LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Biometric Retrieval of Cryptographic Keys

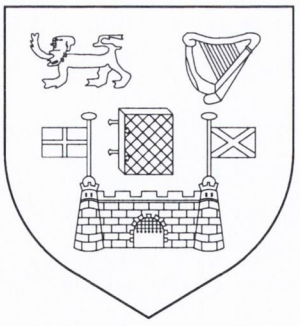
by

Eabhnat Ní Fhloinn
B.A. (Mod) (Hons.)

A Thesis submitted to
University of Dublin
for the degree of

Doctor in Philosophy

School of Mathematics
University of Dublin
Trinity College



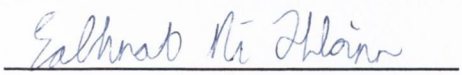
June 2006

TRINITY COLLEGE
21 NOV 2007
LIBRARY DUBLIN
THES S
8282

Declaration

This thesis has not been submitted as an exercise for a degree at this or any other University. Except where otherwise stated, the work described herein has been carried out by the author alone. This thesis may be borrowed or copied upon request with the permission of the Librarian, University of Dublin, Trinity College. The copyright belongs jointly to the University of Dublin and Eabhnat Ní Fhloinn.

Signature of Author



Eabhnat Ní Fhloinn

June 2006

Summary

The problem statement addressed in this thesis is the creation of the first biometric key retrieval scheme capable of protecting private keys used in all three types of public-key cryptosystems: those that rely on the discrete-log problem; those based on elliptic-curve implementations; and those which use the intractability of integer factorisation for security.

In our scheme, we propose that the private key be combined with the individual's biometric, using the exclusive-OR operation, in order to securely protect it. However, private keys in integer-factorisation-based systems, of which RSA [RSA78] is the best known example, are substantially larger than those in the other two types. There is insufficient uncorrupted information in a processed biometric reading to directly combine with the RSA private key. We design a novel approach called "partial hiding" to overcome this problem, making specific portions of the private key publicly available without compromising the overall security of the scheme. The "hidden" portion of the private key can then be securely combined with the biometric reading as before. We consider the most suitable parameters to implement this approach and explore possible attacks on the system.

Our biometric of choice is iris recognition, and we rely on an algorithm by Daugman [Dau93] for the necessary image acquisition and processing stages. However, we propose an improved means of *matching* the resulting iris tem-

plates by introducing the use of error-correcting codes. This approach is more secure than the statistical independence test currently in use for this algorithm, as it makes it unnecessary to store the biometric template explicitly - instead, a one-way hashed version of the iris reading, along with the relevant check digits, is sufficient.

In order to select the most suitable error-correcting code type and parameters for this application, a detailed examination of iris readings is performed. Working with Daugman's algorithm means that it is not possible to access unencrypted processed iris templates, due to commerical proprietary constraints. Therefore, we design a probabilistic model to generate data-strings with identical statistical properties to the processed iris templates and, as a result, choose to work with a $(255, 140)$ Reed-Solomon code over GF(2^8) to correct the iris readings. In addition, we use an open-source iris algorithm [Mas03] to process a database of iris readings [Chi06], and compare the resulting bit-error rates in the iris templates with results reported for Daugman's algorithm. As expected, far more errors occur using the open-source approach. Thus, we propose an alternative error-correction procedure for such systems, involving a concatenation of a 4-bit repetition code and a $(n, 128)$ Reed-Solomon code over GF(2^8), where $218 \leq n \leq 228$.

In this way, we produce a secure biometric key retrieval scheme that can be used for keys up to fourteen times larger than previously achieved; thus, it is suitable for any type of public-key cryptosystem.

Acknowledgements

To my supervisor and mentor, Dr. Michael Purser, a veritable polymath; for sharing your wisdom on a myriad of topics; for knowing when to assist me and when to force me to think for myself; for inspiring me to attempt this work and encouraging me to finish it! To my co-supervisor, Dr. Donal O'Mahony, for lending your advice and assistance when it was most needed.

To the staff and students of the School of Maths in Trinity College Dublin, for your help and guidance through the years, especially Dr. Richard Timoney, for dealing with my legions of demands on his already limited time; Dr. Donal O'Donovan, for convincing me to give Maths a try all those years ago and being there along the way; Dr. Mike Peardon and Dr. Conor Houghton, for your support in particularly difficult times; Karen O'Doherty, for all the chats; Dr. Dave Malone, for always managing to fix my computer account, no matter what I've done to it!

To Dr. Pat Fitzpatrick, Dr. Liam Marnane and Dr. Emanuel Popovici from the Coding and Crypto group in University College Cork, for your insight and encouragement. To Daon Ltd., for your contribution during the first two years of this project, in particular to Dr. Michael Peirce. To Enterprise Ireland, the Department of Education and Trinity College for assisting in funding my studies.

To Steve and John, for cheering me up and calming me down in equal

measures; for a multitude of colourful discussions; for trying so hard to educate me about popular culture. To my other fellow postgrads, past and present, particularly Alan, Justin, Donie, Don, Mick, Richie and Francesca. To my coach, Dan Curran, and all in Trinity College Amateur Boxing Club who have provided welcome stress relief and memorable experiences for the past six years. To the Amnesty International Dublin Central Group and to all the Mornington Singers.

Do mo thuismitheoirí, as ucht bhur síor-chreidiúnt ionam; as ucht an grá agus tacaíocht a thaispeánann sibh dom go deo... fós, teastaíonn uaim bheith díreach cosúil libhse nuair atáim fásta suas! D'Aodaoin agus Caoimhe, mo cháirde speisialta chomh maith le mo dheirfiúracha. To Rose, for being my third sister. To Nan, for teaching me that you should always check that there actually *is* a helicopter behind the tree.

Do Bán, as ucht bheith mar chara dom go síoraí. To Leigh, for all the giggles. To Lorraine, for girlie chats and sage advice. To Kev, for eight years of lunchtime chats. Do Donn, as ucht bheith ann chun éisteacht liom in gcónaí. To Seán, for always being ready with a hug. To Tina, for understanding about boxing, vegetarianism, PhD stress, and apostrophes. To Rory, for all the early morning singing. To JC, for helping me to see beyond the napkins to the wisdom that lay thereon.

A Michael, para todos os abraços, a gargalhada, a felicidade e o amor trouxeste à minha vida; para manter a minha mão durante épocas difíceis; para ser o meu namorado e o meu melhor amigo.

Do mo thuismitheoirí

Le grá

Contents

Declaration i

Summary ii

Acknowledgements iv

List of Figures xiv

List of Tables xv

Abbreviations xvi

1 Introduction 1

1.1 Contribution to Knowledge 3

1.2 Thesis Structure 5

2 Brief Review of PKC, Biometrics, and ECC 7

2.1 Public-Key Cryptography 7

2.1.1 Description of Public-Key Cryptography 8

2.1.2 Elliptic-Curve Cryptography 10

2.1.3 RSA 12

2.2 Biometrics 16

2.2.1 Biometric Identification 17

2.2.2 Fingerprints 19
2.2.3 Iris 22
2.3 Error-Correcting Codes 25
2.3.1 Important Definitions 25
2.3.2 Reed-Solomon Codes 27
2.3.3 Repetition Codes 30
2.3.4 LDPC Codes 31
2.3.5 Hermitian Codes 35

3 Related Work and Motivation 40
3.1 Biometric Key Retrieval 40
3.2 Key Sizes in Public-Key Systems 41
3.3 Partial Key Exposure 44
3.4 Daugman's Iris Recognition Algorithm 46
3.4.1 Projected Polar Coordinate System 48
3.4.2 IrisCode Creation 48
3.4.3 Test of Statistical Independence 50
3.4.4 Hamming Distance Threshold 51
3.4.5 Security of Daugman's Algorithm 53
3.5 Error-Correction in Biometrics 53
3.6 Motivation to Address Bio Key Retrieval for PKC 55

4 Design of Bio Key Retrieval Scheme 57
4.1 Choice of Key Size 57
4.2 Choice of Biometric 58
4.3 Choice of Code Type 59
4.3.1 RS Codes vs Hermitian Codes 61
4.3.2 RS Codes vs LDPC Codes 65

4.4 Detailed Structure of Scheme 68
4.5 Smart Cards 72
4.5.1 Attacks on Smart Cards 72
4.5.2 Biometric Key Retrieval on Smart Cards 73

5 Design for Public-Key Cryptosystems 75
5.1 SDL and Elliptic-Curve PKC Systems 76
5.1.1 Aside: Symmetric Key Systems 76
5.2 IF PKC Systems 77
5.2.1 Overview of Partial Hiding 78
5.2.2 Explanation of Partial Hiding in RSA 79
5.2.3 Partial Hiding in RSA 82
5.2.4 Partial Hiding in CRT RSA 83
5.2.5 Possible Attacks 86
5.3 Partial Hiding vs AES 91

6 Implementation and Analysis: Daugman 95
6.1 Choice of Ground Field for RS Code 96
6.2 Model to Generate Iris-like Data Strings 96
6.2.1 Computer Generation of Iris-like Data Strings 99
6.2.2 Statistical Comparison with Real Iris Data 100
6.3 Further Evidence of Suitability of RS Codes 104
6.4 Choice of Code Parameters 107
6.5 Alternative Iris Recognition Studies 111
6.6 Chosen Parameters 116
6.7 RS Iris Matching Algorithm 116
6.7.1 Program for RS Iris Algorithm 118
6.7.2 Enrolment 118

6.7.3 Verification 119
6.7.4 Programming Challenges 123
6.8 Conclusion 127

7 Implementation and Analysis: Masek **128**
7.1 Masek's Iris Processing Algorithm 129
7.2 Choice of Iris Database 130
7.2.1 Choice of Parameters 130
7.3 Implementation 136
7.4 Results and Analysis 137
7.5 Bit-Error Pattern Analysis 141
7.6 Bit-Error Burst Model 144
7.7 Conclusion 147

8 Discussion: Alternative Approaches **148**
8.1 Alternative Code-Size 149
8.1.1 Encrypting the Check Digits 149
8.1.2 Increasing the Ground Field 152
8.2 Comparison with HAD Report 157
8.2.1 Advantages of our Approach 159
8.3 RS and Repetition Code Concatenation 160
8.3.1 Implementation of RS-Repetition Scheme 163

9 Conclusion **168**
9.1 Summary of Main Conclusions 171
9.2 Summary of Contributions to Knowledge 172
9.3 Future Work 173

A Publications from Thesis **i**

B Gathering Biometric Data iii

B.1 Aims of Fingerprint Study iv

B.2 Structure of Biometric Trial iv

B.3 Implementation of Biometric Trial v

B.4 Problems Experienced by Users vi

B.5 Results of Fingerprint Matching Trial vii

C Berlekamp-Massey Algorithm viii

D Code to Generate Iris Reading Errors x

E Generating Look-Up Tables for RS Codes xiii

E.1 Look-Up Table LUT xiii

E.2 Look-Up Table LUT2 xv

F Code for Iris Authentication xvii

F.1 Enrolment Program xvii

F.2 Verification Program xxiv

G Walk-through of Iris Authentication Program xli

G.1 Enrolment Program xli

G.2 Verification Program xliv

Bibliography l

List of Figures

2.1 Fingerprint whorl, showing delta, ridge ending and bifurcation. 21

2.2 Tanner graph for an $(8, 4)$ linear block code. 33

2.3 Tanner graph showing first half-iteration of message-passing. . 34

2.4 Tanner graph showing second half-iteration of message-passing. 35

3.1 Iris image segmented using Daugman's algorithm 46

4.1 Overview of key protection 69

4.2 Overview of key retrieval 71

4.3 Biometric key retrieval on smart cards 73

5.1 Biometric key protection using partial hiding in RSA 84

6.1 Results of Daugman's iris processing algorithm under unfavourable conditions. 97

6.2 State diagram for probability of a bit error in an iris reading . 98

6.3 Bit-error rate in 1600-bit simulated iris readings 104

6.4 Number of byte errors in 1600-bit simulated iris readings . . . 109

6.5 Results of Daugman's iris processing algorithm under artificially ideal conditions. 112

6.6 Independent results of Daugman's iris processing algorithm from British Telecom Labs (1997) 113

7.1 Iris images segmented using Masek's algorithm 133
7.2 Number of valid bits in correctly-segmented 2048-bit iris readings from CASIA Ver1 database. 134
7.3 Number of valid bits in correctly-segmented 2048-bit iris readings from CASIA Ver2Dev1 database. 135
7.4 Number of valid bits in correctly-segmented 2048-bit iris readings from CASIA Ver2Dev2 database. 135
7.5 Measure of disparity in shortened iris readings of 140 bytes, processed using Masek's algorithm 138
7.6 Bit-error rate for readings in the CASIA Ver1 iris database, processed using Masek's algorithm. 138
7.7 Comparison between bit-error rates for iris readings processed using Daugman's algorithm and Masek's algorithm. 140
7.8 Alternative bit-error rate for CASIA Ver1 database, processed using Masek's algorithm. 142
7.9 Comparison between observed and expected number of error bursts of length t in iris readings 145

8.1 Maximum number of byte errors in 1600-bit simulated iris readings 150
8.2 Codeword in RS code, with duplicated data symbols 154
8.3 RS-Repetition code concatenation scheme 162
8.4 Measure of disparity in iris readings processed using Masek's algorithm, following application of repetition code 164
8.5 Separation between distributions for intra-class and inter-class comparisons in iris readings. 165
8.6 Cross-over point of FAR and FRR in iris readings, for concatenation of $(n, 128)$ RS code with 4-bit repetition code . . . 167

List of Tables

2.1 General comparison of different biometrics 17

3.1 Minimum key sizes recommended by Lenstra and Verheul . . . 42
3.2 Minimum key sizes recommended by NIST 43
3.3 Minimum key sizes recommended by ECRYPT 43
3.4 Output of logical binary operations XOR and AND 50
3.5 False match probabilities for various Hamming Distance criteria 51
3.6 Acceptable number of bits disagreeing per bits compared . . . 52

5.1 Security of revealing $2m$ bits of RSA private key 83
5.2 Number of operations in optimised implementation of AES . . 92

6.1 Normal z-scores for 1600-bit simulated iris readings compared with Daugman's results 102
6.2 Normal z-scores for 1304-bit simulated iris readings compared with Daugman's results 103
6.3 Expected number of bit errors per byte 107
6.4 Possible parameters for a $(255, k)$ RS code over GF(2^8). 108
6.5 Mean number of byte errors in 1600-bit simulated iris readings, with 11% bit-error rate 110
6.6 Mean number of byte errors in 1304-bit simulated iris readings, with 11% bit-error rate 111

6.7 Mean number of byte errors in 1600-bit simulated iris readings, with 8.9% bit-error rate 115
6.8 Mean number of byte errors in 1304-bit simulated iris readings, with 8.9% bit-error rate 115
6.9 Look-up table for GF(2^8) 125

7.1 Decidability of CASIA Ver1 database for various radial and angular resolutions 132
7.2 Summary of results of Masek's iris processing algorithm on three CASIA iris databases 132
7.3 Comparison between mean bit-error rates in iris readings using Daugman's implementation and Masek's implementation . . . 139
7.4 Comparison between expected and observed number of error-bursts of length t in iris readings 143
7.5 Normal z-scores for comparisons between Masek's data and simulated data 146

8.1 Percentage of 1600-bit simulated iris readings within error-correcting capacity of $(255, k)$ RS code over GF(2^8) 151
8.2 Summary of main steps of HAD algorithm and our system. . . 158
8.3 Table showing FAR and FRR for various values of n in an $(n, 128)$ RS code. 166

B.1 Results of biometric trial vii

Abbreviations

BDF	Boneh, Durfee and Frankel system
CRT	Chinese Remainder Theorem
DDH	Decisional Diffie-Hellman
DES	Data Encryption Standard
DL	Discrete Logarithm
DSA	Digital Signature Algorithm
ECC	Elliptic-Curve Cryptography
HD	Hamming Distance
IC	Integrated Circuit
IF	Integer Factorisation
LDPC	Low-Density Parity-Check codes
LSB	Least Significant Bit
LUT	Look-Up Table
MSB	Most Significant Bit
PKC	Public Key Cryptography
RS	Reed-Solomon codes
RSA	Rivest, Shamir and Adleman algorithm
SHA	Secure Hash Algorithm
XOR	Exclusive-OR operation

Chapter 1

Introduction

"Begin at the beginning, and go on till you come to the end: then stop."

- Lewis Carroll

Public-key cryptography (PKC) is based on the use of *key-pairs* to encrypt and decrypt data, with each key-pair consisting of a unique public key and a corresponding private key. An individual's public key is (as the name suggests) openly available at all times but his private key must always be kept secret. Secure storage of this private key is of prime importance in PKC; if this is compromised, the encrypted message is exposed. Private keys tend to be protected by passwords and may be stored either on the hard drive of a computer or possibly on a smart-card. However, passwords frequently consist of easily-guessed words or phrases, or are written beside the user's computer, and smart-cards may be stolen and reverse-engineered.

In this thesis, we propose a more secure means of protecting an individual's private key, through the use of biometrics. A biometric measures a person's unique physical or behavioural characteristics in order to recognize or authenticate his identity [LW01]. The main aim of this thesis could thus be summarised as follows:

Problem Statement: *Design a biometric key retrieval scheme capable of protecting keys from all types of public-key cryptosystems.*

There are two main stages in this process: *key protection*, in which the biometric is combined with the private key (this need only be done once for each private key); and *key retrieval*, which allows an individual to retrieve his private key as needed. These stages consist of the following steps:

1. **Key Protection**

   (a) An individual submits to a biometric reading;

   (b) Bits from this reading are combined with his private key (using the exclusive-OR (XOR) operation) to produce a secure bit-string;

   (c) This secure bit-string is stored on a suitable medium until the individual needs his private key to decrypt a message.

   Private Key $\otimes$ Biometric Reading $\rightarrow$ Secure Bit-String

2. **Key Retrieval**

   (a) The individual submits to a new biometric reading;

   (b) This is XORed with the secure bit-string to retrieve the private key;

   (c) The key can now be used for decryption purposes.

   Secure Bit-String $\otimes$ Biometric Reading $\rightarrow$ Private Key

Should an attacker gain access to the secure bit string, he will be unable to extract the private key without possessing the correct biometric reading.

As we aim to make this scheme suitable for *all* types of public-key cryptosystems, we must consider the variable lengths of private keys, which depend on the algorithm underlying the system. While some have private keys

shorter than 160 bits, others (such as RSA [RSA78]) require keys of 1024-2048 bits for security. It proved necessary to develop an additional technique for dealing with keys in the higher range. We term this novel approach "partial hiding". It consists, quite simply, of the deliberate exposure of specific sections of the private key, while all other sections must remain secret. These "secret" bits can then be XORed with the biometric reading as described above. We draw on research aimed at compromising the RSA algorithm to show that it is possible to make public large portions of the private key in RSA without adversely affecting the security of the system.

Clearly, a biometric key retrieval scheme will only work if the biometric readings used each time are identical. Given that no two consecutive biometric readings are exactly the same, we propose the use of error-correcting codes to overcome this obstacle. We reduce an individual's first (or "enrolment") biometric reading to a fixed-length codeword. All subsequent readings are viewed as corrupted codewords, with any bits which differ from the enrolment codeword treated as errors. These are corrected using an error-correcting code, and the codeword can then be used to retrieve the private key. There are several important choices to be made if this is to be successful: for example, the most suitable biometric identifier; the optimum error-correcting code for the trait in question; as well as the most appropriate code parameters to correct the expected number of errors within that biometric. We explore each of these issues in detail within the thesis.

