$ and find that the upper bound is 1.5. □

Theorem 4.7. The expected increase in the number of point additions and doublings required by a protected NAF compared to a NAF is an increase of 1.25 times the number of operations.

Proof. The expected length of a NAF is $\frac{5N}{9}$ symbols (from Theorem 4.4). Since the number of additions and doublings is three per symbol, it is expected that there will be a total of $\frac{5N}{3}$ operations for a protected NAF. For an unprotected NAF, it is expected that there will be $\frac{4N}{3}$ operations. This is because the length of a run of zeros is expected to be 2 (from Theorem 4.1), four operations (three doublings and one addition) are expected to be required per run of zeros, and on average there are $N/3$ runs of zeros in the NAF. Given the above numbers of operations expected to be required by the NAF and protected NAF, the required increase can then be found to be $\left(\frac{5N}{3}\right) / \left(\frac{4N}{3}\right) = \frac{5}{4} = 1.25$. □

Theorem 4.8. The smallest increase in the number of point additions and doublings required when using the double and add always binary algorithm compared to the unprotected unsigned binary algorithm is an increase of 0 operations.

Proof. If the scalar contains no zeros, the unsigned binary algorithm requires a doubling and addition for each bit of the scalar. These operations are the same as those required by the double and add always binary algorithm. □
Theorem 4.9. The greatest increase in the number of point additions and doublings required by the double and add always binary algorithm compared to the unprotected unsigned binary algorithm is an increase of 2 times the number of operations.

Proof. The greatest increase in operations is obtained when the scalar contains only one non-zero digit. In this case, the double and add always algorithm requires $2N$ operations and the unsigned binary algorithm requires $N + 1$ operations and the value of the required increase is $\frac{2N}{N+1} \approx 2$. □

Theorem 4.10. The expected increase in the number of point additions and doublings required by the double and add always binary algorithm compared to the unprotected unsigned binary algorithm is an increase of $1\frac{1}{3}$ times the number of operations.

Proof. The expected number of operations required by the unsigned binary algorithm is $\frac{3N}{2}$ (since half the digits are expected to be non-zero), and the double and add always algorithm always requires $2N$ operations. The required increase is therefore $2N/\frac{3N}{2} = \frac{4}{3} = 1\frac{1}{3}$. □

We can also examine the ratio (double and add always algorithm : unprotected NAF) to see how much efficiency would be lost if Algorithm 4.1 was used instead of the unprotected NAF.

Theorem 4.11. The smallest increase in the number of point additions and doublings required by the double and add always binary algorithm compared to the unprotected NAF is an increase of $1\frac{1}{3}$ times the number of operations.

Proof. For the increase under consideration to be as small as possible, the number of operations required by the NAF must be maximal. This implies that as many bits as possible in the NAF must be non-zero. This maximum number of bits is $\frac{N}{2}$, and thus the maximum number of operations required by the NAF is $1.5N$ operations. The required ratio is therefore $\frac{2N}{1.5N} = 1.333$. □

Theorem 4.12. The greatest increase in the number of point additions and doublings required by the double and add always binary algorithm compared to the unprotected NAF is an increase of 2 times the number of operations.

Proof. For the increase under consideration to be as small as possible, the number of operations required by the NAF must be minimal. This implies that as few bits
Table 4.3: Minimum, expected and maximum values for the ratio of the total number of point additions and doublings required by one algorithm to that of another for various algorithms

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Min.</th>
<th>Expected</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected naf : unprotected naf</td>
<td>1</td>
<td>1.25</td>
<td>1.5</td>
</tr>
<tr>
<td>double &amp; add always : unprotected</td>
<td>1</td>
<td>1.333</td>
<td>2</td>
</tr>
<tr>
<td>unsigned binary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>double &amp; add always : unprotected</td>
<td>1.333</td>
<td>1.5</td>
<td>2</td>
</tr>
</tbody>
</table>

as possible in the NAF must be non-zero. This minimal number of bits is 1, and thus the minimum number of operations required by the NAF is $N + 1$ operations. The required ratio is therefore $\frac{2N}{N+1} \approx 2$.

**Theorem 4.13.** The expected increase in the number of point additions and doublings required by the double and add always binary algorithm compared to the unprotected NAF is an increase of 1.5 times the number of operations.

**Proof.** The expected number of operations required by the NAF is $\frac{4N}{3}$ (since $\frac{1}{3}$ of the digits are expected to be non-zero), and the double and add always algorithm always requires $2N$ operations. The required increase is therefore $2N/\frac{4N}{3} = \frac{3}{2} = 1.5$.

Table 4.3 provides a summary of the above ratios.

If the bounds for the ratio (double and add always : unprotected NAF) of 1.333 and 2 are compared with the bounds on the (protected NAF : unprotected NAF) of 1 and 1.5, it can be seen that the protected NAF has a much smaller efficiency impact than the double and add always algorithm. Also, the upper bound of the (protected NAF : unprotected NAF) ratio is significantly smaller than the upper bound of the (double and add always algorithm : unprotected unsigned binary algorithm) ratio, indicating that the extra cost of a protected NAF algorithm compared to the original NAF algorithm is much less than the cost of a double and add always algorithm compared to the original unsigned binary algorithm.

Table 4.4 provides a comparison of the new method with other countermeasures from Section 4.2, including the expected number of additions and doublings and the number of points required for each of the algorithms. It shows that
the universal exponentiation method (Section 4.2.4) and the non-deterministic right-to-left method (Section 4.2.9) both require prohibitively large amounts of memory for a smart card. On the other hand, using the same formula for point addition and doubling (Section 4.2.5) has a significant performance impact and is slower than the new countermeasure proposed here. Using a Montgomery ladder (Section 4.2.2) without any special addition or doubling algorithm is also slower. However, using the algorithms proposed in [BJ02] can make the speed of this method only slightly slower than the protected NAF. Although the Montgomery ladder with the algorithm from [FGKS02] is faster than the protected NAF, this method is not suitable in cases where a parallel processor is not available. The countermeasure which splits point addition into two parts, each of which is indistinguishable from a doubling using SPA (Section 4.2.6) was published soon after the new protected NAF. It is also faster and requires less memory. However, great care must be exercised when using the split addition countermeasure, since the difference in addition and doubling algorithms may cause slight discrepancies in the power trace (for example, conflicts when accessing memory may be different between the routines and this discrepancy may be detectable [GG02]).

If the new protected algorithm is compared with Möller’s algorithm (Section 4.2.3), the new algorithm is somewhat slower because Möller’s algorithm uses a windowing method to reduce the number of additions required. How much slower it is depends on the size of window used. If the smallest possible size of window is used, Table 4.4 shows that Möller’s (protected) method would have the same speed as the unprotected unsigned binary method. However, the new method proposed has an advantage when used on devices with limited memory because it requires enough memory for at most four points (one or two points in a precomputation and two points to store outputs from the algorithm), whereas Möller’s algorithm requires enough memory for a minimum of five points (at least four points in the precomputation and one point to store the output of the algorithm).

The width-\textit{w} NAF method [OT03] (see Section 4.2.8) was published after the new protected NAF method proposed here and is based on ideas from both Möller’s algorithm and our NAF method. If the version requiring dummy operations is used with the smallest possible window ($w = 2$), then although the method is slightly faster, it requires more memory (four or five points depending on whether the negative of a point as well as the point is stored in the precomputation). On
Table 4.4: Comparison of expected number of additions and doublings

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Adds</th>
<th>Doubles</th>
<th>Time as number of Adds, Assuming(^{a}):</th>
<th>Minimum Points in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>D = A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D = 0.7 A</td>
<td></td>
</tr>
<tr>
<td>One add/dbl. formula and NAF  (\frac{1}{3}N)</td>
<td>(N)</td>
<td>(N)</td>
<td>1.85(N)(^{b}) 1.85(N)</td>
<td>2 to 3</td>
</tr>
<tr>
<td>Universal Exponentiation</td>
<td>1.25(N)</td>
<td>0</td>
<td>1.25(N)(^{c}) 1.25(N)</td>
<td>Large</td>
</tr>
<tr>
<td>Double and add always</td>
<td>(N)</td>
<td>(N)</td>
<td>2(N) 1.7(N)</td>
<td>3</td>
</tr>
<tr>
<td>Montgomery ladder</td>
<td>(N)</td>
<td>(N)</td>
<td>2(N) 1.7(N)</td>
<td>2</td>
</tr>
<tr>
<td>Montgomery ladder, [BJ02]</td>
<td>(N)</td>
<td>(N)</td>
<td>1.54(N)(^{d}) 1.46(N)</td>
<td>2</td>
</tr>
<tr>
<td>Montgomery ladder, [FGKS02]</td>
<td>(N)</td>
<td>(N)</td>
<td>0.77(N)(^{f}) 0.77(N)</td>
<td>2</td>
</tr>
<tr>
<td>Protected NAF</td>
<td>(\frac{5}{6}N)</td>
<td>(\frac{10}{9}N)</td>
<td>1.667(N) 1.333(N)</td>
<td>3 to 4</td>
</tr>
<tr>
<td>Unprotected unsigned binary</td>
<td>(\frac{1}{2}N)</td>
<td>(N)</td>
<td>1.5(N) 1.2(N)</td>
<td>2</td>
</tr>
<tr>
<td>Non-deterministic R-to-L</td>
<td>(\frac{1}{2}N)</td>
<td>(N)</td>
<td>1.5(N) 1.2(N)</td>
<td>(N + 1)</td>
</tr>
<tr>
<td>Möller (min. window)</td>
<td>(\frac{1}{2}N)</td>
<td>(N)</td>
<td>1.5(N) 1.2(N)</td>
<td>5</td>
</tr>
<tr>
<td>Width-(w = 2) NAF, no dummy</td>
<td>(\frac{1}{2}N)</td>
<td>(N)</td>
<td>1.5(N) 1.2(N)</td>
<td>3 to 5</td>
</tr>
<tr>
<td>Width-(w = 2) NAF, dummy</td>
<td>(\frac{1}{2}N)</td>
<td>(N)</td>
<td>1.5(N) 1.2(N)</td>
<td>4 to 5</td>
</tr>
<tr>
<td>Splitting point addn. with NAF</td>
<td>(\frac{1}{3}N)</td>
<td>(N)</td>
<td>- 1.15(N)(^{g})</td>
<td>2 to 3</td>
</tr>
<tr>
<td>Unprotected NAF</td>
<td>(\frac{1}{3}N)</td>
<td>(N)</td>
<td>1.333(N) 1.033(N)</td>
<td>2 to 3</td>
</tr>
</tbody>
</table>

\(^{a}\)Times are indicative only since requirements for different methods may preclude the use of some point coordinate systems, thus causing a significant degree of variance in addition and doubling times applicable to each algorithm.

\(^{b}\)Assuming that an add (or double) takes 18/13 of the time of a normal addition.

\(^{c}\)This figure may actually be much larger in relation to the other algorithms (i.e. similar to the value for using the one add/double formula) since it is necessary to either never double or else use the one formula for addition and doubling.

\(^{d}\)Assuming that a [BJ02] addition takes the time of 10/13 of a normal addition.

\(^{e}\)Assuming that a [BJ02] addition and doubling take 19/13 as long as an ordinary addition.

\(^{f}\)Assuming that a parallel processor is available and parallel addition and doubling takes 10/13 of the time of a normal addition.

\(^{g}\)Assuming addition takes 18/13 and doubling takes 9/13 of the time of a normal addition.
the other hand, the version which does not require dummy operations is slightly faster than the new NAF proposed here (requiring only $\frac{1}{2}N$ instead of $\frac{5}{9}N$ point additions and $N$ instead of $\frac{10}{9}N$ point doublings) and requires the same amount of memory if addition and subtraction can not be distinguished using SPA (so that the negatives of points are therefore not stored). If addition and subtraction can be distinguished, this method must store more points than the new protected NAF. The algorithm for conversion of the scalar to the format required by the relevant scalar multiplication algorithm would also be slightly faster in the case of the width-$w$ NAF since the algorithm iterates a fewer number of times ($N/2$ times instead of an average of $\frac{5}{9}N$ times) and each loop would probably be faster (since only one if statement is required instead of two and the time for other calculations would be similar). However, the time required by this formatting algorithm will be very small in comparison to the overall scalar multiplication time.

### 4.4 Conclusion

We have presented a new algorithm to convert a scalar to a signed digit representation that is resistant to SPA. On average, the new method takes about 80% of the time of the existing defence of adding in every loop of the binary algorithm, and takes 25% more time than the signed binary algorithm using the original NAF. The ratio of the time taken by the new algorithm to the time taken by the unprotected unsigned binary algorithm using a NAF scalar (if we assume that additions and doublings take the same amount of time) is bounded by the values 1 and 1.5. Also, the extra cost of a protected NAF algorithm compared to the original NAF algorithm is much less than the cost of a protected unsigned binary algorithm compared to the original unsigned binary algorithm.

As stated at the start of Section 4.3.3, if the scalars allowed are restricted to those such that the protected NAF is of length $5N/9$ symbols (appropriately rounded to the nearest bit length), then the average performance figures quoted above are in fact always achieved. However, it is up to a specific implementor to decide on what restrictions to place on the allowed scalars. As we have shown, for a given bit length, it is easy to compute the proportion of scalars which lead to a given protected NAF length, along the lines we have indicated above using the analysis of the Markov process, and so the implementor can choose to trade
off loss of available scalars against information leakage to SPA. We have shown that the extreme case of restricting the scalars to a fixed protected NAF length leads to zero information leakage at the expense of a loss of a fraction of at most $1 - \frac{1}{a}$ of available scalars.

The new algorithm compares favourably with other previously published countermeasures in terms of either speed or memory usage. However, two methods (which were published after the protected NAF method) provide a slightly faster implementation with the same or a slightly smaller amount of memory usage.
Chapter 5

The Security of Fixed versus Random Elliptic Curves

The cryptographic security of fixed versus random elliptic curves over the field $GF(p)$ is examined in this chapter. The underlying assumption of the analysis is that a large precomputation to aid in the breaking of the elliptic curve discrete logarithm problem (ECDLP) can be made for a fixed curve. However, in the case of a random curve, it is likely that a much smaller amount of computing power is available. Given this assumption, it is intuitively obvious that fixed curves are less secure than random curves, but quantifying the loss of security is much more difficult. On the other hand, implementations using fixed curves can have many advantages over those using random curves, such as using less bandwidth, code size and processing time. Since fixed curves are so attractive from an implementation point of view and have been included in various standards, their security compared to that of random curves is examined here in detail.

The discussion is restricted to curves over the field $GF(p)$ where $p$ is a large prime. Firstly an overview of the benefits of using fixed curves is provided, followed by an examination of existing methods of software attack on elliptic curves and their impact on the security of fixed curves. Included in the examination is a variant of Pollard’s rho method which can be used to break more than one ECDLP on the one curve. A lower bound on the expected number of iterations required to solve a subsequent ECDLP using this method is then presented, as well as an approximation for the number of remaining iterations to solve an ECDLP when
a given number of iterations have already been performed. The threat to fixed curves due to hardware attacks and optimizations for curves with special properties is also examined. It is concluded that despite the above issues regarding the security of fixed curves, using a fixed curve is not significantly less secure than using a random curve. In particular, adding approximately 5 bits to the size of a fixed curve compared to a random curve to avoid general software attacks and another 6 bits to avoid attacks on special moduli and \( a \) parameters (i.e. a total of 11 bits) is sufficient to obtain an equivalent level of security.

5.1 Overview of Fixed Curve Benefits

One drawback of an ECC is the complexity of generating a secure elliptic curve. The complexity is high enough to render it infeasible to generate a randomly chosen but secure elliptic curve on a mobile device (e.g. telephones, PDAs and smart cards) due to the time, memory and code size required to count the points on the curve and ensure that other security requirements [BSS99, Section V.7] are met. For example, the Schoof-Elkies-Atkin (SEA) point counting algorithm is the best known point counting algorithm for a randomly chosen EC over \( GF(p) \) and has complexity \( O \left( \log^8(p) \right) \) [BSS99]. It has been implemented in the miracl library in conjunction with Pollard’s lambda method and takes 2-3 minutes on a 180 MHz Pentium Pro to count the points on a 160 bit curve and 3.5 - 5.5 minutes for a 192 bit curve [Sco99]. On a smart card platform, it would take much longer—a 10 MHz smart card could be expected to take at least 36 minutes to count the points on a 160 bit curve based on processor speed. However, it is likely that code size and memory limitations would preclude the algorithm from being programmed onto such a smart card in the first place.

Even if a mobile device could generate a secure elliptic curve, there would still be other costs, such as the bandwidth required to transmit the curve to other parties. The cost of transmitting a curve over \( GF(p) \) is that of transmitting four numbers modulo \( p \). These numbers are the two curve constants \( a \) and \( b \) as well as the base point and the mobile device’s public key in compressed format. Added to the bandwidth and time costs, there is also the cost of a substantially increased code size associated with generating a curve on the mobile device.

On the other hand, if a fixed curve is used, we know that it is feasible to implement an associated ECC on a mobile device since various implementations have
been reported (for example, the implementations in [HNM98] and Chapter 3). When using a fixed curve, the mobile device is only required to generate a secret key using a random number generator and to transmit the corresponding public key to the other parties. The random number generator is needed in any case by some signature algorithms, and the scalar multiplication routine required to generate the public key will already be available for use in the protocols utilized by the mobile device. Therefore, any extra code associated with key selection is minimal when using a fixed curve. Other advantages of using fixed curves include being able to choose special parameters to increase the efficiency of the implementation (see Sections 3.1.1, 3.2.6 and 5.3.3) and a reduced bandwidth requirement since only one number modulo \( p \) (the mobile device’s compressed public key) must be transmitted to other parties. The fact that the fixed curve parameters are required to be publicly available is not a disadvantage when compared with random curves because the curve parameters of random curves must also be made public before the curve can be used. While these issues may not be major for all mobile devices (e.g. in some applications the random curve could be generated by a server and bandwidth usage might not be a problem), the difficulties associated with using random curves have caused various standards organizations to include fixed curves in their standards, such as the WAP curves [WAP01] and the NIST curves [NIS00].

Whilst a fixed curve may be an attractive option due to efficiency reasons, it also offers a single target for people all over the world to attack. On the other hand, if random curves are utilized, there are many more curves in use throughout the world, so that a group of attackers no longer has one target, but many targets to attack. The random curves used may also be constantly changed, making the number of possible targets to attack even greater. Furthermore, attacking one curve will not give the attackers any advantage if they wish to attack a different curve at a later date. Thus the computational power deployed to break a fixed curve is likely to be much greater than that deployed to break a random curve. In addition to this, if a fixed curve is broken, all users of that curve are affected. On the other hand, if a random curve used by a small number of people is broken, the overall impact is much smaller than if a fixed curve used by many people all over the world is broken.

Given the above observations, it would appear intuitively obvious that using a random curve provides a higher level of security than a fixed curve. However,
exactly how much extra security a random curve provides and whether the amount of extra security is significant is much less clear. Previously, there have been no publications examining whether the decision to use a fixed curve compromises the security of a cryptosystem, and the significance of any such compromise. This issue is investigated in detail in this chapter.

5.2 Existing Methods of Attack

In this section the efficiencies of different methods available to attack the ECDLP are examined. Only those attacks applicable to arbitrary elliptic curves are considered. These attacks are then used in the following section to analyse the security of fixed curves compared to random curves.

5.2.1 Pohlig-Hellman Algorithm

The Pohlig-Hellman [PH78] algorithm breaks the ECDLP down into several different ECDLPs, one in each prime order subgroup of the elliptic curve group. Obviously, the hardest one of these to solve is in the subgroup of largest prime order, and thus the attack is resisted by requiring the order of this subgroup to be at least 160 bits [BSS99, p.98]. For the rest of this chapter, we assume that (if applicable) the Pohlig-Hellman algorithm has been used to reduce the ECDLP to an ECDLP in the subgroup of largest prime order.

5.2.2 Index Calculus and Related Methods

There are currently no index calculus or related methods applicable to elliptic curves. Indeed, it is believed to be unlikely that such attacks will ever be possible [HKT00]. Therefore these methods are considered no further in this chapter.

5.2.3 Shanks’s Baby-Step Giant-Step Method

The baby-step giant-step (BSGS) method of Shanks [Sha71] has a precomputation for each curve. A balanced version is often given in the literature (e.g. [BSS99]). We give an unbalanced version below which takes advantage of the fact that the negative of an elliptic curve point can be calculated “for free,” in a similar manner
to Shanks’s original proposal. Let $n$, $Q$, $z$, $m$, $R$ and $d$ be defined as follows:

\[
\begin{align*}
n &= \text{The prime order of the base point } P \\
Q &= \text{The point whose ECDL is to be found} \\
z &= \text{The value of the ECDLP. That is, } Q = [z]P \\
m &= \text{Number of points in the precomputation} \\
d &= \left\lceil \frac{n}{2m} \right\rceil \\
R &= [d]P .
\end{align*}
\]

Then the precomputation of giant steps can be calculated as:

\[
S_\alpha = [\alpha] R \text{ for } 0 \leq \alpha < m
\]

and the ECDLP can be solved by finding the baby steps:

\[
R_\beta = Q - [\beta] P \text{ for } 0 \leq \beta < d
\]

until an $R_\beta$ value is found which is the same as $S_\alpha$ or $-S_\alpha$ for some $\alpha$. The solution to the ECDLP is then:

\[
\begin{align*}
z &= \alpha d + \beta \text{ if } R_\beta = S_\alpha \\
or \quad z &= n - \alpha d + \beta \text{ if } R_\beta = -S_\alpha .
\end{align*}
\]

There are approximately $m$ elliptic curve additions required in the precomputation, and on average $\frac{d}{2}$ further elliptic curve additions required to solve the ECDLP. Thus, on average, it requires approximately $\frac{4m^2 + n}{4m}$ operations to solve one ECDLP. This value is at its minimum of $\sqrt{n}$ operations when $m \approx \sqrt{\frac{n}{2}}$.

### 5.2.4 Pollard’s Rho Method

Pollard’s rho method [Pol78] is currently the best method known for solving the general ECDLP [WZ99]. The method searches for a collision in a pseudo-random walk through the points on the curve. If the iterating function defining the pseudo-random walk is independent of the point whose discrete logarithm is to be found, then the same calculations can be used to find more than one discrete logarithm on the one curve. Kuhn and Struik [KS01] provide an analysis of the
Table 5.1: Definitions for Pollard’s rho method

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>The prime order of the base point ( P ).</td>
</tr>
<tr>
<td>( Q_k )</td>
<td>The points whose ECDLs are to be found. That is, ( Q_k = [z_k]P ), where we wish to find ( z_k ) for ( k \geq 0 ).</td>
</tr>
<tr>
<td>( R_{k,0} )</td>
<td>([u_{k,0}]P + [w_{k,0}]Q_k), where ( u_{k,0} ) and ( w_{k,0} ) are randomly chosen constants and ( w_{k,0} \neq 0 ).</td>
</tr>
<tr>
<td>( R_{k,i} )</td>
<td>The ( i )th point in the pseudo-random walk to solve the ECDLP for ( Q_k ). Note that ( R_{k,i} = [u_{k,i}]P + [w_{k,i}]Q_k ).</td>
</tr>
<tr>
<td>( s )</td>
<td>The number of equations defining the pseudo-random walk.</td>
</tr>
<tr>
<td>( f(R) )</td>
<td>A function mapping a point ( R ) to a number between 1 and ( s ).</td>
</tr>
<tr>
<td>( g(R_{k,i}) )</td>
<td>A function returning the next value in the pseudo-random walk, ( R_{k,i+1} ). It is defined as: ( g(R_{k,i}) = [h_{f(R_{k,i})}]R_{k,i} + [c_{f(R_{k,i})}]P ), where ( c_j ) and ( h_j ) are constants for ( 1 \leq j \leq s ).</td>
</tr>
<tr>
<td>( u_{k,i+1} )</td>
<td>( h_{f(R_{k,i})}u_{k,i} + c_{f(R_{k,i})} \pmod{n} ) for ( 0 \leq i ).</td>
</tr>
<tr>
<td>( w_{k,i+1} )</td>
<td>( h_{f(R_{k,i})}w_{k,i} \pmod{n} ) for ( 0 \leq i ).</td>
</tr>
</tbody>
</table>

The expected running time of such a method, which is described as follows. Let the definitions in Table 5.1 be given. The pseudo-random walk function to solve the \( k \)th ECDLP, \( g(R_{k,i}) \), is defined to be as follows:

\[
g(R_{k,i}) = [h_{f(R_{k,i})}]R_{k,i} + [c_{f(R_{k,i})}]P
\]

where \( h_j \) and \( c_j \) are constants. Note that the next value in the pseudo-random walk to solve the \( k \)th ECDLP \( R_{k,i+1} \) is determined only by \( P \) and \( R_{k,i} \), not \( P \), \( Q_k \) and \( R_{k,i} \). In order to maximize efficiency, \( h_j \) should be set to 1 for all but one of the possible values of \( j \), in which case \( h_j \) should be set to 2 and \( c_j \) should be set to zero. If this is done, each iteration of the method will require only one elliptic curve addition. This random walk is similar to a special case of the “combined walk” proposed by Teske [Tes98]. We note that currently there is no proof that the above random walk is sufficiently random for the theoretical results (which assume the randomness of the walk) to hold. However, it differs from the random walk with such a proof proposed by Teske [Tes98] in approximately \( 1/s \) cases where \( s \) is the number of equations defining the pseudo-random walk and \( s \approx 20 \) gives optimal performance [Tes98]. Since the random walk proposed here differs from Teske’s random walk in only about \( 1/20 \) cases, it is expected to perform randomly enough.

There are two different types of collisions which can occur, a collision with a point on the current pseudo-random walk and a collision with a point on a
5.2. Existing Methods of Attack

previous pseudo-random walk. They can be solved as follows:

If \( R_{k,i} = R_{k,j} \)
then \([u_{k,i}]P + [w_{k,i}]Q_k = [u_{k,j}]P + [w_{k,j}]Q_k \)
with a solution of \( Q_k = \left[ \frac{u_{k,j} - u_{k,i}}{w_{k,i} - w_{k,j}} \right] P \).

Otherwise, \( R_{k,i} = R_{l,j} \)
where \( R_{l,j} = [u_{l,j}]P + [w_{l,j}]Q_l \)
and \( Q_l = [z_l]P \).