1.1 Contribution to Knowledge

In order to achieve our goal of creating a biometric key retrieval scheme for public-key cryptography, we drew on research from three different disciplines:

biometrics, PKC and error-correcting codes. This thesis makes five major contributions to knowledge in this field:

1. A new approach to the protection of private keys in public-key cryptosystems is proposed.
    - For the first time, biometrics, error-correcting codes and public-key cryptography have been successfully combined to produce a key retrieval scheme. Furthermore, this scheme is suitable for *all* public-key systems.
    - The scheme represents a more secure form of protecting private keys than current methods, such as passwords or PINs.
2. An extension of this approach to include (for the first time) high security-level symmetric systems in biometric key retrival is suggested.
    - Keys generated by symmetric algorithms such as AES-192 and AES-256, which could never previously be protected by biometric key retrieval schemes, can be protected by our scheme.
3. The novel concept of "partial hiding" of private keys is proposed for the first time.
    - Exploiting partial exposure of private keys in a positive manner to enable the key to be protected by a biometric is entirely new.
4. An original means of matching high-quality iris templates (processed iris images) is explored.
    - The use of Reed-Soloman (RS) codes to correct errors in iris templates and thus reliably match these templates is a new and important step in biometric identification.

- The use of erasures within RS codes to increase the error-correcting capacity of the code to the level needed for iris recognition is also new. The "masking" bits (those labelled as "corrupted" by the image processing algorithm) are treated as erasures rather than errors to achieve this.
- In the case of RS codewords over larger ground fields, more information bits are needed than are provided by the iris reading. An original means of duplicating the existing biometric data and subsequently adapting the error-correction procedure is suggested.

5. An improved means of matching iris readings which have been more poorly processed, using an open-source algorithm, is presented.
   - A concatenation of repetition codes with Reed-Solomon codes is suggested for poor-quality iris templates. The repetition codes deal with the random noise in the biometric data while the Reed-Solomon codes correct the bursts of errors which occur.
   - These codes have not previously been combined for this purpose, and are an improvement on an existing concatenated scheme involving Reed-Solomon codes, as they allow far more data to be securely protected.

1.2 Thesis Structure

The background chapter is split into three main sections - public-key cryptography, biometrics and error-correcting codes - to address the different fields of study combined in this thesis. We then move on, in Chapter 3, to discuss related work from five areas relevant to the thesis, and explain our motivation

in creating a biometric key-retrieval scheme for public-key cryptosystems.

The next two chapters deal with the design of our scheme. In Chapter 4, we give a general overview of the system, and address three major issues: the key sizes we need to protect; the most suitable biometric for this purpose; and the optimum error-correcting code to use. Chapter 5 looks at how the scheme should be designed for use in different types of public-key cryptosystems. The novel concept of "partial hiding" is explored as a means of reducing the amount of private key data from large keys which must be securely stored.

Chapter 6 and 7 detail different implementations of our scheme, based on two iris recognition algorithms, one by Daugman [Dau93] and one by Masek [Mas03]. The first of these is a commercial, black-boxed algorithm with highly impressive results; the second is an open-source implementation with far poorer processing capabilities. Reed-Solomon codes are used to correct the iris readings before matching in both cases. Chapter 6 describes how the optimum code parameters for use in conjunction with Daugman's iris algorithm are determined, using a simulation model to analyse the expected number of byte errors. Chapter 7 reports the results of using these same parameters on a database of iris readings processed using Masek's algorithm.

In Chapter 8, three outstanding issues are dealt with: the possibility of using an alternative Reed-Solomon code size; a comparison between our biometric key retrieval scheme and another produced concurrently [HAD06]; and an alternative error-correction scheme which would allow poorer iris data to be used for biometric key retrieval.

The final chapter summarises our achievements in producing biometric key-retrieval schemes for each type of public-key cryptosystem. Suggestions for future work in this area are also provided. For clarity and completeness, appendices detailing C code written for the thesis are also included.

Chapter 2

A Brief Review of Public-Key Cryptography, Biometrics, and Error-Correctiong Codes

"Always remember that you are absolutely unique. Just like everyone else."

- Margaret Mead

This background chapter is divided into three main sections, which provide a brief overview of the necessary material from public-key cryptography, biometrics and error-correcting codes for a full appreciation of this thesis.

2.1 Public-Key Cryptography

The concept of public-key cryptography (PKC) first came to the fore in April 1976, with the publication of Diffie and Hellman's appropriately-titled paper "New Directions in Cryptography" [DH76]. However, it has since emerged that earlier work was secretly done in this area in the Communications-Electronics Security Group (CESG), which is the Information Security sec-

tion of GCHQ in the United Kingdom. J.H. Ellis produced a CESG report as early as January 1970 outlining his idea for a new form of cryptography, which he called *"Non-Secret Encryption" (NSE)* [Ell70]. This paper contains an existence theorem describing the idea behind what we now know as PKC. Because of the highly confidential nature of his work, however, this research was not made public knowledge until many years later.

The first workable implementation of PKC in CESG was suggested by Clifford Cocks [Coc73]. In essence, the scheme he described is a special case of the RSA algorithm, described below. Shortly afterwards, Malcolm Williamson, also of CESG, produced a different rendering of PKC [Wil74], which he developed into a scheme which corresponds to what we now know as Diffie-Hellman, although this also only came to light many years later.

2.1.1 Description of Public-Key Cryptography

Despite its ambiguous origins, the idea behind PKC was a major breakthrough for cryptography. Previously, cipher systems were *symmetric*, using a single key for both encryption and decryption. Such systems are fast to implement, but serious problems arise with the transportation of this key from sender to receiver. In a public-key system, each user has two keys: a public key which is openly available and a private key which is kept secret at all times. To send an encrypted message, an individual looks up the public key of the recipient, encrypts the message with this key, and sends it over an open channel. On receipt, the recipient decrypts the message using his private key. As no other key will decrypt the message, and the private key is kept secret at all times, this provides security and privacy to both individuals.

Public-key systems can also be used for digital signatures [DH76]. In this case, a user signs messages using his private key. The recipient then veri-

fies the signature using the signer's public key. Certain public-key systems, such as RSA [RSA78], are suitable for encryption or digital signatures, while others, such as DSA [Nat00], are only suitable for one of these purposes.

One of the most obvious attacks on public-key systems involves an attacker, E, impersonating another individual [Koh78]. E creates a public/private key pair and publishes it falsely as belonging to another user, A. When an individual sends an encrypted message to A, he encrypts it with this public-key, not knowing that the actual holder of the key pair is E. In this way, E can intercept all messages intended for A and decrypt them. E can then re-encrypt the message using A's *actual* public key and send the message to A so that nobody is aware of the breach of security.

For this reason, it is extremely important that public keys are *certified* by a Trusted Third Party (TTP) - namely, a Certification Authority (CA). A CA is an institute which ensures that an individual's identity is genuine and then issues a certificate confirming his identity and public key, signed by the CA. In general, a CA is so large and well-known that impersonation is not really an issue, and so certified public keys are deemed to be trustworthy.

PKC makes frequent use of *hash functions* to reduce the length of a message to be encrypted, thereby reducing the amount of work that must be done. Hash functions take in long strings of data and compress them by mapping them into pseudo-random shorter strings. A good hash function should be:

1. *One-way:* Given a message, m, it should be easy to compute the hash, $H(m)$; but given $H(m)$, it should be hard to compute m;

2. *Collision-resistant:* Given m, it should be hard to find m' such that $H(m) = H(m')$;

3. *Randomising:* If a single bit of the input changes, 50% of the output should change.

Most major PK algorithms rely for their security on either of two intractable mathematical problems - factorising large integers or the discrete logarithm problem.

Problem 2.1.1 (Integer Factorisation Problem) *Given a positive integer n, find its prime factorisation i.e. write $n = p_1^{e_1} p_2^{e_2} ... p_k^{e_k}$ where the p_i are pairwise distinct primes and each $e_i \geq 1$.*

Problem 2.1.2 (Discrete Log Problem) *Given integers p, α and y, find x such that $y \equiv \alpha^x \ mod \ p$.*

Most modern cryptography books cover the main public-key algorithms and their underlying security in detail, notably [vT05].

2.1.2 Elliptic-Curve Cryptography

Elliptic curves and their complexities have been studied closely in number theory for over a century. As Lang stated in 1978 *"It is possible to write endlessly on Elliptic Curves (This is not a threat)"* [Lan78]. However, it was not until 1985 that the concept of elliptic-curve cryptography was independently suggested by Koblitz [Kob87] and Miller [Mil86]. It implies that any public-key algorithm based on the discrete-logarithm problem of modular arithmetic can also be implemented using elliptic curves. It should be remembered that elliptic-curve cryptography does not consist of a new algorithm based on elliptic curves over finite fields, but instead a different implementation of existing public-key algorithms. The mathematical basis for the security of elliptic-curve cryptosystems is the intractability of

the elliptic-curve discrete-log problem [JMV01, p 2]. As this appears to be considerably harder than the discrete-log problem, smaller parameters than usual can be used in elliptic-curve cryptography to give the same level of security. This is because there are no known sub-exponential attacks on the elliptic-curve DL problem, which allows for faster computation speeds and shorter key-lengths, two major advantages of elliptic-curve implementations of public-key algorithms.

2.1.2.1 Description of Elliptic Curves

Definition Let $p > 3$ be an odd prime. An *elliptic curve* E over the finite field $\mathbb{F}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b \ , \tag{2.1}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. The set $E(\mathbb{F}_p)$ consists of all points (x, y), $x, y \in \mathbb{F}_p$, which satisfy the above equation, together with a special point $\mathcal{O}$ called the *point at infinity* [JMV01, pp 11].

A point P on the curve is represented by an x-coordinate and a y-coordinate i.e. $P = (xP, yP)$. For all such points on the curve, $P + \mathcal{O} = \mathcal{O} + P = P$. Thus, the point at infinity, $\mathcal{O}$, is in fact the zero-point. The negative of a point P is effectively its reflection in the x-axis, so that if $P = (xP, yP)$, then $-P = (xP, -yP)$. If Q is another point on the curve such that $Q \neq -P$, then the point $R = P + Q$ on the curve is given by (xR, yR), where:

$$\begin{aligned} xR &= \lambda^2 - xP - xQ \ , \\ yR &= \lambda(xP - xR) - yP \ , \end{aligned} \tag{2.2}$$

where

$$\lambda = \begin{cases} \frac{yQ-yP}{xQ-xP} & \text{if} \quad P \neq Q \\ \\ \frac{3(xP)^2+a}{2yP} & \text{if} \quad P = Q \end{cases} . \tag{2.3}$$

The set of points $E(\mathbb{F}_p)$ together with this addition operation forms a group with $\mathcal{O}$ as the identity. Numerous books and articles have been written on the subject but a good source is [HMV04].

2.1.3 RSA

RSA came to public attention in 1977, deriving its name from its three inventors Rivest, Shamir and Adleman [RSA78]. It can be used both for encryption and digital signatures, and its security relies on both the intractability of factorising large integers and the discrete log problem.

Two large distinct prime numbers p and q are chosen randomly. For maximum security, p and q should be of the same length. The product $N = pq$ is computed and this value N is called the *modulus*. Then the encryption key, e, is chosen at random such that e is relatively prime to $(p-1)(q-1)$. Using the Euclidean Algorithm, the decryption key, d, is computed such that

$$ed \equiv 1 \bmod (p-1)(q-1) \ . \tag{2.4}$$

Now e and N form the public key, while d is the private key. It is of great importance that p and q be kept secret, as otherwise d can be found. They can be discarded at this stage if so desired, but retaining their values can help to speed up private key operations in the algorithm, using the Chinese Remainder Theorem (CRT) version of RSA [QC82]. In this approach, two

separate quantities

$$\begin{aligned} d_p &= d \bmod (p-1) \ , \\ d_q &= d \bmod (q-1) \end{aligned} \tag{2.5}$$

are used to encrypt and decrypt each message.

2.1.3.1 Encryption

If we let m be the message we wish to encrypt and c be the corresponding ciphertext, then encryption merely consists of

$$m^e \bmod N = c \ . \tag{2.6}$$

Decryption is accomplished by raising the ciphertext c to the power of the private key d which gives us

$$\begin{aligned} c^d &= m^{ed} \\ &= m^{1+k(p-1)(q-1)} \\ &= m \bmod N \ . \end{aligned} \tag{2.7}$$

2.1.3.2 Digital Signatures

In the case of digital signatures, the sender typically calculates the hash $H(m)$ of his message and then sends the message together with the signature

$$H(m)^d \bmod N \ . \tag{2.8}$$

In order to validate the signature, the recipient raises it to the power e mod N to get

$$\left(H(m)^d\right)^e = H(m) \bmod N \ , \tag{2.9}$$

which he compares with the hash of the received message. If these are equal, the signature is taken to be valid.

2.1.3.3 Security

The presumed intractability of factorising large integers means that the modulus N cannot be factorised if this algorithm is correctly implemented. In the encryption process, the discrete log problem ensures that, even though the sender knows that $c^d = m \bmod N$, and possesses c, m and N, he cannot determine d, the receiver's private key. Discrete logs also feature in the security of the digital signature scheme, as the signature $H(m)^d$ is sent to the receiver along with $H(m)$. Were it not for the discrete log problem, the receiver could calculate the sender's private key d from this information.

2.1.3.4 Frustrating General Attacks on RSA

Possible attacks on RSA are well-documented and strict guidelines as to the implementation of the algorithm are in place to counteract these attacks. RSA Labs' recommend [Kal03] that RSA moduli of minimum size $n = 1024$ bits are "safe" until at least 2010 against such powerful factorising algorithms as the quadratic sieve [Pom85] and the number field sieve [Pol93]. However, they advise that keys of size 2048 bits are already in common use.

It has been established that factorising the modulus, N, and computing the decryption key d from the public key (N, e) are computationally equivalent [RSA78]. Thus, it is important that the primes p and q, whose product creates the modulus, are carefully selected so that factorising N is infeasible. p and q should be large and about the same bitlength to frustrate the elliptic curve factorising algorithm [Len87]. At the same time, the difference between p and q should not be too small, to avoid factorisation by Fermat

factorisation. Otherwise, if $p - q$ is small, then

$$\begin{aligned} p &\approx q \\ \Rightarrow p &\approx \sqrt{N} \end{aligned} \tag{2.10}$$

and an attacker could factorise N with Fermat's factorisation technique, using $x^2 - y^2 = (x+y)(x-y)$ in order to find $x^2 - y^2 = N$. The requirement for p and q to be "strong primes" is overcome provided p and q are sufficiently large [RS98]. p is a *strong prime* if $(p+1)$ has a large prime factor; $(p-1)$ has a large prime factor, p'; and $(p'-1)$ has a large prime factor.

Another area of vulnerability is the choice of the encryption key e and subsequently the decryption key d. If e is small ($e = 3$ is a common choice), then this speeds up the encryption process. However, if this is the case, it is important that each message is "salted" prior to encryption by appending an independent, random bitstring to the message. Otherwise, supposing that $e = 3$ and that the same message is sent to three or more people, if an attacker intercepts three of these encrypted messages, he will be able to decrypt the message using the Chinese Remainder Theorem [Knu98]. Salting the messages successfully counteracts this threat.

Given that it is advantageous to choose a small value for the encryption key e, one might be tempted to try the same for the decryption key d. However, an attack due to Wiener [Wie90] shows that if d is small compared to the modulus ($d < N^{1/4}$), then it can be successfully computed from the public key (N, e), using continued fractions. This attack is not possible if the decryption key is about the same size as the modulus N.

While it is convenient, and even advisable, to allow users to share a common encryption key, it is important that a unique modulus N is chosen for each user. If a common modulus is used, then a user can factorise N success-

fully from knowledge of his own public/private key pair, and subsequently obtain the private keys of all other users in the system [DeL84].

Although RSA can be used for both encryption and digital signatures, in most systems a user holds two key pairs, one for each purpose. This prevents an attacker from submitting an encrypted message, supposedly for signature, but actually resulting in decryption of the message. This attack is also prevented by hashing any messages for signature.

Having now reviewed the aspects of public-key cryptography that will be relevant to the work in this thesis, we move on to look at biometrics.

2.2 Biometrics

Although certain biometrics, such as fingerprints, have been used as personal identifiers for centuries, there has been increased interest of late in exploiting a range of other characteristics, in an attempt to improve the overall security and reliability of identification systems. Currently, physical rather than behavioural characteristics are more commonly used for biometric identification. Table 2.1 [JHP00] compares characteristics of common biometrics under various headings: *Universality* implies that each person must possess the characteristic; *Uniqueness* means that no two people share an identical trait; *Permanence* refers to the stability of the characteristic, meaning that it neither changes nor can be changed; *Collectability* indicates that the biometric is easily measured and clearly presentable to a sensor; *Performance* assesses the system's accuracy, speed, robustness and so on; *Acceptability* judges the extent to which people are willing to accept a certain biometric as an identifier.

Biometrics	Face	Finger	Hand	Iris	Retina	Signature	Voice
Universality	High	Med	Med	High	High	Low	Med
Uniqueness	Low	High	Med	High	High	Low	Low
Permanence	Med	High	Med	High	Med	Low	Low
Collectability	High	Med	High	Med	Low	High	Med
Performance	Low	High	Med	High	High	Low	Low
Acceptability	High	Med	Med	Low	Low	High	High

Table 2.1: Summary of the relative merits of various biometrics under different headings, as outlined by Jain, Hong and Pankanti [JHP00].

2.2.1 Biometric Identification

Biometric identification is made up of two stages: *enrolment* and *verification* [SRS+99]. Enrolment refers to the process of obtaining several readings of the biometric in question from a new user and extracting notable features to store in a template for comparison purposes later on. Verification consists of acquiring a new reading from the user and comparing the result with the previously-stored template in order to accept or reject the user.

The accuracy and effectiveness of biometric identification is restricted by numerous issues, such as the precision of the biometric sensor (particularly in a practical day-to-day setting); distortions in the image due to sweat, dirt, moisture content, temperature, lighting or pressure; distortions due to the positioning and orientation of the biometric; perceivable variations between different individuals' biometrics, and so on. Thus, biometric readings are not perfect constants [HKQ00], and there will usually be some small difference between two consecutive readings for the same individual. There are two different measures used to rate biometric accuracy:

Definition The *False Acceptance Rate (FAR)* is the probability that a biometric system will incorrectly identify an individual or will fail to reject an impostor.

Definition The *False Rejection Rate (FRR)* is the probability that the system will fail to identify or verify an enrolee. [MR99]

These can be expressed mathematically as

$$FAR = \frac{NFA}{NIA}, \qquad \begin{aligned} NFA &= \text{number of false acceptances,} \\ NIA &= \text{number of impostor attempts,} \end{aligned}$$

and

$$FRR = \frac{NFR}{NEA}, \qquad \begin{aligned} NFR &= \text{number of false rejections,} \\ NEA &= \text{number of enrolee attempts.} \end{aligned}$$

Clearly, these measures are interdependent; decreasing the FAR increases the FRR and vice versa. For example, if the matching criterion is low, then the number of impostors accepted will be higher and the number of enrolees incorrectly rejected will be lower. Therefore, it is extremely important that the correct trade-off between these two measures be chosen in order to provide sufficient security for the desired implementation. In a high-security military system, for example, a low FAR and a high FRR would be chosen, as it is preferable to mildly inconvenience an authorised user, who may be forced to make several verification attempts, than it is to grant access to an unauthorised user. In a commercial situation, however, in which speed of access is deemed more important, a high FAR and a low FRR would be more suitable.