Therefore, \([u_{k,i}]P + [w_{k,i}]Q_k = [u_{l,j}]P + [w_{l,j}z_l]P \)
with a solution of \( Q_k = \left[ \frac{u_{l,j} + w_{l,j}z_l - u_{k,i}}{w_{k,i}} \right] P \).

In order to detect collisions, we need to store the points on the pseudo-random walk \( R_{k,i} \) and compare the current point on the random walk with previous points. However, in order to save storage space, \( R_{k,i} \), \( u_{k,i} \) and \( w_{k,i} \) are only stored if \( R_{k,i} \) is a distinguished point. Distinguished points are defined as those points with some easily checkable property (such as having ten leading zeros in the \( x \) coordinate).

Because only distinguished points are stored, we will not always detect a collision as soon as it occurs, but rather at the next distinguished point on the pseudo-random walk. Because \( R_{k,i+1} \) only depends on \( R_{k,i} \) and \( P \), once a collision occurs between \( R_{k,i} \) and \( R_{l,j} \) we know that \( R_{k,i+m} = R_{l,j+m} \) for all \( m \geq 0 \). For this reason, a collision is guaranteed to be detected at the next distinguished point.

We emphasize that if distinguished points are used, the random walk definition must be independent of the values \( Q_1, Q_2, \ldots \) in order for the following results to hold. We note that the particular random walk recommended by Kuhn and Struik in [KS01] (originally recommended by Teske [Tes98]) to solve a single ECDLP should not be used to solve multiple ECDLPS when using distinguished points, because it must depend on \( Q_i \) in order to be useful. Although Kuhn and Struik provide an analysis of the time required to solve multiple ECDLPS, they do not specify a suitable random walk to use in this situation. The problem with the random walk depending on any \( Q_i \) is that a different random walk must be used for each ECDLP to be solved. This in turn means that any collisions of non-distinguished points from different random walks are not detected because the random walks take different paths after the collision. Only collisions of distin-
guished points are detected in this case. Of course, if all points are distinguished then the random walk may depend on $Q_i$ since all collisions are detected. However, in most practical situations, not all points will be distinguished. Figure 5.1 shows the results of using a different random walk for each ECDLP compared to using the one random walk described above. It is easily seen that while some advantage is gained from the distinguished points from previous (different) random walks, that advantage is quite small compared to the advantage gained if only one random walk is used.

We now wish to know how much of an improvement previous calculations can offer to the speed with which the solution to a subsequent ECDLP is found.

Let $Z_i$ = The number of iterations required to solve the $i^{th}$ ECDLP.

$T_i$ = The total number of iterations to solve the first $i$ ECDLPs.

Note: $T_i \geq i + 1$.

Obviously, the expected value of $Z_1$, $E(Z_1)$, is the same as that of the traditional Pollard’s rho method, namely $[Tes98]$:

$$E(T_1) = E(Z_1) \approx \sqrt{\frac{\pi n}{2}}.$$

We note that Wiener and Zuccherato $[WZ99]$ have been able to improve this figure by a factor of $\sqrt{2}$ by restricting the random walk to points with distinct $x$ coordinates. For simplicity, we have not included this optimization in the description of Pollard’s rho method in this section. However, by changing $n$ to $n/2$ in the following discussion, its effect can be taken into consideration. In Section 5.3 this optimization is taken into account in the calculations performed.

It is also possible to parallelize Pollard’s rho algorithm $[vOW99]$ to obtain a linear speedup. However, it is not necessary to include such parallelization directly in the model since it can be taken into account by increasing the speed at which calculations can be made. For example, a parallelized version running on five computers each at speed $x$ will complete in the same time as a non-parallelized version on a single computer running at speed $5x$. We therefore account for any increase in speed due to parallelization by setting the speed at which computations are performed to an appropriate value.

We now wish to find $E(T_i)$ and $E(Z_i)$ for $i > 1$. Kuhn and Struik $[KS01]$
Fig. 5.1: Actual iterations to solve 50 ECDLPs on a 25-bit curve averaged over 200 trials with 1 in 400 points distinguished and $s = 5$. 

5.2. Existing Methods of Attack

One random walk
50 different random walks
provide an approximation for the expected value of \( Z_{i+1} \) as:

\[
E(Z_{i+1}) \approx \sqrt{\frac{\pi n}{2}} \left( \frac{2i}{i} \right) \frac{1}{4^i} \quad \text{for } i \ll n^{\frac{1}{4}} \quad (5.1)
\]

and the expected value of \( T_{i+1} \) as:

\[
E(T_{i+1}) \approx \sqrt{\frac{\pi n}{2}} \sum_{t=0}^{i-1} \left( \frac{2t}{t} \right) \frac{1}{4^t} \quad \text{for } i \ll n^{\frac{1}{4}} \quad (5.2)
\]

\[
\approx 2 \sqrt{\frac{i}{\pi}} E(Z_1) \quad (5.3)
\]

This dissertation proves a new result which gives a lower bound on the expected values above. The lower bound is quite similar to the approximations above, but replaces \( \sqrt{\frac{\pi n}{2}} \) with \( E(Z_1) \). This is stated formally in the following theorem:

**Theorem 5.1.** Let \( Z_i \) and \( T_i \) be defined as above. Then the following inequalities hold:

\[
E(Z_{i+1}) \geq \left( \frac{2i}{i} \right) \frac{1}{4^i} E(Z_1) \quad \text{for } i \geq 1 \quad (5.4)
\]

\[
E(Z_{i+1}) \geq \frac{1}{2i} E(T_i) \quad \text{for } i \geq 1 \quad . \quad (5.5)
\]

Substituting the first few values of \( i \) into (5.4) gives:

\[
E(Z_2) \geq \frac{1}{2} E(Z_1)
\]

\[
E(Z_3) \geq \frac{3}{4} \cdot \frac{1}{2} E(Z_1)
\]

\[
E(Z_4) \geq \frac{5}{6} \cdot \frac{3}{4} \cdot \frac{1}{2} E(Z_1) .
\]

The proof of Theorem 5.1 is dependent on the following lemma:

**Lemma 5.1.**

\[
\sum_{k_1=1}^{t-\alpha} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} \frac{k_{\alpha-1}}{n} \geq \frac{t(t-\alpha)}{2\alpha n} \sum_{k_1=1}^{t-\alpha} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} \frac{k_{\alpha-1}}{n} \quad (5.6)
\]
Proof. With a change of variables, (5.6) can be rewritten as:

\[ \sum_{k_1=1}^{c} \sum_{k_2=1}^{k_1} \cdots \sum_{k_\beta=1}^{k_{\beta-1}} k_1k_2 \cdots k_\beta \geq \frac{c(c + \beta)}{2\beta} \sum_{k_1=1}^{c} \sum_{k_2=1}^{k_1} \cdots \sum_{k_{\beta-1}=1}^{k_{\beta-2}} k_1k_2 \cdots k_{\beta-1} \]

and by subtracting \( \left( c \sum_{k_1=1}^{c} \sum_{k_2=1}^{k_1} \cdots \sum_{k_{\beta-1}=1}^{k_{\beta-2}} k_1k_2 \cdots k_{\beta-1} \right) \) from both sides we obtain:

\[ \sum_{k_1=1}^{c-1} \sum_{k_2=1}^{k_1} \cdots \sum_{k_\beta=1}^{k_{\beta-1}} k_1k_2 \cdots k_\beta \geq \frac{c(c - \beta)}{2\beta} \sum_{k_1=1}^{c} \sum_{k_2=1}^{k_1} \cdots \sum_{k_{\beta-1}=1}^{k_{\beta-2}} k_1k_2 \cdots k_{\beta-1} \tag{5.7} \]

We now proceed to prove (5.7) by using two nested proofs by induction. Firstly we prove by induction on \( \beta \), and to complete the proof, we need to use induction on \( c \) for a specific value of \( \beta \).

Firstly, we prove (5.7) true for \( \beta = 2, c \geq 1 \). When \( \beta = 2 \), we have:

\[
\sum_{k_1=1}^{c-1} \sum_{k_2=1}^{k_1} k_1k_2 = \sum_{k_1=1}^{c-1} k_1^2(k_1 + 1) / 2 \\
= \frac{1}{2} \left( \sum_{k_1=1}^{c-1} k_1^3 + \sum_{k_1=1}^{c-1} k_1^2 \right) \\
= \frac{1}{2} \left( \frac{c^2(c - 1)^2}{4} + \frac{c(c - 1)(2c - 1)}{6} \right) \\
= \frac{c(3c - 2)(c - 1)(c + 1)}{24} \\
= \frac{c(c - \frac{2}{3})(c - 1)(c + 1)}{8} \\
\geq \frac{c^2(c - 2)(c + 1)}{8} \\
= \frac{c(c - 2)}{4} \sum_{k_1=1}^{c} k_1
\]

which completes the proof.

We now assume (5.7) true for \( \beta = \alpha - 1 \) and \( c \geq 1 \), and prove that this implies (5.7) is true for \( \beta = \alpha \). In particular, assume (5.7) for \( \beta = \alpha - 1 \)
Chapter 5. The Security of Fixed versus Random Elliptic Curves

and \( c = n + 1 \), as in (5.8).

\[
\sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} \\
\geq \frac{(n+1)(n+1-\alpha+1)}{2(\alpha-1)} \sum_{k_1=1}^{n+1} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-2}=1}^{k_{\alpha-3}} k_1 k_2 \ldots k_{\alpha-2} .
\]

(5.8)

That is, when \( n + 2 > \alpha \) we assume:

\[
\sum_{k_1=1}^{n+1} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-2}=1}^{k_{\alpha-3}} k_1 k_2 \ldots k_{\alpha-2} \\
\leq \frac{2(\alpha-1)}{(n+1)(n-\alpha+2)} \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} .
\]

(5.9)

We are now able to proceed to prove:

\[
\sum_{k_1=1}^{c-1} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha} \geq \frac{c(c-\alpha)}{2\alpha} \sum_{k_1=1}^{c} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} 
\]

(5.10)

which is obviously true when \( \alpha \geq c \geq 1 \), since the RHS \( \geq 0 \) and the LHS \( \leq 0 \).

We prove (5.10) when \( c > \alpha \) by induction. We let the “first” case be \( c = \alpha - 1 \). This is obviously true since \( \alpha \geq c \). We now assume that (5.10) is true for some value \( n \) of \( c \) such that \( n \geq \alpha - 1 \). That is, \( n + 2 > \alpha \).

Substituting \( n \) for \( c \) in (5.10) gives (5.11) which we assume is true:

\[
\sum_{k_1=1}^{n-1} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha} \geq \frac{n(n-\alpha)}{2\alpha} \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} 
\]

(5.11)

\[
\sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha} \geq n \left( \frac{n-\alpha}{2\alpha} + 1 \right) \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} \\
= \frac{n(n+\alpha)}{2\alpha} \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} 
\]

(5.12)

We now proceed to prove (5.10) for \( c = n + 1 \) given (5.11). That is, we proceed
5.2. Existing Methods of Attack

to prove:

\[ \sum_{k_1=1, k_2=1}^{n} \cdots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha} \]

\[ \geq \frac{(n+1)(n+1 - \alpha)}{2^\alpha} \sum_{k_1=1}^{n+1} \sum_{k_2=1}^{k_1} \cdots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} . \]  (5.13)

From (5.12), (5.13) is true if:

\[ n(n+\alpha) \sum_{k_1=1, k_2=1}^{n} \cdots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} \]

\[ \geq (n+1)(n+1 - \alpha) \sum_{k_1=1, k_2=1}^{n+1} \cdots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} . \]  (5.14)

Rearranging the RHS of (5.14) and then substituting (5.9) into it gives:

\[ (n+1)(n+1 - \alpha) \sum_{k_1=1, k_2=1}^{n+1} \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} \]

\[ = (n+1)(n+1 - \alpha) \sum_{k_1=1, k_2=1}^{n} \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} \]

\[ + (n+1)^2 (n+1 - \alpha) \sum_{k_1=1, k_2=1}^{n+1} \sum_{k_{\alpha-2}=1}^{k_{\alpha-3}} k_1 k_2 \cdots k_{\alpha-2} \]

\[ \leq (n+1)(n+1 - \alpha) \sum_{k_1=1, k_2=1}^{n} \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} \]

\[ + \frac{2(\alpha - 1)(n+1)^2 (n+1 - \alpha)}{(n+1)(n-\alpha+2)} \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \cdots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} \]

\[ = (n+1)(n+1 - \alpha) \frac{(n+\alpha)}{n+2 - \alpha} \sum_{k_1=1, k_2=1}^{n} \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \cdots k_{\alpha-1} \]

\[ = Q . \]

Now LHS of (5.14) ≥ Q \implies LHS of (5.14) ≥ RHS of (5.14). Thus if we prove LHS of (5.14) ≥ Q, or in other words LHS of (5.14) − Q ≥ 0, we will have proved
LHS of (5.14) ≥ RHS of (5.14).

LHS of (5.14) − Q

\[
\begin{align*}
&= n(n + \alpha) \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} \\
&\quad - \frac{(n + 1)(n + 1 - \alpha)(n + \alpha)}{n + 2 - \alpha} \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} \\
&= \frac{(n + \alpha)(\alpha - 1)}{n + 2 - \alpha} \sum_{k_1=1}^{n} \sum_{k_2=1}^{k_1} \ldots \sum_{k_{\alpha-1}=1}^{k_{\alpha-2}} k_1 k_2 \ldots k_{\alpha-1} \\
&= R.
\end{align*}
\]

Now \( R \geq 0 \) when \( n + 2 > \alpha \). Thus we have proved that (5.14) is true when \( n + 2 > \alpha \), and hence (5.13) is true given \( n + 2 > \alpha \), (5.11) and (5.9).

Since given (5.9) is true and (5.10) is true for \( c = n \) and \( n + 2 \geq \alpha \) implies that (5.10) is true for \( c = n + 1 \) and since (5.10) is obviously true for \( 1 \leq c \leq \alpha \), (5.10) is true for all \( c \geq 1 \) given (5.9).

Since (5.7) is true for \( \beta = 2 \) and (5.7) is true for \( \beta - 1 \) implies (5.7) is true for \( \beta \), (5.7) is true for all \( \beta \geq 2 \). This completes the proof. \( \square \)

It is now possible to prove Theorem 5.1 as follows:

**Proof.** In order to find \( E(T_i) \) and \( E(Z_i) \) for \( i > 1 \) we start with the formulae given in [vOW99]:

\[
\Pr(T_1 > t) = \prod_{j=1}^{t-1} \left( 1 - \frac{j}{n} \right)
\]

\[
E(T_i) = \sum_{j=1}^{\infty} j \cdot \Pr(T_i = j)
\]  
\( (5.15) \)

\[
= \sum_{j=1}^{\infty} j \cdot (\Pr(T_i > j - 1) - \Pr(T_i > j))
\]

\[
= \sum_{j=0}^{\infty} \Pr(T_i > j).
\]  
\( (5.16) \)
5.2. Existing Methods of Attack

We now give \( \Pr (T_i = t) \) and \( \Pr (T_{i+1} > t) \):

\[
\Pr (T_i = t) = \frac{t-i}{n} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{t-i} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{t-i} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i} \left( 1 - \frac{k}{n} \right) \tag{5.17}
\]

\[
\Pr (T_{i+1} > t) = \Pr (T_i > t) + \Pr (T_{i+1} > t \text{ and } T_i \leq t)
= \Pr (T_i > t)
+ \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{t-i} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{t-i} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i} \left( 1 - \frac{k}{n} \right) \tag{5.18}
\]

Using (5.15) and (5.17), we derive an expression for \( \mathbb{E}(T_i) \):

\[
\mathbb{E}(T_i) = \sum_{t=i+1}^{n+i} \frac{t-i}{n} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{t-i} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{t-i} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i} \left( 1 - \frac{k}{n} \right)
= (i+1) \left( \frac{1}{n} \right)^i + \sum_{t=i+2}^{n+i} \frac{t-i}{n} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{t-i} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{t-i} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i} \left( 1 - \frac{k}{n} \right) \tag{5.19}
\]

We now derive an expression for \( \mathbb{E}(T_{i+1}) \) from (5.16) and (5.18):

\[
\mathbb{E}(T_{i+1}) = \sum_{t=0}^{\infty} \Pr (T_{i+1} > t)
= 2 + \sum_{t=i+2}^{n+i} \Pr (T_{i+1} > t)
= 2 + \sum_{t=i+2}^{n+i} \Pr (T_i > t)
+ \sum_{t=i+2}^{n+i} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{t-i} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{t-i} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i} \left( 1 - \frac{k}{n} \right)
\]
Chapter 5. The Security of Fixed versus Random Elliptic Curves

\[ E(T_{i+1}) = E(T_i) + 1 - \Pr(T_i > i + 1) \]
\[ + \sum_{t=i+2}^{n+i} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{k_{i-2}} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i-1} \left( 1 - \frac{k}{n} \right) \]
\[ = E(T_i) + \left( \frac{1}{n} \right)^i \]
\[ + \sum_{t=i+2}^{n+i} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{k_{i-2}} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i-1} \left( 1 - \frac{k}{n} \right). \quad (5.20) \]

We then substitute into (5.20) the inequality of Lemma 5.1 which is given again in (5.21):

\[ \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{k_{i-2}} \frac{k_{i-1}}{n} \geq \frac{t(t - \alpha)}{2\alpha n} \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{k_{i-2}} \frac{k_{i-1}}{n} \quad (5.21) \]

\[ \therefore E(T_{i+1}) \geq E(T_i) + \left( \frac{1}{n} \right)^i \]
\[ + \sum_{t=i+2}^{n+i} \frac{t(t - i)}{2n i} \left( \sum_{k_1=1}^{t-i} \frac{k_1}{n} \sum_{k_2=1}^{k_1} \frac{k_2}{n} \cdots \sum_{k_{i-1}=1}^{k_{i-2}} \frac{k_{i-1}}{n} \right) \prod_{k=1}^{t-i-1} \left( 1 - \frac{k}{n} \right) \quad (5.22) \]

and substituting (5.19) into (5.22) gives:

\[ E(T_{i+1}) \geq E(T_i) + \frac{1}{2i} E(T_i) \]
\[ = \frac{2i + 1}{2i} E(T_i) \]
\[ : E(T_{i+1}) \geq \prod_{k=1}^{i} \left( \frac{2k + 1}{2k} \right) E(T_1) \quad (5.23) \]

Similarly,

\[ E(T_i) \geq \prod_{k=1}^{i-1} \left( \frac{2k + 1}{2k} \right) E(T_1) \quad (5.24) \]
and note that

\[ E(T_{i+1}) = E(Z_{i+1}) + E(T_i) \quad (5.25) \]

By substituting (5.25) into (5.23) and then (5.24) into (5.26) we obtain:

\[
\begin{align*}
E(Z_{i+1}) + E(T_i) & \geq E(T_i) + \frac{1}{2i} E(T_i) \\
E(Z_{i+1}) & \geq \frac{1}{2i} E(T_i) \\
E(Z_{i+1}) & \geq \frac{1}{2i} \prod_{k=1}^{i-1} \left( \frac{2k+1}{2k} \right) E(T_i) \\
& = \prod_{k=1}^{i} \left( \frac{2k-1}{2k} \right) E(Z_1).
\end{align*}
\]

That is, we have:

\[
\begin{align*}
E(Z_2) & \geq \frac{1}{2} E(Z_1) \\
E(Z_3) & \geq \frac{3}{4} \cdot \frac{1}{2} E(Z_1) \\
E(Z_4) & \geq \frac{5}{6} \cdot \frac{3}{4} \cdot \frac{1}{2} E(Z_1) \\
E(Z_i) & \geq \frac{(2(i-1))!}{2^{2(i-1)}((i-1)!)^2} E(Z_1) \quad \text{for } i \geq 2 \quad (5.28) \\
E(Z_{i+1}) & \geq \left( \frac{2i}{i} \right) \frac{1}{4^i} E(Z_1) \quad \text{for } i \geq 1. \quad (5.29)
\end{align*}
\]

Equations (5.29) and (5.27) complete the proof of (5.4) and (5.5) respectively.

\[ \square \]

It is easily seen that the left and right hand sides of Lemma 5.1 and (5.21) are equal when \( \alpha = 1 \), since they give the formula for the sum of numbers from 1 to \( t-1 \) in that case. Therefore, in the case of \( i = 1 \) in the above proof, the greater than or equal to signs can be replaced with an equals sign in (5.22), (5.23) and (5.29), giving the results:

\[
\begin{align*}
E(Z_2) & = \frac{1}{2} E(Z_1) \\
\text{and } E(T_2) & = \frac{3}{2} E(Z_1).
\end{align*}
\]
In order to study the behaviour of $E(Z_i)$ more easily, (5.28) can be approximated using Stirling’s formula which states [GB73, p.373]:

$$n! \approx (2\pi)\frac{1}{2} n^{n+\frac{1}{2}} e^{-n} \quad \text{for large } n .$$  \hspace{1cm} (5.30)

Substituting (5.30) where possible in (5.28), we obtain:

$$E(Z_i) \geq \frac{1}{\sqrt{\pi i}} E(Z_1) \quad \text{for large } i .$$  \hspace{1cm} (5.31)

The above results lead us to expect the second ECDLP to be solved in half the time of the first, the third ECDLP to be solved in no less than three-eighths of the time of the first, and so on. As stated in [KS01], (5.1) and (5.2) are good approximations. We provide experimental evidence of this in Fig. 5.2, which shows the actual number of iterations to solve 50 ECDLPs on a 32-bit curve averaged over 200 trials, as well as the bound in (5.4). Note that since $E(Z_1)$ has been taken to be $\sqrt{\frac{\pi}{2}}$, the bound is the same as the expected value provided in (5.1).

We also use $E(Z_i \mid r \text{ previous iterations})$ in our analysis in the following section. An approximation is given by Theorem 5.2:

**Theorem 5.2.** Let $Z_i$ and $n$ be defined as above. Then:

$$E(Z_i \mid r \text{ previous iterations}) \approx \sqrt{\frac{\pi n}{2}} e^{\frac{r^2}{2n}} \left( 1 - \Phi\left( \frac{r}{\sqrt{2n}} \right) \right)$$  \hspace{1cm} (5.32)

where $\Phi(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt$.

**Proof.** Suppose that during a random walk to solve previous ECDLPs, as defined in Table 5.1, $r$ iterations have been performed. Then for $Z_i$, the number of iterations to solve a new ECDLP,

$$\Pr(Z_i > z) = \left( 1 - \frac{r}{n} \right) \left( 1 - \frac{r+1}{n} \right) \cdots \left( 1 - \frac{r+z-1}{n} \right)$$

and taking logarithms we have:

$$\ln(\Pr(Z_i > z)) = \ln \left( 1 - \frac{r}{n} \right) + \ln \left( 1 - \frac{r+1}{n} \right) + \cdots + \ln \left( 1 - \frac{r+z-1}{n} \right) .$$  \hspace{1cm} (5.33)
5.2. Existing Methods of Attack

Fig. 5.2: Actual iterations to solve 50 ECDLPs on a 32-bit curve, averaged over 200 trials, with $s = 21$ and $1$ in $800$ points distinguished. The theoretical bound from (5.4) is also shown.
Now we can expand $\ln(1 + x)$ using a Maclaurin series as:

$$
\ln (1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots .
$$

Using this expansion and assuming $r, z \leq \sqrt{n}$, we can approximate (5.33) as:

$$
\ln (\Pr (Z > z)) \approx - \sum_{j=r}^{r+z-1} \frac{j}{n} - \frac{1}{2} \sum_{j=r}^{r+z-1} \frac{j^2}{n^2} - \cdots
$$

$$
\approx - \frac{z(2r + z - 1)}{2n} - O \left( \frac{1}{\sqrt{n}} \right)
$$

$$
\therefore \Pr (Z > z) \approx e^{-\frac{z}{2\sqrt{n}}} (z + 2r - 1). \quad (5.34)
$$

Substituting (5.34) into (5.16) produces:

$$
\mathbb{E}(Z_i) \approx \sum_{z=0}^{\infty} e^{-\frac{z}{2\sqrt{n}}} (z + 2r - 1)
$$

$$
\approx \int_0^\infty e^{-\frac{z}{2\sqrt{n}}} (z + 2r - 1) dz
$$

$$
= \int_{r - \frac{1}{2}}^{\infty} e^{-\frac{y^2}{2\sqrt{n}}} (y^2 - (r - \frac{1}{2})^2) dy \quad \text{where} \quad y = z + r - \frac{1}{2}
$$

$$
= e^\frac{1}{2\sqrt{n}} (r - \frac{1}{2})^2 \int_{r - \frac{1}{2}}^{\infty} e^{-\frac{y^2}{2\sqrt{n}}} dy
$$

$$
= e^\frac{1}{2\sqrt{n}} (r - \frac{1}{2})^2 \sqrt{2n} \int_{\frac{1}{\sqrt{2\sqrt{n}}} (r - \frac{1}{2})}^{\infty} e^{-t^2} dt \quad \text{where} \quad t = \frac{y}{\sqrt{2n}} . \quad (5.35)
$$

Now the error function $\Phi(x)$ is defined as [GR94, p.938]:

$$
\Phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt
$$

and from [GR94, p.354],

$$
\int_0^\infty e^{t^2} dt = \frac{\sqrt{\pi}}{2}
$$

$$
\therefore 1 - \Phi(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt . \quad (5.36)
$$
5.3. Security Comparison

Substituting (5.36) into (5.35) gives:

\[
E(Z_i) \approx e^{\frac{1}{2\pi n}(r-\frac{1}{2})^2 \sqrt{\frac{2\pi n}{2}}} \left(1 - \Phi \left(\frac{r}{\sqrt{2n}}\right)\right) \\
\approx e^{\frac{r^2}{2\pi}} \sqrt{\frac{\pi n}{2}} \left(1 - \Phi \left(\frac{r}{\sqrt{n}}\right)\right).
\] (5.37)

This completes the proof. \(\square\)

If we take into account the optimization of Wiener and Zuccherato [WZ99], (5.32) becomes:

\[
E(Z_i \mid r \text{ previous iterations}) \approx \sqrt{\frac{\pi n}{2} e^2 \left(1 - \Phi \left(\frac{r}{\sqrt{n}}\right)\right)}
\] (5.38)

with \(\Phi(x)\) defined as before.