Strictly speaking, a biometric scheme is an *authentication* scheme if the user must produce some claim of identity in conjunction with their biometric.

The resulting search of the database is known as a *one-to-one* search, as the biometric is only tested against the specific identity template to which the individual lays claim. A biometric *identification* scheme is one in which the user need only present his biometric to the sensor. A *one-to-many* search of the database is performed in which many templates are considered to see if any provide a possible match for the user. For a general overview of the most common types of biometric identification, see [Fhl05].

2.2.1.1 Security of Biometrics

The usefulness of any identification scheme is entirely dependant on the security of the system. No matter how reliably a biometric may identify an individual, this is of little use if the system can be easily attacked. One major objection to the use of biometrics is that, if a user's biometric is stolen or compromised in some way, there exist very few (or sometimes no) alternative means to securely re-enrol him. Thus, through no fault of his own, such a user may face exclusion from the system [Sch99]. For example, if a fingerprint is the chosen biometric, then nine alternatives exist if one is compromised, but if we are dealing with face recognition, this option does not exist. Other possible threats include insider theft, where a systems operator steals a user's template, or outside acquisition, in which a user's biometric is obtained when he voluntarily submits to a reading for some seemingly innocuous purpose. Thus, any system using biometric identification must also provide countermeasures against any such security breaches.

2.2.2 Fingerprints

Fingerprints are widely accepted to be unique physical identifiers of an individual. The patterns on each finger-tip are dependant on the initial condi-

tions of the embryonic mesoderm [JP99] and there is no known link between fingerprints in related individuals (even identical twins) or even between different fingers on the same hand. The widespread use of fingerprinting as a form of criminal identification meant that there were some negative associations to be overcome in introducing the biometric in everyday applications, but it has become constantly more acceptable to the general public. Numerous applications of fingerprint matching exist, including access control to buildings or PCs; credit-card verifications; and, most recently, monitoring of individuals entering the United States.

2.2.2.1 History of Fingerprint Matching

Fingerprints are the oldest form of biometric identification in existence. Fingerprint matching techniques were first used as early as the 16th century, and by 1880, Henry Fauld had proposed their uniqueness [JP99]. Sir Francis Galton introduced the idea of minutiae features in 1888 [LG91] and this was followed in 1899 by Edward Henry's famous "Henry system" [JP99] for indexing fingerprints. By 1926, the FBI had begun creating a database of fingerprints of known criminals [SRS+99], and this system was automated in the 1960s. Nowadays, fingerprinting is the most well-known biometric in use.

2.2.2.2 Methods of Fingerprint Matching

There are two main ways in which fingerprints are obtained: the traditional "inked" method and the modern "inkless" method. In the traditional method, the individual's finger is pressed into ink and an impression of the fingertip is taken. The modern approach can be implemented in numerous ways: for example using optical frustrated total internal reflection [KT84] or capacitance-based solid-state sensors [YHB+97]. Solid-state sensors tend to

Figure 2.1: A fingerprint whorl, in which the triangle represents a delta; the circle, a ridge ending; the cross, a ridge bifurcation. The image is 512 x 512 with 256 grey levels and is scanned at 512 dpi resolution [JP99].

be less affected than optical sensors by the dirt and oil that can remain on the sensor after a reading, but can be sensitive to the skin moisture level.

When the fingerprint image has been obtained, a method of matching prints must be employed. One of the most common approaches involves extracting and comparing minutiae points from the fingerprint, as shown in Figure 2.1. Minutiae points are locations where a fingerprint ridge splits in two, called a bifurcation, or where a ridge simply ends. Other critical points in the fingerprint are known are cores and deltas. These points are extracted using a series of image processing steps. Another popular method involves using the entire fingerprint image in order to employ a pattern recognition algorithm, such as correlation [SRS+99].

2.2.2.3 Security of Fingerprint Matching

Public perception of the security of biometric schemes is of huge importance if biometrics are to become an integral part of modern life. Books and films that depict scenes in which a dead individual's finger is used to impersonate him and gain access to a secure area reinforce scepticism and mistrust among the public. In fingerprinting matching systems, measures known as "live and well detection" have long been proposed to combat the use of fake or dead fingers. These measures also protect against latent prints, which occur when traces of the previous fingerprint remain on the sensor and are processed as a new print. However, many fingerprinting systems were dealt a severe blow by the publication of a paper by a Japenese cryptographer [MMYH02], in which eleven different fingerprint systems were fooled by using "gummy fingers" (artificial prints moulded in gelatine and placed over the finger). While the technology exists to prevent such attacks, it is important that these security measures be included in *all* sensors to combat the problem.

2.2.3 Iris

While fingerprints are the most widely accepted biometric in use today, there can be little doubt but that the iris is emerging as a serious contender in the race to discover the most efficient form of reliable personal recognition. The iris is formed by a process known as "chaotic morphogenesis", meaning that there is no genetic influence on its structure [Cam99]. Each iris has a unique texture which, like fingerprints, is developed from initial conditions in the embryonic mesoderm [Dav90] so that even the two irises in one person show no additional correlation. One objection to the use of irises stems from "iridology", which claims that systematic changes occur in the iris pattern

reflecting health, age, mood and the future. However, numerous studies ([SWM79], [Kni88]) have long since discredited iridology as medical fraud.

Some of the features which protect the iris (eyelashes, eyelids, cornea) also make it more difficult to obtain a clear image, obscuring certain areas or creating a curved, reflecting surface. Further difficulties are posed by the frequent stretching and contracting of the iris in response to variations in light intensity, although the texture itself remains unchanged.

Iris recognition schemes are currently in use in airports such as Heathrow, Frankfurt and New York JFK, as well as in ATMs for several major banking chains, in numerous state penitentiaries in the United States, and in various private business applications.

2.2.3.1 History of Iris Recognition

While the idea of using irises for identification purposes is often perceived to be relatively new, it was actually first proposed in 1885 for use in the Parisian penal system [Ber85]. However, this implementation was based merely on visually inspecting the iris and observing the colour patterns. In 1936 the concept of using iris feature patterns was introduced by Frank Burch at a meeting of the American Academy of Ophthalmology, although it would appear that it was not until 1949 that this was documented in an ophthalmology textbook by James Doggart [Dog49]. The idea of the iris providing an optical fingerprint was further explored by ophthalmologists in the 1960s, who noted the unique and unchanging nature of the iris over several decades [Adl65]. However it was not until the late 1980s that two ophthalmologists, Aran Safir and Leonard Flom, patented the idea [FS87]. They then approached John Daugman who developed the algorithm [Dau93] currently underlying all commercial iris recognition systems. The imaging difficulties previously

cited are dealt with in an efficient manner, exploiting these natural traits as advantages rather than disadvantages.

2.2.3.2 Methods of Iris Recognition

If Daugman's approach had been feature-based, different readings would possibly describe different features, and the resulting bit-strings would be of various lengths. Instead he uses a phase sequence description of the iris, meaning that the length is always the same, regardless of the image quality. An image of the user's eye is obtained and the iris is segmented from the overall image, as illustrated in Figure 3.1. An "IrisCode" is then calculated based on a phase sequence description of the iris. This IrisCode is 256 bytes in length and is accompanied by a further 256 "masking" bytes which provide information as to which bits of the image have been corrupted and should be ignored. In order to perform iris recognition, the IrisCodes from two irises are compared using a test of statistical independence. If two different irises are used, this test will be passed; if two IrisCodes computed from the same iris are compared, the test will be failed. Thus, in order to successfully identify an individual, this statistical test needs to be *failed.*

Shortly after this algorithm was first suggested, a research group in the David Sarnoff Research Centre proposed an alternative iris recognition scheme [WAH+96], which is based on Laplacian filters and uses a normalised correlation technique for iris matching. A couple of years later, Boles and Boashash suggested a means of iris matching by calculating 1D signals using zero-crossings of the wavelet transform [BB98]. Much work has been undertaken in the area of iris recognition in the last few years, with suggestions ranging from the use of competitive learning [LLBK01] to decision schemes [CKKL04]; texture analysis [ZTW00] to Hough transforms [TMTR02]; and

most recently, characterising key local variations [MTWZ04] or independent component analysis [NBPK05].

Having provided the necessary background information to biometric identification, we shall now look at error-correcting codes, which we propose to use as a means of matching processed biometric images.

2.3 Error-Correcting Codes

Modern forms of communication have resulted in the vast majority of information being transmitted in binary form - namely, a sequence of bits, each of which have the value 0 or 1. Clearly, there is an ever-present danger of errors in such transmissions, for a wide variety of reasons ranging from a momentary short-circuit in a system to a scratch on a compact disc. Error-correcting codes are used to detect and correct such errors in order to restore transmissions to their original form, and we recommend texts such as [Pur95] or [HP03] for a fuller description of the capabilities of error-correcting codes.

2.3.1 Important Definitions

If any valid message is defined to be a *codeword*, then a *code* is the set of all codewords. The *distance* between any two codewords is the number of bits which differ between them. The *code-distance*, d, is then the minimum distance between any two codewords [Pur95, p 7]. If

$$d = 2t + 1 \ , \tag{2.11}$$

then up to $2t$ errors can be *detected* and up to t of these *corrected*; with t errors or less, the corrupted codeword is automatically closer to one valid codeword than any other, but if there are more than t errors, there is no way

of knowing to which codeword one should correct. Therefore, a good code has as large a code-distance as possible, enabling more errors to be corrected.

Each codeword in the code is of the same length, usually represented by n. It is divided into k *information symbols* and $(n-k)$ *check digits*. Such a code is known as a systematic (n, k) code. The *code-rate* is then the ratio of information symbols to the code-length, and is given by $\frac{k}{n}$.

In the case of *cyclic* codes, each cyclic rotation of a codeword gives another codeword. In order to more easily understand and study cyclic codes, we map them to polynomials. Thus, a codeword such as $c_0c_1c_2c_3$ can be represented as $c(x) = c_0 + c_1x + c_2x^2 + c_3x^3$. There exists a *generating polynomial*, $g(x)$, which divides every codeword, $c(x)$. In an (n, k) code, the degree of $g(x)$ is $(n-k)$. There is also a *generator matrix*, G, which is a $(k \times n)$ matrix, defined as

$$G = (I, P) \ , \tag{2.12}$$

where I is the $(k \times k)$ identity matrix and P is a $(k \times (n-k))$ *parity matrix*. This leads to the definition of the *parity-check matrix*, H, given by

$$H = (-P^T, I) \ , \tag{2.13}$$

where P^T represents the *transpose* of the parity matrix (i.e. the rows and columns are interchanged). Clearly,

$$GH^T = 0 \ . \tag{2.14}$$

Both the generator and parity-check matrices can be represented in a more systematic form for different codes, which facilitates easier calculation.

Another important quantity in error-correcting codes is known as a *syndrome*, $\vec{s}$. One way in which this can be calculated is

$$\vec{s} = \vec{c}H^T \ , \tag{2.15}$$

where $\vec{c}$ represents a codeword and H^T is the transpose of the parity-check matrix, as before. Clearly, $\vec{s}$ is an $(n-k)$ vector. If $\vec{c}$ is an uncorrupted codeword, then its syndrome should always be zero.

There are two main decoding techniques, known as *hard-decision decoding* and *soft-decision decoding*. The latter attaches probabilities to each bit transmitted which are then used on receipt to determine the most likely sequence of bits.

Finally, it is important to clearly distinguish between the two types of discrepancies present in error-correcting codes:

Errors: Unknown in value, number and location;

Erasures: Unknown in value, but known in number and location.

2.3.2 Reed-Solomon Codes

Reed-Solomon (RS) codes were first introduced in 1960 by I.S. Reed and G. Solomon [RS60] and are now one of the most practical codes in existence, used by both NASA (National Aeronautics and Space Administration) and ESA (European Space Agency), and also in compact discs [HHL+00, pp 127].

RS codes are a special instance of the cyclic BCH codes and can be viewed as non-binary BCH codes over a ground field GF(q). As with all BCH codes, m consecutive roots are chosen, but for RS codes, all of these roots are in the ground field itself, rather than an extension of the ground field [Pur95, p 59]. Thus, if α is a root in GF(q), the other roots can be represented by α^i, each of which has minimum polynomial $(x-\alpha^i)$. This leads to a very simple expression for the generating polynomial:

$$g(x) = \prod(x - \alpha^i) \ . \tag{2.16}$$

Now if we let $(n-k) = m$, then the distance

$$\begin{aligned} d &= m+1 \\ &= n-k+1 \ , \end{aligned} \tag{2.17}$$

which is the maximum possible value for d. Thus, RS codes are *maximum distance separable (MDS)* codes. The length of an RS code is $n = q-1$, as all the roots satisfy the equation $x^{q-1} = 1$. The parity-check matrix for RS codes can be written

$$H = \begin{pmatrix} \alpha^{n-1} & \alpha^{n-2} & \dots & \alpha^2 & \alpha & 1 \\ (\alpha^{n-1})^2 & (\alpha^{n-2})^2 & \dots & (\alpha^2)^2 & \alpha^2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^i & (\alpha^{n-2})^i & \dots & (\alpha^2)^i & \alpha^i & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ (\alpha^{n-1})^m & (\alpha^{n-2})^m & \dots & (\alpha^2)^m & \alpha^m & 1 \end{pmatrix} . \tag{2.18}$$

RS codes also handle groups of bits together, rather than single bits. As a result, a short burst of single-bit errors becomes one symbol error [Pur95, p 7]. This can be an advantage because bit errors do not always occur at random, uniform intervals; if errors have a specific cause, they occur in "bursts" with several consecutive bits affected.

2.3.2.1 Decoding Reed-Solomon Codes

The steps involved in correcting the errors in an RS code are essentially those of any non-binary BCH code, although they are simplified by the fact that all the roots are in the ground field, meaning only one non-binary field is required. In order to deal with both erasures and errors, it is necessary to modify the usual error-correcting technique (described in [Pur95, p 59]).

Let Y_i be the value of the i^{th} error at location X_i and $\tilde{Y}_i$ the value of the i^{th} erasure at location $\tilde{X}_i$. Let $d-1=2t$, where d is the distance of the code, and assume there are r errors and s erasures. These can be corrected, provided

$$2r+s \leq d-1 \ . \tag{2.19}$$

With these symbols, syndromes can be defined as

$$S_k = \sum_{i=1}^{s} \tilde{Y}_i \tilde{X}_i^{k+1} + \sum_{j=1}^{r} Y_j X_j^{k+1} \ , \tag{2.20}$$

where S_k is the syndrome arising from root α^k, and $k=0,\ldots,2t-1$. Now suppose that the $\tilde{X}_i$ satisfy

$$f_0 x^s + f_1 x^{s-1} + \cdots + f_{s-1} x + f_s = 0 \ , \tag{2.21}$$

where $f_0=1$. These f_i can be calculated from $\tilde{X}_i$, whose values are known. Now, for $1 \leq i \leq 2t-s$, let

$$\begin{aligned} T_i &= \sum_{j=0}^{s} S_{s+i-j-1} f_j \\ &= \sum_{j=0}^{s} f_j \Big(\sum_{k=1}^{r} Y_k X_k^{s+i-j-1} + \sum_{k=1}^{s} \tilde{Y}_k \tilde{X}_k^{s+i-j-1} \Big) \\ &= \sum_{k=1}^{r} Y_k X_k^i f(X_k) + \sum_{k=1}^{s} \tilde{Y}_k \tilde{X}_k^i f(\tilde{X}_k) \\ &= \sum_{k=1}^{r} Y_k X_k^i f(X_k) \ , \end{aligned} \tag{2.22}$$

as $f(\tilde{X}_k)=0$ by construction. If $E_k = Y_k f(X_k)$, then

$$T_i = \sum_{k=1}^{r} E_k X_k^i \ . \tag{2.23}$$

With these definitions in place, the correction process is as follows:

1. Calculate f_j for $j = 0, \ldots, s$, from the known erasure locators $\tilde{X}_k$, where $k = 1, \ldots, s$.

2. Determine the syndromes S_k for $k = 0, \ldots, 2t - 1$.

3. Find the values of $T_i = \sum_{j=0}^{s} S_{s+i-j-1} f_j$, for $i = 1, \ldots, 2t - s$.

4. Assuming that the error locators, X_k, are roots of a polynomial

$$p(x) = p_0 x^r + p_1 x^{r-1} + \cdots + p_r \ , \tag{2.24}$$

with $p_0 = 1$, calculate the values of p_i by solving

$$T_{i+r} + p_1 T_{i+r-1} + \cdots + p_r T_i = 0 \ , \tag{2.25}$$

for $i = 1, \ldots, t$ (or less if there are fewer than t errors). The Berlekamp-Massey algorithm [Mas69] (see Appendix C) can be used to do this.

5. Now solve $p(x)$ to find its roots, the error locators X_k, for $k = 1, \ldots, r$.

6. Determine E_k from $T_i = \sum_{k=1}^{r} E_k X_k^i$, using Gaussian Elimination with partial pivoting.

7. Using $E_k = Y_k f(X_k)$, calculate the error values, Y_k.

8. Finally, to find the erasure values $\tilde{Y}_i$, use the formula for S_k, relying again on Gaussian Elimination with partial pivoting.

The values of the erasures and the locations and values of the errors have now been found, so the corrupted codeword is restored to its original value.

2.3.3 Repetition Codes

Repetition codes are perhaps the simplest type of error-correcting code in existence. The data is expanded by a factor of x, meaning that each bit is

repeated x times in succession. If the data then becomes corrupted in any way, it is corrected by looking at each x bits in turn and determining whether 0s or 1s are more dominant. For example, if the data is expanded by a factor of 3, then every trio of bits containing two zeros is corrected to all zeros, and every trio consisting of two ones is corrected to all ones. The data is then reduced by a factor of 3 to restore the original codeword.

2.3.4 LDPC Codes

Low-density parity-check (LDPC) codes were first proposed in 1962 by R.G. Gallager [Gal62]. LDPC codes are linear block codes and are constructed using sparse random parity-check matrices. Shannon's Theorem [Sha48] states that almost error-free communication can be achieved if a suitable code is chosen in which the block-size, n, is sufficiently large and the code-rate is less than the capacity of the channel [Pur95]. Given an optimal decoder, LDPC codes approach the Shannon limits for a wide variety of channels [Mac99]. However, they are computationally complex to implement, and as a result, little work was done in this area until 1981, when Tanner [Tan81] produced a graphical representation of LDPC codes, known as Tanner graphs. In the mid-nineties, as a result of vastly improved computing power, research began again on LDPC codes, both expanding on the work done by Gallager [MN96] and proposing new types of codes, such as Tornado codes [LMS+97] or other irregular LDPC codes [RSU01].

If we consider the case of a binary LDPC code (which can easily be generalized to the non-binary case), the *weight*, w, of a matrix is the number of 1s it contains [Mac99]. The *density* of a source of random bits is the expected fraction of bits whose value is 1; such a source is considered to be *sparse* if its density is less than 0.5.

Definition A *low-density parity-check code* is a linear block code in which the parity-check matrix, H, is sparse with a low density of 1s. [Rya04]

An (n, k) LDPC code, with $m = (n - k)$, is said to be *regular* if H has exactly w_c 1s per column and $w_r = w_c(n/m)$ 1s per row, where $w_c << m$. Gallager showed that such codes have excellent distance properties, provided $w_c \geq 3$ and $w_r \geq w_c$. If H is low-density, but the number of 1s in the rows and columns is not constant, then it is an *irregular* LDPC code. Generally, irregular LDPC codes outperform regular LDPC codes [PT04].