5.3 Security Comparison

This section contains a detailed comparison of the security of fixed versus random curves. Firstly, it examines “equivalent security” by assuming that a precomputation can be made to aid in solving the ECDLP on a fixed curve. All other attributes of the curves are assumed to be equal for this comparison. Secondly, special properties of curves which may give a greater amount of speed to an ECC implementation and are therefore likely to be used in fixed curves are examined in relation to the ECDLP. Unfortunately, such properties also make it faster to find an ECDL and this issue is addressed. Thirdly, special purpose hardware is considered and finally all of the results are combined into an overall recommendation.

Throughout this section, we use \(n_F\) and \(n_R\) to denote the orders of the prime (sub)groups of the fixed and random curves respectively where both curves have an “equivalent” level of security.

5.3.1 Equivalent Security I

In order to compare the security of fixed and random curves, we need a definition of equivalence of security. We give one possible definition below:
Definition 5.1 (Equivalent Security I). Assume that there are $2^\nu$ users of a fixed curve (for example, $2^{32}$ users), and that if these users moved to random curves, there would be $2^\omega$ users per curve, and a total of $2^{\nu-\omega}$ random curves in use. Then we say that the fixed curve has equivalent security I (ES-I) to the random curves if it takes the same expected number of computations to break the ECDLP for all users of the fixed curve as it does to break the ECDLP for all users of the random curves.

5.3.1.1 BSGS and ES-I

In order to apply the ES-I definition to attacks using the BSGS method, we assume that there are $2^\nu$ users of a fixed curve and that if random curves were used, each user would have a different random curve (i.e., $\omega = 0$). In the case for the fixed curve, the precomputation need only be performed once. Therefore, the total amount of computation to find all ECDLs is $V \approx m_F + 2^\nu \frac{n_F}{m_F}$ where $m_F$ is the size of the precomputation. We find that $V$ is at its minimum value of $V_{\text{min}} \approx 2^{\nu/2} \sqrt{n_F}$ for any value of $n_F$ when $m_F \approx 2^{(\nu/2)-1} \sqrt{n_F}$.

We now set $V_{\text{min}}$ equal to the minimum time taken to solve all ECDLps on the random curves (using Pollard’s rho method with the optimization of Wiener and Zuccherato since it is fastest), and solve for $n_F$:

$$V_{\text{min}} = 2^{\nu/2} \sqrt{n_F} = 2^\nu \frac{n_F}{2} \approx 2^\nu n_R.$$

Therefore we need to increase the order of a fixed curve by $\nu$ bits to satisfy ES-I under the assumption of a BSGS attack.

5.3.1.2 Pollard’s Rho and ES-I

Using the definition for ES-I and the result in (5.3), if the fixed and random curves have the same order, then when $(\nu - \omega)$ is sufficiently large, it takes approximately $2^{\nu-\omega}$ times as long to solve all of the ECDLps on the random curves as it does on the fixed curve. Therefore, to decide how much bigger the fixed curve should be than the random curves for an equivalent level of security, we set:

$$\frac{\sqrt{\pi n_F}}{2} = 2^{\nu-\omega} \frac{\sqrt{\pi n_R}}{2}.$$

That is, $n_F = 2^{\nu-\omega} n_R$. 
5.3. Security Comparison

5.3.1.3 ES-I Result

By setting \( \nu = 32 \) (for the approximate number of people in the world) and \( \omega = 0 \) (every user has his/her own random curve) we can give a very conservative worst case estimate that about 32 bits should be added to the curve order of a fixed curve compared to a random curve. This then ensures that the number of group operations required to solve all ECDLPs on the fixed curve is equivalent to the number required to solve all ECDLPs on random curves.

5.3.2 Equivalent Security II

The analysis using ES-I is based on the assumption that we must break all users’ ECDLPs on a fixed curve to break the cryptosystem. In practice, we actually find it unacceptable for even one ECDLP on the fixed curve to be broken, and therefore present a new definition of “equivalent security”. Firstly, suppose that we can perform \( S(t) \) iterations per year at time \( t \), where \( t = 0 \) corresponds to the creation of the fixed curve under consideration. Using Moore’s law [LV01], which states that the efficiency of computers increases exponentially over time, we could set \( S(t) = \kappa_1 e^{\kappa_2 t} \). We then determine the number of iterations performed in \( \tau \) years as:

\[
I(\tau) = \int_0^{\tau} S(t) \, dt, \quad I(\tau) = \frac{\kappa_1 e^{\kappa_2 \tau}}{\kappa_2} - \frac{\kappa_1}{\kappa_2}. \tag{5.39}
\]

**Definition 5.2 (Equivalent Security II).** The value obtained in (5.39) can be taken as the number of iterations used to create an available precomputation \( \tau \) years after the fixed curve has been released. This value can be used to find how many iterations are expected to remain to solve the first ECDLP. A calculation can then be made to determine how many extra bits need to be added to the order of the fixed curve to ensure that at time \( \tau \), the first ECDLP on the fixed curve (using a precomputation) is as hard as an ECDLP on a random curve. We define a fixed curve and a random curve with curve orders satisfying these conditions to have equivalent security II (ES-II).

This definition will give a better estimate of the required increase in curve order than that given by using the definition of ES-I. However, we stress that there is really no significant improvement of the time to solve the first ECDLP on
a fixed curve compared to a random curve if the precomputation time is included. This has been shown by Kuhn and Struik [KS01], who prove the following bounds on the time to solve one out of $k$ ECDLPs (denoted as $Z_{\text{DLP}(1:k)}$):

\[
E(Z_1) - k \leq E(Z_{(1:k)}) \\
E(Z_{(1:k)}) \leq E(Z_1).
\]

Since $k$ is much less than $E(Z_1)$, we can approximate $E(Z_{(1:k)})$ with $E(Z_1)$.

We ignore the precomputation time when calculating ES-II because there is a greater incentive to break ECDLPs on a fixed curve than a random curve (this is because more ECDLPs may be solved for the same effort). Again, we note that if it is feasible to solve one ECDLP on a fixed curve of a particular size, then it is also possible to solve one ECDLP on a random curve of the same size in the same manner.

In the examples that follow, we use the table given by Lenstra and Verheul [LV01], which gives the minimum cryptographic key size required for security in a certain year. In particular, they give a minimum key size of 135 bits for the year 2002. In order to estimate how many iterations of a precomputation could be made per year, we assume that it would take the entire year of 2002 to break a 135 bit ECC using Pollard’s rho method. Since a 135 bit ECC is actually considered to be secure in 2002, this is a very optimistic estimation of the current computing power available. As in [LV01], we also assume that available computing power per dollar is doubling every 18 months and the available budget to spend on such computing power is doubling every ten years. To satisfy the doubling of budget and computing power requirements, we must solve:

\[
S(t) = S(0) \cdot 2^{\frac{t}{18}} \cdot 2^{\frac{t}{10}}
\]

that is,

\[
e^{\kappa_2 t} = 2^{\frac{t}{18} + \frac{t}{10}}
\]

for $\kappa_2$, giving $\kappa_2 = \frac{23}{30} \ln(2)$. To allow a 135 bit elliptic curve to be solved in 2002, we set the number of iterations in the first year, $I(1)$ (where $I(\tau)$ is defined in (5.39)) to be the number of iterations expected to be required to solve such a
curve. That is, we set

\[ I(1) = \frac{\sqrt{\pi \cdot 2^{135}}}{2} \]

that is, \( \frac{\kappa_1}{\kappa_2} \left( e^{\kappa_2 - 1} \right) = \frac{\sqrt{\pi \cdot 2^{135}}}{2} \)

\[ \therefore \kappa_1 = \frac{\frac{23}{30} \ln(2) \sqrt{\pi \cdot 2^{133}}}{2^{33} - 1} \]

\[ \approx 1.343018162 \sqrt{2^{133}} \]

Note that with this value of \( \kappa_1 \), \( t = 0 \) corresponds to the beginning of the year 2002.

For a curve size \( v \) bits, we multiply \( \kappa_1 \) by \( \frac{135^2}{v^2} \) because the modular multiplication operations used by the point addition in each iteration have a complexity proportional to the square of the number of bits in the curve. This takes account of the fact that each iteration on a curve larger than 135 bits is slower than an iteration on a 135 bit curve. The above adjustment to \( \kappa_1 \) is necessary to allow our calculations to be used in conjunction with Table 1 of [LV01], as otherwise inaccuracies occur when the bit size of the curve is significantly larger than 135 bits. (This adjustment was omitted from the ES-I calculations because its omission simplified the calculations considerably and its effect was not significant since the fixed and random curves were close in size; its inclusion would have meant one extra bit of difference between fixed and random curve orders.)

Table 5.2 shows the sizes of various fixed curves and two sizes of corresponding random curves satisfying ES-II. In order to determine the precomputation size, the last year in which the fixed curve would be considered secure was obtained from Table 1 of [LV01]. The shorter precomputation ends at the beginning of that year and starts at the beginning of 2002. This then allows one year after the completion of the precomputation to break any desired ECDLPS. The longer precomputation goes for an extra year, and finishes at the same time the curve becomes insecure. We believe that the shorter precomputation is adequate to find how many extra bits are required for a fixed curve to achieve ES-II. However, conservative users may wish to use the longer precomputation.

In order to demonstrate how the calculations in Table 5.2 were made, we perform an example calculation for a fixed curve with group order \( n \approx 2^{160} \). Since a 160 bit curve is only secure until 2019, the number of years to find the
Table 5.2: Loss in security of a fixed curve according to ES-II

<table>
<thead>
<tr>
<th>Fixed curve bit size</th>
<th>136</th>
<th>144</th>
<th>152</th>
<th>160</th>
<th>168</th>
<th>176</th>
<th>184</th>
<th>192</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last yr. secure (γ)</td>
<td>2003</td>
<td>2008</td>
<td>2014</td>
<td>2019</td>
<td>2024</td>
<td>2030</td>
<td>2035</td>
<td>2041</td>
<td>2046</td>
</tr>
<tr>
<td>Precomputation time of γ - 2002 years</td>
<td>Precomp. size</td>
<td>267.3</td>
<td>272.2</td>
<td>276.7</td>
<td>280.4</td>
<td>284.1</td>
<td>288.5</td>
<td>292.2</td>
<td>296.7</td>
</tr>
<tr>
<td>BSGS eq. rand. size</td>
<td>133.7</td>
<td>140.0</td>
<td>147.0</td>
<td>155.6</td>
<td>164.2</td>
<td>171.3</td>
<td>179.9</td>
<td>186.9</td>
<td>195.5</td>
</tr>
<tr>
<td>BSGS bits lost</td>
<td>2.3</td>
<td>4.0</td>
<td>5.0</td>
<td>4.4</td>
<td>3.8</td>
<td>4.7</td>
<td>4.1</td>
<td>5.1</td>
<td>4.5</td>
</tr>
<tr>
<td>Poll. eq. rand. size</td>
<td>134.3</td>
<td>141.3</td>
<td>148.6</td>
<td>157.0</td>
<td>165.5</td>
<td>172.8</td>
<td>181.2</td>
<td>188.5</td>
<td>196.9</td>
</tr>
<tr>
<td>Pollard bits lost</td>
<td>1.7</td>
<td>2.7</td>
<td>3.4</td>
<td>3.0</td>
<td>2.5</td>
<td>3.2</td>
<td>2.8</td>
<td>3.5</td>
<td>3.1</td>
</tr>
<tr>
<td>Precomputation time of γ + 1 - 2002 years</td>
<td>Precomp. size</td>
<td>268.7</td>
<td>273.0</td>
<td>277.5</td>
<td>281.1</td>
<td>284.8</td>
<td>289.3</td>
<td>293.0</td>
<td>297.5</td>
</tr>
<tr>
<td>BSGS eq. rand. size</td>
<td>130.9</td>
<td>138.4</td>
<td>145.4</td>
<td>154.1</td>
<td>162.3</td>
<td>169.7</td>
<td>178.3</td>
<td>186.4</td>
<td>193.9</td>
</tr>
<tr>
<td>BSGS bits lost</td>
<td>5.1</td>
<td>5.6</td>
<td>6.6</td>
<td>5.9</td>
<td>5.3</td>
<td>6.3</td>
<td>5.7</td>
<td>6.6</td>
<td>6.1</td>
</tr>
<tr>
<td>Poll. eq. rand. size</td>
<td>132.5</td>
<td>140.1</td>
<td>147.3</td>
<td>155.8</td>
<td>164.3</td>
<td>171.5</td>
<td>180.0</td>
<td>187.2</td>
<td>195.7</td>
</tr>
<tr>
<td>Pollard bits lost</td>
<td>3.5</td>
<td>3.9</td>
<td>4.7</td>
<td>4.2</td>
<td>3.7</td>
<td>4.5</td>
<td>4.0</td>
<td>4.8</td>
<td>4.3</td>
</tr>
</tbody>
</table>

The precomputation is 2019 - 2002 = 17 years. The number of iterations, $m$, in the precomputation is:

$$m = \frac{k_1}{k_2} (e^{527} - 1)$$

$$= \frac{135^2}{160^2} 1.34\sqrt{2^{133}} \frac{30}{23\ln(2)} \left( e^{\frac{2\pi}{\sqrt{3}\ln(2)} - 17} \right)$$

$$= 2^{80.38}.$$ 

For the BSGS method, any ECDL can then be found using an average of $\frac{n}{m} = 2^{77.62}$ iterations. We then set this equivalent to the number of iterations required to solve an ECDLP on a random curve for an unknown size using Pollard’s rho method (since it is faster than the BSGS method):

$$2^{77.62} = \sqrt{\pi n_R/2}$$

$$n_R = 2^{155.6}.$$ 

Therefore about five bits of security have been lost in seventeen years if the BSGS method is used to solve the ECDLP on the fixed curve. Note that while the time for each iteration on the fixed curve will be slightly different to that required on the random curve, because the curves are close in size, the effect is minimal (about 0.2 fewer bits are actually lost).

For Pollard’s rho method, we find how much loss in security occurs for the same curve and precomputation as were used for the BSGS method. We substitute
the size of the seventeen year precomputation \(2^{80.38}\) as \(r\) in (5.38) to find how many iterations are expected to remain to solve the ECDLP.

\[
E(Z_1) \approx \frac{\sqrt{2^{160}\pi}}{2} \left(2^{-160} \right)^{ \left(2^{160.76}\right)} \left(1 - \Phi \left(\frac{2^{80.38}}{2^{80}}\right) \right)
\]

\[
\approx 2^{78.34}
\]

We then ascertain the size of a random curve generated in 2019 which has equivalent security to the fixed curve (as defined by ES-II) as follows:

\[
2^{78.34} = \frac{\sqrt{\pi n_R}}{2}
\]

\[
n_R \approx 2^{157.0}
\]

Therefore, in this case, using Pollard’s rho method, only 3 bits of security are lost, as opposed to 5 bits using the BSGS method. In fact, Table 5.2 shows that using the BSGS method with a precomputation causes a greater decrease in the security of a fixed curve.

Given that Pollard’s rho method is generally accepted as being more efficient than the BSGS method on average, it may at first seem surprising that a greater loss of fixed curve security occurs due to the BSGS method than to Pollard’s rho method. This apparent contradiction can be explained by observing that the calculations assume that the first ECDLP to be solved is not known until after the calculation of the precomputation. If the first ECDLP was known before the precomputation for Pollard’s rho method began, the precomputation could be directed at solving that particular ECDLP, and it would be expected to be solved at about the same time as the precomputation in Table 5.2 finished. This is not surprising since the curve becomes insecure at the same time as this occurs. Although the final result on fixed curve security hinges on the efficiency of the BSGS method, the new results for Pollard’s rho method are still necessary to the analysis because they enable us to show that this is in fact the case.

We conclude that given the above assumptions, adding about five bits to the order of a fixed curve compared to a random curve will give approximately the same level of security as defined by ES-II when attacks are performed using the BSGS or Pollard’s rho method.
5.3.3 Curves with Special Properties

Standardized fixed curves often have special properties in order to increase the speed at which ECCs can operate. For example, the curves over $GF(p)$ specified by NIST [NIS00] use primes for which fast modular reduction algorithms are available as well as setting the $a$ parameter of the elliptic curve to $-3$ for faster point addition algorithms. On the other hand, the WAP specification [WAP01] sometimes sets the $a$ parameter to 0 for the same reason. Unfortunately, these settings mean that attacks on these curves can be performed at a faster rate also.

Using one of the generalized Mersenne primes specified by NIST as the modulus for the curve can make an implementation up to 3.7 times as fast, based on figures from [BHL01]. Even if the modular reduction took no time at all, then this would only lead to an implementation up to 4.5 times as fast. Therefore, based on the relationship between the curve size and the complexity of the BSGS and Pollard’s rho methods, adding $2 \cdot \log_2(4.5) = 4.34$ bits to the curve size would overcome this problem.

Using an $a$ parameter of $-3$ can reduce the number of squares and multiplies required for a point addition from 10 to 8, so that addition is about 1.25 times as fast. Using an $a$ parameter of 0 can reduce the number of squares and multiplies to 7 instead of 10, so that addition is about 1.43 times as fast. Increasing the curve size by $2 \cdot \log_2(1.43) = 1.03$ bits would overcome problems due to special values for the $a$ parameter.

To avoid any attacker obtaining an advantage when attacking a fixed curve with a special modulus or $a$ parameter, we suggest that an increase of 5.4 bits (1.03 + 4.34) will provide a more than adequate level of security.

5.3.4 Special Purpose Hardware

It is possible to build special purpose hardware to attack elliptic curve cryptosystems which would be considerably faster than a software attack using equipment of the same value. Lenstra and Verheul [LV01, Section 3.2.5] provide an analysis of the difference in cost between hardware and software implementations and conclude that in the elliptic curve case, for curves over the field $GF(p)$, software is more than 2,000 ≈ $2^{11}$ times more expensive than hardware.

As another example, an MPC190 security processor from Motorola [Mot03] running at 66 MHz can perform 1000 internet key exchanges (IKE) on a 155 bit
elliptic curve per second. Therefore, one scalar multiplication on such a device takes less than 1 ms. A Pentium IV 1.8 GHz machine can compute one scalar multiplication on a 160 bit curve in 2.66 ms, or about $73 \approx 2^{6.2}$ times slower taking into account the processor speed.

While some would suggest that extra bits should be added to fixed curves over $GF(p)$ to resist attacks due to special purpose hardware, we argue that hardware availability forms an equal threat to both fixed and random curves. Hardware such as the MPC190 security processor is able to perform calculations for any elliptic curve, not just a single curve. If attackers are able to invest in hardware to attack fixed curves, then that hardware can just as easily be used to attack random curves. While those who see hardware as a greater threat to fixed curves than random curves may suggest adding some extra bits to the fixed curve (22 bits based on the estimation of Lenstra and Verheul or 13-15 bits based on the MPC190 speed), we believe that such an action is unnecessary, given the equal susceptibility of fixed and random curves to hardware attacks.

5.3.5 Results of Analysis and Performance Effects

By combining the results of the previous subsections we can determine how many extra bits should be added to a fixed curve for security equivalent to a random curve. Very conservative users may wish to add a total of 38 bits for curves over $GF(p)$ (6 bits for special curve attacks and 32 bits to achieve ES-I). However, we believe a more realistic approach is to add approximately 11 bits for curves over $GF(p)$ (being 5 bits to achieve ES-II and 6 bits for special curve attacks). Of course, if the fixed curve has neither a special modulus nor a special $a$ parameter, the 6 bits for special curve attacks need not be added, and in that case only an extra 5 extra bits are required for the fixed curve.

While adding extra bits to a fixed curve does increase the time required to perform elliptic curve operations on such curves, the increase is still small enough for fixed curves to be attractive. For comparison, Table 5.3 shows timings using the MIRACL library [Sha00] for a single scalar multiplication on both a fixed and random curve. In all cases, the Comba optimization from the MIRACL library has been used which unravels and reorganizes the programme loops implicit in the field multiplication and reduction processes. The curves recommended by NIST [NIS00] were used as the fixed curves. These curves have an $a$ parameter of $-3$ and a generalized Mersenne number as the modulus, allowing a fast modular
Table 5.3: Time in milliseconds of elliptic curve scalar multiplication using the
miracl library [Sha00] on a Pentium iv 1.8 GHz

<table>
<thead>
<tr>
<th>Bit size</th>
<th>Random curve</th>
<th>Fixed curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>2.66</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>4.46</td>
<td>2.19</td>
</tr>
<tr>
<td>224</td>
<td>6.76</td>
<td>3.34</td>
</tr>
</tbody>
</table>

reduction algorithm. The table shows that a fixed curve with these properties is
still faster than a random curve 32 bits smaller than it. Therefore, if fixed curves
take advantage of the availability of special moduli and parameters, the 11 extra
bits we recommend adding to the size of a fixed curve will not have any serious
impact on performance compared to random curves. In fact, the fixed curve may
be faster than a random curve implementation not using these special features
but with an equivalent level of security. We note that in practice 32 (rather
than 11) extra bits are likely to be added to make the modulus size a multiple
of the word size of the processor being used, but that this does not change our

5.4 Conclusion

We have analysed the ECDLP on a fixed versus a random curve over GF(p) and
found that if the order of the fixed curve with special properties is 11 bits larger
than that of the random curve, an equivalent level of security is achieved if pre-
viously published attacks are used to solve the ECDLP.

We have given a lower bound on the expected value to solve more than one
ECDLP on the one curve using Pollard’s rho method, given an approximation
of the expected time to solve an ECDLP using Pollard’s rho method given that
a precomputation of a certain size already exists, and proposed two definitions
of “equivalent security.” We have given examples which support the conclusion
that approximately 32 bits should be added to the size of a fixed curve to have
equivalent security to that of a random curve using the first definition, ES-I, but
only 5 bits need to be added using the preferred definition, ES-II.

Attacks taking advantage of special purpose hardware have been considered,
but it was concluded that special purpose hardware forms an equal threat to
both fixed and random curves, implying that this attack does not require the
size of fixed curves to be increased compared to random curves. Also, attacks
taking advantage of fixed curves using a special modulus or a parameter have been investigated, and a recommendation made to add 6 bits to the size of the fixed curve to resist these attacks.

Taking all attacks into consideration, we recommend adding 11 bits to the size of a fixed curve compared to a random curve, being 6 bits to resist attacks due to a special modulus or a parameter, and 5 bits to achieve ES-II. However, if the fixed curve does not have a special modulus or a parameter, the addition of only 5 bits to the curve order is necessary. These results show that there is no security problem associated with the use of fixed curves, provided the order of the fixed curve is increased by a small amount. Such an increase in the size of the fixed curve has a minimal performance impact, whilst allowing realization of the many benefits associated with the use of fixed curves.
Chapter 6

Provably Secure Elliptic Curve Protocols

A major goal of modern cryptography is to enable two or more users on an insecure (adversary controlled) network to communicate in a confidential manner and/or ensure that such communications are authentic. In order to realize this goal, symmetric key cryptographic tools are often used due to their efficiency compared to public key techniques. However, use of such tools requires the creation of a secret key (which is typically at least 100 bits long) known only to the users communicating with each other. Because of the impracticality of each possible pair of users sharing a long term secret key, public key and/or password-based techniques are used to generate such a key when it is required. An advantage of this method of key generation is to keep different sessions independent, which enables the avoiding of replay attacks (since the wrong key will have been used for the replay) and lessens the impact of key compromise (since only one session will be exposed, not all previous communications).

In order to provide assurance of the security of key exchange protocols, a formal proof of security can be used. This chapter uses the Canetti-Krawczyk proof model for this purpose, since it offers the advantage of a modular proof with reusable components. The first part of the chapter provides a proof of security for a password-based authentication mechanism, followed by an analysis of its use in key exchange protocols. The second part of the chapter uses the Canetti-Krawczyk model to deal with secure tripartite (three party) key exchange.
protocols. Tripartite key exchange protocols are particularly suited to ECCs because of the availability of bilinear mappings on elliptic curves which allow more efficient tripartite key exchange protocols.

The chapter begins with a description of the Canetti-Krawczyk proof model, followed by a formal definition of a password-based authenticator and its proof of security. The details of how the authenticator can be used in various key exchange protocols are then provided, followed by a comparison of the performance of those protocols with the ones using a similar password-based authentication mechanism by Halevi and Krawczyk. Tripartite key exchange is then discussed, beginning with a definition of security of tripartite key exchange protocols in the Canetti-Krawczyk model. A tripartite key exchange protocol is then proven secure in this model and combined with a signature based and an encryption based authentication mechanism. Finally, the efficiency of various implementation options for the tripartite protocols is explored.

6.1 Overview of the Canetti-Krawczyk Approach

In the past, a trial and error approach to the security of cryptographic protocols has been taken where cryptographic protocols were proposed together with informal security analysis. However, such protocols were sometimes wrong and usually only partially analysable (see for example the flaws documented by Lowe [Low96]). Indeed, in some cases, flaws have come to light years after a protocol’s proposal and acceptance by the community as being secure.

These problems have led to the development of various formal methods of proving the security of a protocol. Bellare and Rogaway [BR93, BR95] first proposed a formal model for proving the security of protocols. Although their initial model only covered the case where two parties already share a long-term secret, it has been extended, by themselves and others, to cover all the main types of authenticated key exchange (AKE) protocol. The proofs follow the style of most proofs in modern cryptography by reducing the security of the protocol to the security of some underlying primitive. A limitation of these proofs is that they tend to be complex and difficult for practitioners to design and use. Even more important from our viewpoint is that they are monolithic, fragile and error prone [JP01, WC01]. A small change in the protocol structure can destroy the proof and leave no indication of how to repair it.
6.1. Overview of the Canetti-Krawczyk Approach

Fig. 6.1: Graphical representation of an authenticator

This paper works in the model adopted by Canetti and Krawczyk [CK01a] which we refer to hereafter as the ck-model. Two previous works of Bellare and Rogaway [BR93] and Bellare, Canetti and Krawczyk [BCK98] form the basis of the ck-model. The former uses the indistinguishability of [GM84] for defining security while the latter postulates a two-step methodology for a substantial simplification in designing provably secure cryptographic protocols. As a consequence, the ck-model inherits the aforementioned properties of [BR93, BCK98]. Its modularity is gained by applying a protocol translation tool, called an authenticator, to protocols proven secure in a much simplified adversarial setting where authentication of the communication links is not required. This is shown graphically in Figure 6.1. The result of such an application is secure protocols in the unsimplified adversarial setting where the full capabilities of the adversary are modelled. Moreover, various basic parts of a protocol can be proven secure independently of each other and then combined to form a single secure protocol. This leads to simpler, less error-prone proofs and the ability to construct a large number of secure protocols from a much smaller number of basic secure components.

Here a description of the ck-model is given. Further details can be found in [BCK98] and [CK01a]. The ck-model defines protocol principals who may simultaneously run multiple local copies of a message driven protocol. Each local copy is called a session and has its own local state. A powerful adversary attempts to break the protocol by interacting with the principals. The adversary controls all communications between principals. However, the adversary must be efficient
in the sense of being a probabilistic polynomial time algorithm. The model uses the following terminologies:

**Matching** Two sessions are matching if each session has the same session identifier and the purpose of each session is to communicate between the particular two parties running the sessions.

**Partner** A partner to a session is a party running a matching session.

**Session-state reveal** In a session-state reveal, the internal state of the session within the party running the session is revealed to the adversary, a special message is added to the party’s output to indicate the occurrence of the session-state reveal and the session produces no more output.

**Session-key query** By performing a session-key query, the adversary learns the secret key output by the session.

**Corruption** The adversary is able to corrupt any principal, thereby learning all information in the memory of that principal (e.g. long-term keys, session states and session keys). The adversary may impersonate a corrupted principal, although the corrupted principal itself is not activated again and produces no further output or messages.

**Completed session** A completed session is a session which has ended its run and whose local state is erased.

**Expired session** An expired session is a session whose session key and any related session state has been erased from the memory of the party running the session. The adversary can not perform a session-key query on an expired session.

**Locally exposed session** A session is locally exposed if the adversary has performed a session-state reveal or a session-key query on the session, or has corrupted the party owning the session before the session was expired within the party.

**Exposed session** An exposed session is one which is either locally exposed or has a matching locally exposed session.
Test-session query As well as its other activities, the adversary may choose one session as the test-session at any time during its run. This session must be completed, unexpired and unexposed at the time. A random value $b \overset{R}{\leftarrow} \{0, 1\}$ is chosen. If $b = 0$ then the adversary is provided with the actual value of the session-key, $\kappa$. Otherwise the adversary is provided with a value randomly chosen from the distribution of possible session keys for the protocol. The adversary can then continue with its normal execution, but can not expose the test-session. (The adversary can not perform session-state reveals, session-key queries or partner’s corruption on the test-session or any matching session. However, as soon as the test-session expires at a partner, the restriction on that partner’s corruption is lifted and the partner can in fact be corrupted.)

An informal definition of session key security can now be provided. A formal definition is provided later in this section.

**Definition 6.1 (Informal).** An AKE protocol is called session key (SK-) secure if the following two conditions are met. Firstly, if two uncorrupted parties complete matching sessions, then they both accept the same key. Secondly, suppose a session key is exchanged between two uncorrupted parties and has not been revealed by the adversary. Then the adversary cannot distinguish the key from a random string with probability greater than $1/2$ plus a negligible function in the security parameter.

Another way of stating the restriction on the adversary’s probability of distinguishing between a key and a random string is to say that if the adversary is given either the key or a random string, each with probability $1/2$, the adversary cannot correctly determine which one it received with a probability greater than $1/2$ plus a negligible function in the security parameter. (That is, it can do no better than randomly guess which one it received.) This requirement is somewhat similar to the indistinguishability requirement of an encryption scheme (see Appendix A.1). However, the definition of indistinguishability of an encryption scheme goes a step further to define the advantage of an encryption scheme in such a way that it should be negligible for secure schemes, whereas here the restriction is placed directly on the probability of guessing correctly, not on the adversary’s advantage.

Two adversarial models are defined: the unauthenticated-links adversarial model (UM) and the authenticated-links adversarial model (AM). The only differ-
ence between the two is the amount of control the adversary has over the communications lines between principals. The UM corresponds to the “real world” where the adversary completely controls the network in use, and may modify or create messages from any party to any other party. The AM is a restricted version of the UM where the adversary may choose whether or not to deliver a message, but if a message is delivered, it must have been created by the specified sender and be delivered to the specified recipient without alteration. In addition, any such message may only be delivered once. In this way, authentication mechanisms can be separated from key exchange mechanisms by proving the key exchange secure in the AM, and then applying an authentication mechanism to the key exchange messages so that the overall protocol is secure in the UM. The definition of SK-security in the UM must be relaxed slightly when password-based authentication mechanisms are used, which is addressed in Section 6.2.3. The formal definition of session key security is given in Definition 6.2, followed by an informal definition of an authenticator in Definition 6.3.

**Definition 6.2 (Formal [CK01a]).** A KE protocol $\pi$ is called session key (SK-) secure in the AM (respectively UM) if the following two properties hold for any adversary $A$ (respectively $U$) in the AM (respectively UM).

1. Protocol $\pi$ satisfies the property that if two uncorrupted parties complete matching sessions then they both output the same key.

2. The probability that $A$ (respectively $U$) guesses correctly the bit $b$ from the test-session (i.e. outputs $b' = b$) is no more than $1/2$ plus a negligible function in the security parameter.

**Definition 6.3 (Informal).** An authenticator is a protocol translator that takes an SK-secure protocol in the AM to an SK-secure protocol in the UM.

Authenticators can be constructed using one or more message transmission (MT-) authenticators. An MT-authenticator is a protocol which delivers one message in the UM in an authenticated manner. To translate an SK-secure protocol in the AM to an SK-secure protocol in the UM an MT-authenticator can be applied to each message and the resultant sub-protocols combined to form one overall SK-secure protocol in the UM.

If the SK-secure protocol in the AM consists of more than one message, the resultant protocol can be optimized to reduce the number and size of messages,
involving reorder and reuse of message components. Such an optimization process was performed by Canetti and Krawczyk [CK01a] when they illustrated the use of their model by applying a signature based authenticator to Diffie-Hellman key exchange. Their optimization involved using messages from the Diffie-Hellman key exchange as the random nonces required by the authenticator and “piggy-backing” or joining the common flows. However, use of the Diffie-Hellman messages as nonces for the authenticator creates the situation where the second message of the protocol is created before the first message has been accepted as authentic (because the messages must essentially be reordered to enable the Diffie-Hellman messages to be used as nonces). Whilst such a scenario does not pose a problem for Diffie-Hellman key exchange (or any of the other protocols described in this chapter), it is worth noting that in some protocols, this could constitute a security flaw.

In order to define the security of an MT-authenticator, it is necessary to formally define an MT-protocol in the AM as follows.

**Definition 6.4 (MT-Protocol).** An MT-protocol in the AM is one with the following properties:

- Upon activation within party A on external request \((B, m)\), party A sends the message \((A, B, m)\) to party B and outputs ‘A sent \(m\) to B.’

- Upon receipt of a message \((A, B, m)\), B outputs ‘B received \(m\) from A.’

An MT-authenticator is defined to be secure if it emulates an MT-protocol in the UM. Emulation is defined to occur when the global output (which consists of the concatenation of the cumulative output of all parties and the output of the adversary) in the AM is computationally indistinguishable from the global output in the UM. (The output of the adversary is a function of its internal states at the end of the interaction.) Note that in the UM, it is necessary to augment a protocol with an initialization function to allow the required bootstrapping of the cryptographic authentication functions. The CK-approach can now be summarized in the following three steps.

**CK1** Design a basic protocol and prove it SK-secure in the AM.

**CK2** Design an MT-authenticator and prove that it is secure.
Chapter 6. Provably Secure Elliptic Curve Protocols

CK3 Apply the $mt$-authenticator to the $AM$ protocol to produce an automatically secure $UM$ protocol. If necessary, reorder and reuse message components to optimize the resulting protocol.

6.2 Password-Based Protocols Secure in the CK Model

While public key cryptography can be used to provide a secure method of key exchange, it is not always practical to use it exclusively due to the inconvenience and expense of securely storing full length cryptographic keys in some applications. Secure password based key exchange mechanisms (where the only secret information held by one or more of the parties is a short password) are necessary in such environments. One such example is mobile environments, where memory may be scarce and the devices in use not secure. However, because of the short length of the password, special care must be taken when designing protocols to ensure that both the password and the key finally exchanged are secret.

Although password-based protocols that have been proven secure in other proof models do exist, such as those in [KOY01] and [Mac02], they do not use a public key for the server and therefore are not amenable to the CK-model, since it is impossible to separate the key exchange and authentication mechanisms of such protocols [HK99]. Thus the advantages of the modular approach used by the CK-model can not be realized by these protocols, making the proof of security of a password-based authenticator and demonstration of its application in the CK-model worthwhile.

6.2.1 Password-Based Protocols and Their Constraints

Because of the difficulty humans have remembering secrets even of a length of only 7 or 8 characters [FK89, Wu99], passwords have a very small amount of entropy. For example, if all upper and lower case letters as well as the digits 0 to 9 are used in an 8 character password, only 48 bits of entropy are possible. This means that it is possible for an attacker to test all possible passwords in a relatively short amount of time and leads to the requirement that off-line dictionary attacks must be infeasible (that is, an adversary with access to transcripts of one or more sessions must not be able to eliminate a significant number of possible
passwords). In addition, on-line dictionary attacks must be infeasible (that is, an active adversary must not be able to abuse the protocol in a way that allows him to eliminate a significant number of possible passwords). Note that in an on-line attack, the adversary can guess a password and attempt to impersonate the user, so at least one password can be eliminated per protocol run. We require that no more than one password can be eliminated per protocol run with non-negligible probability. Other more general security properties required are key authentication (parties participating in the key exchange know the identity of all other parties who could possibly hold a copy of the key [MvOV96, p.490]) and key freshness [MvOV96, p.494] (key freshness is necessary since it is assumed the adversary is able to find the value of old keys). It is also desirable that the protocol be efficient in terms of the number of operations performed and the number of messages transmitted.

The main focus of this section is on the security and application of a password based authentication mechanism (called the Encrypted Challenge-Response protocol) which was proposed by Halevi and Krawczyk [HK99] and used in their key exchange protocols. While these key exchange protocols do not have an associated proof of security, the authentication mechanism itself does. However, Boyarsky [Boy99] has criticized both this proof and an earlier version [HK98] due to an inadequate definition of security. In addition, while Halevi and Krawczyk state that their security formalization and proof provide a basis for a password-based equivalent of an authenticator in the ck-model, the proof they provide can not be used for this purpose for a couple of reasons. Firstly, the proof assumes that there is only one uncorrupted party (in addition to the server) in existence. A valid proof for the ck-model must allow any (polynomially bounded) number of uncorrupted parties. Secondly, the protocol does not actually enable the transmission of an authenticated message, as required by a proper authenticator in the ck-model; the only achievement is that the server knows the client responded to the server’s nonce.

In order to overcome these problems, a formal description of an mt-authenticator based on the work of Halevi and Krawczyk is provided here, as well as a proof of its security in the ck-model. This mt-authenticator can be used for constructing authenticators which transform protocols in the simplified adversarial setting to secure protocols in the full adversarial setting, as described in [BCK98]. As a demonstration of the applications of our new mt-authenticator,
it is applied to two protocols proposed in the literature for the simplified adversarial setting, thereby obtaining two secure key exchange protocols using passwords. Whilst it is possible to obtain other password-based key exchange protocols using this method, the two presented in this paper are the most efficient that can be built from currently available components proven secure in the CK-model. Finally, the new password-based key exchange protocols are compared with those proposed by Halevi and Krawczyk [HK99] for performance and efficiency.

### 6.2.2 The Authenticator

The Encrypted Challenge-Response protocol of Halevi and Krawczyk [HK99] shown by Protocol 6.1 is designed to authenticate a client to a server and is based on a randomized encryption of a shared password. It requires a function $F$ such that with fixed input strings $\pi, x$, the induced functions $F(\pi; \cdot)$ and $F(\cdot; x)$ are one-to-one. For example, the concatenation function $F(x; y) = (x \parallel y)$ satisfies the requirement.

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>$B$’s public key, $e_B$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td>Pick random nonce $N_B$</td>
<td></td>
</tr>
</tbody>
</table>

\[
c = E_{e_B}(F(\pi; A, B, N_B))
\]

\[
\frac{N_B}{A, N_B, c}
\]

\[
v = D_{d_B}(c)
\]

Check $v = F(\pi; A, B, N_B)$

**Protocol 6.1:** Halevi and Krawczyk’s Encrypted Challenge-Response protocol

The only achievement of Protocol 6.1 is to provide assurance to the server that the client responded to its challenge. It does not provide authenticated message transmission because it does not deliver any message from the client to the server. Although Halevi and Krawczyk have slightly modified the mechanism so that they can use it to provide authentication in their key agreement protocols, there is no formal proof that the key exchange protocols are secure or that messages from client to server are properly authenticated, even if this intuitively seems to be the case. Therefore, the Encrypted Challenge-Response protocol has been modified here to form a new mt-authenticator for messages from a client to
6.2. Password-Based Protocols Secure in the CK Model

a server and its security level is discussed in Section 6.2.3. The modifications
give the authenticator a similar format to that of the authenticators proposed
in [BCK98]. The authenticator is denoted by $\lambda_{p\text{-}enc}$ and Protocol 6.2 gives its
specification. The following subsections give a detailed description of various
aspects of the authenticator.

<table>
<thead>
<tr>
<th>$A$ (Client)</th>
<th>$B$ (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password, $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Public key of $B$, $e_B$</td>
<td>Public key, $e_B$</td>
</tr>
<tr>
<td>Unique message, $m$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td>$c = E_{e_B}(m, N_B, A, \pi)$</td>
<td>$N_B \in_R {0,1}^k$</td>
</tr>
<tr>
<td>$m, N_B \rightarrow c$</td>
<td>$m, c \rightarrow v = D_{d_B}(c)$</td>
</tr>
<tr>
<td>$v \equiv (m, N_B, A, \pi)$</td>
<td></td>
</tr>
</tbody>
</table>

**Protocol 6.2:** Password based authenticator, $\lambda_{p\text{-}enc}$

As for the signature authenticator in [BCK98], it would be possible to omit
the message $m$ from some of the protocol flows. In particular, the first protocol
flow only needs to notify $B$ that there is a message waiting for delivery. The
second flow does not need to carry the entire message, but does need some unique
identifier which is able to bind $N_B$ to $m$, so that $A$ can determine the session
to which $N_B$ belongs. (Since all messages are unique in the CK-model, it is
possible to use the message itself for this purpose.) The message which is outside
the encryption in the third flow can also be omitted. However, its inclusion is
intended to enable $B$ to easily determine to which session the message belongs.
(Of course, the message inside the encryption cannot be deleted.)

### 6.2.2.1 The Parties

Let there be a total of $n$ parties, $P_1, \ldots, P_n$. We split the parties into two disjoint
sets, the set of all servers of size $s$ and the set of all clients of size $c$ where
$(c + s) = n$. The values $n$, $c$ and $s$ are polynomial in the security parameter $k$
and may be written as $n(k)$, $c(k)$ and $s(k)$ respectively.
6.2.2.2 Initialization

The initialization function $I_{\text{p-enc}}$ consists of two parts, $I_{\text{pwd}}$ and $I_{\text{pubKey}}$. $I_{\text{pwd}}$ invokes a password generation algorithm once for each pair of (client, server) parties ($C_i, S_j$). This algorithm randomly chooses a password, denoted $\pi_{C_i,S_j}$ (or simply $\pi$ when the password is shared between $A$ and $B$), from a password dictionary, $\mathcal{D}$, and assigns the password to $C_i$ and $S_j$.

$I_{\text{pubKey}}$ invokes, once for each server party $S_j$, the key generation algorithm of an asymmetric randomized encryption scheme which is indistinguishable under adaptive chosen ciphertext attack (also known as chosen ciphertext postprocessing attack [BDPR98, DDN01]) with security parameter $k$. Let $\mathcal{E}$ and $\mathcal{D}$ denote the encryption and decryption algorithms. (Note that $\mathcal{E}$ is a randomized encryption since otherwise the attacker could just try the encryption for different values of $\pi_{C_i,S_j}$ and see which one matched the ciphertext sent from $C_i$ to $S_j$ in the third protocol flow in Protocol 6.2.) Let $e_{S_j}$ and $d_{S_j}$ denote the encryption and decryption keys associated with server party $S_j$. The public keys, $e_{S_j}$, are distributed to all client parties $C_i$.

The public information is all public keys: $I_0 = \{e_{S_1}, \ldots, e_{S_s}\}$. The (completely) private information of each server party $S_j$ is the private key $d_{S_j}$. Shared private information for each $S_j$ is the set of passwords shared by $S_j$ with each $C_i$, $\{\pi_{C_1,S_j}, \ldots, \pi_{C_s,S_j}\}$. Shared private information for each client $C_i$ is the set of passwords shared by $C_i$ with each server $S_j$, $\{\pi_{C_i,S_1}, \ldots, \pi_{C_i,S_s}\}$.

6.2.2.3 Protocol Description

Since the majority of the following discussion is for only one client and server pair, we denote the client by $A$ and the server by $B$ for simplicity. The protocol begins with the initialization function $I_{\text{p-enc}}$ described above. Each server then sets the number of unsuccessful attempts to complete the protocol with each client to 0. The protocol then proceeds according to the following rules:

- If a client $A$ is activated with an external request to send a unique message $m$ to server $B$, then $A$ outputs ‘$A$ sent message $m$ to $B$’ and sends ‘message: $m$’ to $B$.

- Upon receipt of ‘message: $m$’ from $A$, $B$ chooses a random value $N_B \in \{0,1\}^k$ and sends ‘challenge: $m, N_B$’ to $A$.
Upon receipt of ‘challenge: \(m, N_B\)’ from \(B\), client \(A\) sends ‘encryption: \(m, E_{e_B}(m, N_B, A, \pi)\)’ to \(B\).

Upon receipt of ‘encryption: \(m, E_{e_B}(m, N_B, A, \pi)\)’ from \(A\), party \(B\) accepts \(m\) from \(A\) if, when decrypted, \(m\) is the same as the cleartext \(m\), \(A\) is a valid client, \(\pi\) is the password shared between \(A\) and \(B\), \(B\) has previously sent ‘challenge: \(m, N_B\)’ (where \(m\) and \(N_B\) match those in the encryption and the challenge is still outstanding) and the number of unsuccessful attempts to complete the protocol with \(A\) is less than or equal to \(\gamma\). If \(B\) accepts \(m\) from \(A\) then \(B\) removes ‘challenge: \(m, N_B\)’ from the list of outstanding challenges and outputs ‘\(B\) received \(m\) from \(A\)’. If the value \(A\) in the encryption is the identity of a valid client, but \(B\) does not accept the message \(m\) from \(A\), then \(B\) increases the number of unsuccessful attempts to complete the protocol with \(A\) by one. Otherwise, if the identity in the encryption is not a valid client, \(B\) does nothing.

### 6.2.3 Proof of Security of Authenticator

Various notational conventions and definitions of security of an encryption scheme which are commonly used in the literature are provided in Appendix A. It is assumed the reader is familiar with these notations and definitions of security.

In order to complete the proof of security of the password-based authenticator, it is necessary to use the following lemma.

**Lemma 6.1.** Let \(p\) be a positive integer and let \(a\) and \(b\) be real numbers. Then the following inequality holds:

\[
pa^{p-1}b \geq a^p - (a - b)^p, \quad \text{for } b \leq \frac{a}{p}.
\]  

(6.1)

**Proof.** We know

\[
a^p - (a - b)^p = pa^{p-1}b + \sum_{i=2}^{p} \binom{p}{i} (-1)^{i-1} a^{p-i}b^i
\]

and using this equality, (6.1) can be rewritten as:

\[
0 \geq \sum_{i=2}^{p} \binom{p}{i} (-1)^{i-1} a^{p-i}b^i.
\]  

(6.2)
We know that (6.2) will hold when:

\[
\left( \frac{p}{2i} \right)^{a^{p-2i}b^{2i}} \geq \left( \frac{p}{2i+1} \right)^{a^{p-2i-1}b^{2i+1}}, \quad \text{for } i \in \mathbb{Z}, \ 1 \leq i \leq \left\lfloor \frac{p-1}{2} \right\rfloor.
\]

Rearranging, we obtain:

\[
1 \geq \frac{\left( \frac{p}{2i+1} \right)^{a}}{\left( \frac{p}{2i} \right)^{b}}.
\]

\[
\therefore 1 \geq \frac{(p-2i)b}{(2i+1)a}, \quad \text{for all } i \text{ such that } 1 \leq i \leq \left\lfloor \frac{p-1}{2} \right\rfloor.
\]

Since \( p \geq \frac{(p-2i)}{(2i+1)} \), (6.1) is satisfied when:

\[
b \leq \frac{a}{p}.
\]

\[\square\]

Given certain assumptions on the security of the encryption scheme, the maximum number of unsuccessful attempts per client to complete \( \lambda_{p-\text{enc}} \) and the size of the password dictionary, the following theorem states the level of security achieved by \( \lambda_{p-\text{enc}} \).

**Theorem 6.1.** Assume that the encryption scheme in use is secure in the sense of \( \text{IND-CCA} \) (that is, the advantage of an adversary \( \mathcal{F} \) against the encryption scheme \( \Pi, \ Adv_{\Pi,\mathcal{F}}^{\text{ind-CCA}}(k) \), is negligible when the time complexity of \( \mathcal{F} \) is polynomially bounded). Also assume that \( \gamma \) is the maximum number of unsuccessful attempts per client to complete the protocol with a server, and the passwords are randomly chosen from a dictionary \( \mathcal{D} \) of size \( |\mathcal{D}| \). Then the output of protocol \( \lambda_{p-\text{enc}} \) is the same as that of an MT-protocol in unauthenticated networks with probability \( (1 - \epsilon(k)) \) where:

\[
\epsilon(k) \leq \left( \epsilon_0(k) + \left( 1 - \frac{\gamma + 1}{|\mathcal{D}|} \right)^{s(k)c(k)} \right).
\]  

(6.3)
and \( \epsilon_0(k) \) is negligible.

Because the protocol \( \lambda_{p^{\text{ENC}}} \) is password based, this is the best security level we can hope to achieve. That is, the probability that \( \lambda_{p^{\text{ENC}}} \) in the UM is different to the MT protocol in the AM is no more than a negligible function plus the probability that the adversary randomly but correctly guesses at least one password. In practice \( \lambda_{p^{\text{ENC}}} \) does not achieve ‘emulation’ in the sense of [BCK98] due to the small size of the dictionary which makes \( \epsilon(k) \) non-negligible. An adversary can always guess a password and attempt to use it in \( \lambda_{p^{\text{ENC}}} \). If the attempt to complete the protocol is unsuccessful, the adversary can eliminate that password from the list of possible passwords for the user it attempted to impersonate. However, we show that the probability of the adversary doing better than this is negligible.

Because the probability that the output of \( \lambda_{p^{\text{ENC}}} \) is different to that of MT is not negligible, \( \lambda_{p^{\text{ENC}}} \) can not be used to create sk-secure protocols using the original definition of sk-security. However, the definition can be modified so that the probability of correctly distinguishing the key from a random string is \( 1/2 + \delta \) where \( \delta \) is no longer negligible, but no more than a negligible function plus half of the probability of randomly guessing at least one password. In our case, \( \delta = \epsilon(k)/2 + \omega(k) \) where \( \epsilon(k) \) is defined above and \( \omega(k) \) is negligible. This can be shown by letting \( D \) be the event that the adversary guesses the session key correctly and \( E \) be the event that the output of the key exchange protocol in the UM is identical to the output of the key exchange protocol in the AM. Then:

\[
\Pr(D) = \Pr(D|E)\Pr(E) + \Pr(D|\neg E)\Pr(\neg E) \\
\leq (1/2 + \omega_1(k))(1 - \epsilon(k)) + 1 \cdot \epsilon(k) \\
= 1/2 + \epsilon(k)/2 + \omega(k), \text{ where } \omega(k) \text{ and } \omega_1(k) \text{ are negligible.}
\]

We call protocols that satisfy the new definition of security \textit{password-based session key (PBSK-) secure}.

Now that the lower level of security provided by \( \lambda_{p^{\text{ENC}}} \) compared to other MT-authenticators has been justified, the proof of Theorem 6.1 is shown:

\textit{Proof}. Let \( \mathcal{U} \) be a UM-adversary that interacts with \( \lambda_{p^{\text{ENC}}} \). We construct an AM-adversary \( \mathcal{A} \) such that the output in the UM and AM is identical with probability \( 1 - \epsilon(k) \). Adversary \( \mathcal{A} \) runs \( \mathcal{U} \) on the following simulated interaction with a set of parties running \( \lambda_{p^{\text{ENC}}} \).

1. First \( \mathcal{A} \) chooses and distributes keys for the imitated parties, according
to function $I_{\text{P-enc}}$.

2. Next, when $\mathcal{U}$ activates some imitated client party $A'$ for sending a message $m$ to imitated server party $B'$, adversary $\mathcal{A}$ activates client party $A$ in the authenticated network to send $m$ to server $B$. In addition, $\mathcal{A}$ continues the interaction between $\mathcal{U}$ and the imitated parties running $\lambda_{\text{P-enc}}$.

3. When some imitated party $B'$ outputs `$B'$ received $\hat{m}$ from $A'$', adversary $\mathcal{A}$ activates party $B$ in the authenticated-links model with incoming message $\hat{m}$ from $A$.

4. When $\mathcal{U}$ corrupts a party, $\mathcal{A}$ corrupts the same party in the authenticated network and hands the corresponding information (from the simulated run) to $\mathcal{U}$.

5. Finally, $\mathcal{A}$ outputs whatever $\mathcal{U}$ outputs.

We first need to show that the above description of the behaviour of $\mathcal{A}$ is a legitimate behaviour of an $\lambda\text{-adversary}$. The above steps are easy to verify as legal moves for $\mathcal{A}$, except for Step 3. In that case, let $\mathcal{B}$ denote the event that imitated party $B'$ outputs `$B'$ received $\hat{m}$ from $A'$' where $A'$ is uncorrupted and the message $(\hat{m}, A, B)$ is not currently in the set $U$ of undelivered messages. In other words, $\mathcal{B}$ is the event where $B'$ outputs `$B'$ received $\hat{m}$ from $A'$', and either $A$ was not activated for sending $\hat{m}$ to $B$ or $B$ has already had the same output before. In this event we say that $\mathcal{U}$ broke party $A'$.

If $\mathcal{B}$ does not occur (that is, Step 3 can always (legally) be carried out), then the above construction is as required. It remains to show that event $\mathcal{B}$ occurs only with low probability. Assume that event $\mathcal{B}$ occurs with probability $\gamma(k)$.