In order to represent these codes graphically, a Tanner graph using a *bipartite* graphing system is constructed. This consists of *nodes* which are connected to *edges*; the nodes may be separated into two types, *variable nodes* and *check nodes*, and the edges may only connect two nodes of different types [Rya04]. Two nodes are considered to be *neighbouring* nodes if they are connected by an edge. There are m check nodes and n variable nodes. If an element h_{ij} in the parity-check matrix H is equal to 1, then check node i is connected to variable node j in the Tanner graph.

For example, if we consider an $(8, 4)$ linear block code in which we have

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \tag{2.26}$$

then the corresponding Tanner graph for this code is given in Figure 2.2. Note that this is an example of an *irregular* LDPC code, because the number of 1s in each column of H is not constant. A *cycle* of length x in a Tanner graph is a path consisting of x edges which loops back on itself. The smallest such cycle which can exist is a length-4 cycle. Cycles adversely affect the performance of the decoding algorithm for LDPC codes, as explained below.

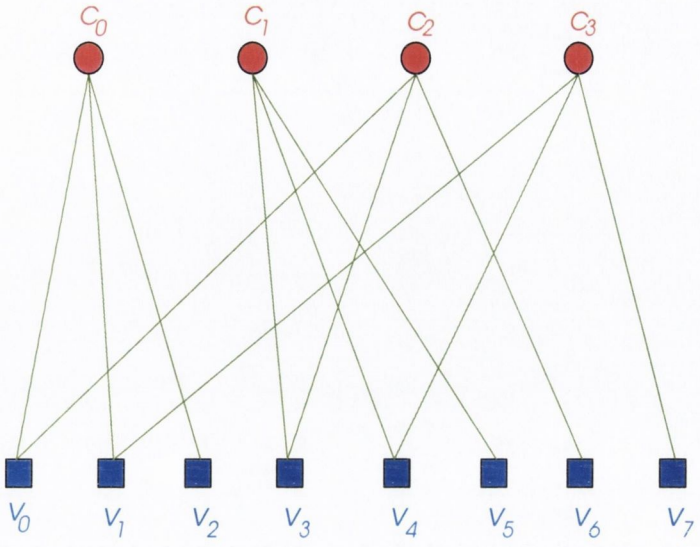


Figure 2.2: Tanner graph for an $(8, 4)$ linear block code, in which $c_0, \ldots, c_3$ represent the check nodes and $v_0, \ldots, v_7$ represent the variable nodes.

2.3.4.1 Decoding LDPC Codes

The decoding algorithm which is used for LDPC codes is an iterative probability-based approach, first suggested by Gallager [Gal62]. We shall outline this highly complex process briefly below; a fuller explanation of the algorithm can be found in Ryan's excellent introduction [Rya04] to LDPC codes.

The decoding procedure is known as the *message passing* algorithm and is based on the *a posteriori* probability that a given bit, b_i, in the codeword, $\vec{c}$, equals 1, given the received word, $\vec{r}$. This can be written as

$$\text{Prob}\ (b_i = 1 \mid \vec{r})\ . \tag{2.27}$$

A *message* is then the probability that a 1 or a 0 has been transmitted. There are two main stages of message passing, the first involving probabilities of variable nodes and and the second, probabilities of check nodes. The process is most easily explained in terms of Tanner graphs.

The first half-iteration of the algorithm consists of each variable node, v_i,

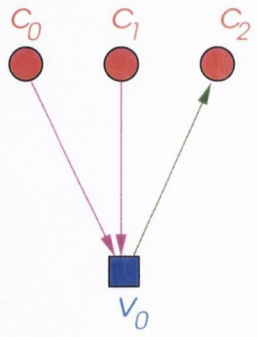


Figure 2.3: Section of Tanner graph corresponding to parity-check matrix H with column 0 given by $[1\ 1\ 1\ 0\ 0\ \ldots 0]^T$. Arrows show message passing from node v_0 to node c_2.

processing its input messages and sending the resulting output message to the neighbouring check nodes. The output message contains

$$\text{Prob}\ (v_0 = x \mid \text{input messages})\ , \tag{2.28}$$

where $x \in \{0, 1\}$. This is illustrated in Figure 2.3, for a parity-check matrix whose first column is represented by $[1\ 1\ 1\ 0\ 0\ \ldots 0]^T$. This means that there are edges from c_0, c_1 and c_2 to v_0. In this first stage, the variable node, v_0, processes messages from c_0 and c_1 and sends the result to c_2. Note that all the messages received by the variable node are taken into account, except that of the recipient check node, c_2, itself.

For the second half-iteration, the situation is reversed, with each check node, c_i, processing its input messages and passing the resulting output message to the neighbouring variable nodes, as shown in Figure 2.4. The output message in this case contains

$$\text{Prob (check equation } c_0 \text{ is satisfied} \mid \text{input messages})\ . \tag{2.29}$$

Again, this probability is based on all the messages received by the check node, c_0, excluding that of the variable node, v_3, itself.

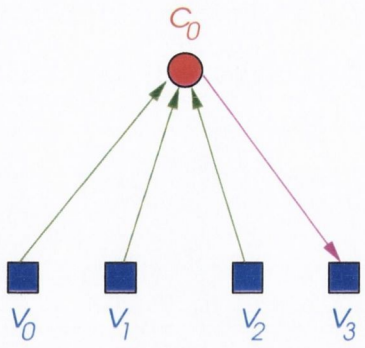


Figure 2.4: Section of Tanner graph corresponding to parity-check matrix H with row 0 given by [1 1 1 1 0 0 ... 0]. Arrows show message passing from node c_0 to node v_3.

The algorithm continues thus until a specified number of iterations has been completed or a certain criterion has been satisfied, for example $\vec{c^*}H^T = 0$, where $\vec{c^*}$ is a possible codeword [Rya04]. At this stage, the decoder calculates the *a posteriori* probability which will be used to make decisions about the bits b_i.

It is important that there are no cycles present in an LDPC code, as their presence violates the assumption that the messages are statistically independent of each other. Length-four cycles are particularly damaging to the decoding process, although not all examples may be harmful [LTK+01].

2.3.5 Hermitian Codes

Hermitian codes belong to a larger class of codes known as *algebraic geometry* codes, of which Reed-Solomon codes are a simpler example. The mathematics behind algebraic geometry codes is highly complex and will not be covered in detail here, but can be found in [Mus01]. We shall instead outline the structure of Hermitian codes, relying heavily on the explanations provided in [HvLP98], [O'S00] and [Pop02].

Consider the field $K = \mathrm{GF}(q^2)$ and the ring of polynomials in two variables $K[U, V]$. Hermitian codes are based on *Hermitian curves*, which are defined over K by the inhomogeneous equation

$$U^{q+1} + V^{q+1} + 1 = 0 \ . \tag{2.30}$$

This gives us the homogeneous equation

$$U^{q+1} + V^{q+1} + W^{q+1} = 0 \ , \tag{2.31}$$

which has $(q + 1)$ points at infinity. We can rewrite equation (2.31) as

$$U\bar{U} + V\bar{V} + W\bar{W} = 0 \ , \tag{2.32}$$

where $\bar{U} = U^q$ represents the conjugate of U. The similarity between this and equating a Hermitian form over the complex numbers to zero has resulted in these curves becoming known as "Hermitian" curves [HvLP98, page 879].

It is more convenient, when constructing a code, to have exactly one point at infinity, rather than $(q + 1)$, and so we transform equation (2.31) to get

$$X^{q+1} = Y^q Z + Y Z^q \ , \tag{2.33}$$

where $X = W$, $Y = bU + abU - aV$ and $Z = bU - V$, with $a^q + a = -1$ and $b^{q+1} = -1$. This gives us the inhomogeneous equation

$$X^{q+1} = Y^q + Y \ , \tag{2.34}$$

in which $P_\infty = (0 : 1 : 0)$ is the only point at infinity. A detailed derivation of these equations can be found in [HvLP98, p. 879].

For the Hermitian curve, the *genus*, or complexity of the curve, is

$$g = \frac{q(q-1)}{2} \ , \tag{2.35}$$

and there are $n = q^3$ solutions to this equation over $\mathrm{GF}(q^2)$.

Definition Let P_i be the n points on the curve with co-ordinates $(x_i : y_i : 1)$, for $i = 1, \ldots, n$. A *divisor*, D, of a curve is defined to be

$$D = \sum_i a_i P_i \ , \tag{2.36}$$

with $a_i \in \mathbb{Z}$ and $a_i = 0$ for all but a finite number of points P_i. The *support* of a divisor is then the set of points with nonzero coefficient.

For Hermitian codes, we have two divisors,

$$D = \sum_{i=1}^{n} P_i \text{ and } G = mP_\infty \ , \tag{2.37}$$

which have disjoint support. Focusing on codes in which G has only a single point in its support allows for the longest code lengths and so, we will restrict ourselves to these.

Definition Let $L(G) = L(mP_\infty)$ be the set of rational functions on a Hermitian curve with a pole of multiplicity at most m in P_∞. Then $L(G)$ is a finite-dimensional vector space over GF(q), and $l(G)$ is the dimension of this space. Let a be a non-negative integer. If

$$l(aP) = l((a+1)P) \ , \tag{2.38}$$

then a is a *gap* of P. Any non-negative integer that is not a gap is called a *nongap* of P [HvLP98, p. 891].

Let x and y be defined as the rational functions X/Z and Y/Z respectively. The functions x and y have poles of order q and $(q+1)$ respectively at P_∞. Let $\rho(x^i y^j) = iq + j(q+1)$. The set of nongaps for our code is then

$$\Lambda = \{\rho(x^i y^j) : i, j \geq 0\} \ . \tag{2.39}$$

Thus, a basis for the vector space $L(G)$ is given by

$$L(G) = \langle x^i y^j : i \geq 0, 0 < j \leq q \text{ and } \rho(x^i y^j) \leq m \rangle \ . \tag{2.40}$$

We can then define our Hermitian code to be

$$C_m = C_\Omega(D, G) = \{\vec{c} \in \mathrm{GF}(q^2)^n : \sum_{k=1}^{n} f(P_k) c_k = 0, f \in L(G)\} \ . \tag{2.41}$$

We only include points with non-zero co-ordinates in our code, giving us a code-length of $(q^3 - q)$. To encode this, we must solve the system of equations

$$H\vec{c} = 0 \ , \tag{2.42}$$

where $\vec{c}$ is the codeword, and H the parity-check matrix, in which $H_k = f(P_k)$. If our received vector, $\vec{r} = \vec{c} + \vec{e}$, then we express the syndromes as

$$\begin{aligned} S_e(f) &= \sum_{k=1}^{n} r_k f(P_k) \\ &= \sum_{k=1}^{n} e_k f(P_k) \ . \end{aligned} \tag{2.43}$$

Implementation-wise, this can be accomplished with similar complexity to that of q Reed-Solomon codes. The same hardware can be used to implement both encoding and syndrome generation, which is a significant saving [Pop02].

In addition, the minimum distance, $d(C_m)$, of a Hermitian code is

$$d(C_m) \geq n - k(C_m) + 1 - g \ . \tag{2.44}$$

Clearly, this is smaller than in Reed-Solomon codes, in which $d = n - k + 1$, but the length of Hermitian codes can be significantly greater. An outline of the various parameters possible for Hermitian codes can be found in [YK91].

2.3.5.1 Decoding Hermitian Codes

The decoding process in Hermitian codes is complex and several different solutions have been proposed, although many of these have proved particularly difficult to implement in hardware. Therefore, we shall give a brief overview of the decoding approach successfully employed by Popovici [Pop02] to implement decoding using FPGA technology on a microchip.

The approach is divided into two main sections: finding the location of the errors and finding their values. As with Reed-Solomon codes, we begin by calculating the syndromes, using equation (2.43). The next challenge is to compute the error locator polynomial. Kötter [K96] developed a method of using a generalisation of the Berlekamp-Massey [Mas69] algorithm, known as the Sakata algorithm [Sak90], in order to calculate this polynomial, provided all the syndromes are known. However, only $(m + 1)$ of the syndromes will have been successfully computed from equation (2.43). Therefore, an additional procedure known as *majority voting* [FR93] is used to calculate any remaining syndromes. In this way, the error locations can be obtained.

When this has been completed, the values of the errors must be determined. This is also done in a similar manner to Reed-Solomon codes, by calculating an error evaluator polynomial which results in the error values being decoded. Popovici has designed this in such a way that the entire decoder can fit onto an FPGA chip. A more detailed description of this procedure is outside the scope of this thesis but can be found in [Pop02].

We shall now go on to consider the application of the three areas discussed in the background chapter to the overall idea explored in this thesis.

Chapter 3

Related Work and Motivation

"One never notices what has been done;
one can only see what remains to be done." - Marie Curie

In order to design a biometric key retrieval scheme suitable for use in public-key cryptosystems, there are numerous important issues which must be examined. We must first consider the state of the art in five relevant areas: biometric key retrieval schemes already in existence; predicted lengths of private keys in PKC systems; studies of the effects of partial key exposure in RSA; a detailed overview of Daugman's iris recognition algorithm; and the use of error-correcting codes to correct biometric readings.

3.1 Biometric Key Retrieval

Originally, biometric key retrieval schemes involved the use of a trusted bit-replacement algorithm to hide a cryptographic key within the enrolment template. On verification, the key bits would be retrieved from the appropriate locations using this secret algorithm, and the key would then be available for use. However such methods are vulnerable to attack as the key is retrieved

from the same location in a template for each different user in the system.

Bodo [Bod94] suggested another strategy: to derive the key directly from the biometric template. However, such a template is not sufficiently consistent to be used as a key, and if it is ever compromised, it cannot be changed or revoked, a major disadvantage.

Another approach, "Biometric Encryption", [SRS+01] uses a fingerprint reading together with a "token" to retrieve a cryptographic key. The secure data token is known as the "Bioscrypt", and is generated mathematically by combining the user's biometric reading and the key when the user first enrols. Neither the key nor the biometric reading can be recreated by an attacker who gains access to the token. The system is based on a complicated mathematical process involving correlation and Fourier transforms. However, the maximum number of bits which can be produced is approximately 128 [Sou01]. Thus, in order to implement this scheme in PKC, it is necessary to generate a 128-bit symmetric key, and use this to encrypt the user's private key for a public-key cryptosystem.

Concurrently, but independently, a team in Cambridge [HAD06] (including Daugman, whose iris recognition scheme is described in this chapter) proposed a biometric key retrieval scheme similar to that outlined in this thesis. They use error-correcting codes to correct the errors in iris readings, but can only protect a 140-bit key. We contrast these schemes in Chapter 8.

3.2 Key Sizes in Public-Key Systems

As mentioned in the previous chapter, public-key systems in use today can be split into three main types - those that rely on the intractability of integer factorisation (IF) for their security; those that rely on the discrete-log

problem (DL); and those that are based on elliptic-curve implementations. We briefly review the main findings of some major recent studies in relation to the predicted key-sizes necessary for such systems.

Lenstra and Verheul [LV01] produced a detailed estimate of comparative key sizes across various cryptosystems. Four main factors are taken into account: the length of time that information must remain secret; the security margin, defined as the year until which a user trusts DES [Nat93] (publicly broken in 1997); the projected computing environment; and predicted advances in cryptanalysis. Their estimates, partially reproduced in Table 3.1, assume that DES was only safe until 1982, although formulae are provided to alter these values if it is felt that DES should be trusted for longer.

Year	IF PK System Modulus Size		SDL PK System	Elliptic Curve
	Comp-Based	Cost-Based	Subgroup Size	Key Size
2010	1369	1056	138	146/160
2020	1881	1472	151	161/188
2030	2493	2016	165	176/215
2040	3214	2656	179	191/244

Table 3.1: Minimum key sizes for the given time periods, as recommended by Lenstra and Verheul [LV01]. IF refers to integer-factorisation-based schemes; SDL to subgroup discrete log systems. Comp-based are computationally-based estimates, significantly higher than cost-based estimates.

Two different estimates are given for the projected size of moduli in IF PK cryptosystems; the first based on computational considerations and the second on a cost-based analysis. The SDL PK systems in the table are the branch of PK cryptosystems which rely on the difficulty of computing dis-

crete logs in a *subgroup* of the multiplicative group of a finite field, of which DSA [Nat00] is the most well-known example. Their study excludes systems, such as ElGamal [ElG85], based on the "computational Diffie-Hellman assumption", viewed as being a stronger assumption than the problem of solving discrete-logs. Two separate values are given for the elliptic-curve cases, based on differing projected rates of progress made in attacking such systems. The "key size" here refers to the subgroup size needed.

In the last year, NIST (National Institute of Standards and Technology) [BBB+05] and ECRYPT (European Network of Excellence in Cryptology) [Eur05] independently produced reports outlining their recommended minimum key sizes for various types of cryptosystems. The relevant information is summarised in Tables 3.2 and 3.3.

Year	IF PK System Modulus Size	SDL PK System Subgroup Size	Elliptic Curve Key Size
Up to 2010	1024	160	160
Up to 2030	2048	224	224
Beyond 2030	3072	256	256

Table 3.2: Minimum key sizes, as recommended by NIST [BBB+05].

Year	IF PK System Modulus Size	SDL PK System Subgroup Size	Elliptic-Curve Key Size
Up to 2025	1248	160	160
Up to 2035	2432	224	224
Beyond 2035	3248	256	256

Table 3.3: Minimum key sizes, as recommended by ECRYPT [Eur05].

As can be seen from these tables, the required key sizes for IF PK systems are far higher than those of the other two systems. This will lead us to create the notion of "partial hiding", which builds on work done in the area of partial key exposure.

3.3 Partial Key Exposure

Several papers have been produced on the partial exposure of the private key in RSA. Although they focus on the security of the system following an *unintentional* exposure of part of the private key, they are relevant to our proposal of deliberately revealing sections of the RSA private key, as they describe possible attacks which may result.

In 1998, Boneh, Durfee and Frankel (BDF) [BDF98] explored several attacks on RSA following partial exposure of the private key. Interestingly, for low-exponent RSA, they showed that the most significant half of the bits of the private key d are automatically "leaked" and that this has no adverse effect on security. Their results for low-exponent RSA also show that only a quarter of the LSBs of d are sufficient for an attacker to obtain all of d.

However, Steinfeld and Zheng [SZ04] went on to show that this result only holds if no more than the least significant bit of p and q is equal, where p and q are primes such that the modulus $N = pq$. In the case of low-exponent RSA in which the m LSBs of p and q are identical, they show that the estimated running time for the BDF attack is too low by a factor in the order of 2^m, making it ineffective for such systems. BDF's erroneous estimate is as a result of an incorrect application of Hensel lifting. Hensel's lemma [NZM91, p. 87] is only applicable if the solutions of an equation are non-singular, whereas BDF use it for an equation in which all the solutions are singular. However,

this does not affect the majority of parameter choices for low-exponent RSA, in which p and q are chosen randomly and independently; indeed, p and q are only likely to have m equal LSBs if they are carefully chosen to do so.

Steinfeld and Zheng go on to prove that, if a system with m equal LSBs in p and q is secure with no bits exposed, it is secure if up to $2m$ LSBs of d are exposed. We shall make use of this result in our subsequent calculations.

Hinek, Low and Teske [HLT03] investigate multi-prime RSA (3 primes or more) in relation to partial exposure of the private key. They claim that the BDF attack is completely ineffective for 3-prime RSA as the number of solutions is exponential in the bit-size of the RSA modulus.

Blömer and May presented a paper entitled "New Partial Key Exposure Attacks on RSA" [BM03], in which they produce even stronger results for RSA than those suggested by BDF. In addition, they also produce attacks on partial key exposure in the Chinese Remainder Theorem (CRT) version of RSA [QC82], suggesting that for low-exponent RSA, half the bits of $d_p = d \bmod (p-1)$ are sufficient to factorize the modulus N.

Canetti et al [CDH+00] suggested using *All-Or-Nothing Transforms* (AONTs) and *Exposure-Resilient Functions* (ERFs) to protect a secret key even when *almost all* of the secret key has been exposed. An AONT is defined [Riv97] as an efficiently computable transformation T on strings such that

- for any string x, given *all* of $T(x)$, one can efficiently recover x
- there exists some threshold l such that any polynomial-time adversary that (adaptively) learns all but l bits of $T(x)$ obtains *no* information about x.

From this, an ERF is described as being a deterministic function whose output appears random even if *almost all* the bits of the input are revealed.

Instead of storing a secret key as usual, the key is stored with the AONT applied to it. If an AONT can be created with the threshold value l far lower than the size of the output of the AONT, then security against almost total exposure can be obtained. In order to accomplish this, the AONT is based on a suitable ERF.

3.4 Daugman's Iris Recognition Algorithm

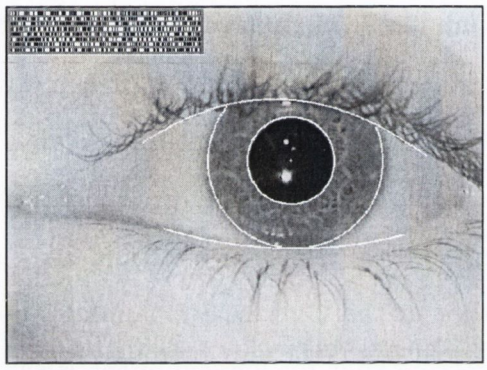


Figure 3.1: Iris image processed using Daugman's algorithm, with IrisCode visible in top left corner. The image was taken monochromatically in near-infrared light in the 700-900nm band at a rough distance of 35cm. [Dau93]

In an iris recognition scheme, it is important to accurately locate the iris within the captured image. The advised equipment and conditions for obtaining such an image in Daugman's algorithm are explained in detail in [Dau04, p 2]; for our purposes, it is sufficient to note that such imaging takes place under near-infrared wavelengths designed to eliminate the necessity of shining visible light into a subject's eye. This makes it more difficult to discern the inner and outer boundaries (pupil and limbus) of the iris as, under infrared light, the iris and sclera often appear to be of almost

identical brightness. A further complication is that the pupil and iris are not necessarily concentric; frequently, the pupil centre is nasal and inferior to the iris centre, with a radius ranging from 0.1 to 0.8 of the iris radius [Dau04].

To detect the boundary of the pupil, Daugman proposes an optimization solution, which searches for the maximum "spike" in the rate-of-change of the summed brightness of a circle, for a series of circles of increasing radii and varying centre points [Dau04]. A circle whose centre point equals that of the pupil should show a considerably larger "spike" than other circles. This is implemented by using the integrodifferential operator

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{r,x_0,y_0} \frac{I(x,y)}{2\pi r} ds \right| \quad , \tag{3.1}$$

where $I(x,y)$ represents a raw input image as described in [Dau93]. This operator searches the image domain (x,y) for the maximum partial derivative of the normalized contour integral of $I(x,y)$ along a circular arc ds of radius r and centre coordinates (x_0,y_0), with respect to increasing radius. The symbol $*$ represents a convolution, an integral which expresses the amount of overlap of one function as it is shifted over another. The integrodifferential operator is convoluted with a radial Gaussian

$$G_\sigma(r) = \frac{1}{\sqrt{2\pi}\sigma} e^{-((r-r_0)^2/2\sigma^2)} \quad , \tag{3.2}$$

with centre r_0 and standard deviation σ, which acts as a smoothing function.

The overall effect of this operator is to detect circular edges by searching with increasing radius at successively finer scales of analysis through the three parameter space of centre coordinates (x_0,y_0) and radius r. A similar approach is used to determine the outer boundary of the iris and to detect the presence of eyelids, with the circular path replaced by an arcuate one. This isolates the iris from the rest of the image.

3.4.1 Projected Polar Coordinate System

In order to create a system that is invariant to the size, location and orientation of the iris within an image and the size of the pupil within the iris, Daugman uses a doubly dimensionless projected polar coordinate system, in which both the polar variable and the radial variable are dimensionless.

Each point in the iris is assigned a pair of dimensionless real coordinates (r, θ), where $r \in [0, 1]$ and $\theta \in [0, 2\pi]$. Thus, each iris image $I(x, y)$ is mapped to a doubly dimensionless polar coordinate system by

$$I(x(r, \theta), y(r, \theta)) \rightarrow I(r, \theta) \ , \tag{3.3}$$

where $x(r, \theta)$ and $y(r, \theta)$ are linear combinations of the set of pupillary boundary points $(x_p(\theta), y_p(\theta))$ and the set of limbus boundary points $(x_s(\theta), y_s(\theta))$:

$$\begin{aligned} x(r, \theta) &= (1 - r)x_p(\theta) + rx_s(\theta) \ , \\ y(r, \theta) &= (1 - r)y_p(\theta) + ry_s(\theta) \ . \end{aligned} \tag{3.4}$$

Because the radial variable r varies from the inner (pupillary) to the outer (limbic) boundaries as a unit interval, it corrects for the elastic pattern deformation in the iris whenever the pupil changes its size.

3.4.2 IrisCode Creation

Once the iris has been located, the IrisCode is constructed by demodulation of the iris pattern. The aim of this process is to obtain the phase information. Complex-valued 2D Gabor wavelets are used to create a sequence of phasors from the iris (where a phasor is a vector in the complex plane). Each phasor angle is quantized into the complex quadrant in which it lies for each local patch (r_0, θ_0) of the iris, and this is done across the iris at many different

scales (α, β, ω) of analysis [Dau99]. These Gabor filters are defined as:

$$H(r,\theta) = e^{-i\omega(\theta-\theta_0)} e^{-(r-r_0)^2/\alpha^2} e^{-(\theta-\theta_0)^2/\beta^2} \quad , \tag{3.5}$$

where α and β are constants representing the wavelet size, ω is the frequency (inversely proportional to β) and (r_0, θ_0) are the polar coordinates of the iris portion for which the phasor coordinates are to be calculated.

Each bit h in the IrisCode can be treated as a coordinate of one of the four vertices of the logical unit square in the complex plane. The value of h is calculated by projecting the relevant section of the iris onto these complex-valued 2D Gabor wavelets:

$$h_{\{\Re,\Im\}} = \text{sgn}_{\{\Re,\Im\}} \int_\rho \int_\phi I(\rho,\phi) e^{-i\omega(\theta_0-\phi)} e^{-(r_0-\rho)^2/\alpha^2} e^{-(\theta_0-\phi)^2/\beta^2} \rho d\rho d\phi \tag{3.6}$$

where $h_{\{\Re,\Im\}}$ is a complex-valued bit whose real and imaginary bits are 1 or 0 (represented by sgn), depending on the sign of the 2D integral. The raw iris image is denoted $I(\rho, \phi)$. This is repeated until 2048 bits (256 bytes) have been extracted from the iris.

In the most recent revision [Dau04], a further 256 "masking bytes" are computed, signifying which regions are obscured by eyelashes, eyelids, specular reflections and so forth, and should be ignored in the demodulation code. No information has yet been made available as to the formation or exact structure of this IrisMask.

Although the system at this stage has achieved invariance to the position and size of the iris, and the pupillary dilation, it does not correct for the orientation of the iris within the image. To do this, the phase code is compared at many discrete orientations by cyclic scrolling of the angle.

3.4.3 Test of Statistical Independence

Recognition of irises by their IrisCodes depends on the *failure* of a test of statistical independence. Two events are said to be *statistically independent* if the occurrence of one event does not influence the likelihood of occurrence of the other [MvOV97, pp 51]. Applying this definition to random variables, it means that knowledge about the value of one random variable tells us nothing about the value of the other. This test of statistical independence is designed so that if two IrisCodes from two different irises are compared, it is virtually guaranteed to be passed, whereas it will uniquely be failed if two IrisCodes from the same iris are compared.

In order to implement this test, the logical operations of Exclusive-Or (XOR), represented by $\otimes$, and AND, represented by $\cap$, are used. The outputs of these operations are summarised in Table 3.4. Firstly, the two IrisCodes

Input A	Input B	Output A $\otimes$ B	Output A $\cap$ B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 3.4: Outputs of logical binary operations XOR ($\otimes$) and AND ($\cap$)

are XORed together, to identify any corresponding bits that differ. The result is then ANDed with each of the masks, to ensure that only bits which are considered to have been uncorrupted in both IrisCodes are included. A fractional *Hamming Distance (HD)* is used as a measure of the dissimilarity between the two irises. The HD between two vectors is defined as the number

of bit positions at which the two vectors have differing values [Pur95, pp 7]:

$$\mathrm{HD} = \frac{\|(\mathrm{IrisCode}_A \otimes \mathrm{IrisCode}_B) \cap \mathrm{IrisMask}_A \cap \mathrm{IrisMask}_B\|}{\|\mathrm{IrisMask}_A \cap \mathrm{IrisMask}_B\|} \quad . \tag{3.7}$$

Because the denominator computes the number of bits remaining once both masks have been taken into account, a score of 0 represents a perfect match.

3.4.4 Hamming Distance Threshold

In Table 3.5, the single false match probabilities for various HD criteria are generated. Based on this data and numerous studies, Daugman states that

Hamming Distance Criterion	Odds of False Match
0.29	1 in 13 billion
0.31	1 in 185 million
0.33	1 in 4 million
0.35	1 in 133,000

Table 3.5: False match probabilities for various HD criteria. [Dau04]

the most suitable HD threshold for iris recognition is 0.333. This means that even quite a poor match between corresponding bits, with slightly less than 70% agreement, can be very convincing proof of an iris match. As seen in the table, the odds of a single false match are low enough to allow very high confidence levels, even when large databases are being searched. However, a further consideration when choosing a HD threshold is that significant ethnic differences exist in the amount of the iris visible in an image. Daugman estimates that 79% of the iris is visible in a typical Caucasian, due to occlusion from the upper eyelid and the effect of gravity. However, in a Japanese or Korean individual, only about 64% is typically visible, as occlusion from the lower lids also occurs [Dau02]. If fewer bits are available to compare, the

likelihood of a false match is increased, as random agreement is statistically more likely. This means that using raw HD as a threshold is unsuitable. Thus, Daugman's search engine automatically adjusts the acceptable HD depending on the number of valid bits to be compared [Dau01].

Number of Bits Compared	Approximate % of Iris Visible	Maximum Acceptable Fraction of Bits Disagreeing
500	43%	0.27
700	61%	0.31
911	**79%**	**0.33**
1000	87%	0.34
1152	**100%**	**0.35**

Table 3.6: Comparison between the number of bits compared (approximate amount of the iris visible) and the maximum number of bits that may disagree without affecting the validity of the match. With less than 911 bits, the HD is adversely affected, whereas with 1152 bits or more, 100% visibility of the iris can be assumed. [Dau01]

Table 3.6 displays the advantage of having as large a portion of the iris visible as possible in order to ensure a valid match. The higher the number of bits compared, the higher the fraction of bits that can differ and still provide an accurate match. Two significant points are highlighted: if less than 911 bits remain after invalid bits are excluded, then the HD will be adversely affected, but if 1152 bits or greater are compared, this is considered to be the equivalent to 100% visibility of the iris [Dau01].

3.4.5 Security of Daugman's Algorithm

As with all biometrics, countermeasures must be in place to ensure that an attacker cannot fool the system and impersonate another user. In an extreme case, such an attack may involve using a dead iris; in a less dramatic situation, a contact lens with a fake iris pattern printed on it may be worn. Fortunately, both attacks are prevented by the design of Daugman's algorithm which exploits a trait which might otherwise have been deemed an impediment - namely, the constant oscillations in pupil size ("hippus") that occur even in unchanging light [Adl65]. Observing changes in pupil diameter relative to iris diameter over a sequence of frames provides a simple test against any such attacks [Dau93, p 1150].

A further protection against fake irises on contact lenses [Dau00, pp 14-16] details how printing a pattern onto a contact lens adds an identifiable signature to the 2D Fourier power spectrum, which can thus be used to detect any such attempted falsifications. This also suggests protection against an attack [TKZ02] in which a digital image of an iris is printed onto a sheet, a hole is cut in a suitable position, and the attacker's eye positioned behind the sheet. The hippus in the attacker's eye can be observed by the system, successfully bypassing the initial security check. However, presuming that a similar effect is observed when a digital image is printed onto a sheet as when it is printed onto a contact lens, it should be possible to detect such an attack by the spurious energy visible in the Fourier power spectrum.

3.5 Error-Correction in Biometrics

The use of error-correcting codes to correct iris readings and compare them with previously-stored templates was first suggested by Davida, Frankel and

Matt [DFM99]. Their scheme consists of generating M biometric templates for each user and using majority decoding to obtain a k-bit template, to which check digits are then appended. For correction purposes, they suggest the use of bounded distance decoding, in which a count of the Hamming weight of the error vector is calculated. However, their scheme is not practical in its presented form because it lacks sufficient error-tolerance [JW99]. It also requires that each biometric scan be repeated numerous times, assuming that the errors in the scans are entirely independent.

Juels and Wattenbery [JW99] produced a "fuzzy commitment" scheme using fingerprints, which they suggest could also be applied to some type of biometric key retrieval scheme, with minor modifications. Rather than taking numerous readings, as is done in [DFM99], they instead treat the biometric input as a corrupted codeword.

Dodis et al [DRS04] describe a general technique for creating strong cryptographic keys from noisy data such as biometrics, relying on a range of areas such as error-correction, randomness extraction and password authentication.

Martini and Beinlich [MB03] suggested the use of low density parity check (LDPC) codes to create a "virtual PIN" from an iris scan. However, the exact details are not available and so, detailed comment must be reserved.

As mentioned previously, the concurrent work of the Cambridge team [HAD06] proposed the use of error-correcting codes to correct iris readings. They suggest the use of Reed-Solomon codes in conjunction with Hadamard codes for this purpose.

3.6 Motivation to Address Biometric Key Retrieval for Public-Key Cryptosystems

The protection of private keys within public-key cryptosystems is of paramount importance from a security point-of-view. However, this must be achieved in such a way that a legitimate user can readily access his private key whenever necessary. This means that the systems need to use some type of identification scheme to distinguish between such a user and an attacker. Methods of personal identification can be divided into three main types [LW01]:

1. Something you know (password, PIN, personal information).
2. Something you have (smart-card, key-card, token).
3. Something you are (biometric).

There are numerous disadvantages associated with the first two types: passwords frequently consist of easily-guessed words or phrases, or are written beside the user's computer; PINs may be forgotten, or observed while being typed into a numeric key-pad; key-cards may be lost, or loaned to an unauthorised individual. While no system is ever entirely secure, biometric schemes provide us with a more reliable and convenient means of personal identification. In some cases, there is no need for a user to possess any additional token or knowledge in conjunction with the biometric; in other cases, a biometric may be combined with one of the first two types of identification for increased security.

Biometric key retrieval schemes to date have not succeeded in reliably reproducing a sufficient number of bits to be directly used in public-key cryptosystems. Biometric Encryption [SRS+01] can deal with a maximum of 128 bits, while even the very recent work by the Cambridge team [HAD06]

has an upper limit of 140 bits. The only solution currently proposed is to use a symmetric key (generated, for example, by AES [DR02]) to encrypt the private key from a public-key cryptosystem, and subsequently protect the symmetric key with the biometric reading.

Although an algorithm such as AES can be implemented at high-speed, it adds an extra layer of complexity to the overall system. However, a far more serious objection to this approach lies in the fact that, while AES-128 (which uses a 128-bit key) is most commonly used today, AES-192 or AES-256 (with 192-bit and 256-bit keys respectively) are advisable for highest security level implementations [NSA03]. Neither of these could currently be protected using biometric key retrieval, nor could they be used to protect PKC systems. Therefore, the advantages of creating a biometric key retrieval scheme that can deal directly with larger key sizes are twofold:

1. It allows us to directly accommodate the private keys in PK sytems, without the added complexity and decrease in speed associated with using an algorithm such as AES to encrypt the private keys, before encrypting the symmetric key with the biometric.

2. It is a necessary step to ensure that biometric key retrieval can be used to protect and retrieve symmetric keys in higher-security symmetric systems, such as AES-192 or AES-256.

Given that the main advantage of introducing biometrics to protect cryptographic keys is to increase the overall security, it would constitute a significant advance in the field if we were to create a scheme that can accommodate high-security key sizes.

Chapter 4

Design of Biometric Key Retrieval Scheme

"The big lie of computer security is that security improves by imposing complex passwords on users. In real life, people write down anything they can't remember. Security is increased by designing for the way humans actually behave." - Jakob Nielsen

In our initial design for a biometric key retrieval scheme, there are three major decisions which must be made: the size of private key which we aim to protect and retrieve; the biometric we will use to do so; and the most suitable error-correcting code to correct readings of this biometric to ensure repeatability. We will now look at each of these issues in turn.

4.1 Choice of Key Size

As mentioned, it is important to estimate the key-sizes that will need to be protected. In the previous chapter, we explored three of the most recent studies on predicted key-sizes across various types of public-key cryptosystems.

In the case of RSA, our private key, d, will be chosen to be approximately as large as the modulus, for maximum security. All three studies predict that the commonly-used modulus size of 1024 bits will only be secure for another couple of years, and so we choose to protect a key size of 2048 bits for RSA. In systems relying on the subgroup-discrete-log problem or elliptic-curve-based systems, key sizes that are ten times smaller are predicted to be secure for another twenty years, and so we will aim to protect key sizes of that order. Given the enormous size difference between keys for such systems and those for RSA, it is likely that an additional mechanism will be required to deal with the latter, but we discuss this further in the next chapter.

4.2 Choice of Biometric

If we are to use biometric readings to protect and retrieve a private key, and also introduce error-correcting codes as an effective means of matching these biometric templates, it is vital that this be done in conjunction with a stable biometric and a highly dependable biometric processing system. Unlike many other biometrics (notably fingerprints), the iris benefits from a high level of natural protection in the body [Dau93], being unaffected by dirt, weather conditions, sweat and various other external influences which can adversely affect a reading. Although it is an internal organ, it is externally visible, which allows an image to be taken in an unintrusive fashion. It also contains an intrinsic polar geometry, which makes it significantly easier to apply a natural coordinate system to the iris than to other biometrics.

While numerous algorithms for iris recognition have been suggested (as outlined in the background chapter), none has come close to the impressive reliability demonstrated over the last decade by Daugman's approach. An

in-depth report was produced by the U.K. National Physical Laboratory in 2001 [MKCK01], following a detailed study of the performance and reliability of various biometrics. In 2.73 million cross-comparisons, there were no false matches reported for Daugman's iris recognition technology. In addition, each iris reading is processed into a 256-byte IrisCode by Daugman's algorithm, providing us with ample bits for use in key retrieval. Therefore, we propose to use his method to segment the iris from the image and process the iris reading into a biometric template, but we then suggest a more secure means of iris matching, involving error-correcting codes.

4.3 Choice of Code Type

Although Daugman's use of Hamming Distances to match iris readings seems to effectively differentiate between individuals' identities, it would be advantageous to introduce error-correcting codes in their place. The principal benefit from this approach is that we avoid storing the user's entire biometric template explicitly. By representing the iris reading as the information symbols of a codeword, we can hash the codeword and store only this hashed entity, along with the corresponding check digits, without losing any of the information contained in the reading. Given that the potential theft of biometrics is one of the main objections to their usage, this would greatly increase the security of the system. A further advantage of this approach is that it reduces the overall amount of data to be stored, which is particularly welcome in situations involving large databases of individuals or in which limited storage space is available, for example in low-capacity smart cards.