There are a number of ways in which $\mathcal{B}$ could occur. Firstly, $B$ could output the same nonce twice, coupled with the same message. However, the probability of this occurring is $\epsilon(k) = 2^{-k}$, which is a negligible function in the security parameter, $k$.

Obviously $\mathcal{U}$ can attempt to send a message as if from $A$ by guessing the password and including the guess in the final message of the protocol. If a maximum of $\gamma$ unsuccessful login attempts are allowed for each client, then $\mathcal{U}$ has a probability of at most $\frac{\gamma+1}{|\mathcal{B}|}$ of succeeding for one particular client and server pair without obtaining any information about the password (apart from the contents
of $D$). Therefore $U$ has probability at most $\left(1 - \left(1 - \frac{\epsilon_1(k)}{2^k}\right)^{s(k)c(k)}\right)$ of succeeding for at least one client and server pair. Then we show that the probability that $B$ occurs is negligibly higher than this. That is, if $B$ occurs with probability $\epsilon(k)$ and the function $\epsilon_2(k) = \epsilon(k) - \epsilon_1(k) - \left(1 - \left(1 - \frac{\epsilon_1(k)}{2^k}\right)^{s(k)c(k)}\right)$ is not negligible, then we show that the advantage $\text{Adv}^{\text{ind-cca}}_{\Pi,F}(k)$ associated with the encryption scheme for a polynomial time adversary $F$ is not negligible, which contradicts the assumption that the encryption scheme is secure.

Let $\Pi$ denote the encryption scheme in use. As noted in Appendix A.3, since the advantage of $F$ attacking the indistinguishability of the cryptosystem, $\text{Adv}^{\text{ind-cca}}_{\Pi,F}(k)$, is negligible, so is the advantage of an adversary $F$ attacking the left-or-right indistinguishability of the cryptosystem, $\text{Adv}^{\text{lor-cca}}_{\Pi,F}(k)$. (An adversary attacking left-or-right indistinguishability is provided with an oracle which always returns the encryption of either the left or right of a pair of input plaintexts. The adversary must guess whether the oracle encrypts the left or right plaintext. Indistinguishability is a special case of left-or-right indistinguishability where the adversary may only make one such oracle query.)

From this point the proof is similar to that of Theorem 5.3 in [AB01]. However, a few modifications are required for this particular situation. Let $F$ be an adversary having polynomial time complexity, and attacking LOR-CCA of $\Pi$. Given an encryption key $pk$, a left-or-right encryption oracle $\mathcal{E}_{pk}(\mathcal{L}R(\cdot, b))$ and a decryption oracle $D_{sk}(\cdot)$, adversary $F$ runs $U$ on the following simulated interaction with a set of parties running $\lambda_{P-ENC}$.

1. First $F$ chooses and distributes keys for the imitated parties according to function $I_{P-ENC}$ with the exception that the public encryption key associated with some server party $B^*$, chosen at random from the set of servers $S$, is replaced with the input key $pk$. $A^*$ is chosen at random from the set of clients $C$. Note that $F$ knows the password shared between $A^*$ and $B^*$, $\pi$.

2. If party $A^*$ or party $B^*$ is corrupted then the simulation is aborted and $F$ fails.

3. If $U$ activates any parties other than $A^*$ or $B^*$ to do anything, then $F$ has the necessary keys and acts according to protocol $\lambda_{P-ENC}$.

4. If $A^*$ is activated by $U$ to send the first message of the protocol $\lambda_{P-ENC}$ for
the message $m$ to any server party $R$, then $A^*$ outputs ‘$A^*$ sent message $m$ to $R$’ and sends ‘message: $m$’ to $R$.

5. If party $A^*$ is activated by $U$ to send the third message of the protocol $\lambda_{\text{enc}}$ for the message $m$ (where $A^*$ has previously output ‘$A^*$ sent message $m$ to $R$’) and nonce $N_R$ of the server $R$, where $R$ is not $B^*$ then $\mathcal{F}$ finds the necessary encryption and sends ‘encryption: $m, \mathcal{E}_{e_R}(m, N_R, A^*, \pi_{AR})$’ where the public key of $R$ is $e_R$ and $\pi_{AR}$ is the password shared between $A^*$ and $R$.

6. If party $A^*$ is activated by $U$ to send the third message of the protocol $\lambda_{\text{enc}}$ for the message $m$ and nonce $N_B$ of the server $B^*$ (where $A^*$ has previously output ‘$A^*$ sent message $m$ to $B^*$’), then $\mathcal{F}$ queries the encryption oracle with $\mathcal{E}_{pk}(\mathcal{L}R((m \parallel N_B \parallel A^* \parallel r), (m \parallel N_B \parallel A^* \parallel \pi), b))$ and receives output $C$, where $r$ is newly chosen for each oracle query and $r \sim_{\mathcal{D}}$. $\mathcal{F}$ then sends ‘encryption: $m, C$’ to $B^*$.

7. If $B^*$ is activated by $U$ to respond to ‘message: $\tilde{m}$’ with the second message of the protocol $\lambda_{\text{enc}}$, then $\mathcal{F}$ randomly generates $\tilde{N}_{B^*}$ and causes $B^*$ to respond with ‘challenge: $(\tilde{m}, \tilde{N}_{B^*})$’.

8. If $U$ activates $B^*$ with ‘encryption: $m, C$’ where $C$ is the output of the encryption oracle, and when the encryption oracle was queried, the corresponding plaintexts were $(m \parallel N_B \parallel A^* \parallel r)$ and $(m \parallel N_B \parallel A^* \parallel \pi)$ and $B^*$ had previously sent ‘challenge: $(m, N_B)$’ (and the challenge is still outstanding) then $B^*$ outputs ‘$B^*$ received $m$ from $A^*$’. 

9. If $U$ activates $B^*$ with ‘encryption: $m, C$’ where $C$ is not an output of the encryption oracle, $\mathcal{F}$ queries its decryption oracle and finds $p \leftarrow \mathcal{D}_{sk}(C)$. If $p$ is of the form $(m \parallel N_B \parallel P \parallel \pi_{PB})$ where $P$ is the identity of a party (possibly $A^*$) and $\pi_{PB}$ is the password shared between $P$ and $B^*$, and $B^*$ had previously sent ‘challenge: $(m, N_B)$’ (and the challenge is still outstanding) then $B^*$ outputs ‘$B^*$ received $m$ from $P$’ and removes the challenge from the list of outstanding challenges. If $P$ is actually $A^*$ then if the attempt was successful (that is, $B^*$ output the “received” message), then $\mathcal{F}$ guesses that the bit $b$ associated with the $\mathcal{E}_{pk}(\mathcal{L}R(\cdot, \cdot, b))$ oracle is 1. If the attempt was unsuccessful, $\mathcal{F}$ keeps a running total of the number of unsuccessful attempts to complete the protocol for $P$ and allows a maximum
of \( \gamma \) attempts for each client. (That is, after the \( \gamma+1 \)th unsuccessful attempt to complete the protocol purportedly from \( P \), \( B^* \) will no longer accept any message from \( P \).) If \( \gamma + 1 \) attempts have been made for \( A^* \) or \( U \) finishes (and there was no successful attempt for \( A^* \) which had not used the \( \mathcal{E}_{pk}(\mathcal{L}\mathcal{R}(\cdot, \cdot, b)) \) oracle) then \( \mathcal{F} \) guesses that the bit \( b \) associated with the \( \mathcal{E}_{pk}(\mathcal{L}\mathcal{R}(\cdot, \cdot, b)) \) oracle is 0.

Note that \( B \) could be caused by \( B^* \) outputting the same message twice. However, since all messages are unique, \( A^* \) sent the message only once. With probability \( (1 - 2^{-k}) \) (where \( k \) is the length of a nonce), the challenge \( N_{B^*} \) in the encryption is different to the challenge \( A^* \) encrypted. Thus \( \mathcal{F} \) never asked for a ciphertext from the \( \mathcal{E}_{pk}(\mathcal{L}\mathcal{R}(\cdot, \cdot, b)) \) oracle to substitute as this encryption and \( \mathcal{F} \) will detect that \( U \) has successfully broken the encryption scheme.

Note that \( U \)'s view of the interaction with \( \mathcal{F} \), conditional on the event that \( \mathcal{F} \) does not abort the simulation is identically distributed to \( U \)'s view of a real interaction with an unauthenticated network if the bit \( b \) associated with the \( \mathcal{E}_{pk}(\mathcal{L}\mathcal{R}(\cdot, \cdot, b)) \) oracle is 1. (This is because \( A^* \) and \( B^* \) are randomly chosen.) Therefore, the probability of guessing \( b \) correctly is the same as that of a successful forgery between \( A^* \) and \( B^* \), which is \( 1 - \left( 1 - \frac{\gamma + 1}{|\mathcal{D}|} \right)^{s(k)c(k)} - \epsilon_2'(k) \). On the other hand, if the bit \( b \) is 0, since no information given to \( U \) depends in any way on the password, the likelihood of a successful forgery is no more than that of being successful using random guesses, \( \frac{\gamma + 1}{|\mathcal{D}|} \). Therefore, the advantage of \( \mathcal{F} \), as defined in Appendix A.3, is:

\[
\text{Adv}_{\Pi, \mathcal{F}}^{\text{lor-cca}} = \Pr[\text{Exp}_{\Pi, \mathcal{F}}^{\text{lor-cca}-1}(k) = 1] - \Pr[\text{Exp}_{\Pi, \mathcal{F}}^{\text{lor-cca}-0}(k) = 1] \\
\geq 1 - \left( 1 - \frac{\gamma + 1}{|\mathcal{D}|} \right)^{s(k)c(k)} - \epsilon_2'(k) - \left( \frac{\gamma + 1}{|\mathcal{D}|} \right) \\
= 1 - y - \left( (1 - y)^{p(k)} - \epsilon_2'(k) \right) \]

where \( y = \frac{\gamma + 1}{|\mathcal{D}|} \) and \( p(k) = s(k)c(k) \). Now \( g(k) \) is less than or equal to the advantage of the adversary in breaking the encryption scheme. If \( g(k) \) is not negligible, we have a non-negligible advantage in breaking the encryption scheme, which contradicts our original assumption. Therefore, assume \( g(k) \) is negligible.
Using the definition of negligibility, this implies:

\[
  g(k) = 1 - y - \left( (1 - y)^{p(k)} - \epsilon_2'(k) \right)^{\frac{1}{k^c}} \leq \frac{1}{k^c} \quad \text{for } c \geq 1 \text{ and } k \geq k_c
\]

\[
  (1 - y)^{p(k)} - \left( (1 - y) - \frac{1}{k^c} \right)^{p(k)} \geq \epsilon_2'(k) \quad (6.4)
\]

Lemma 6.1 states that:

\[
p(k)a^{p(k)-1}b \geq a^{p(k)} - (a - b)^{p(k)}
\]

when \( b \leq \frac{a}{p(k)} \). Substituting this into (6.4) gives \( \frac{p(k)(1-y)^{p(k)-1}}{k^c} \geq \epsilon_2'(k) \) when \( k \geq \left( \frac{p(k)}{1-y} \right)^{\frac{1}{k^c}} = k_d \). Since \( (1 - y) \) is less than one, \( (1 - y)^{p(k)-1} \) is also less than one, which implies \( \frac{p(k)}{k^c} \geq \epsilon_2'(k) \). Now assume that \( p(k) \leq k^c \) for \( k \geq k_c \). Then \( \frac{1}{k^c} \geq \epsilon_2'(k) \) for \( k \geq \max(k_c, k_d) \). Therefore \( \epsilon_2'(k) \) is negligible, and from the definition of \( \epsilon_2'(k) \),

\[
  \epsilon(k) = \epsilon_1(k) + \epsilon_2(k) + \left( 1 - \left( 1 - \frac{\gamma + 1}{|S|} \right)^{s(k)c(k)} \right)
\]

where \( \epsilon_1(k) \) is also negligible. Therefore,

\[
  \epsilon(k) \leq \left( \epsilon_0(k) + \left( 1 - \left( 1 - \frac{\gamma + 1}{|S|} \right)^{s(k)c(k)} \right) \right)
\]

where \( \epsilon_0(k) \) is negligible, and this completes the proof. \( \square \)

6.2.3.1 Variations of \( \lambda_{P ENC} \)

6.2.3.1.1 Encryption Indistinguishable under Chosen Plaintext Attack

Note that the decryption oracle in the proof of security in Section 6.2.3 is required to be able to tell whether other clients have successfully authenticated themselves. If each client had a different (personal) public key for \( B \), then the decryption oracle would not be necessary, and the proof could be completed if the encryption scheme was non-malleable to chosen plaintext attack. The proof would be a combination of parts of the above proof with the proof of Theorem 5.3 in [AB01]. Boyarsky [Boy99] has noted (without providing the proof) that this condition is sufficient for security against a static adversary. However, if the CK-
model and definition of security are used as above, the scheme can be proven secure against a dynamic adversary.

### 6.2.3.1.2 Using a Public Password

Halevi and Krawczyk [HK99] address the problem of a client having insufficient storage for a server’s public key by using *public passwords*. A public password is the hash of the server’s public key and must be kept by the client. The public password may be seen by other parties, but must not be able to be modified by them. The authenticator must be modified so that the server sends its public key to the client in the second message flow and the client checks that the hash of the public key it received is the same as the public password. As stated by Halevi and Krawczyk [HK99, p. 247], if the generator of the public keys can be trusted not to look for collisions, the hash function need only have second preimage resistance. Otherwise, it must have (strong) collision resistance. The proof in Section 6.2.3 remains substantially the same if public passwords are used. However, the definition of $\epsilon'_2(k)$ must be changed to be:

$$
\epsilon'_2(k) \overset{\text{def}}{=} \left( \epsilon(k) - \epsilon_1(k) - \epsilon_3(k) - \left( 1 - \left( 1 - \frac{\gamma + 1}{|D|} \right)^{s(k)c(k)} \right) \right)
$$

where $\epsilon_3(k)$ is the probability that the client is sent a public key which is not the server’s public key but whose hash is the same as the public password. Since the hash function has either second preimage resistance or (strong) collision resistance, $\epsilon_3(k)$ is negligible. Subsequent expressions involving $\epsilon_1(k)$ in the proof should replace $\epsilon_1(k)$ with $(\epsilon_1(k) + \epsilon_3(k))$.

### 6.2.4 Some Applications of the Authenticator

We now show how the password-based authenticator $\lambda_{p-\text{enc}}$ can be used in practice by applying it to some key exchange protocols which have been proposed in the literature and proven secure in the AM. We then compare the resultant provably secure protocols in the UM with the two password-based key exchanges proposed by Halevi and Krawczyk [HK99] for performance and efficiency. As stated in Section 6.2.1, it is necessary to use $\lambda_{p-\text{enc}}$ to provide authenticated password-based message transmission in the construction of these protocols if they are to be provably secure. The original Encrypted Challenge-Response protocol of Halevi and Krawczyk cannot be used to produce provably secure protocols (unless, of
Table 6.1: Possible provably secure protocols in the UM using $\lambda_{\text{p-enc}}$

<table>
<thead>
<tr>
<th>Name</th>
<th>AM Protocol</th>
<th>Authenticators</th>
<th>Message One</th>
<th>Message Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>2DHPS</td>
<td>2DH (Diffie-Hellman, [CK01a])</td>
<td>$\lambda_{\text{p-enc}}$</td>
<td>$\lambda_{\text{sig}} ([\text{BCK98}])$</td>
<td></td>
</tr>
<tr>
<td>2DHSP</td>
<td>2DH</td>
<td>$\lambda_{\text{sig}}$</td>
<td>$\lambda_{\text{p-enc}}$</td>
<td></td>
</tr>
<tr>
<td>2DHPE</td>
<td>2DH</td>
<td>$\lambda_{\text{p-enc}}$</td>
<td>$\lambda_{\text{enc}} ([\text{BCK98}])$</td>
<td></td>
</tr>
<tr>
<td>2DHEP</td>
<td>2DH</td>
<td>$\lambda_{\text{enc}}$</td>
<td>$\lambda_{\text{p-enc}}$</td>
<td></td>
</tr>
<tr>
<td>DHMP</td>
<td>DHM (2DH variant, [TBGN03])</td>
<td>$\lambda_{\text{p-enc}}$</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ENCP</td>
<td>ENC (Encryption [CK01a])</td>
<td>$\lambda_{\text{p-enc}}$</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

course, one wishes to find a new proof of security for the entire newly generated protocol.

The protocols in this section are described using discrete logarithm notation, although it could easily be converted to EC notation instead. Let $p$ and $q$ be two primes such that $q|(p - 1)$ and the length of $q$ is $k$ bits. Let $G = \langle g \rangle$ be a subgroup of $\mathbb{Z}_p^*$ of order $q$. The parameters $(g, p, q)$ are assumed to be publicly known and all arithmetic is performed in $\mathbb{Z}_p^*$ unless otherwise indicated.

6.2.4.1 Existing Components in the CK-model

Several components in the CK-model exist which can be reused to derive new secure protocols in the UM. We present two SK-secure protocols in the AM and one encryption-based MT-authenticator here which are used later to derive two new provably secure UM protocols. It is possible to easily obtain other provably secure password-based key exchange protocols using the other components available, and these were considered. However, the two presented in this section are the most efficient and useful for practical situations that can be built from currently available components. Table 6.1 shows a list of the protocols it would be possible to generate using $\lambda_{\text{p-enc}}$ and other currently available components.

6.2.4.1.1 Provably Secure Protocols in the AM Each of the AM protocols below contains a session identifier (sid) [CK01a] whose value is not specified in any of the descriptions of the protocols, but is assumed to be known by protocol participants before the protocol runs. In practice, the value can be determined during protocol execution.

Although the session identifier may not be explicitly visible in an MT-
authenticator, it is included as part of the message \( m \). We stress that the session identifier is fundamentally critical to the security of the model. Its role is to assist two parties to maintain a particular session among all concurrent sessions, and the value of the session identifier must be unique to the two parties participating in the protocol.

One method of choosing the session identifier would be to have either the initiator or responder of the protocol choose it. However this approach allows replay of messages because both parties can not guarantee the uniqueness of session identifiers. A countermeasure to overcome this problem is to require each party to maintain a list of all the messages received in the past and to accept only new messages. Clearly, this approach is impractical.

A secure yet practical method of ensuring the uniqueness of the session identifier is to enforce contributions from both parties. Each party then knows that the session identifier is fresh and unique since their individual inputs form part of it. Such a session identifier may be agreed as a prologue to the protocol or else during its execution. As an example, \( sid \) in Protocol 6.10 may be replaced by \((g^x || g^y)\) where || represents concatenation. Although such a value of \( sid \) can not be used in the first message because \( g^y \) is not known at the time the message is sent, it must be used in all other messages. Omission of \( sid \) from the first message does not pose a security flaw because \( sid \) is included when \( A \) provides authentication for the first message (in the third message flow) [CK01a].

### 6.2.4.1.1.1 The Diffie-Hellman Key Exchange.

The well-known Diffie-Hellman protocol shown by Protocol 6.3 and named 2DH has been proven secure in the AM under the decision Diffie-Hellman assumption in [CK01a]. This protocol has previously been extended to be secure in the UM using a signature-based authenticator [BCK98].

\[
\begin{align*}
A & \\
A \in \mathbb{Z}_q & \\
u & = g^x & \\
B & \\
y \in \mathbb{Z}_q & \\
t & = g^y & \\
A, sid, u & \\
B, sid, t & \\
K' & = t^x = (g^y)^x \quad K = u^y = (g^x)^y
\end{align*}
\]

**Protocol 6.3: 2DH (Diffie-Hellman)**
6.2.4.1.1.2 The ENC Protocol. Protocol ENC shown in Protocol 6.4 is proven secure in the AM in [CK01a] under the assumption that the encryption scheme is secure against chosen ciphertext attack and that \( \{f_r\}_{r \in \{0,1\}^k} \) is a pseudorandom function family as defined in [GGM86]. It is assumed that A has the authentic public key of B, \( e_B \), and that B has the corresponding private key, \( d_B \).

<table>
<thead>
<tr>
<th>A</th>
<th>Known: B’s public key, ( e_B )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( r \in_R {0,1}^k )</td>
</tr>
<tr>
<td></td>
<td>( c = \mathcal{E}_e_B(r) )</td>
</tr>
<tr>
<td>( A, sid, c )</td>
<td>( A, sid, c )</td>
</tr>
<tr>
<td></td>
<td>( K = f_r(A, B, sid) )</td>
</tr>
</tbody>
</table>

Protocols 6.4: ENC (Encryption based key exchange)

6.2.4.1.2 An Encryption-based MT-authenticator The encryption-based MT-authenticator [BCK98] of Protocol 6.5, \( \lambda_{\text{ENC}} \), uses an IND-CCA public key encryption scheme and a secure message authentication (MAC) scheme, denoted \( \mathcal{M} \). It is similar to \( \lambda_{\text{p-ENC}} \) and ENC in that party B needs to have possession of a public and private key pair with the public key being known to party A. This MT-authenticator is used to construct an authenticator for the AM protocol consisting of two message flows.

<table>
<thead>
<tr>
<th>A</th>
<th>Known: B’s public key, ( e_B )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( v_A \in_R {0,1}^k )</td>
</tr>
<tr>
<td></td>
<td>( c = \mathcal{E}_{e_A}(v_A) )</td>
</tr>
<tr>
<td>( d = \mathcal{M}_{e_A}(m, A) )</td>
<td>( m, c )</td>
</tr>
<tr>
<td>( m, d' )</td>
<td>( v_A' = \mathcal{D}_{d_A}(c) )</td>
</tr>
<tr>
<td>Check ( d' = d )</td>
<td>( d' = \mathcal{M}_{e_A}(m, A) )</td>
</tr>
</tbody>
</table>

Protocols 6.5: Encryption-based MT-authenticator, \( \lambda_{\text{ENC}} \)
Although the MAC scheme could be implemented using the cipher-block-chaining (CBC) mode of a block cipher [MvOV96], it is more efficient to use a MAC scheme based on the use of a hash function [GB01]. One such example is the HMAC scheme proposed by Bellare, Canetti and Krawczyk [BCK96] and shown as Algorithm 6.1. The complexity of the scheme is approximately equivalent to two hash functions and therefore quite low in comparison to other cryptographic primitives. Verification of a MAC value is performed by finding the correct MAC value using the MAC algorithm and checking whether it is identical to the value to be verified.

To find \( d = \mathcal{M}_v(m) \) where:
- \( \mathcal{M} \) is the MAC function, implemented using HMAC,
- \( v \) is the secret MAC key (it should be at least \( l \) bits long where \( l \) is defined below),
- \( m \) is the message whose MAC is required,
- \( \mathcal{H} \) is the hash function on which to base the MAC (e.g. SHA-1 [NIS95] or MD5 [Riv92]),
- \( z \) is the length in bytes of a hashing block of \( \mathcal{H} \) (e.g. \( z = 64 \) for SHA-1 and MD5), and
- \( l \) is the length in bits of the output of \( \mathcal{H} \) (e.g. \( l = 160 \) for SHA-1 and \( l = 128 \) for MD5)

Algorithm:
- Let \( ipad = 0x36 \) repeated \( z \) times.
- Let \( opad = 0x5C \) repeated \( z \) times.
- Append zeros to \( v \) until it is \( z \) bytes long. Call this string \( w \).
- \( d = \mathcal{H}((w \oplus opad) \| \mathcal{H}((w \oplus ipad) \| m)) \) (where \( \oplus \) indicates the exclusive or operation and \( \| \) indicates concatenation).
- Return the \( l \)-bit string \( d \).

Algorithm 6.1: HMAC function [BCK96]

6.2.4.2 Key Exchanges of Halevi and Krawczyk

Since our MT-authenticator is based on the work of Halevi and Krawczyk [HK99], we compare their results with ours. Their proposal has two key exchanges, one with and one without support for forward secrecy. As previously mentioned, the
proof of the authentication mechanism used in these protocols is not adequate for use in the ck-model. Both protocols are claimed to be resistant to off-line dictionary guessing attacks and use the following additional terminologies and notations.

The public password of the user is denoted by $ppwd = \mathcal{H}(e_B)$ where $\mathcal{H}$ is a collision-resistant hash function, e.g. SHA-1 [NIS95]. The function $\mathcal{F}$ is the same as that described in Section 6.2.2. The protocols also require the pseudorandom function family described in [GGM86], $\{f_r\}_{r \in \mathbb{R}\{0,1\}^k}$.

### 6.2.4.2.1 Mutual Authentication and Key Exchange

Protocol 6.6 describes the first key exchange with mutual authentication in [HK99] which does not provide forward secrecy. Hereafter, we refer to it as HK. Mutual authentication of HK follows the general design of SKEME [Kra96] where the server $B$ uses the key $r$ chosen by the client $A$ to authenticate itself. Explicitly doing this can be viewed as providing key confirmation.

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Public password, $ppwd$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td></td>
<td>Public key, $e_B$</td>
</tr>
<tr>
<td>$w_B \in R {0,1}^k$</td>
<td>$w_B \in R {0,1}^k$</td>
</tr>
</tbody>
</table>

\[
ppwd = \mathcal{H}(e_B) \\
\begin{align*}
w_B, e_B & \leftarrow R \\
r & \in R \{0,1\}^k \\
y & = f_r(w_B, B, A) \\
c & = \mathcal{E}_{e_B}(r, \mathcal{F}(\pi; w_B, r, A, B))
\end{align*}
\]

\[
\begin{align*}
(r', v) & = \mathcal{D}_{d_B}(c) \\
\text{Check } v & = \mathcal{F}(\pi; w_B, r', A, B) \\
y' & = f_{r'}(w_B, B, A) \\
\text{Check } y & = y'
\end{align*}
\]

\[
K = f_r(y) \\
K' = f_{r'}(y')
\]

Protocol 6.6: HK

### 6.2.4.2.2 Mutual Authentication and Diffie-Hellman Key Exchange

Protocol 6.7 shows an enhanced version of HK, named HKDH, which achieves perfect forward secrecy by using Diffie-Hellman key exchange. Note that since $x$
is chosen randomly from $\mathbb{Z}_q$ in every run, the random nonce $w_B$ in Protocol 6.6 has been removed as freshness can be ensured by just using $x$. Rather than using a Diffie-Hellman key, this enhanced protocol uses $K = f_r(g^{xy})$ so that computing $K$ requires both breaking the Diffie-Hellman primitive and the encryption function.