In order to determine the most suitable code type to correct the errors in iris readings, several issues must be taken into account:

1. The *type* of data with which we will be working;

2. The *predicted error rates* within the data;

3. The *dispersion* of these errors within the data.

We will not be dealing with a continuous stream of data, such as might occur in a satellite transmission, but rather with a limited amount of data from each iris reading. The predicted error rates within readings will affect the *parameters* of our code choice, more so that the code type itself; however, the dispersion of the errors is a major factor in our decision. We now presume the following:

Assumption: The errors within iris readings occur in bursts.

The basis for this assumption is a communication from Daugman that suggested that the errors within iris readings tend to occur in bursts, for two main reasons [Dau02]:

1. Obstructions such as eyelashes and eyelids will affect a number of bits in succession;

2. Infrared light makes it difficult to distinguish between the iris and sclera, which can lead to slight displacements of the polar coordinate system, causing errors at the erroneous boundary.

As a result, bit errors are not randomly dispersed throughout the data, but rather are clumped together in bursts. We will go on to prove this assumption in Chapters 6 and 7.

We need a code type which deals efficiently with bursts of errors in relatively short data strings in order to make the system more trustworthy.

Reed-Solomon codes are specifically designed to deal with bursts of errors, as several consecutive bit errors result in only one symbol error. They have a fixed threshold below which errors are guaranteed to be corrected, which is a desirable characteristic in biometric recognition, as it leads to greater consistency and accuracy. They have been in use for many years and have proven themselves to be highly reliable for numerous applications. However, as a direct result of this longevity and to ensure that we are working with the optimum technology available today, it would be advisable to now consider whether some more recent codes, such as LDPC or Hermitian codes, might be more suitable for our application. Therefore, we produce comparisons between these codes and RS codes, to determine the best choice for our biometric key retrieval scheme.

4.3.1 RS Codes vs Hermitian Codes

As Hermitian codes are only beginning to emerge as a viable choice, few performance comparisons with RS codes have yet been produced. One early such comparison [WJ97] concluded that, although Hermitian codes show great potential for applications such as low-rate satellite communications, the complexity of their decoding algorithms means that considerable progress is necessary before they could be considered competitive with RS codes. We now analyse the suitability of Hermitian codes for our particular application.

Let our Hermitian code over GF(q^2) have the following parameters:

$$\begin{aligned} \text{Code-length: } n &= q^3 \ , \\ \text{Distance: } d &= q^3 - m \ , \\ \text{Information Symbols: } k &\geq m + 1 - g, \text{ with } g < m < n \ , \\ \text{Genus: } g &= \frac{q^2 - q}{2} \ , \end{aligned}$$

giving $n-k+1 \leq q^3-m+g$. We will compare this code with an (n', k') RS code with distance $d' = n' - k' + 1$. There are three possible cases which we need to consider:

1. Both codes are over the same ground field and have the same code-rates, meaning that they must have different code-lengths.
2. Both codes have the same code-length and distance, meaning that they must be over different ground fields and have different code-rates.
3. Both codes have the same code-length and code-rate, meaning that they must be over different ground fields and have different distances.

4.3.1.1 Case 1

Firstly, we consider the case where Hermitian and RS codes are over the same ground field, GF(q^2), and have the same code-rates, so that

$$\frac{k'}{n'} = \frac{k}{n} \ . \tag{4.1}$$

This means that our code-lengths, defined to be $n = q^3$ and $n' = q^2$, must be different. From equation (4.1) and the code parameters, we can deduce that

$$\begin{aligned} \frac{k'}{n'} &= \frac{k}{n} \\ &\approx \frac{m+1-\frac{q^2-q}{2}}{q^3} \\ &= \frac{\left[\frac{m+1}{q} - \frac{q-1}{2}\right]}{q^2} \ . \end{aligned} \tag{4.2}$$

We know that the code-length n' of the RS code is q^2, which means that the numerator in equation (4.2) must be equal to k'. Thus, we can write the

code-distance, d', for the RS code as

$$\begin{aligned} d' &= n' - k' + 1 \\ &\approx q^2 - \frac{m+1}{q} + \frac{q-1}{2} \ . \end{aligned} \tag{4.3}$$

We want to compare this with the distance for the Hermitian code, $d = q^3 - m$. For large $m \approx q^3 - r$, with r small, the code-rate is nearly equal to 1, and the RS distance is larger than the Hermitian one. For smaller values of m, resulting in lower code-rates, the situation is reversed. Recall, from our background chapter, that a larger distance results in the code being able to correct a higher number of errors. Therefore, for lower code-rates, Hermitian codes would be expected to perform better than RS codes. However, this is only achieved by having a much longer code-length, q times greater than that of the RS code. This means that there would also be a considerable increase in the number of information symbols, k, required. These k symbols come directly from the available data. Therefore, Hermitian codes would be far more suitable for applications in which there is a long stream of data, such as telecommunications, rather than for error-correction in biometrics, in which a limited block of data is present.

4.3.1.2 Case 2

Alternatively, we could compare Hermitian and RS codes of the same length $n = n' = q^3$ and distance $d = d'$, but necessarily different ground fields. Thus, we have

$$\begin{aligned} d' &= n' - k' + 1 \\ &= q^3 - k' + 1 \\ &= q^3 - m \\ &= d \end{aligned} \tag{4.4}$$

so $k' = m+1$. This means that the RS and Hermitian code-rates respectively are given by

$$\frac{k'}{n'} = \frac{m+1}{q^3} \text{ and } \frac{k}{n} \geq \frac{m+1-g}{q^3} \ . \tag{4.5}$$

To achieve the same distance, the Hermitian code-rate is less than that of the RS code, meaning that more check digits are required. This is not surprising, given that RS codes are MDS (maximum-distance separable) codes. A lower code-rate is a disadvantage, as this makes the code less efficient (as the ratio of information bits, k, to the length, n, of each codeword is poor). The only possible advantage is the potentially simpler calculations associated with the use of a smaller ground field for the Hermitian codes. However, this advantage is eliminated by the fact that RS calculations are straightforward while Hermitian decoding is very complex.

4.3.1.3 Case 3

This final case is similar to the previous one, in that the Hermitian and RS codes are of the same length, $n = n' = q^3$, and necessarily different ground fields. However, in this case, we keep the code-rates identical, meaning that we also have $k = k'$. This gives us

$$\begin{aligned} d &= q^3 - m \\ &\geq n' - (k' - 1 + g) \\ &\geq d' - g \ . \end{aligned} \tag{4.6}$$

However, as RS codes are MDS codes, we know that $d' \geq d$. Therefore, we have

$$d' - g \leq d \leq d' \ , \tag{4.7}$$

meaning that the distance of the Hermitian code is reduced and the error-correcting performance degraded. As in the previous case, a smaller ground

field is used by the Hermitian code, but once again, any advantage associated with the simpler calculations in such a field is far outweighed by the complexity of decoding Hermitian codes.

4.3.1.4 Conclusion

We have compared Hermitian and RS codes under all possible conditions. The only situation in which Hermitian codes are superior to RS codes is in Case 1, where a larger distance is achieved over the same ground field and code-rate. However, this increase is at the expense of a substantial increase in the information symbols required for each codeword, which would cause problems in an application with limited amounts of data, such as biometric identification, as there simply would not be enough data to fill the required number of data slots. In any case, if greater code-distance is required for the application, a more efficient approach would be to increase the length of the RS code in use (resulting in a change in ground field). As shown in Cases 2 and 3, for the same length, RS codes give superior results than Hermitian codes, with a far simpler decoding procedure. Overall, we conclude that there is no advantage to introducing Hermitian codes in this particular application; indeed, they would result in a poorer performance than that of RS codes.

4.3.2 RS Codes vs LDPC Codes

We will now consider whether LDPC codes would be more suitable than RS codes for our application. The use of LDPC codes for iris recognition is not new, as it would appear to have been implemented by Martini and Beinlich [MB03], although, as mentioned previously, details of their scheme are sketchy. Several performance comparisons between LDPC codes and RS codes have been produced, although most of these are uninformative for our

purposes, as they are unfairly biased in favour of LDPC codes. This is due to the fact that many are conducted on data which do not contain bursts of errors, and also that they tend to be set up for soft decision decoding, which is better suited to LDPC codes than RS codes [MR03].

Results from studies that do include bursts of errors are mixed: [PT04] and [WS98] report improved performance from LDPC over RS codes; [RN04] finds that LDPC codes offer no advantage over RS codes. The inconsistent nature of the conclusions drawn from these and other comparisons between RS and LDPC codes shows the need for continued work in this area.

It is not possible to produce a systematic comparison similar to that done in the previous section for Hermitian codes, due to the large structural differences between LDPC and RS codes. Therefore, we must instead consider the suitability of LDPC codes for our specific system.

LDPC codes cannot be constructed in a similar manner to RS codes, by simply choosing a ground field, code-length and number of information symbols. Instead, their development is largely heuristic, and they tend to have only an *expected* minimum distance, meaning that the error-correcting capacity of the code is probabilistic. This is a clear disadvantage in a scheme such as iris recognition, in which the number of errors present is taken as an indicator of whether or not a person is who they claim to be. As such, it is important that a certain threshold of errors is always corrected, and, beyond that number, the system should fail and reject the user's claim.

The "best" LDPC codes (those that come closest to channel capacity) are based on randomly generated irregular graphs [PT04]. These irregular graphs are then classified into groups, based on probability distributions, and properties are proven about the general characteristics of that group. However, [PT04] showed that the performance of *individual* codes with each group

varies substantially. They conclude that, while some distributions yield good codes after only a few iterations of code generation, other require a long time to do so, making them difficult to use in practice. Thus, we cannot predict precisely whether a particular LDPC code will perform well without implementing and testing it robustly. By contract, RS codes have well-defined error-correcting capacities and correct all errors below that threshold.

In order to select a suitable LDPC code to correct iris readings, we would need to carefully examine the iris data to determine the number of errors we aim to correct; generate (through analysis of Tanner graphs, or manipulations of the parity-check matrix) a "good" LDPC code, which is predicted to correct this number of errors; test this code on a large amount of data, to determine its actual performance; and if its performance is poor, start again from scratch, generating a new code, which in turn must be robustly tested. If the iris recognition system is altered, resulting in a greater number of errors, it would involve the generation of a new "good" LDPC code, with all the additional complications. If an RS code was in use, however, it is relatively simple to adjust the parameters to make it more suitable to correct the resulting errors, as we will show later in this thesis.

In addition, LDPC codes demand very large memory requirements, as every edge represents a unique message which must be passed during the message-passing algorithm for decoding. Thus, they are expensive and complex to implement in hardware. Finally, unlike RS codes, LDPC codes are not specifically designed to efficiently correct bursts of errors. Clearly, the "burstiness" of the data will dictate how much more suitable RS codes are to correct it than LDPC codes, and so we will attempt to quantify this "burstiness" in Section 6.3. Thus, it is clear that, for our implementation, RS codes are a far more suitable choice to correct errors in iris readings.

4.4 Detailed Structure of Scheme

Having now chosen the most suitable key sizes, biometric, and error-correcting code for our system, we can begin to design it in detail. Our proposed scheme employs biometric authentication to protect and retrieve the private key in a public-key cryptosystem. The structure of this scheme can be outlined as follows:

Key Protection:

1. An individual enrols in the system by providing a claim of identity and submitting to an iris reading. The claim of identity may be in any form deemed suitable.

2. The iris is segmented from the rest of the image, and an IrisCode and corresponding IrisMask are created using Daugman's algorithm.

3. The IrisCode is shortened by deleting from it any bits identified by the IrisMask as being corrupt.

4. Check digits are calculated for the shortened IrisCode and appended to it to form a codeword (with the shortened IrisCode acting as the information bits of the codeword).

5. Bits from the codeword are XORed with the individual's private key in order to form a secure entity.

6. A hash of the information bits is stored in the individual's file, along with the check digits and the enrolment IrisMask, referenced by the identity claim. The secure entity is stored on a suitable medium, such as a smart card (see Section 4.5).

Figure 4.1 shows the key protection stage.

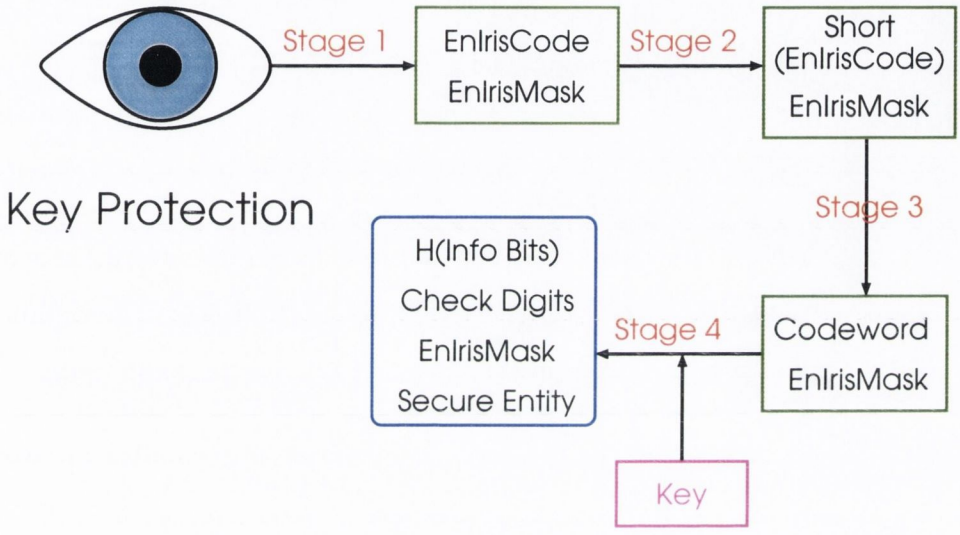


Figure 4.1: Overview of Key Protection. STAGE 1: Process iris using Daugman's algorithm. STAGE 2: Shorten enrolment IrisCode (EnIrisCode) by deleting bits identified in EnIrisMask as corrupt. STAGE 3: Calculate check digits for Short(EnIrisCode) and append to create codeword. STAGE 4: XOR codeword with key to create secure entity, and store secure entity, hash of information bits of codeword, check digits and EnIrisMask in file.

Key Retrieval:

1. The individual produces the same claim of identity and submits to a new iris reading.

2. The claim of identity retrieves the hashed information bits, the corresponding check digits and the enrolment IrisMask from the individual's file.

3. The new iris reading is processed as before, using Daugman's algorithm, and the bits identified by the enrolment IrisMask are again deleted.

4. This shortened reading is treated as a corrupted codeword, in which the verification IrisMask identifies erasures and all other discrepancies are treated as errors. It is corrected using a Reed-Solomon code.

5. If the hashed result of the corrected information bits equals that stored previously for the individual, verification is considered to be valid.

6. These corrected bits are then XORed with the secure entity on the smart card to "retrieve" the individual's private key.

Figures 4.2 shows the key retrieval stage. This scheme could be implemented directly on a PC, if so desired, or preferably on a smart card, as we will discuss in the next section.

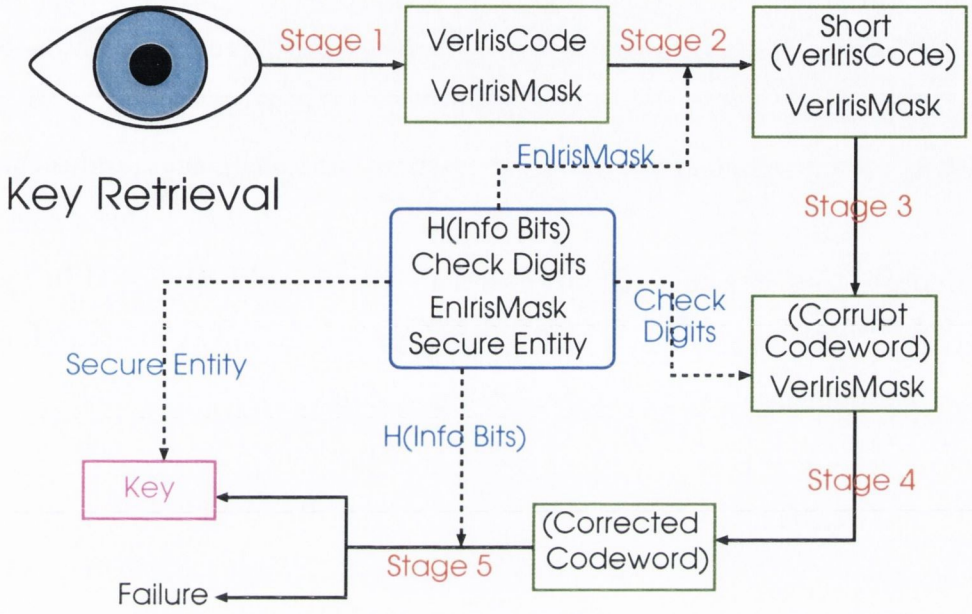


Figure 4.2: Overview of Key Retrieval. STAGE 1: Process iris using Daugman's algorithm. STAGE 2: Shorten verification IrisCode (VerIrisCode) by deleting bits identified in EnIrisMask as corrupt. STAGE 3: Retrieve check digits from file and append to Short(VerIrisCode) to create corrupted codeword. STAGE 4: Correct codeword using RS code, treating bits from VerIrisMask as erasures and other discrepancies as errors. STAGE 5: Hash information bits of corrected codeword and verify this equals previously stored hash. If successful, key is retrieved by XORing codeword with secure entity.

4.5 Smart Cards

Before investigating how smart cards could be used in our application, we briefly review some general information about them. A smart card, while similar in size and appearance to a plastic credit card, is considerably more complex. The idea was first patented in 1970 by the Japanese inventor, Kunitaka Arimura, and further developed in 1974 by Roland Moréno in France [Hen01, pp 42]. Some good overviews of smart card technology and security can be found in [RE04], [Car06] and [Sun06]. Not all smart cards contain microprocessors, but for our purposes here, we are only interested in those that do. For these smart cards, the data is processed on the card itself, and so they can be used for cryptographic applications.

The storage capacity of smart cards currently ranges from 8,000-128,000 bits [Car06], depending on the type of card and its intended use, but this continues to increase with improvements in technology. Older smart cards are based on 8-bit microprocessors, but 32-bit Reduced Instruction Set Computing (RISC) processors running at 25-32 MHz are becoming more common. They are considered to be highly secure as any information stored on the chip itself is difficult to duplicate or disrupt. They must also conform to numerous international standards, such as the ISO7816 series, which range from basic physical structure to electronic signals and protocols. Work is also beginning on introducing efficient elliptic-curve cryptosystems to smart cards [SKS06].

4.5.1 Attacks on Smart Cards

An excellent overview of the attacks possible on smart cards is to be found in [vT05]. These attacks can be divided into logical and invasive attacks. *Logical attacks* involve the recovery of secret information from the card without

physically damaging the card itself. Examples of this are differential power analysis, in which statistical analysis of the power consumption of the card is used to decipher secret information, or timing attacks, which study the execution time of different processes to determine when keys are being used. *Invasive attacks* consist of direct physical attacks on the smart card. These are expensive to implement and require a high level of technical expertise, and aim to recover information stored on the card through reverse-engineering, using optical or scanning electron microscopes. In general, preventative measures can be implemented on the card itself to counteract these attacks.