### Protocol 6.7: HKDH

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Public password, $ppwd$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td></td>
<td>Public key, $e_B$</td>
</tr>
<tr>
<td></td>
<td>$x \in_R \mathbb{Z}_q$</td>
</tr>
<tr>
<td></td>
<td>$u = g^x$</td>
</tr>
</tbody>
</table>

\[ ppwd \overset{?}{=} \mathcal{H}(e_B) \]
\[ r \in_R \{0, 1\}^k \]
\[ y \in_R \mathbb{Z}_q \]
\[ v = g^y \]
\[ t = \mathcal{F}(\pi; u, v, r, A, B) \]
\[ c = \mathcal{E}_{e_B}(r, t) \]
\[ z = f_r(c) \]

\[ u, e_B \]

\[ v, c \rightarrow \]

\[ (r', t') = D_{d_B}(c) \]

Check $t' = \mathcal{F}(\pi; u, v, r', A, B)$
\[ z' = f_{r'}(c) \]

Check $z \overset{?}{=} z'$
\[ K = f_r(u^y) = f_r(g^{xy}) \]
\[ K' = f_{r'}(v^x) = f_{r'}(g^{xy}) \]

### 6.2.4.3 Application of $\lambda_{P-ENC}$

This section applies $\lambda_{P-ENC}$ to the sk-secure protocols in the AM described in Section 6.2.4.1.1 to obtain two secure protocols in the UM. It is intended that $\lambda_{P-ENC}$ be used in an unbalanced networking setting, where the client $A$ (typically the initiator) shares only a password with the server, while the server $B$ (typically the responder) possesses a public and private key pair. Moreover, we only show resultant protocols in the UM that do not require the client to possess a secret key (other than the password).

#### 6.2.4.3.1 Application of $\lambda_{P-ENC}$ to 2DH

Since 2DH has two message flows, we need to construct a valid authenticator for two-message protocols in the AM.
We cannot apply our password-based $\text{mt}$-authenticator to both message flows of $2\text{DH}$ because it would violate the assumption that the client’s only secret key is the password. Thus we combine $\lambda_{\text{p-enc}}$ with $\lambda_{\text{enc}}$ to form a valid authenticator. More precisely, $\lambda_{\text{p-enc}}$ is applied to the first flow of $2\text{DH}$ while $\lambda_{\text{enc}}$ is applied to the second flow of $2\text{DH}$. This application yields a new PBSK-secure protocol in the $\text{um}$, $2\text{DHPE}$. Protocol 6.8 shown an unoptimized version of the protocol. However, this version can be improved in the following ways:

- The nonce $N_B$ generated by $B$ can be replaced with $g^y$ in the second and third message flows, since $g^y$ is also a random value generated by $B$. However, whether this will result in a better protocol may depend on the implementation. If (as in the case of ordinary DL systems) $g^y$ is significantly larger than the minimum size of $N_B$, then it may be more efficient to use a separate $N_B$ to enable a shorter plaintext length to be encrypted in the third message. If, as in the case of ECCS, $g^y$ and $N_B$ are approximately the same length, it may be best to use $g^y$ instead of $N_B$.

- The encrypted nonce from $A$, $\mathcal{E}_{e_B}(v_A)$ can be moved from the fifth message to the first message. This enables the response to $A$’s nonce, $\mathcal{M}_{v_A}(B, \text{sid}, g^y, A)$ to be moved up to the second message of the protocol, making the last three messages redundant.

- The identities of $A$ and $B$ can be omitted from the beginning of all of the messages, and $g^x$ can be omitted from the beginning of the second and third messages since the session identifier can be used to determine to which session the messages belong. (In the specification of the $\text{mt}$-authenticators, the messages were unique and the entire message from the AM was included at the start of each UM message for this purpose since there were no session identifiers.)

Protocols 6.9, 6.10 and 6.11 show the optimized versions of the protocol. Protocol 6.9 uses a separate nonce $N_B$, whereas Protocol 6.10 uses the Diffie-Hellman value $g^y$ in place of the nonce $N_B$. Protocol 6.11 assumes that the client does not know the server’s public key, and therefore uses a public password, as do the Halevi and Krawczyk protocols. This protocol also uses a separate nonce, although modification of the protocol to use the Diffie-Hellman value in place of the nonce is straightforward.
### Protocol 6.8: Unoptimized 2DHPE

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Pub. key of $B$, $e_B$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td>$x \in R \mathbb{Z}_q$, $u = g^x$</td>
<td>$y \in R \mathbb{Z}_q$</td>
</tr>
<tr>
<td>$N_B \in {0, 1}^k$</td>
<td></td>
</tr>
<tr>
<td>$c_1 = \mathcal{E}_{c_B}(A, sid, u, N_B, A, \pi)$</td>
<td>$v_1 = \mathcal{D}_{d_B}(c_1)$</td>
</tr>
<tr>
<td>$v_A \in {0, 1}^k$</td>
<td>Check $v_1 \equiv (A, sid, u, N_B, A, \pi)$</td>
</tr>
<tr>
<td>$c_2 = \mathcal{E}_{c_B}(v_A)$</td>
<td>$t = g^y$</td>
</tr>
<tr>
<td>$d = \mathcal{M}_{v_A}(B, sid, t, A)$</td>
<td>$d' = \mathcal{M}_{v'_A}(B, sid, t, A)$</td>
</tr>
</tbody>
</table>

**Check** $d' \overset{?}{=} d$

$K' = t^x = (g^y)^x$

### Protocol 6.9: Optimized 2DHPE using a separate nonce

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Pub. key of $B$, $e_B$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td>$x \in R \mathbb{Z}_q$, $v_A \in {0, 1}^k$</td>
<td>$y \in R \mathbb{Z}_q$</td>
</tr>
<tr>
<td>$u = g^x$</td>
<td></td>
</tr>
<tr>
<td>$c_2 = \mathcal{E}_{c_B}(v_A)$</td>
<td>$N_B \in {0, 1}^k$</td>
</tr>
<tr>
<td>$t = g^y$</td>
<td></td>
</tr>
<tr>
<td>$v'<em>A = \mathcal{D}</em>{d_B}(c_2)$</td>
<td></td>
</tr>
<tr>
<td>$d' = \mathcal{M}_{v'_A}(B, sid, t, A)$</td>
<td></td>
</tr>
</tbody>
</table>

**Check** $d' \overset{?}{=} d$

$K' = t^x = (g^y)^x$

$K = u^y = (g^x)^y$
Chapter 6. Provably Secure Elliptic Curve Protocols

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Pub. key of $B$, $e_B$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td>$x \in \mathbb{Z}_q$, $v_A \in {0,1}^k$</td>
<td>$y \in \mathbb{Z}_q$</td>
</tr>
<tr>
<td>$u = g^x$</td>
<td>$t = g^y$</td>
</tr>
<tr>
<td>$c_2 = \mathcal{E}_{e_B}(v_A)$</td>
<td>$v_A' = \mathcal{D}_{d_B}(c_2)$</td>
</tr>
<tr>
<td>$d = \mathcal{M}_{v_A}(B, \text{sid}, t, A)$</td>
<td>$d' = \mathcal{M}_{v_A'}(B, \text{sid}, t, A)$</td>
</tr>
<tr>
<td>Check $d \overset{?}{=} d'$</td>
<td>Check $v_1 \overset{?}{=} (A, \text{sid}, u, t, \pi)$</td>
</tr>
<tr>
<td>$c_1 = \mathcal{E}_{e_B}(A, \text{sid}, u, t, \pi)$</td>
<td>$v_1 = \mathcal{D}_{d_B}(c_1)$</td>
</tr>
<tr>
<td>$K' = t^x = (g^y)^x$</td>
<td>$K = u^y = (g^x)^y$</td>
</tr>
</tbody>
</table>

Protocol 6.10: Optimized 2DHPE using Diffie-Hellman value as nonce

<table>
<thead>
<tr>
<th>A (Client)</th>
<th>B (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known: Password $\pi$</td>
<td>Known: Password of $A$, $\pi$</td>
</tr>
<tr>
<td>Public password, $pwd$</td>
<td>Secret key, $d_B$</td>
</tr>
<tr>
<td>$x \in \mathbb{Z}_q$</td>
<td>Public key, $e_B$</td>
</tr>
<tr>
<td>$y \in \mathbb{Z}_q$, $N_B \in {0,1}^k$</td>
<td>$t = g^y$</td>
</tr>
<tr>
<td>Check $pwd \overset{?}{=} \mathcal{H}(e_B)$</td>
<td>$v_A' = \mathcal{D}_{d_B}(c_2)$</td>
</tr>
<tr>
<td>$u = g^x$</td>
<td>$d' = \mathcal{M}_{v_A'}(B, \text{sid}, t, A)$</td>
</tr>
<tr>
<td>$c_1 = \mathcal{E}_{e_B}(A, \text{sid}, u, N_B, \pi)$</td>
<td>$v_1 = \mathcal{D}_{d_B}(c_1)$</td>
</tr>
<tr>
<td>$v_A \in {0,1}^k$</td>
<td>Check $v_1 \overset{?}{=} (A, \text{sid}, u, N_B, \pi)$</td>
</tr>
<tr>
<td>$c_2 = \mathcal{E}_{e_B}(v_A)$</td>
<td>$v_A' = \mathcal{D}_{d_B}(c_2)$</td>
</tr>
<tr>
<td>$d = \mathcal{M}_{v_A}(B, \text{sid}, t, A)$</td>
<td>$d' = \mathcal{M}_{v_A'}(B, \text{sid}, t, A)$</td>
</tr>
<tr>
<td>$\text{sid}, e_B, t, N_B$</td>
<td>$\text{sid}, d'$</td>
</tr>
<tr>
<td>$K' = t^x = (g^y)^x$</td>
<td>$K = u^y = (g^x)^y$</td>
</tr>
</tbody>
</table>

Protocol 6.11: Optimized 2DHPE using a public password and separate nonce
6.2. Password-Based Protocols Secure in the CK Model

6.2.4.3.2 Application of $\lambda_{P\rightarrow ENC}$ to ENC  Since ENC consists of only one message flow, $\lambda_{P\rightarrow ENC}$ is the only authenticator which needs to be applied and no substantial optimization or reordering of messages is required. However, as for Protocol 6.10, only the session identifier is used to identify to which session the message belongs, instead of the entire AM message. Protocol 6.12 shows the new UM protocol which is named ENCP, and Protocol 6.13 shows a variant that uses a public password. Note that client to server authentication is explicit through use of $\pi$, but server to client authentication is implicit through use of the server’s public key, $e_B$.

**Protocol 6.12: ENCP**

\[
\begin{align*}
A \text{ (Client)} & \quad B \text{ (Server)} \\
\text{Known: } & \quad \text{Password } \pi \\
& \quad \text{Pub. key of } B, \; e_B \\
& \quad r \in_R \{0,1\}^k, c_1 = \mathcal{E}_{eb}(r) \\
& \quad \xrightarrow{sid, c_1} \\
& \quad c_2 = \mathcal{E}_{eb}(A, sid, c_1, N_B, \pi) \\
& \quad \xrightarrow{sid, N_B} \\
& \quad v = D_{db}(c_2) \\
& \quad \text{Check } v \overset{?}{=} (A, sid, c_1, N_B, \pi) \\
& \quad r' = D_{db}(c_1) \\
& \quad K = f_r(A, B, sid) \\
& \quad K' = f_{r'}(A, B, sid)
\end{align*}
\]

**Protocol 6.13: ENCP using a public password**

\[
\begin{align*}
A \text{ (Client)} & \quad B \text{ (Server)} \\
\text{Known: } & \quad \text{Password } \pi \\
& \quad \text{Public password, } ppwd \\
& \quad \xrightarrow{sid, e_B, N_B} \\
& \quad r \in_R \{0,1\}^k, c_1 = \mathcal{E}_{eb}(r) \\
& \quad c_2 = \mathcal{E}_{eb}(A, sid, c_1, N_B, \pi) \\
& \quad \xrightarrow{sid, c_1, c_2} \\
& \quad v = D_{db}(c_2) \\
& \quad \text{Check } v \overset{?}{=} (A, sid, c_1, N_B, \pi) \\
& \quad r' = D_{db}(c_1) \\
& \quad K = f_r(A, B, sid) \\
& \quad K' = f_{r'}(A, B, sid)
\end{align*}
\]
6.2.4.4 Protocol Performance Comparison

We now compare our results in Section 6.2.4.3 with the two password-based protocols of Halevi and Krawczyk [HK99]. Protocols 2DHPE and HKDH are compared with each other because they provide forward secrecy and protocols ENC and HK are compared with each other because they do not provide forward secrecy.

Two aspects of protocol performance are examined, namely computational requirements and number of message flows. The results are summarized in Table 6.2 which indicates how many operations a particular protocol needs to perform for a successful protocol run. A number in brackets indicates that that number of operations can be computed off-line. These figures are discussed further the following subsections.

Table 6.2: Protocol computational loads (where $A$ is the client and $B$ is the server)

<table>
<thead>
<tr>
<th>Computational Operations</th>
<th>Protocol</th>
<th>2DHPE</th>
<th>HKDH</th>
<th>ENC</th>
<th>HK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>1(1)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>0</td>
</tr>
<tr>
<td>Asymmetric Encryption</td>
<td>1(1)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1(1)</td>
</tr>
<tr>
<td>Asymmetric Decryption</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Count</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client initiated, no pub. pwd.</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Client initiated, with pub. pwd.</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Server initiated, with pub. pwd.</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Server initiated, no pub. pwd.</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

6.2.4.4.1 Computational Requirements. The typical scenario in which password-based protocols are used involves a client with somewhat limited computational resources. Therefore we want to shift as much computation as possible to the server which will minimize the computational requirements of the client. In this regard, our UM protocol 2DHPE is as efficient as HKDH because both require two on-line computations on the client side. Although the server in 2DHPE needs to perform two public key decryptions instead of the one in HKDH, we stress that one of the reasons for using password-based protocols is to facilitate more efficient clients.

Similarly, on-line computations for an ENC client are the same as those for a HK client, although an ENC server must perform one more decryption than a HK
server. It is important to note that there is a double encryption in ENCP where
\( c_1 = E_{e_B}(r) \) is encrypted again in the last message using the same public key
of the server, \( e_B \). Although it seems intuitively that we can remove the double
encryption by replacing \( c_1 \) with \( r \) in the third message of ENCP (and removing
\( c_1 \) from the first message), removing the double encryption alters the protocol in
such a way that the security proof is no longer valid. On the other hand, since
the encryption necessary to calculate \( c_1 \) can be performed off-line in any case,
removing it will not save any on-line computational costs for the client (although
it would save one on-line decryption at the server).

6.2.4.4.2 Number of Message Flows. The number of message flows in a
protocol may play a vital role in its performance. Obviously an extra message
exchange in a protocol will increase the time required to complete a successful
protocol run. Depending on the circumstances, a smaller number of messages
may neutralize any disadvantage due to the protocol being more computationally
intensive on the server side. Initiation by the server in the Halevi and Krawczyk
protocols is an unusual protocol setting which does not allow clients to com-
municate with the server at will. Although this can be overcome by adding a
message flow at the beginning of the protocol so that the client can trigger the
start of the protocol, it also increases the number of messages in the protocol from
three to four. In most circumstances, more message exchanges increase protocol
completion time due to factors such as propagation delay.

Since both HKDH and HK are initiated by the server instead of the client, the
versions of our protocols 2DHPE and ENCP that are initiated by the client and do
not use a public password have the advantage of taking less time to complete the
protocol if initiation must be performed by the client. We argue that initiation
by clients is a more natural setting as it allows the clients to communicate with
the server at will, and that in many cases it is not unreasonable to expect the
client to store the server’s public key. However, if there is a situation where the
client is unable to store the server’s public key and the client must therefore use
a public password, then although 2DHPE requires the same number of message
flows as HKDH and HK, the ENCP protocol still has the advantage of requiring
one less message. This is the case for both client initiated and server initiated
versions.
6.3 Tripartite Key Exchange

Tripartite key exchange is where three parties exchange a secret key. This area is particularly suited to protocols and implementations using pairings on elliptic curves, such as the Weil or Tate pairing [ZLK02]. Here the security of tripartite key exchange in the context of the ck-model is investigated. The notation used in this section is as follows:

\( A, B, C \) : Protocol participants exchanging a secret key

\( G_1 \) : Points on an elliptic curve with a suitable pairing.

\( G_2 \) : A group of the same size as \( G_1 \).

\( n \) : The order of \( G_1 \) and \( G_2 \).

\( P \) : The base point of the elliptic curve.

\( e : G_1 \times G_1 \rightarrow G_2 \) : An admissible bilinear map (the properties of such a map are specified below).

\( \sigma^X_Y \) : Signature by \( X \) intended for \( Y \). Specifying the intended recipient clarifies the purpose of each signature in protocol descriptions, although in practice the intended recipient need not be specified.

\( X \cdot_e Y \) : Encryption by \( X \) intended for \( Y \). Specifying the sender clarifies the purpose of each encryption in protocol descriptions, although in practice the sender does not necessarily need to be specified.

An admissible bilinear map must satisfy the following three properties [BF01]:

- The map must be bilinear; that is, \( e([a]P, [b]Q) = e(P, Q)^{ab} \) for all \( P, Q \in G_1 \) and all \( a, b \in \mathbb{Z} \).

- The map must be non-degenerate; that is, it must not send all pairs in \( G_1 \times G_1 \) to the identity in \( G_2 \).

- The map must be computable; that is, there must be an efficient algorithm to compute \( e(P, Q) \) for any \( P, Q \in G_1 \).

It is possible to construct an admissible bilinear map based on either the Weil pairing or Tate pairing over an elliptic curve [Hes02, BF01].
6.3.1 Definition of Secure Tripartite Key Exchange

The definition of SK-security in the AM and UM provided by Canetti and Krawczyk in [CK01a] is restricted to the case of two participating parties. It is necessary to extend the definition to cater for at least three parties for use with tripartite key exchange. Here it is extended to apply to \( t \) parties. In order to provide the definition, it is necessary to redefine the term *matching* from Section 6.1 as follows:

**Matching** A set of \( u \) sessions (where each session is run by a different party) are *matching* if each session has the same session identifier and the purpose of each session is to communicate between the party running the session and the other \( u - 1 \) parties running the other sessions. Any subset of these sessions are also said to be matching. In particular, any two sessions from the set are said to be matching. There are a total of \( \binom{u}{2} \) such pairs.

The informal and formal definitions of SK-security when more than two parties are involved in the key exchange are provided below.

**Definition 6.5 (Informal).** A \( t \)-party KE protocol is called session key (SK-) secure if the following three conditions are met. Firstly, if \( t \) uncorrupted parties complete a set of matching sessions, then they all accept the same key. Secondly, each party who participated in the protocol knows the identities of the other \( t - 1 \) parties who also know the key. Thirdly, suppose a session key is exchanged between \( t \) uncorrupted parties and has not been revealed by the adversary. Then the adversary cannot distinguish the key from a random string with probability greater than \( 1/2 \) plus a negligible function in the security parameter.

Another way of stating the restriction on the adversary’s probability of distinguishing between a key and a random string is to say that if the adversary is given either the key or a random string, each with probability \( 1/2 \), the adversary cannot correctly determine which one it received with a probability greater than \( 1/2 \) plus a negligible function in the security parameter. (That is, it can do no better than randomly guess which one it received.) This requirement is somewhat similar to the indistinguishability requirement of an encryption scheme (see Appendix A.1). However, the definition of indistinguishability of an encryption scheme goes a step further to define the advantage of an encryption scheme in
such a way that it should be negligible for secure schemes, whereas here the restriction is placed directly on the probability of guessing correctly, not on the adversary's advantage.

**Definition 6.6 (Formal).** A $t$-party KE protocol $\pi$ is called session key (SK-) secure in the AM (respectively UM) if the following three properties hold for any adversary $A$ (respectively $U$) in the AM (respectively UM).

1. Protocol $\pi$ satisfies the property that if $t$ uncorrupted parties complete a set of $t$ matching sessions then they all output the same key.

2. Protocol $\pi$ satisfies the property that if $t$ uncorrupted parties complete a set of $t$ matching sessions then each of the $t$ parties knows the identities of the other $t - 1$ parties who participated in the protocol.

3. The probability that $A$ (respectively $U$) guesses correctly the bit $b$ from the test-session (i.e. outputs $b' = b$) is no more than $1/2$ plus a negligible function in the security parameter.

The first and third requirements of the definition of SK-security in the $t$ party case directly correspond to the two requirements of SK-security in the two party case. The other requirement in the $t$ party case that each party know the identities of the other participating $t - 1$ parties must be explicitly specified because each party does not necessarily exchange messages with every other party participating in the protocol. In the two party case, specification of this requirement is unnecessary because each party knows that the identity of the one other participant is the identity of the person with whom he or she exchanged the protocol messages.

The above definition of SK-security assumes that all uncorrupted parties behave according to the protocol specification. The definition does not cater for the case where uncorrupted parties send incorrect information to other parties. An example of such a situation could be where party $E$ generates input for a tripartite key exchange protocol, and sends it to party $A$. $A$ then sends his own input as well as that from $E$ to $C$, but says that the input from $E$ actually came from $B$. $C$ then believes that he shares a key with $A$ and $B$, when in fact he shares a key with $A$ and $E$. Such a situation does not arise in the two party case because there are not enough parties for data to be sent using an intermediary to guarantee its authenticity. A definition of SK-security for key agreement between
more than two parties when uncorrupted parties do not behave according to the protocol specification is an open problem.

### 6.3.2 Tripartite Key Exchange Protocol in the AM

Joux has described an unauthenticated broadcast tripartite key exchange protocol which requires only one round [Jou00, Pat02b], shown in Protocol 6.14. This protocol can be used as a building block in the ck-model to form authenticated tripartite key exchange protocols. However, because there are no authenticators available for broadcast protocols, proving this version of the protocol secure in the AM is not a useful exercise. Therefore, Protocol 6.14 has been modified here to create the unicast version of the protocol in the AM shown by Protocol 6.15.

The messages in this version are almost identical to those of Protocol 6.14, the only difference being the addition of the identity of the third protocol participant in some messages and a session identifier (as discussed in Section 6.2.4.1.1), \( \text{sid} \), in all messages. It is assumed that messages in the AM implicitly specify sender and receiver.

**Protocol 6.14:** Joux broadcast protocol

\[
\begin{align*}
A \rightarrow B, C : \quad & [a]P \quad a \in \mathbb{Z}_n \\
B \rightarrow A, C : \quad & [b]P \quad b \in \mathbb{Z}_n \\
C \rightarrow A, B : \quad & [c]P \quad c \in \mathbb{Z}_n \\
\text{Key} : \quad & e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}
\]

The six messages of Protocol 6.15 can be reduced to four messages by allowing one party to act as messenger between the other two parties. Such a protocol has been created in the course of this research and is shown by Protocol 6.16.

In order to prove the security of Protocols 6.15 and 6.16 in the AM, it is necessary to assume that the Decisional Bilinear Diffie-Hellman Problem (DBDH) is hard. The assumption has been studied in [CL02], and can be described similarly to the Decisional Diffie-Hellman assumption of [CK01a] as follows:

**Definition 6.7 (DBDH assumption).** Let \( e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2 \) be an admissible bilinear map that takes as input two elements of \( \mathbb{G}_1 \) and outputs an element of \( \mathbb{G}_2 \). Let \( n \) be the order of \( \mathbb{G}_1 \) and \( \mathbb{G}_2 \), and let \( P \) be an element of \( \mathbb{G}_1 \). Let two probability distributions of tuples of seven elements, \( Q_0 \) and \( Q_1 \), be defined as:

\[
Q_0 = \{(\mathbb{G}_1, \mathbb{G}_2, P, [a]P, [b]P, [c]P, e(P, P)^{abc}) : \ a, b, c \in \mathbb{Z}_n\} \quad \text{and}
\]
Protocol 6.15: Joux protocol in the AM without broadcast messages

\begin{align*}
A &\text{ on input } (A, B, C, s) : \\
A &\rightarrow B : \ (C, \text{sid}, [a]P), \quad a \in_R \mathbb{Z}_n \\
A &\rightarrow C : \ (B, \text{sid}, [a]P), \quad a \in_R \mathbb{Z}_n \\
B &\text{ on receipt of } (C, s, [a]P) : \\
B &\rightarrow A : \ (\text{sid}, [b]P), \quad b \in_R \mathbb{Z}_n \\
B &\rightarrow C : \ (\text{sid}, [b]P), \quad b \in_R \mathbb{Z}_n \\
C &\text{ on receipt of } (B, s, [a]P) : \\
C &\rightarrow A : \ (\text{sid}, [c]P), \quad c \in_R \mathbb{Z}_n \\
C &\rightarrow B : \ (\text{sid}, [c]P), \quad c \in_R \mathbb{Z}_n \\
\text{Key} : e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}

Protocol 6.16: Variant of Joux protocol in the AM that can be used with authenticators to create efficient UM protocols

\begin{align*}
B &\rightarrow A : \ (\text{sid}, [b]P) \quad \text{(where } b \in_R \mathbb{Z}_n) \\
C &\rightarrow A : \ (\text{sid}, [c]P) \quad \text{(where } c \in_R \mathbb{Z}_n) \\
A &\rightarrow B : \ (\text{sid}, C, [a]P, [c]P) \quad \text{(where } a \in_R \mathbb{Z}_n) \\
A &\rightarrow C : \ (\text{sid}, B, [a]P, [b]P) \\
\text{Key} : e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}
\[ Q_1 = \{ (G_1, G_2, P, [a]P, [b]P, [c]P, e(P, P)^d) : a, b, c, d \in R \mathbb{Z}_n \} \].

Then the DBDH assumption states that \( Q_0 \) and \( Q_1 \) are computationally indistinguishable.

**Theorem 6.2.** Given the DBDH assumption, Protocols 6.15 and 6.16 are both SK-secure in the AM.

**Proof.** The proof is almost identical to that of the Diffie-Hellman key exchange provided in [CK01a]. However, changes are required to cater for the participation of three parties instead of two and the use of the DBDH assumption instead of the decisional Diffie-Hellman assumption. To see that the first requirement of Definition 6.6 is met (if three uncorrupted parties complete matching sessions then they all output the same key), note that if \( A, B \) and \( C \) are uncorrupted, then they all compute the same key. Note that the session identifier, \( sid \), uniquely binds the values \([a]P, [b]P \) and \([c]P \) to these particular matching sessions and differentiates them from other values that the parties may exchange in other (possibly simultaneous) sessions.