4.5.2 Biometric Key Retrieval on Smart Cards

The design we described in Section 4.4 requires the user to provide a claim of identity which is then accepted or rejected based on their biometric reading. Although the claim could be something as simple as an identity badge, a smart card would seem to be an ideal alternative.

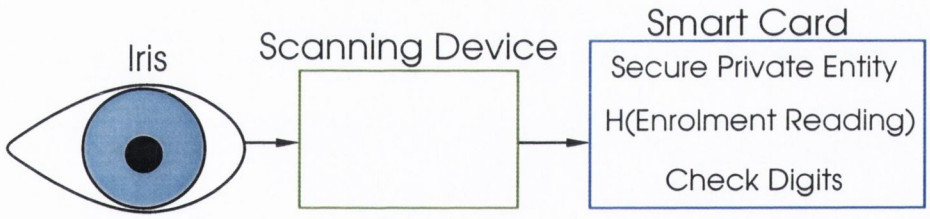


Figure 4.3: Biometric key retrieval on smart cards. The card stores the secure private entity (the private key XORed with the enrolment iris reading), a hash of the enrolment reading and the corresponding check digits. The user submits to a new iris reading, and inserts the smart card into the scanning device, which transmits the iris reading to the card. All subsequent calculations are carried out inside the card.

Figure 4.3 shows a possible implementation involving smart cards. The private key is stored as a secure entity on the card, having been XORed with the iris reading at the enrolment stage. In order to recover the key, the user inserts the card into an iris-scanning device of some sort and submits to an iris reading as usual. The scanning device transmits the iris reading to the card, at which stage it is corrected in the usual fashion and used to restore the private key. It is important that, once the card has been removed from the transmitter, the session ceases and the decoded private key is instantly destroyed, to maintain security in the aftermath of the transaction. In this way, none of the stored data leaves the smart card, enhancing the overall security. Even if this card was stolen, without the correct biometric, the attacker could not impersonate the user and access their private key.

Having now outlined the general design of our biometric key retrieval scheme, we proceed, in the next chapter, to consider what adjustments may be necessary to implement this scheme for the various types of public-key cryptosystems.

Chapter 5

Specific Design for Biometric Key Retrieval for Public-Key Cryptosystems

"A small key opens big doors."

- Turkish Proverb.

In the previous chapter, we decided to use iris readings, corrected using Reed-Solomon codes, in our biometric key retrieval scheme. We also detailed the various key sizes that we would aim to protect for the three different types of public-key cryptosystems: those based on the subgroup-discrete-log problem (SDL); elliptic-curve-based systems; and those which rely on the intractability of integer factorisation (IF) for their security. As observed previously, the key sizes in IF systems are ten times larger than those of the other two, and as such, will necessitate some modification of our scheme for successful key protection. We now consider how biometric key retrieval would be implemented for each type of PKC system.

5.1 SDL and Elliptic-Curve PKC Systems

Given that we aim to protect and retrieve keys of the same order for SDL and elliptic-curve-based PKC systems, both can be approached in a similar manner. Based on the various studies we reviewed in Chapter 3, key sizes in the range of 160-224 bits for these PKC systems should be safe for at least another twenty years. Using Daugman's algorithm to process iris readings results in a data-string, the IrisCode, of 2048 bits. The design of our biometric key retrieval scheme, outlined in the previous chapter, involves creating a codeword from the valid bits of the IrisCode, and XORing this with the private key in order to protect the key. Clearly, there will always be a sufficient number of valid bits in an IrisCode to directly XOR with private keys of this size. Given that there is no reason to believe that any particular section of the iris reading is more secure than any other section, we suggest that the least significant bits (LSBs) be used for this purpose.

If it is decided to employ smart cards for this process, as described in the previous chapter, some additional complications arise for elliptic-curve systems, due to the complexity involved in their hardware implementation. However, such complications are likely to become less marked with continued advances in engineering.

5.1.1 Aside: Symmetric Key Systems

As an aside, it is worth noting that the above system could also be employed for all types of symmetric key systems. Most significantly, it could be used for AES-192 and AES-256, which have key lengths of 192 and 256 bits respectively, and therefore could not be used in any previous biometric key retrieval scheme. As mentioned earlier, these versions of AES are recommended for

the highest security levels, where the use of biometrics as an identifier would be highly desirable.

5.2 IF PKC Systems

Developing biometric key retrieval schemes for PKC systems, such as RSA, which rely on integer factorisation for their security, presents us with an additional challenge. Their private keys are substantially larger than those of other PKC systems. There are insufficient uncorrupted bits in an IrisCode to perform a direct XOR between valid bits and the private key, meaning that the key cannot be protected as above. Even allowing for improved techniques of biometric acquisition, there exists a finite amount of extractible information in any biometric. The size of private keys in IF PKC systems, however, could increase indefinitely with improved computing power.

Therefore, in order to implement biometric key-retrieval for IF PK algorithms, there are three approaches to be considered:

1. The information content of biometrics could be supplemented (for example, by padding the biometric with pseudo-random numbers).

2. The biometric could be used as an AES key to encrypt the private key.

3. The amount of data from a private key which is securely hidden could be reduced, without adversely affecting security.

The first option, while initially appearing to be the most straightforward, introduces numerous complications, as the amount of pseudo-random data needed will rapidly eclipse the biometric information available. This means that we are hugely reliant on the pseudo-random generator used and its ability to produce distinct, but repeatable, sequences for different inputs.

The second option, that of creating a symmetric key (such as an AES key) from the biometric and using it to encrypt the private key has previously been suggested [Sou01] (though not implemented) as a means of using current biometric key retrieval schemes for PKC systems. The final option - that of reducing the amount of the private key which must be be kept secret at all times - has not been previously attempted. We propose to investigate this novel idea, to determine if it would be possible to merely hide specific portions of the key. This idea is interesting in its own right as a mathematical problem, but it may also prove preferable to option 2 above, in terms of ease of implementation, speed or security. Therefore, following our exploration of this topic, we will compare its benefits with those of option 2, to determine the optimum choice.

5.2.1 Overview of Partial Hiding

We begin by formulating the notion of "partial hiding", in which sections of the private key are made publicly available with no subsequent loss of security. In mathematical notation, the idea of partial hiding could be expressed by letting P be the private key of length p, S be the bits not hidden and X be the bits hidden [FP05]. Using the exclusive-OR operation (represented by $\otimes$) to combine these, we have

$$P = S' \otimes X' \ , \tag{5.1}$$

where S' and X' represent bit-strings of length p, consisting partly of S or X respectively and padded with zeros in positions where bits were extracted. For example, suppose P is a 10-bit string, and X consists of the four "middle"

bits, then we would have

$$\begin{aligned} P &= (1011010010)_2 \\ &= (1010000010)_2 \otimes (0001010000)_2 \\ &= S' \otimes X' \ . \end{aligned} \tag{5.2}$$

It is vital that X is sufficiently large to render a brute force attack impractical; if an attacker has access to the public key, the algorithm and S, he should not be able to determine P by trying all possible values of X.

There are several ways in which the bits of X could be extracted from P: we could remove the least significant bits (LSBs); the most significant bits (MSBs); or a random scattering of bits from throughout the key. Randomly selected bits would seem initially to give a greater level of security. However, if the bits are randomly selected according to some secret rule, then we also need to store their original position in the key, to recover them later on. Storing the *positions*, as well as the *values* of the bits, within the n-bit allowance we choose for X reduces the overall number of bits that can be hidden thus - as we could only store $n/2$ bits of the key and $n/2$ positions, instead of n bits of the key, a clear disadvantage. Therefore, we chose to focus instead on exposing either LSBs or MSBs or both.

5.2.2 Explanation of Partial Hiding in RSA

When considering the deliberate exposure of bits from the RSA private key, it is logical to examine work done on attacks on RSA following *accidental* exposure of parts of the key. As mentioned in our related work chapter, [BDF98] showed that the most significant half of the private key d is automatically "leaked" in low-exponent RSA, but this does not affect the security of the system. We now give an explanation as to why this is the case, based

on their work.

Let $N = pq$ be an n-bit RSA modulus. Assume the primes p and q are distinct and close to $\sqrt{N}$, such that

$$p, q < 2\sqrt{N} \ . \tag{5.3}$$

This implies that

$$p + q < 4\sqrt{N} \ . \tag{5.4}$$

Suppose we have a small public-key exponent, e. Then, for some k,

$$ed = k\phi(N) + 1 \ , \tag{5.5}$$

where $\phi(N) = N - p - q + 1$. Note that $k \in \{1, \cdots, e-1\}$. Equation (5.5) can be re-written:

$$ed - k\phi(N) = ed - k(N - p - q + 1) = 1 \ . \tag{5.6}$$

From this, we can write

$$d = \frac{1 + k(N - p - q + 1)}{e} \ . \tag{5.7}$$

Now let

$$\tilde{d} = \left\lfloor \frac{1 + k(N+1)}{e} \right\rfloor \ . \tag{5.8}$$

Given that

$$0 \leq \tilde{d} - d \leq \frac{k(p+q)}{e} \leq \frac{4k\sqrt{N}}{e} < 4\sqrt{N} \ , \tag{5.9}$$

this means that $\tilde{d}$ and d match on the $(\frac{n}{2} - 2)$ MSBs of d. Therefore, if an attacker can find the value of k, he will know the MS half of the bits of d. Given that $1 \leq k \leq e - 1$, all candidate values for k can be found, allowing the attacker to compile a small list of possible values for the MS half of the bits of d. In this way, the MS half of the bits of d are "leaked" in low-exponent RSA.

[BDF98] also details an attack showing a vulnerability if a quarter of the LSBs are exposed. Their argument is as follows: Suppose $p \not\equiv q \bmod 4$, and that we are given the LS $n/4$-bits of d, which we will term

$$d_0 \equiv d \bmod 2^{\frac{n}{4}} \ . \tag{5.10}$$

Reducing equation (5.6) mod $2^{\frac{n}{4}}$, and letting $q = \frac{N}{p}$, we have $p \bmod 2^{\frac{n}{4}}$ as a solution for x in the equation:

$$ed_0 \equiv 1 + k\left(N - x - \frac{N}{x} + 1\right) \bmod 2^{\frac{n}{4}} \ , \tag{5.11}$$

which can be re-written as a quadratic equation:

$$kx^2 + (ed_0 - k(N+1) - 1)x + kN \equiv 0 \bmod 2^{\frac{n}{4}} \ . \tag{5.12}$$

An attacker tries each possible value for k in the range $1 \leq k \leq e-1$, and finds all solutions for $x \bmod 2^{\frac{n}{4}}$ in each case. One of these solutions will satisfy

$$x \equiv p \bmod 2^{\frac{n}{4}} \ , \tag{5.13}$$

and so allow the attacker to factor N and break the system. BDF show that this can be achieved within time polynomial in n and e - see [BDF98] for a detailed proof. To find the solutions for $x \bmod 2^{\frac{n}{4}}$, the BDF attack relies on an algorithm [Red96] that is only valid if

$$\left(\frac{p-q}{2}\right)^2 \equiv 1 \bmod 8 \ . \tag{5.14}$$

Thus, when $p \equiv q \bmod 4$, this approach no longer works.

Steinfeld and Zheng [SZ04] went on to show that the time for the BDF attack increases by a factor of 2^α when the α LSBs of p and q are equal. They show that, in fact, in order to mount a polynomial-time attack, $(\frac{n}{4} + \alpha)$ LSBs of d are needed, and if the m LSBs of p and q are equal, up to $2m$ LSBs of d can safely be exposed, without being vulnerable to the above attack.

5.2.3 Partial Hiding in RSA

We have chosen to focus on deliberately exposing some of the LSBs and MSBs in an RSA private key, keeping only the "middle" bits secret. We base our system on an adaptation of that proposed in [SZ04], setting the m LSBs of p and q to be equal. If low-exponent RSA in this form is secure with no bits exposed, it is secure if up to $2m$ LSBs are exposed. As discussed previously, a key-size of 2048 bits is now the most appropriate for RSA. Thus, let $N = pq$ be an RSA modulus of size $n = \log_2 N = 2048$ bits, with p and q each 1024 bits long. When choosing a "small" exponent e, common values previously used, such as $e = 3$ or $e = 17$, are no longer considered safe [LV01]; thus, we let $e = 2^{16} + 1 = 65537$. Now, with

$$m = \frac{n}{4}(1 - \epsilon) \quad , \tag{5.15}$$

with small ϵ, in a secure system, we can expose $n/2$ (=1024) MSBs of d and $(n/2)(1 - \epsilon)$ $(= 1024 - 1024\epsilon)$ LSBs of d and still have a secure system.

We now determine the optimum value for ϵ. With $\epsilon < 0.5$, there is a considerable reduction in the cost of computation [SZ04]. However, we must ensure that 2^y is too large to exhaustively search for the y unknown bits of p or q, where $y = (\epsilon n/4)$, that would enough for an attacker to break the system. Table 5.1 shows various possible values of ϵ and the corresponding values of 2^y. We reject values such as $\epsilon = 0.125$, as 2^{64} is too low for security. However, with $\epsilon = 0.25$, we have 2^{128}, which is sufficiently large to be considered safe. Thus, the 384 LSBs of p and q are set to be equal. In order to generate these primes, simply find p in the usual fashion, fix the 384 LSBs of q to be identical to p, and produce a prime q of this form [SZ04]. This is expected to be as efficient as the standard independent primes generation algorithm for random RSA moduli, where each candidate for q is chosen independently

Size of N	Size of p (or q)	Value of ϵ	Value of m	Value of 2^y
2048	1024	0.5	256	2^{256}
2048	1024	0.25	384	2^{128}
2048	1024	0.125	448	2^{64}

Table 5.1: Table showing the relationship between various values of ϵ and the subsequent 2^y, which reflects the security of the overall system.

of p as a random odd integer.

With $\epsilon = 0.25$, the 1024 MSBs and the 768 LSBs of d can be exposed for our system, and if the original RSA system was secure, then this new system will also be. We need to keep 256 bits securely hidden at all times, in bit positions from 769 to 1024 inclusive, assuming an offset of 1 and working from right to left. In terms of our biometric key retrieval scheme, the only alteration that needs to be made is at Stage 4, as illustrated in Figure 5.1. Instead of the entire private key being XORed with the codeword at this point, only the "secret portion" from bits 769-1024 is protected in this way.

5.2.4 Partial Hiding in CRT RSA

RSA is frequently implemented using the CRT RSA approach [QC82], which relies on the Chinese Remainder Theorem to speed up calculations. Therefore, we now investigate whether partial hiding can be used in CRT RSA, in which two separate quantities

$$\begin{aligned} d_p &= d \bmod (p-1) \ , \\ d_q &= d \bmod (q-1) \end{aligned} \tag{5.16}$$

are used to encrypt and decrypt each message. By combining the public RSA information with either p or q, both d_p and d_q can be calculated. Thus,

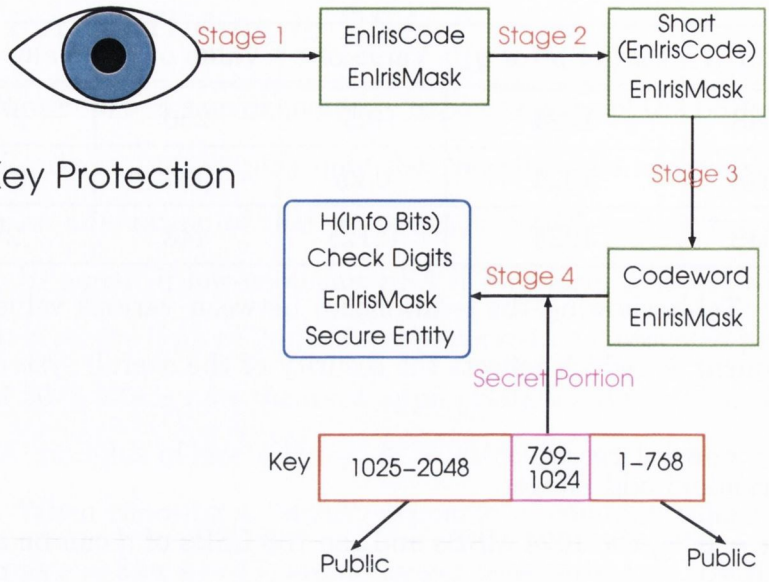


Figure 5.1: Key Protection, using partial hiding. At stage 4, 2048-bit RSA private key is split into 3 sections. Bits 1-768 and 1025-2048 are public; bits 769-1024 are kept secret and XORed with the codeword to keep them secure.

while we need to retain the values of p and q, we cannot make these publicly known in their entirety. There are two main approaches to consider:

1. Partially store p and q. Partially store d_p and d_q. Restore these hidden portions to the "public" portions of d_p and d_q each time.

2. Partially store p and q. Partially store d. Generate d_p and d_q each time using both partially stored and publicly known information.

For option 1, certain restrictions must be imposed immediately: less than 50% of the LSBs (or MSBs) of d_p or d_q can be made public, as an attack [BM03] exists on low public exponent CRT RSA if more than this amount is exposed. If the RSA modulus is of size $n = 2048$, then p and q consist of 1024 bits, and d_p and d_q are also approximately this size. The 384 LSBs

of p and q are identical in our system; the 640 MSBs are not. Coppersmith [Cop97] describes a method of factoring N given the MS half of the bits of p. Thus, the 640 MSBs of p and q should be securely hidden using the iris template. We must also store over half of the bits of d_p and d_q, meaning a total of more than 2300 bits of data need to be XORed with the iris in this scheme. However, an iris template consists of only 2048 bits of data, and so a direct XOR with the RSA information is not possible. Thus, we turn our attention to the second approach.

For option 2, we again choose to partially hide the 640 MSBs of both p and q. As suggested above for RSA, we store only the 256 "middle" bits of d, which gives us a total of 1536 bits to be stored, which could easily be hidden by XORing with an iris template.

We must now consider whether we have introduced additional vulnerability to the system by partially revealing p and q. In our private key, d, bits 1-768 and 1025-2048 are made public, with only bits 769-1024 securely hidden. In addition, we now propose to reveal bits 1-384 of p (and q). If it is possible to determine either the least significant half or the most significant half of d_p as a result, then our system is not secure.

p is prime, so the LSB of p will always equal 1. Thus, to obtain $(p-1)$, we simply change this LSB to 0. Let p' be the 384 LSBs of p that are publicly available. Then we have

$$p = t(2^{384}) + p' \ , \tag{5.17}$$

where p and t are not publicly known. This means that

$$p - 1 = t(2^{384}) + p' - 1 \ . \tag{5.18}$$

Therefore, we can write

$$ed_p = 1 + k(t2^{384} + p' - 1) \ . \tag{5.19}$$

But since

$$k < e < 2^{16} + 1 \ , \tag{5.20}$$

an attacker can guess the value of k by a brute-force search, allowing him to calculate

$$d_p = e^{-1}(1 + k(p' - 1)) \bmod 2^{384} \ , \tag{5.21}$$

which will give him the 384 LSBs of d_p. However, the 512 LSBs of d_p are needed in order for the key to be in danger [BM03]. Given that there are 2^{128} possible combinations for the remaining 128 unknown bits, the key is still safe from attack. Therefore, partial hiding can be applied to CRT RSA.

5.2.5 Possible Attacks

Having designed our system for partial hiding in RSA, we must now examine its resistance to attacks on partial key exposure. The most basic attack consists simply of a brute force approach, where an attacker tries all possible combinations for the hidden portion of the private key, d. We are proposing to hide bits 769-1024 of a 2048-bit d. This means that only 256 bits must be uncovered to break the system. However, an attacker would need to calculate the values of all 256 of these bits before being in a position to judge if any were correct. The parameters we have chosen for this system are secure under current and predicted computing power.