To see that the second requirement of Definition 6.6 is met by Protocol 6.15 (if three uncorrupted parties complete matching sessions then each of the three parties knows the identities of the other two parties who participated in the protocol), notice that the initiator of the protocol specifies who the other two participants are in the first message (one being the receiver of the message and the other explicitly specified in the message). The other two protocol participants each know the identities of the other two protocol participants because one is specified in a message received by that participant, and the other is the sender of that message.

To see that the second requirement of Definition 6.6 is met by Protocol 6.16, note that \( B \) and \( C \) each know the other two protocol participants because they each receive a message from \( A \) (one participant) specifying the other participant. Also, \( A \) must know who the other protocol participants are to produce these messages in the first place.

To see that the third requirement of Definition 6.6 is met by both protocols (the probability that \( A \) guesses correctly the bit \( b \) saying whether it received the key or a random value for the test session it chose is no more than \( 1/2 \) plus a negligible function in the security parameter), assume to the contrary there is a KE-adversary \( A \) in the AM against the protocol that has a non-negligible advantage in guessing correctly whether the response to a test-query is real or random.
Out of this attacker \( A \), we construct a distinguisher \( D \) that distinguishes between the distributions \( Q_0 \) and \( Q_1 \) with non-negligible probability; thus reaching a contradiction with the above DBDH assumption. The input to \( D \) is denoted by \((G_1, G_2, P, e, \alpha, \beta, \gamma, \delta)\) and is chosen from \( Q_0 \) or \( Q_1 \) each with probability \( 1/2 \). Let \( l \) be an upper bound on the number of sessions invoked by \( A \) in any interaction. Algorithm 6.2 describes the distinguisher \( D \) and uses adversary \( A \) as a subroutine.

**Algorithm 6.2:** Building a distinguisher for DBDH

First note that the run of \( A \) by \( D \) (up to the point where \( A \) stops or \( D \) aborts \( A \)'s run) is identical to a normal run of \( A \) against the above protocol.

Consider the case in which the test session \( s \) chosen by \( A \) coincides with the session chosen at random by \( D \) (i.e., the \( r \)-th session as chosen in Step 1). In this case, the response to the test-query of \( A \) is \( \delta \). Thus, if the input to \( D \) came from
Q₀ then the response was the actual value of the key exchanged between A, B and C during the test-session s (since, by construction, the session key exchanged in Step 4 of Algorithm 6.2 is \( \delta = e(P, P)^{abc} \)). On the other hand, if the input to D came from Q₁ then the response to the test query was a random pairing, i.e. a random value from the distribution of keys generated by the protocol.

In addition, the input to D was chosen with probability 1/2 from Q₀ and with probability 1/2 from Q₁ and so the distribution of responses provided by D to the test query of A is the same as specified in the definition of KE-security. In this case, the probability that A guesses correctly whether the test value was “real” or “random” is \( 1/2 + \epsilon \) for non-negligible \( \epsilon \). By the above argument this is equivalent to guessing whether the input to the distinguisher D came from Q₀ or Q₁, respectively. Thus, by outputting the same bit as A, the distinguisher D guesses correctly the input distribution Q₀ or Q₁ with the same probability 1/2 + \( \epsilon \) as A did.

Now consider the case in which the \( r \)-th session is not chosen as a test-session. In this case D always ends outputting a random bit, and thus its probability to guess correctly the input distribution is 1/2.

Since the first case (in which the test-session and the \( r \)-th session coincide) happens with probability 1/l while the other case happens with probability 1−1/l we find that the overall probability of D to guess correctly is 1/2 + \( \epsilon/l \), and thus D succeeds in distinguishing Q₀ from Q₁ with non-negligible advantage. Since this contradicts the original DBDH assumption, the assumption that there is a KE-adversary A in the AM against the protocol that has a non-negligible advantage in guessing correctly whether the response to a test-query is real or random is false. Hence the third requirement of Definition 6.6 is met and this completes the proof.

\[ \square \]

It is possible to modify the protocol so that the use of the DBDH assumption in the proof can be replaced with the use of a random oracle. This also requires the proof to use the assumption that the Bilinear Diffie-Hellman (BDH) problem is hard. Let \( e : G₁ × G₁ → G₂ \) be an admissible bilinear map that takes as input two elements of \( G₁ \) and outputs an element of \( G₂ \). Let \( n \) be the order of \( G₁ \) and \( G₂ \), and let \( P \) be an element of \( G₁ \). Then the BDH problem [ZLK02] is to find \( e(P, P)^{abc} \) when given \( (G₁, G₂, P, [a]P, [b]P, [c]P) \), where \( a, b \) and \( c ∈ R Zₙ \). If the BDH problem is hard, there is no polynomial time algorithm to solve the BDH problem with non-negligible probability.
One way to modify the protocol to use this proof method is to combine \( e(P, P)^{abc} \) with some sort of hash function to produce the key (e.g. \( H( e(P, P)^{abc} ) \)) or a keyed hash function \( H( e(P, P)^{abc} ([a]P, [b]P, [c]P)) \). The logic of the proof is based on the observation that since the hash function is completely random, the adversary can only obtain information about the session key by querying the hash function oracle with the input that would have been used to generate the session key. However, if the adversary is able to produce such a value with which to query the oracle, then the adversary is also able to break the BDH problem, which was assumed to be hard. The formal proof proceeds in a similar fashion to that of the proof using the DBDH assumption.

### 6.3.3 Applying Authenticators to the Joux Protocol

In order to create an sk-secure protocol in the UM, it is necessary to apply one or more authenticators to the Joux protocol. Here we focus on two authenticators originally proposed in [BCK98], \( \lambda_{\text{sig}} \) and \( \lambda_{\text{enc}} \). Their specification is given by Protocols 6.17 and 6.5.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>( r_B \in_R {0, 1}^k )</td>
</tr>
<tr>
<td>( m, r_B )</td>
<td></td>
</tr>
<tr>
<td>( m, \sigma_A^B(m, r_B, B) )</td>
<td></td>
</tr>
</tbody>
</table>

**Protocol 6.17:** Signature-based MT-authenticator, \( \lambda_{\text{sig}} \)

Although the Joux protocol can be written as a broadcast protocol, there are currently no authenticators available in the CK-model which can be applied to a broadcast message. In fact, providing secure authenticators for multicast messages is currently an open research problem. Authenticators must therefore be applied to a unicast version of the Joux protocol.

Applying \( \lambda_{\text{sig}} \) to each message of the Joux protocol (Protocol 6.15) results in Protocol 6.18. However, it is possible to optimize this protocol to produce a much more efficient version. This can be done by using \([a]P\) in place of \( r_A \) and \( r'_A \), \([b]P\) in place of \( r_B \) and \( r'_B \), and \([c]P\) in place of \( r_C \) and \( r'_C \) to avoid creating and transmitting these extra nonces. In addition, in most cases, the two signatures produced by each party can be combined to a single signature containing one copy of each of the items originally contained in the two separate signatures. Finally,
only the session identifier needs to be included at the beginning of each UM message to determine to which session the messages belong. (In the specification of the MT-authenticators, the messages were unique and the entire message from the AM was included at the start of each UM message for this purpose since there were no session identifiers.) The resultant protocol in the UM is shown by Protocol 6.19. The session identifier has not been given a specific value here, but Section 6.2.4.1.1 discusses how to choose an appropriate value. The resulting protocol requires a total of five messages and four signatures. It is possible to combine $\sigma_A^B(A, sid, [a]P, [b]P)$ and $\sigma_C^B(C, sid, [b]P, [c]P)$ from Protocol 6.19 into one signature at the expense of an extra message, as shown by Protocol 6.20. Protocol 6.21 is another possible UM protocol where some messages have been combined after the authenticator has been applied to create a broadcast protocol. It has five broadcasts and three signatures.

The $\lambda_{\text{sig}}$ authenticator can be applied to Protocol 6.16 to produce Protocol 6.22. This protocol can be optimized in a similar way to Protocol 6.18 to produce a protocol in the UM which requires five messages but only three signatures, shown as Protocol 6.23.

A protocol resulting from applying the $\lambda_{\text{enc}}$ authenticator to the AM Joux protocol (Protocol 6.15) is described by Protocol 6.24 and an optimized version is described by Protocol 6.25. For clarity, the encryption notation used is $X^Y(z)$ and indicates an encryption of the message $z$ created by $X$ to be decrypted by $Y$. The optimized protocol requires a total of five messages, six encryptions and six MACs. Allowing messages to be broadcast does not change these requirements.

Another protocol using $\lambda_{\text{enc}}$ can be constructed in the UM, by using the variant of the Joux protocol in the AM (Protocol 6.16). The unoptimized protocol is shown by Protocol 6.26. The optimized version is shown by Protocol 6.27 and requires five messages, four encryptions and four MACs. In a broadcast version of the protocol, the last two messages can be combined into one broadcast so that only four messages are required. However, the same number of encryptions and MACs are still required by the broadcast version.

### 6.3.4 Efficiency of Joux Based Protocols in the UM

Table 6.3 shows the efficiency of each of the different optimized protocols in the UM described in Section 6.3.3. The table shows that the efficiency of each scheme depends heavily on the signature or encryption scheme chosen for the
Chapter 6. Provably Secure Elliptic Curve Protocols

Protocol 6.18: Joux protocol authenticated with $\lambda_{sig}$, not optimized


Protocol 6.19: Joux protocol authenticated with $\lambda_{sig}$
6.3. Tripartite Key Exchange

\begin{align*}
A & \rightarrow B : \quad (C, \text{sid}, [a]P) \quad \text{(where } a \in \mathbb{Z}_n) \\
B & \rightarrow C : \quad (A, \text{sid}, [a]P, [b]P) \quad \text{(where } b \in \mathbb{Z}_n) \\
C & \rightarrow A : \quad \left( \text{sid}, [b]P, [c]P, \sigma^A_B (A, B, \text{sid}, [a]P, [b]P, [c]P) \right) \quad \text{(where } c \in \mathbb{Z}_n) \\
A & \rightarrow B : \quad \left( \text{sid}, [c]P, \sigma^A_C (B, C, \text{sid}, [a]P, [b]P, [c]P) \right) \\
B & \rightarrow C : \quad \sigma^A_C (A, C, \text{sid}, [a]P, [b]P, [c]P), \quad \sigma^B_C (B, C, \text{sid}, [a]P, [b]P, [c]P) \\
B \text{ or } C & \rightarrow A : \quad \sigma^B_C (A, C, \text{sid}, [a]P, [b]P, [c]P) \\
\text{Key} : & \quad e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}

**Protocol 6.20:** Joux protocol authenticated with $\lambda_{\text{sig}}$ using a minimal number of signatures

\begin{align*}
A & \rightarrow B, C : \quad (\text{sid}, [a]P) \quad \text{(where } a \in \mathbb{Z}_n) \\
B & \rightarrow C, A : \quad (\text{sid}, [b]P) \quad \text{(where } b \in \mathbb{Z}_n) \\
C & \rightarrow A, B : \quad \left( \text{sid}, [c]P, \sigma^A_B (\text{sid}, A, B, [a]P, [b]P, [c]P) \right) \quad \text{(where } c \in \mathbb{Z}_n) \\
A & \rightarrow B, C : \quad \left( \text{sid}, \sigma^B_C (B, C, \text{sid}, [a]P, [b]P, [c]P) \right) \\
B & \rightarrow A, C : \quad \sigma^C_A (A, C, \text{sid}, [a]P, [b]P, [c]P) \\
\text{Key} : & \quad e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}

**Protocol 6.21:** Joux protocol authenticated with $\lambda_{\text{sig}}$, broadcast version
## Protocol 6.22: Variant of Joux protocol authenticated with $\lambda_{\text{sig}}$, not optimized

<table>
<thead>
<tr>
<th>Step</th>
<th>Message</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B \rightarrow A$</td>
<td>$(\text{sid}, [b]P)$</td>
<td>(where $b \in \mathbb{Z}_n$)</td>
</tr>
<tr>
<td>$A \rightarrow B$</td>
<td>$(\text{sid}, [b]P, r_A)$</td>
<td>(where $r_A \in {0, 1}^k$)</td>
</tr>
<tr>
<td>$B \rightarrow A$</td>
<td>$(\text{sid}, [b]P, \sigma^B_B (\text{sid}, [b]P, r_A, A))$</td>
<td></td>
</tr>
<tr>
<td>$C \rightarrow A$</td>
<td>$(\text{sid}, [c]P)$</td>
<td>(where $c \in \mathbb{Z}_n$)</td>
</tr>
<tr>
<td>$A \rightarrow C$</td>
<td>$(\text{sid}, [c]P, r'_A)$</td>
<td>(where $r'_A \in {0, 1}^k$)</td>
</tr>
<tr>
<td>$C \rightarrow A$</td>
<td>$(\text{sid}, [c]P, \sigma^C_C (\text{sid}, [c]P, r'_A, A))$</td>
<td></td>
</tr>
<tr>
<td>$A \rightarrow B$</td>
<td>$(\text{sid}, C, [a]P, [c]P)$</td>
<td>(where $a \in \mathbb{Z}_n$)</td>
</tr>
<tr>
<td>$B \rightarrow A$</td>
<td>$(\text{sid}, C, [a]P, [c]P, r_B)$</td>
<td>(where $r_B \in {0, 1}^k$)</td>
</tr>
<tr>
<td>$A \rightarrow C$</td>
<td>$(\text{sid}, B, [a]P, [b]P)$</td>
<td></td>
</tr>
<tr>
<td>$C \rightarrow A$</td>
<td>$(\text{sid}, B, [a]P, [b]P, r_C)$</td>
<td>(where $r_C \in {0, 1}^k$)</td>
</tr>
</tbody>
</table>

**Key:** $e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c$

## Protocol 6.23: Variant of Joux protocol authenticated with $\lambda_{\text{sig}}$

<table>
<thead>
<tr>
<th>Step</th>
<th>Message</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B$</td>
<td>$(C, \text{sid}, [a]P)$</td>
<td>(where $a \in \mathbb{Z}_n$)</td>
</tr>
<tr>
<td>$B \rightarrow C$</td>
<td>$(A, \text{sid}, [a]P, [b]P, \sigma^A_B (A, \text{sid}, [a]P, [b]P))$</td>
<td>(where $b \in \mathbb{Z}_n$)</td>
</tr>
<tr>
<td>$C \rightarrow A$</td>
<td>$(\text{sid}, [b]P, [c]P, \sigma^C_A (A, \text{sid}, [a]P, [c]P), \sigma^B_B (A, \text{sid}, [a]P, [b]P))$</td>
<td>(where $c \in \mathbb{Z}_n$)</td>
</tr>
<tr>
<td>$A$ or $B \rightarrow C$</td>
<td>$(\text{sid}, [c]P, \sigma^{BC}_A (B, C, \text{sid}, [a]P, [b]P, [c]P))$</td>
<td></td>
</tr>
</tbody>
</table>

**Key:** $e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c$
6.3. Tripartite Key Exchange

\[
\begin{array}{ll}
A \rightarrow B : (C, \text{sid}, [a]P) & \text{(where } a \in \mathbb{Z}_n) \\
B \rightarrow A : (C, \text{sid}, [a]P, \mathcal{E}_B(N_{BA})) & \text{(where } N_{BA} \in \{0, 1\}^k) \\
A \rightarrow B : (C, \text{sid}, [a]P, \text{MAC}_{N_{BA}}(C, \text{sid}, [a]P, B)) \\
A \rightarrow C : (B, \text{sid}, [a]P) \\
C \rightarrow A : (B, \text{sid}, [a]P, \mathcal{E}_A(N_{CA})) & \text{(where } N_{CA} \in \{0, 1\}^k) \\
A \rightarrow C : (B, \text{sid}, [a]P, \text{MAC}_{N_{CA}}(B, \text{sid}, [a]P, C)) \\
B \rightarrow A : (\text{sid}, [b]P) & \text{(where } b \in \mathbb{Z}_n) \\
A \rightarrow B : (\text{sid}, [b]P, \mathcal{E}_B(N_{AB})) & \text{(where } N_{AB} \in \{0, 1\}^k) \\
B \rightarrow A : (\text{sid}, [b]P, \text{MAC}_{N_{AB}}(\text{sid}, [b]P, A)) \\
B \rightarrow C : (\text{sid}, [b]P) \\
C \rightarrow B : (\text{sid}, [b]P, \mathcal{E}_B(N_{CB})) & \text{(where } N_{CB} \in \{0, 1\}^k) \\
B \rightarrow C : (\text{sid}, [b]P, \text{MAC}_{N_{CB}}(\text{sid}, [b]P, C)) \\
C \rightarrow A : (\text{sid}, [c]P) & \text{(where } c \in \mathbb{Z}_n) \\
A \rightarrow C : (\text{sid}, [c]P, \mathcal{E}_C(N_{AC})) & \text{(where } N_{AC} \in \{0, 1\}^k) \\
C \rightarrow A : (\text{sid}, [c]P, \text{MAC}_{N_{AC}}(\text{sid}, [c]P, A)) \\
C \rightarrow B : (\text{sid}, [c]P) \\
B \rightarrow C : (\text{sid}, [c]P, \mathcal{E}_C(N_{BC})) & \text{(where } N_{BC} \in \{0, 1\}^k) \\
C \rightarrow B : (\text{sid}, [c]P, \text{MAC}_{N_{BC}}(\text{sid}, [c]P, B)) \\
\end{array}
\]

\[\text{Key : } e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c\]

**Protocol 6.24:** Joux protocol authenticated with λ_{enc}, not optimized
Chapter 6. Provably Secure Elliptic Curve Protocols

Protocol 6.25: Joux protocol authenticated with $\lambda_{\text{ENC}}$

\[
\begin{align*}
A \rightarrow B : & \quad C, \text{sid}, [a]P, A^E_B(N_{AB}), A^E_C(N_{AC}) \\
B \rightarrow C : & \quad A, \text{sid}, [a]P, [b]P, B^E_C(N_{BC}), B^E_A(N_{BA}), \text{MAC}_{N_{AB}}(\text{sid}, [b]P, A), A^E_C(N_{AC}) \\
C \rightarrow A : & \quad \text{sid}, [b]P, [c]P, C^E_A(N_{CA}), C^E_B(N_{CB}), \text{MAC}_{N_{AC}}(\text{sid}, [c]P, A), \\
& \quad \text{MAC}_{N_{BC}}(\text{sid}, [c]P, B), B^E_A(N_{BA}), \text{MAC}_{N_{AB}}(\text{sid}, [b]P, A) \\
A \rightarrow B : & \quad \text{sid}, [c]P, \text{MAC}_{N_{BA}}(\text{sid}, C, [a]P, B), \text{MAC}_{N_{CA}}(\text{sid}, B, [a]P, C), \\
& \quad C^E_B(N_{CB}), \text{MAC}_{N_{BC}}(\text{sid}, [c]P, B) \\
B \rightarrow C : & \quad \text{sid}, \text{MAC}_{N_{CB}}(\text{sid}, [b]P, C), \text{MAC}_{N_{CA}}(\text{sid}, B, [a]P, C) \\
\text{Key} : & \quad e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}
\]

Protocol 6.26: Variant of Joux protocol authenticated with $\lambda_{\text{ENC}}$, not optimized

\[
\begin{align*}
B \rightarrow A : & \quad (\text{sid}, [b]P) \quad \text{(where } b \in R \mathbb{Z}_n) \\
A \rightarrow B : & \quad (\text{sid}, [b]P, A^E_B(N_{AB})) \quad \text{(where } N_{AB} \in R \{0, 1\}^k) \\
B \rightarrow A : & \quad (\text{sid}, [b]P, \text{MAC}_{N_{AB}}(\text{sid}, [b]P, A)) \\
C \rightarrow A : & \quad (\text{sid}, [c]P) \quad \text{(where } c \in R \mathbb{Z}_n) \\
A \rightarrow C : & \quad (\text{sid}, [c]P, A^E_C(N_{AC})) \quad \text{(where } N_{AC} \in R \{0, 1\}^k) \\
C \rightarrow A : & \quad (\text{sid}, [c]P, \text{MAC}_{N_{AC}}(\text{sid}, [c]P, A)) \\
A \rightarrow B : & \quad (C, \text{sid}, [a]P, [c]P) \quad \text{(where } a \in R \mathbb{Z}_n) \\
B \rightarrow A : & \quad (C, \text{sid}, [a]P, [c]P, B^E_A(N_{BA})) \quad \text{(where } N_{BA} \in R \{0, 1\}^k) \\
A \rightarrow B : & \quad (C, \text{sid}, [a]P, [c]P, \text{MAC}_{N_{BA}}(C, \text{sid}, [a]P, [c]P, B)) \\
A \rightarrow C : & \quad (B, \text{sid}, [a]P, [b]P) \\
C \rightarrow A : & \quad (B, \text{sid}, [a]P, [b]P, C^E_A(N_{CA})) \quad \text{(where } N_{CA} \in R \{0, 1\}^k) \\
A \rightarrow C : & \quad (B, \text{sid}, [a]P, [b]P, \text{MAC}_{N_{CA}}(B, \text{sid}, [a]P, [b]P, C)) \\
\text{Key} : & \quad e(P, P)^{abc} = e([b]P, [c]P)^a = e([a]P, [c]P)^b = e([a]P, [b]P)^c
\end{align*}
\]
6.3. Tripartite Key Exchange

A → B :  C, sid, [a]P, A\(\mathcal{E}_B(N_{AB})\), A\(\mathcal{E}_C(N_{AC})\)

B → C :  A, sid, [a]P, [b]P, MAC_{N_{BA}}(sid, [b]P, A), A\(\mathcal{E}_C(N_{AC})\)

C → A :  sid, [b]P, [c]P, C\(\mathcal{E}_A(N_{CA})\), MAC_{N_{AC}}(sid, [c]P, A), B\(\mathcal{E}_A(N_{BA})\), MAC_{N_{AB}}(sid, [b]P, A)


A or B → C :  sid, MAC_{N_{CA}}(sid, B, [a]P, [b]P, C)


**Protocol 6.27:** Variant of Joux protocol authenticated with \(\lambda_{enc}\)

### Table 6.3: Operations and messages required by tripartite U\(M\) protocols

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast used</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N/Y</td>
<td>N/Y</td>
<td>N/Y</td>
</tr>
<tr>
<td>Messages</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td></td>
<td>N/Y</td>
<td>N/Y</td>
</tr>
<tr>
<td>Signatures</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Verifications</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Encryptions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Decryptions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>MACs</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Scalar mults.</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Exponentiations</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Pairings</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Implementation. Since the protocols will be executed using an elliptic curve where a pairing is available that can be used as the basis of an admissible bilinear map (as defined at the beginning of Section 6.3), the suitability of various pairing-based signature and encryption schemes for the above protocols has been investigated. A brief description of each scheme is included below, and the efficiency of each scheme summarized in Tables 6.4 and 6.5.

**Paterson** This identity-based signature scheme was proposed by Paterson in [Pat02a]. The signature requires a total of three scalar multiplications. However, two of these may be performed together to increase the speed of the implementation, so that the total computation time required will be
Table 6.4: Efficiency of signature schemes using pairings

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Signature</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hess</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Paterson</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BLS</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CC</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NR</td>
<td>(1)</td>
<td>1</td>
</tr>
<tr>
<td>LQ</td>
<td>(1)</td>
<td>2</td>
</tr>
<tr>
<td>M-L</td>
<td>(1)</td>
<td>-</td>
</tr>
<tr>
<td>SOK</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(y) indicates an additional $y$ operations required in a precomputation.

Table 6.5: Efficiency of encryption schemes using pairings

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lynn</td>
<td>(1)</td>
<td>-</td>
</tr>
<tr>
<td>BF</td>
<td>(1)</td>
<td>1</td>
</tr>
<tr>
<td>NR</td>
<td>(1)</td>
<td>1</td>
</tr>
<tr>
<td>LQ</td>
<td>(1)</td>
<td>2</td>
</tr>
<tr>
<td>M-L</td>
<td>(1)</td>
<td>-</td>
</tr>
</tbody>
</table>

(y) indicates an additional $y$ operations required in a precomputation.
equivalent to about 2.4 scalar multiplications. The verification requires a total of three pairings. However, one is a constant value for all verifications, and can be stored permanently with the curve parameters. One of the others only depends on public information and the identity of the signer, and so can be precomputed for a particular signer. The third pairing must be performed on-line.

**M-L** This identity-based signcryption scheme was proposed by Malone-Lee in [ML02]. A summary of its efficiency is also provided in [NR03]. The scheme provides non-repudiation if the plaintext is surrendered to the party required to perform an independent verification. The scheme can therefore be used in place of a signature scheme if desired.

**NR** This identity-based signcryption scheme was proposed by Nalla and Reddy in [NR03]. The trusted authority which is required to allow the scheme to be identity-based can also be used to provide non-repudiation.

**BLS** This scheme to provide short signatures was proposed by Boneh, Lynn and Shacham in [BLS01]. The scheme is not identity-based and allows different signatures to be combined, thus saving bandwidth (at the expense of some extra computation). The scheme can also be used for batch verification, which increases verification efficiency if several users all sign the same message.

**Lynn** This scheme to provide authenticated identity-based encryption was proposed by Lynn in [Lyn02]. The scheme does not provide non-repudiation [LQ03] and so can not be used in place of a signature scheme. It is noteworthy that this scheme actually uses fewer pairings than the BF scheme which provides encryption only. However, the Lynn scheme does require use of symmetric encryption and decryption algorithms.

**CC** This scheme to provide an identity-based signature was proposed by Cha and Cheon in [CC03].

**Hess** This scheme was proposed by Hess in [Hes02] and is an identity-based signature scheme. The paper also includes a comparison with the CC, Paterson and SOK schemes.
**SOK** This scheme was proposed by Sakai, Ohgishi and Kasahara and an efficiency analysis is provided in [Hes02].

**BF** This identity-based encryption scheme was proposed by Boneh and Franklin in [BF01].

**LQ** This identity-based signcryption scheme was proposed by Libert and Quisquater in [LQ03]. It can require the use of a symmetric encryption and decryption scheme, and can be used as either a signature or an encryption scheme since it provides non-repudiation because any party can verify the origin of the ciphertext. However, verification of the origin of the plaintext requires the key used for the symmetric encryption to be provided to the party performing the verification. Another property of the scheme is that the symmetric encryption and decryption can be replaced by some extra modular multiplications if the plaintext to be encrypted is only short. The signcryption requires a total of two scalar multiplications, but these can be performed together in the time of about 1.4 scalar multiplications.

Although the signcryption schemes can be used as either a signature or an encryption scheme, care must be taken when performing an efficiency analysis of the resulting UM protocol, since extra signatures may need to be created if a single signature was intended for use by more than one recipient in the original UM protocol.

Since the pairing operation is the most expensive of those performed by the signature and encryption schemes under consideration, the authenticated identity-based encryption scheme of Lynn appears to be the most promising from an efficiency viewpoint. Combining it with the protocol requiring the least number of operations, Protocol 6.27, leads to an implementation of the Joux protocol in the UM requiring three on-line pairings to compute the key (one per party) and four off-line pairings. Four instead of eight off-line pairings are required since some of the off-line pairings can be reused and need not be calculated twice.

Table 6.6 provides a comparison of the number of operations required by Protocol 6.27 and those required by the tripartite protocols proposed by Al-Riyami and Paterson [ARP02] and based on Joux's protocol. The table shows that those protocols proposed in [ARP02] and using broadcast messages (TAK-1 to TAK-4) only require 3 messages, which is less than the most efficient of the protocols proposed here. However, such protocols do not provide authentication for the
Table 6.6: Operations and messages required by Al-Riyami and Paterson’s tri-partite protocols compared with Protocol 6.27

<table>
<thead>
<tr>
<th>Protocol name Broadcast used</th>
<th>TAK-1</th>
<th>TAK-2</th>
<th>TAK-3</th>
<th>TAK-4</th>
<th>TAKC</th>
<th>6.27</th>
<th>6.27</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages Broadcast used</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Messages</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Signatures</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Verifications</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Encryptions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Decryptions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>MACs</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Scalar mults.</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Exponentiations</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Pairings</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>3+(4)</td>
<td>3+(4)</td>
</tr>
</tbody>
</table>

messages. Rather, the mechanism for deriving the shared key is dependent on the public keys of all participants. This provides an indirect form of authentication, because the key will only be shared between all participants if they all believe the same people are participating in the key exchange protocol. However, Shim [Shi03] has proposed a man-in-the-middle attack on the TAK-2 protocol, implying that this method of authentication is inadequate at least in the case of TAK-2. The non-broadcast protocol from [ARP02] (TAKC) is designed to provide key confirmation as well as key authentication. This protocol requires more messages than Protocol 6.27 because it provides key confirmation, which is not provided by Protocol 6.27. It can be seen that if the time for the precomputation of pairings required by Protocol 6.27 is ignored, Protocol 6.27 is generally more efficient in terms of number of operations than those of [ARP02], and requires fewer messages than the TAKC protocol, but more messages than the TAK-1 to TAK-4 protocols.

### 6.4 Conclusion

This chapter has formally described a password-based authenticator, $\lambda_{P-\text{ENC}}$, in the CK-model and formally evaluated its security. It is impossible for any password-based authenticator to achieve the level of security required for a proper MT-authenticator because the adversary can guess passwords and thereby attempt to impersonate a client on-line with non-negligible probability of succeeding due
to the small size of the password dictionary. However, it was proved that $\lambda_{p-\text{ENC}}$ achieves the best possible level of security for a password-based authenticator.

The password-based authenticator was then applied to two protocols that have been proven SK-secure in the AM elsewhere in the literature, resulting in two provably secure key exchange protocols in the UM. Other password-based key exchange protocols can easily be obtained by using $\lambda_{p-\text{ENC}}$ with other components which have been proven secure in the CK-model elsewhere in the literature, but such protocols constructed using presently available components are less efficient than the two presented here. The two new protocols were compared with the two proposed by Halevi and Krawczyk which use the same authentication mechanism, with the result that the new protocols are almost as efficient in terms of number of computations, but better in terms of number of message flows. In addition, the new protocols have associated formal proofs of security, while the Halevi-Krawczyk ones do not. These properties make the new protocols attractive for use in settings where clients must authenticate to a server using a relatively short password.

The CK-model was also used to examine the security of tripartite key exchange protocols based on the Joux protocol. A new definition of security for key exchange protocols with more than two participants was provided, and a proof of security for the Joux protocol in the AM was given. The efficiency of the UM protocols created by combining the Joux AM protocol with signature and encryption based authenticators was analysed, and the efficiency of various pairing based encryption and signature schemes which could be used in the authentication mechanism was summarized. It was concluded that a secure tripartite key exchange protocol could be formed which required three on-line and four off-line pairings. This protocol also compared favourably with other published tripartite key agreement protocols.
Chapter 7

Conclusion

The use of elliptic curves in cryptography is an emerging but important topic in modern information security research. Because elliptic curves offer substantial advantages over other public key cryptosystems in terms of key size, processing time, bandwidth and code size, they are rapidly being accepted for standardization and use in lightweight applications. This dissertation constitutes a significant contribution to the available knowledge of elliptic curves and their applications.

The main outcomes of the dissertation are:

- An understanding of the issues involved in an implementation of an elliptic curve cryptosystem for a lightweight device,
- An elliptic curve scalar multiplication algorithm to resist simple power analysis,
- Knowledge of the security level provided by the fixed curves that are often used on lightweight devices compared to the security level of random curves, and
- Proofs of security for a password-based authentication mechanism and tripartite key exchange protocols.

A more detailed summary of each of these outcomes is provided in Section 7.1, and a description of possible future research directions is given in Section 7.2.
7.1 Summary of Research

In Chapter 2 a summary of the fundamental concepts of an ECC was provided. This included the definition of an elliptic curve, a description of the different levels of arithmetic which are required to implement an ECC (including relevant algorithms), and the security issues and requirements associated with an ECC.

Chapter 3 provided an overview of the algorithms required for the underlying field arithmetic of an elliptic curve and described the different point coordinate systems available for EC point addition and doubling. This included detailed elliptic curve addition and doubling algorithms with low memory usage suitable for implementation on a lightweight device for the most efficient coordinate systems. Although the mathematical formulae upon which such algorithms are based are available in the literature, detailed algorithms with low memory usage suitable for implementation (such as those provided in Chapter 3) have not previously been published for some point coordinate systems. Of those that have been previously published, some are less efficient in terms of either speed or memory usage than those presented in Chapter 3.

Chapter 3 also listed the estimated speed and variable usage for the point addition and doubling algorithms, and then used this information to estimate the efficiency of scalar multiplication using various coordinate system combinations. Details of an actual smart card implementation were included which showed the speed, code size and RAM usage of ECDSA for different coordinate system combinations and scalar multiplication algorithm variations. It was observed that a higher speed of ECC requires higher RAM usage and code size, and on some lightweight devices, RAM and code size restrictions may preclude faster implementations. However, the data provided is sufficient to allow an implementor to choose the fastest implementation option that will still meet memory and code size restrictions on a particular lightweight device. In addition, recommendations of functions for future coprocessors to enable efficient elliptic curve implementations on smart cards were provided.

Chapter 4 provided an introduction to side channel analysis and listed some existing countermeasures for simple and differential power analysis. It then described a new and innovative scalar multiplication algorithm to resist simple power analysis, based on the use of a signed scalar. This algorithm required a maximum of two points in a precomputation and took about 80% of the time required by the existing double and add always countermeasure for SPA used with
the binary algorithm. The chapter included an analysis of the security of the new method, and recommended that to ensure security, only scalars for which the method will require a fixed number of point additions and doublings should be used. The new algorithm compared very favourably with other previously published countermeasures. However, two methods which were published after the publication of the new scalar multiplication method provide a slightly faster implementation with the same or a slightly smaller amount of memory usage.

Chapter 5 investigated the security of fixed versus random elliptic curves. It described the speed, code size and memory usage advantages of using a fixed curve, and why lightweight devices such as smart cards can not randomly generate secure curves. It also described why it seems intuitively obvious that fixed curves are less secure than random curves and mentioned that despite this fact, there are no studies available in the literature examining the issue. It then went on to examine in detail the various methods of attack applicable to an ECC, and developed two new results on the efficiency of Pollard’s rho algorithm for solving multiple ECDLPs. These two results were a lower bound on the expected number of iterations to solve one ECDLP given that a number of others had already been solved on the same curve, and an estimation of the expected number of iterations required to solve an ECDLP given the existence of a precomputation of a certain size.

In order to quantify the security difference between a fixed and random curve, two definitions of “equivalent security” were proposed. The new results on the efficiency of Pollard’s rho method were used in conjunction with the two definitions to provide an estimation of the number of bits of security lost by using a fixed curve. Also considered was the loss in security due to the use of fixed curves with special properties such as a special modulus allowing a fast modular reduction algorithm and a special $a$ parameter allowing fast point operations. The security impact of special purpose hardware was considered, but it was concluded that such hardware posed an equal threat to both fixed and random curves. Overall, it was concluded that increasing the order of a fixed curve with special properties by 11 bits and the order of a fixed curve without special properties by 5 bits would provide an equivalent level of security to a random curve for the foreseeable future. Such an increase in the size of a fixed curve has a minimal impact on performance. Indeed, fixed curves with special properties and of a size commonly used for cryptographic applications can run faster than random curves without
special properties that are 32 bits smaller. These results imply that fixed curves are still a secure and attractive option for most applications.

Chapter 6 investigated some aspects of provably secure key exchange protocols. It began with an overview of the Canetti-Krawczyk proof model, which was used because it allows modular proofs with reusable components. A new proof of security of a password-based authenticator in this model was provided. It was shown that it is impossible for any password-based authenticator to meet the original definition of security in the \textit{ck}-model, but that the password-based authenticator presented in the chapter attained the highest possible level of security. The proof was followed by an analysis of the use and efficiency of the authentication mechanism in provably secure key exchange protocols.

The next part of Chapter 6 used the Canetti-Krawczyk model to examine secure tripartite (three party) key exchange protocols. Tripartite key exchange protocols are particularly suited to \textsc{ecc}s because of the availability of bilinear mappings on elliptic curves, which allow more efficient tripartite key exchange protocols. A new definition of security of tripartite key exchange in the \textit{ck}-model was provided, accompanied by a proof of security of an existing tripartite key exchange protocol. This was followed by an analysis of the efficiency of the protocol when used in conjunction with various authentication mechanisms.

\section{7.2 Directions for Future Research}

A number of possible future research directions building on the content of this dissertation are possible and listed below.

\textbf{Coprocessors tailored to ECCs:} A coprocessor for a smart card which efficiently implemented the recommendations for future coprocessors in Section 3.5 to provide support for \textsc{ecc}s would be a useful contribution. This would be the case particularly if a thorough analysis of its efficiency was presented.

\textbf{Side channel countermeasures:} Many side channel attack countermeasures exist in the current literature. However, some of these countermeasures are unsuitable for any one smart card due to excessive memory usage, interface incompatibility or the existence of a subsequently published attack. In addition, countermeasures to resist one form of attack may inadvertently make
another form of attack easier than it would otherwise have been [YKLM01]. A study of the different countermeasures available and their interactions with one another would be valuable. The study would ideally include an analysis of the speed, code size and memory usage of each countermeasure, and give details of which countermeasures to combine together to resist a variety of different side channel attacks simultaneously.

**Security of fixed versus random ECs over \( GF(2^n) \):** This research provided an analysis of the security of fixed versus random elliptic curves over the field \( GF(p) \). However, curves over other fields are also in use in various ECCs. Therefore, an analysis of the security of fixed versus random ECCs over other fields such as \( GF(2^n) \) would be valuable.

**Efficiency of solving multiple DLPs:** This research provided a lower bound on the expected time required to solve multiple DLPs using Pollard’s rho method. However, a theoretical lower bound on the complexity of any algorithm to solve multiple discrete logarithm problems would be a valuable contribution to current knowledge of the DLP. Although such a lower bound exists for a single general discrete logarithm problem, no such result currently exists for the case of multiple DLPs with the same domain parameters, although a hypothesis as to the value of such a lower bound has been proposed [KS01].

**Secure key exchange protocols:** The Canetti-Krawczyk proof model offers the significant advantages of a modular proof and reusable components. However, there are currently only a small number of components available with associated security proofs. In addition, many of the protocols currently used or standardized do not have an associated security proof. Further research in this area providing more secure components would be valuable, as would research providing proofs of security for currently used or standardized protocols, or a closely related provably secure alternative.

**Multicast Authenticators** The tripartite key exchange protocol proposed by Joux and studied in Chapter 6 is essentially a multicast protocol. However, because there are no multicast authenticators, if the Joux protocol is to be used in conjunction with the CK-model, the protocol must first be converted to a unicast protocol in the AM and then unicast authenticators applied
to produce secure UM protocols. If a multicast UM protocol is desired, messages must then be combined to create such a protocol in the UM. It is highly desirable to use a multicast authenticator in this situation instead, for ease of use and to avoid the generation of errors during the translation of the protocol to and from a multicast protocol. Creation of such multicast authenticators is still an open research problem.
Appendix A

Definitions and Notational Conventions

This appendix contains various definitions and notational conventions used in this dissertation. We begin with some notational conventions. The notation $a \leftarrow B$ indicates that if $B$ is an algorithm then $a$ is assigned its output. If the algorithm is randomized then it flips any coins necessary to generate the output. If $B$ is a set, then $a$ is chosen at random from that set. The notation $a \overset{R}{\leftarrow} B$ or $a \in_R B$ is used in the same way, but emphasizes the random nature of the algorithm $B$ or the random choice from the set $B$. The notation $|x|$ can be used in two ways. If $x$ is a message, it means the length of the message. If $x$ is a set, it means the cardinality of that set.

We now give the definition of a negligible function provided in [Bel02] and use the same notation and definition of an encryption scheme as given in [BDPR98]. A function is negligible if it approaches zero faster than the reciprocal of any polynomial. That is:

**Definition A.1 (Negligible).** A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for every integer $c > 0$ there is an integer $k_c$ such that $\epsilon(k) \leq k^{-c}$ for all $k \geq k_c$.

The notion of a negligible function is often used in the analysis of the security of protocols and cryptographic schemes. If the probability of success of an adversary against such a scheme or protocol is negligible, then the scheme or protocol generally deemed to provide an acceptable level of security.
Definition A.2 (Encryption Scheme). An encryption scheme \( \Pi \) consists of three algorithms \((K, E, D)\) where \( K \) is the key generation which takes a security parameter \( k \in \mathbb{N} \) and returns a randomly selected pair \((pk, sk)\) of matching public and secret keys. The (probabilistic) encryption algorithm \( E \) takes a public key \( pk \) and message \( m \) and produces a ciphertext \( c \). This is denoted \( c \xleftarrow{R} E_{pk}(m) \). The decryption algorithm \( D \) takes a secret key \( sk \) and ciphertext \( c \) and returns either the corresponding plaintext message \( m \) or the special symbol \( \bot \) indicating that the ciphertext \( c \) was invalid.

Of course, for any key pair \((pk, sk)\) generated by \( K \), it is required that all ciphertexts created by encrypting a plaintext message using the public key should be able to be decrypted using the secret key to arrive at the same plaintext message. That is, if \( c \xleftarrow{R} E_{pk}(m) \) then it is required that \( m \xleftarrow{R} D_{sk}(c) \) for any message \( m \). Encryption, decryption and key generation must also be able to be completed in polynomial time.

A decryption oracle under the key \( sk \) (i.e. an oracle returning the decryption of its input using the decryption key \( sk \), without revealing \( sk \) to the user of the oracle) is denoted \( D_{sk}(\cdot) \). An encryption oracle under the key \( pk \) is denoted \( E_{pk}(\cdot) \). If an adversary \( A \) has access to an oracle, it is indicated as a superscript. For example, if \( A \) has access to the decryption oracle \( D_{sk}(\cdot) \), this is denoted \( A^{D_{sk}(\cdot)} \).

A.1 Indistinguishability of an Encryption Scheme

Firstly we present the definition of indistinguishability under adaptive chosen ciphertext attack (IND-CCA) given by Bellare, Desai, Pointcheval and Rogaway [BDPR98], with some slight modification to the notation. This definition (or an equivalent one) is generally used in the literature when referring to indistinguishability of a public key cryptosystem. It basically allows the adversary to encrypt or decrypt any values of its choice, until it chooses two plaintexts on which to be tested. An encryption of one of the plaintexts is then given to the adversary, and it must decide to which one of the plaintexts the ciphertext corresponds. The adversary may encrypt or decrypt any values of its choice whilst arriving at its decision, with the exception that it may not decrypt the ciphertext about which it is making its decision. If the encryption scheme is secure, the adversary should not be able to do better than make a random guess for its decision. In other words, a ciphertext should reveal nothing about its corresponding
A.1. Indistinguishability of an Encryption Scheme

plaintext, even if the adversary already has partial knowledge of the plaintext and can choose plaintexts and ciphertexts and be told the corresponding ciphertexts and plaintexts respectively. We begin the formal definition by defining an experiment and the advantage of the adversary in the experiment.

**Definition A.3** (Exp$_{\Pi,A}^{\text{ind-cca}}$).

Let $\Pi = (K,E,D)$ be an encryption scheme, let $A = (A_{\text{find}}, A_{\text{guess}})$ be an adversary, let $b \in \{0,1\}$ and let $k \in \mathbb{N}$. Then we define an experiment Exp$_{\Pi,A}^{\text{ind-cca}-b}(k)$ as the following sequence of steps:

\[
\begin{align*}
    (pk, sk) & \xleftarrow{\$} K(k) \\
    (x_0, x_1, s) & \leftarrow A_{\text{find}}^{E_{pk}(\cdot), D_{sk}(\cdot)}(k) \\
    y & \leftarrow E_{pk}(x_b) \\
    d & \leftarrow A_{\text{guess}}^{E_{pk}(\cdot), D_{sk}(\cdot)}(k, y, s)
\end{align*}
\]

Return $d$

We require that $A_{\text{guess}}$ does not query $D_{sk}(\cdot)$ on $y$ and that the two messages $(x_0, x_1)$ have equal length. Note that the provision of the encryption oracle $E_{pk}(\cdot)$ is trivial for public key cryptosystems since any message can be encrypted using the public key. However, it is included in the definition for clarity.

**Definition A.4** (Adv$_{\Pi,A}^{\text{ind-cca}}$).

We define the advantage of the adversary $A$ (where $A$ was described in Definition A.3) as:

\[
\begin{align*}
    \text{Adv}_{\Pi,A}^{\text{ind-cca}} &= 2 \cdot \Pr[\text{Exp}_{\Pi,A}^{\text{ind-cca}-b}(k) = b] - 1 \quad (\text{where } \Pr[b = 0] = \frac{1}{2}) \\
    &= \Pr[\text{Exp}_{\Pi,A}^{\text{ind-cca}-1}(k) = 1] - \Pr[\text{Exp}_{\Pi,A}^{\text{ind-cca}-0}(k) = 1]
\end{align*}
\]

According to [BDPR98], $\Pi$ is secure in the sense of IND-CCA if $A$ being polynomial-time implies that $\text{Adv}_{\Pi,A}^{\text{ind-cca}}(\cdot)$ is negligible.

We note that this is precisely the same as the definition of FTG-CCA (find-then-guess security under chosen ciphertext attack) given by Bellare, Desai, Jokipii and Rogaway for symmetric key encryption in [BDJR97]. However, they go a step further to define the advantage of the scheme:
Appendix A. Definitions and Notational Conventions

Definition A.5 (Adv\textsubscript{\textit{ind}}-\textit{cca} \textit{\Pi}).

For any integers $t, q_e, \mu_e, q_d, \mu_d$, we define the advantage of the encryption scheme $\Pi$ as:

$$
\text{Adv}_{\Pi}^{\text{ind--cca}}(k, t, q_e, \mu_e, q_d, \mu_d) = \max_{A_{\text{cca}}} \{ \text{Adv}_{\Pi,A}^{\text{ind--cca}}(k) \}
$$

where the maximum is over all $A$ with time complexity $t$, each making at most $q_e$ queries to the $E_{pk}(\cdot)$ oracle, totalling at most $\mu_e - |x_0|$ bits (where $|x|$ denotes the number of bits in $x$), and also making at most $q_d$ queries to the $D_{sk}(\cdot)$ oracle, totalling at most $\mu_d$ bits.

Note that for a secure encryption scheme, if the running time $t$ of the adversary is polynomially bounded, then so too are $q_e, \mu_e, q_d$ and $\mu_d$. This being the case, $\text{Adv}_{\Pi}^{\text{ind--cca}}(k, t, q_e, \mu_e, q_d, \mu_d)$ is negligible since $\text{Adv}_{\Pi,A}^{\text{ind--cca}}(k)$ is negligible.

A.2 Left-or-Right Indistinguishability

In order to complete the proof of security of the authenticator in Section 6.2, a different definition of indistinguishability is used. This definition can be found in [BDJR97], where it is called left-or-right indistinguishability. It is based on a left-or-right oracle $E_{pk}(\text{LR}(\cdot, \cdot, b))$ where $b \in \{0,1\}$. The oracle takes input $(x_0, x_1)$ and returns $E_{pk}(x_b)$. Note that an adversary with access to such an oracle can always find $E_{pk}(x)$ for any $x$ since this can be found by $E_{pk}(\text{LR}(x, x, b))$. The goal of the adversary is to guess the bit $b$. The difference between this definition and the one above is that the oracle $E_{pk}(\text{LR}(\cdot, \cdot, b))$ may be queried many times, instead of only once. Indistinguishability under a left-or-right chosen ciphertext attack (LOR-CCA) is defined as follows:

Definition A.6 (Exp\textsubscript{\Pi,A}^{\text{lor--cca}}). Let $\Pi = (K, E, D)$ be an encryption scheme, let $A$ be an adversary, let $b \in \{0,1\}$ and let $k \in \mathbb{N}$. Then we define an experiment $\text{Exp}_{\Pi,A}^{\text{lor--cca}}(k)$ as the following sequence of steps:

$$(pk, sk) \xrightarrow{R} K(k)$$

$$d \leftarrow A_{E_{pk}(\text{LR}(\cdot, \cdot, b)), D_{sk}(\cdot)}(k)$$

Return $d$
A.3. Equivalence of Indistinguishability Definitions

We require that $A$ never queries $D_{sk}(\cdot)$ on a ciphertext $C$ output by the $E_{pk}(LR(\cdot, \cdot, b))$ oracle, and that the two messages queried of $E_{pk}(LR(\cdot, \cdot, b))$ always have equal length.

Definition A.7 ($\text{Adv}_{\Pi, A}^{\text{lor-cca}}$).

We define the advantage of the adversary $A$ (where $A$ was described in Definition A.6) as:

$$\text{Adv}_{\Pi, A}^{\text{lor-cca}} = \Pr[\text{Exp}_{\Pi, A}^{\text{lor-cca}-1}(k) = 1] - \Pr[\text{Exp}_{\Pi, A}^{\text{lor-cca}-0}(k) = 1]$$

Definition A.8 ($\text{Adv}_{\Pi}^{\text{lor-cca}}$). For any integers $t, q_e, \mu_e, q_d, \mu_d$, we define the advantage of the encryption scheme $\Pi$ as:

$$\text{Adv}_{\Pi}^{\text{lor-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) = \max_{A_{\text{cca}}} \{\text{Adv}_{\Pi, A}^{\text{lor-cca}}(k)\}$$

where the maximum is over all $A$ with time complexity $t$, each making at most $q_e$ queries to the $E_{pk}(LR(\cdot, \cdot, b))$ oracle, totalling at most $\frac{t}{2}$ bits, and also making at most $q_d$ queries to the $D_{sk}(\cdot)$ oracle, totalling at most $\mu_d$ bits.

The scheme $\Pi$ is said to be LOR-CCA secure if the function $\text{Adv}_{\Pi, A}^{\text{lor-cca}}(\cdot)$ is negligible for any adversary $A$ whose time complexity is polynomial in $k$.

A.3 Equivalence of Indistinguishability Definitions

The different definitions of security in the sense of indistinguishability under chosen ciphertext attack presented above are basically equivalent for an adversary whose time is polynomially bounded. Namely, the following implications hold:

$$\text{Adv}_{\Pi, A}^{\text{ind-cca}}(k) \text{ ng.} \iff \text{Adv}_{\Pi}^{\text{ind-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \text{ ng. (A.1)}$$

$$\text{Adv}_{\Pi}^{\text{ind-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \text{ ng.} \iff \text{Adv}_{\Pi}^{\text{lor-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \text{ ng. (A.2)}$$

$$\text{Adv}_{\Pi}^{\text{ind-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \text{ ng.} \Rightarrow \text{Adv}_{\Pi}^{\text{lor-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \text{ ng. (A.3)}$$

$$\text{Adv}_{\Pi}^{\text{lor-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \text{ ng.} \iff \text{Adv}_{\Pi, A}^{\text{lor-cca}}(k) \text{ ng. (A.4)}$$

$$\text{Adv}_{\Pi, A}^{\text{ind-cca}}(k) \text{ ng.} \iff \text{Adv}_{\Pi, A}^{\text{lor-cca}}(k) \text{ ng. (A.5)}$$
where “ng.” stands for “is negligible.” Equations (A.1) and (A.4) hold because if the time complexity, $t$, of $A$ is polynomially bounded, then so are $q_e, \mu_e, q_d$ and $\mu_d$. Equation (A.2) holds because an \textsc{ind-cca} experiment is a special case of a \textsc{lor-cca} experiment. Equation (A.3) holds due to a result proven by Bellare et al. in [BDJR97]:

$$\text{Adv}_{\text{III}}^{\text{lor-cca}}(k, t, q_e, \mu_e, q_d, \mu_d) \leq q_e \cdot \text{Adv}_{\text{II}}^{\text{ind-cca}}(k, t, q_e, \mu_e, q_d, \mu_d).$$

Since a polynomial multiplied by a negligible function is also a negligible function, this result implies (A.3). The result used in the proof of security of the authenticator is given in (A.5) and follows from (A.1) to (A.4).


Bibliography


[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution (extended abstract). In Advances in


[DeWMPW98] Erik De Win, Serge Mister, Bart Preneel, and Michael Wiener. On the performance of signature schemes based on elliptic curves. In Algorithmic Number Theory: Third International Symposium,


[GB01] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. Summer Course “Cryptography and Computer Security” at


[LQ03] Benoît Libert and Jean-Jacques Quisquater. New identity based signcryption schemes from pairings. Cryptology ePrint Archive,