Our scheme is not vulnerable to the attacks mentioned in [BDF98], as it is based on the Steinfeld-Zheng system, which is resistant to these, as explained in Section 5.2.2. However, Blömer and May [BM03] produced a paper containing several attacks which may be applicible to our scheme, depending on the parameters used. We now look at each relevant attack in detail and discuss whether they pose a threat to the system we have proposed.

5.2.5.1 Blömer-May Attack 1

We begin with their strongest attack, which works for all $e < N^{\frac{7}{8}}$, provided a certain portion of the LSBs of d is exposed. The attack is based on the following theorem, proven in their paper:

Theorem 5.2.1 (Blömer-May) *For every $\epsilon > 0$, there exists N_0 such that for every $N \geq N_0$, the following holds:*
Let (N, e) be an RSA public key with $\alpha = \log_N(e) \leq \frac{7}{8}$. Let d be the private key. Given d_0, M satisfying $d = d_0 \bmod M$ with

$$M \geq N^{\frac{1}{6}+\frac{1}{3}\sqrt{1+6\alpha}+\epsilon}, \tag{5.22}$$

then N can be factored in polynomial time.

We apply this theorem to our system to determine whether a threat is posed by this attack. Our modulus N is of size $n = 2048$ bits, and we have set $e = 2^{16} + 1 = 65537$. We expose the 768 LSBs of d; we must now calculate the minimum number of LSBs required in order for the attack to proceed.

The smallest number of length 2048 bits is 2^{2047}. From the theorem, if M is sufficiently large (where M denotes the number of LSBs known), then our system can be broken. To minimise the number of LSBs needed, set $\epsilon \approx 0$ and neglect this term completely. We now calculate the value of α:

$$\begin{aligned} \alpha &= \log_{2^{2047}}(65537) \\ &= 0.0078 \ . \end{aligned} \tag{5.23}$$

Using (5.22), we have

$$\begin{aligned} M &= \left(2^{2047}\right)^{\frac{1}{6}+\frac{1}{3}\sqrt{1+6(0.0078)}} \\ &= 2^{1039.32} \approx 2^{1040} \ . \end{aligned} \tag{5.24}$$

Thus, a minimum of 1040 LSBs are needed for this attack. As we only expose 768 LSBs, our system is secure against this approach.

5.2.5.2 Blömer-May Attack 2

The next attack to which our scheme may be vulnerable involves a provable attack for almost all $e < N^{\frac{1}{2}}$. Here, a vulnerability is introduced if

$$N^{\alpha+\frac{1}{2}+\epsilon} \leq M \leq 2N^{\alpha+\frac{1}{2}+\epsilon} \ , \tag{5.25}$$

where $0 < \alpha, \epsilon < 1/2$ and M represents the known LSBs again. As before, letting $\alpha, \epsilon \approx 0$ for the smallest number of bits acceptable, we get

$$\begin{aligned} N^{\frac{1}{2}} \leq \ & M \ \leq 2N^{\frac{1}{2}} \\ \implies \left(2^{2047}\right)^{\frac{1}{2}} \leq \ & M \ \leq 2\left(2^{2047}\right)^{\frac{1}{2}} \\ \implies \ & M \ = 2^{1024} \ . \end{aligned} \tag{5.26}$$

Thus, the 1025 LSBs of d need to be exposed to make a system vulnerable to this attack, whereas we only expose the 768 LSBs of d.

5.2.5.3 Blömer-May Attack 3

The final Blömer and May attack which may introduce a vulnerability to our scheme is based on CRT RSA. The attack enables an adversary to recreate d given the least significant half of the bits of d_p and works for low-exponent RSA such as $e = 2^{16} + 1$. In our system, the least significant half of the bits of d_p is only of length 512 bits. We expose the 768 LSBs of d, but in order to determine the value of d_p, it is necessary to know p. However, an attacker knows that the 384 LSBs of p and q are equal, and has access to the value of $N = pq$. We now investigate whether this knowledge introduces a vulnerability into the system by potentially exposing some of the LSBs of p.

We choose artificially small values for our parameters for illustration pur-

poses. Let p and q be 16 bits long each, where

$$\begin{aligned} p &= (1000010101110011)_2 = 34163 \ , \\ q &= (1001010000110011)_2 = 37939 \ . \end{aligned} \tag{5.27}$$

Note that p and q have six equal LSBs, which corresponds to 0.375 of the bits of p, the same fraction as in our proposed system where we have 384 identical bits from a possible 1024 bits. Now

$$\begin{aligned} N &= pq \\ &= (1000010101110011)_2 \times (1001010000110011)_2 \\ &= (10011010100000100010001111101001)_2 \\ &= 1296110057 \ . \end{aligned} \tag{5.28}$$

If we now multiply the common LSBs of p and q, we get

$$\begin{aligned} (110011)_2 \times (110011)_2 &= 51 \times 51 \\ &= 2601 \\ &= (101000101001)_2 \ , \end{aligned} \tag{5.29}$$

which gives an identical six LSBs to that of N. We now consider whether it is possible to work backwards from "101001" to obtain "110011" as the root.

We know that the last six bits are "101001" and that this is the result of multiplying two numbers with six identical LSBs. Let a, b, c, d, e, f represent the six LSBs of p and q. Then we have

$$\ldots abcdef \times \ldots abcdef = \ldots 101001 \ . \tag{5.30}$$

We know that $f \times f = 1$, so this means $f = 1$. This gives us

$$(1 \times e) + (e \times 1) = 0 \ . \tag{5.31}$$

But here $e = 0$ or $e = 1$ satisfies the equation, as we have

$$\begin{aligned} (1 \times 0) + (0 \times 1) &= (0 + 0) = 0 \\ (1 \times 1) + (1 \times 1) &= (1 + 1) = 0 \ . \end{aligned} \tag{5.32}$$

We cannot definitively determine the value of e based on this information. Continuing through the equation, we have

$$(1 \times d) + (e \times e) + (d \times 1) = 0 \text{ or } 1 \ , \tag{5.33}$$

depending on whether $e = 0$ or $e = 1$, as $e = 1$ will lead to 1 being carried over to the next stage. This continues at each stage, with the uncertainty growing each time.

In this simplified example, it would conceivably be possible to test every combination and determine the six LSBs of p, but in our system, it would be necessary to determine 384 such LSBs. The correctness of each bit depends on the correctness of the bit before it. The LSB will always be revealed, but the value of any further bits is protected. Thus, our system is secure against this form of attack.

5.2.5.4 Wiener's Continued Fractions Attack

Wiener's original attack [Wie90] relies on continued fractions to find the value of d. It is successful for short secret exponents, which are up to one-quarter the size of the modulus, N. Our secret exponent is of length 2048 bits, but only 256 of these are securely hidden. We must investigate whether a continued fractions approach could expose a vulnerability in this scheme.

Let our secret exponent, d, consist of three sections: let d_0 represent the known 1024 MSBs, d_1 the unknown "middle" 256 bits and d_2 the known 768 LSBs. Then we have

$$d = d_0 2^{1024} + d_1 2^{768} + d_2 \ . \tag{5.34}$$

This means we can adapt Wiener's notation and write:

$$e\left(d_0 2^{1024} + d_1 2^{768} + d_2\right) = K \operatorname{lcm}(p-1, q-1) + 1 \ . \tag{5.35}$$

This gives us

$$e\left(d_0 2^{1024} + d_1 2^{768} + d_2\right) = \frac{k}{g}(p-1)(q-1) + 1 \ , \tag{5.36}$$

where

$$G = \gcd(p-1, q-1),\ k = \frac{K}{\gcd(K,G)} \text{ and } g = \frac{G}{\gcd(K,G)} \ . \tag{5.37}$$

At this point, Wiener manipulates the formula to read:

$$\frac{e}{pq} = \frac{k}{dg}(1-\delta) \text{ where } \delta = \frac{p+q-1-\frac{g}{k}}{pq} \ . \tag{5.38}$$

If d is small, it can now be found using continued fractions. However, in our case, an attacker would need to be able to extract d_1 in equation (5.36) from the sum in order to be able to launch an attack in this fashion. It would be necessary to express d as a product in which the unknown d_1 was less than a quarter of the bits of N, in place of the sum that is currently used, in order to attempt an adaptation of Wiener's attack, but as this cannot be done, the system is safe from this form of attack.

5.3 Partial Hiding vs AES

We have shown that partial hiding can be safely implemented in RSA, using the specified parameters, allowing us to XOR the hidden portion of the key with an iris reading for use in a biometric key retrieval scheme. We must now consider whether this approach is preferable to the other possibility mentioned at the start of this chapter: namely, the creation of an AES key from the iris reading, which is then used to encrypt the private key. In

our comparison, we consider issues such as speed, ease of implementation, originality and security.

AES is made up of *rounds* [DR02], each of which consists of four main steps, known as SubBytes, ShiftRows, MixColumns, and AddRoundKey. The number of rounds needed depends on the length of the key: with an 128-bit key, there are ten rounds; with an 192-bit key, twelve rounds; while a 256-bit key requires fourteen rounds. The execution of AES can be speeded up by converting the SubBytes, ShiftRows and MixColumns steps into four look-up tables, each with 256 32-bit entries. A round can then be completed in 16 table look-ups and twelve 32-bit XORs for these three steps, and four 32-bit XORs for the AddRoundKey step, giving a total of 16 table look-ups and 16 32-bit XORs per round. Table 5.2 shows the total number of operations required to implement AES, for its various key-lengths. By comparison, the implementation of partial hiding involves a single XOR operation of 256 bits. Even without properly taking into account the table look-ups (which take far longer than XORs to implement, and do not take constant time [Ber05]), partial hiding is at the very least 30 times faster than AES. Therefore, from the speed point of view, while AES is itself very fast to implement,

AES Key-Length	Number of Rounds	Total Number of Table Look-Ups	Total Number of Bits XORed
128	10	160	5120
192	12	192	6144
256	14	224	7168

Table 5.2: Number of operations (table look-ups and XORs) needed in the optimised implementation of AES, for its various key-lengths.

partial hiding is even faster. Any process which improves the speed of biometric key retrieval is to be welcomed; particularly in situations where a number of iris cameras all feed into a single processing system, causing a potential bottleneck.

In terms of ease of implementation, no additional online work is required to use partial hiding: the decision about which bits can be made public has already been made, and all that remains for implementation purposes is to enact a single XOR operation. In this way, it is again preferable to AES.

The partial hiding approach is a novel one, involving an original application of an interesting mathematical exploration into the possibility of revealing specific bits of a large integer without exposing the overall value.

One disadvantage of partial hiding is that, although we have shown that it is resistant to all known attacks, no conclusive proof of security exists for the system. Cryptographic algorithms are potentially vulnerable to new cryptanalysis attacks; this is also the case for AES, as noted in [CJJ+03] which states "since there is no proof of security of AES, legitimate questions about its security are frequently raised". In particular, the algebraic structure of AES has been a cause of concern for some cryptographers [CP02], who fear that this structure may make it more susceptible to attack. It is preferable to protect the private key using an XOR operation rather than the AES algorithm. XOR is not an algorithm, and, provided the input is random, ensures "perfect secrecy" [Sha48] (meaning that the ciphertext gives no additional information about the plaintext). Due to its high information content, the iris is random, and although XOR is susceptible to known plaintext or known ciphertext attacks, these are of no significance here, given that possession of the plaintext or ciphertext indicates possession of the biometric or RSA key, which eradicates the need for such an attack.

Having reviewed both approaches in terms of speed, ease of implementation, originality and security, we conclude that there is no advantage to using AES instead of partial hiding; in fact, it is clear that the quickest secure way to implement biometric key retrieval for RSA is through the use of partial hiding. We have now detailed our proposed design for each PKC system, and go on in the next chapter to discuss our implementation of this system.

Chapter 6

Implementation and Analysis: Daugman

"When it is not in our power to follow what is true, we ought to follow what is most probable." - Rene Descartes

In the previous two chapters, we gave a detailed description of the design of our biometric key-retrieval scheme for PKC systems. We selected the iris as the most suitable biometric, and Reed-Solomon codes as the optimum code to correct iris readings. We established that our design could be modified using partial hiding so that it could be used for all types of PKC systems.

The next step is to verify that iris readings can be corrected using RS codes, and to determine the most suitable code parameters to achieve this. To do so, we need a database of iris images which can be processed into iris templates, or data strings, and subsequently compared against each other. As mentioned in Chapter 4, we have chosen to work with the most accurate iris processing algorithm in existence, namely Daugman's [Dau04], which was described in detail in Chapter 3. The advantage of using this algorithm is that it produces the most consistent and repeatable data strings from iris

images; the disadvantage is that the algorithm is patented and black-boxed, meaning that it has not been possible to obtain processed iris data which has not subsequently been encrypted by the algorithm. Therefore, instead, we rely on the extensive results published by Daugman and generate data strings with statistically identical properties to his processed iris images. We then study the dispersion of errors and the error rates within these data strings, to confirm the suitability of RS codes to correct these errors and to determine the necessary code parameters to do so.

6.1 Choice of Ground Field for RS Code

We opt for a Reed-Solomon code over GF(2^8), as this has a maximum length of 255 octets, or bytes. The ground field of 2^8 is chosen largely because of the natural ease of dealing with octet-based calculations, a clear advantage when working with bytes of data. Furthermore, it is of characteristic two, so the operations of addition and subtraction are identical. This greatly simplifies arithmetic in the field, allowing us to use the XOR (exclusive-OR) function.

As we are working over GF(2^8), it is necessary to ascertain the *byte*-error rate of an iris reading. The number of *bit* errors can be estimated directly from Daugman's results, but we do not know the mean length of bit-error bursts or their distribution. We must calculate this information from our generated data strings, and so, we now detail how these strings are produced.

6.2 Model to Generate Iris-like Data Strings

To generate data strings that are statistically identical to Daugman's processed iris images, we begin by examining Daugman's results [Dau93], shown

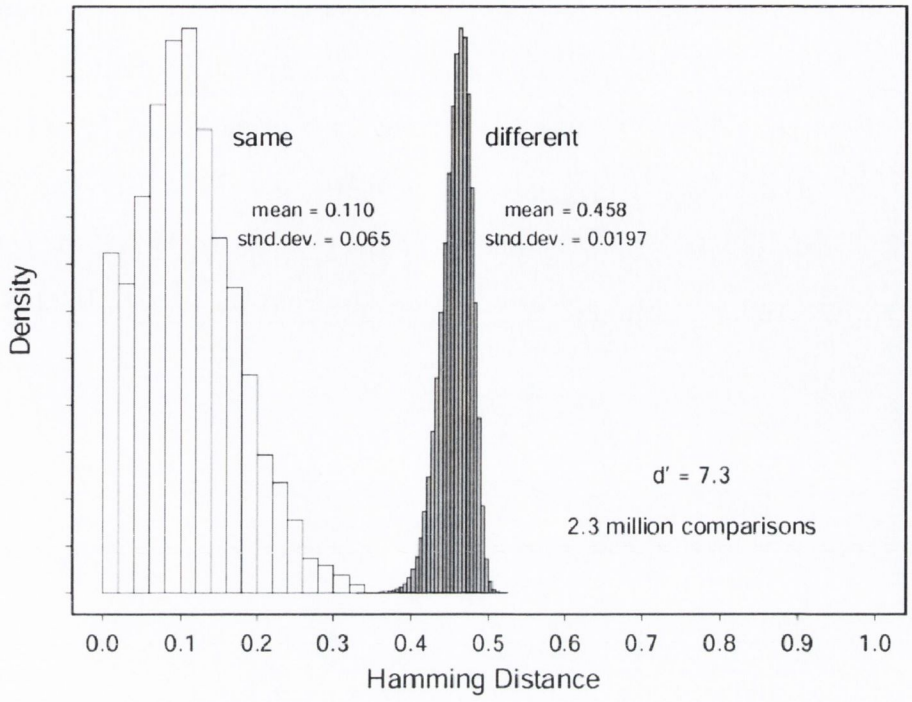


Figure 6.1: Results of Daugman's algorithm for unfavourable conditions [Dau93, Fig 9]. Images were taken at different distances and optical platforms to simulate verification. The left histogram represents identical irises.

in Figure 6.1, based on 2.3 million cross-comparisons between iris templates. The readings from identical irises are marked "same" in the graph, and have a mean of 11% bit errors with a standard deviation of 6.5%. 7070 comparisons were used in the creation of this "same" distribution. Only about 1% of these identical readings differ in greater than 28% of their bits and none have a bit-error rate exceeding 32.7% [Dau03]. Given the huge population involved in this study, we rely on these figures for our model.

The biggest challenge facing us in the construction of such a model lies in the distribution of the erroneous bits. In Chapter 4, we made the assumption that the errors in an iris reading occur in bursts. Thus, the probability that

a bit is in error is influenced by whether or not the previous bit was in error; a fact that must be taken into account in the design of this model.

Let P_e be the probability that the current bit is in error. Let $P_n = (1-P_e)$ be the probability that the current bit is *not* in error. When the current bit is not in error, let p be the probability that the next bit is. When the current bit is in error, let q be the probability that the next bit is not. In this way,

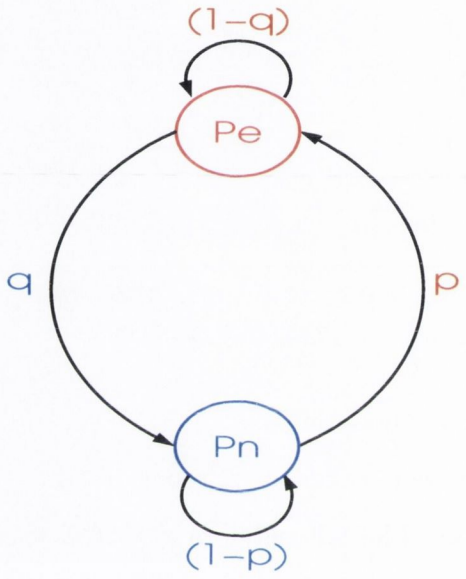


Figure 6.2: State diagram for our model. P_e is the error state and P_n the non-error state. p is the probability of an error given the current bit is not in error; q is the probability of a correct bit given the current bit is in error; $(1-p)$ and $(1-q)$ result in no change in their respective states.

p and q can be regarded as the relative probabilities of changing from one state to another, as shown in Figure 6.2. Thus, we have

$$\begin{aligned} P_e &= pP_n + (1-q)P_e \ , \\ P_n &= qP_e + (1-p)P_n \ . \end{aligned} \tag{6.1}$$

This gives us

$$\begin{aligned} P_e(p+q) &= p \ , \\ P_n(p+q) &= q \ . \end{aligned} \tag{6.2}$$

Rearranging Equation (6.2),

$$P_e = \frac{p}{p+q} \ . \tag{6.3}$$

Thus, we can express p in terms of q as

$$p = \frac{qP_e}{1-P_e} \ . \tag{6.4}$$

P_e is in fact the average bit-error rate for the entire string. Note that if $p = P_e$, then we would have $q = (1-p) = P_n$, which gives us an uncorrelated, random situation. Therefore, in order to include bursts of errors, we want $p < P_e$ which gives us $q < P_n$.

6.2.1 Computer Generation of Iris-like Data Strings

We now produce a computer simulation of this model. We wish to generate an array of N bits, for $N = 1600$ and $N = 1304$, as these correspond to typical Caucasian and Korean images respectively. We let 0 represent an uncorrupted bit and 1 represent an error, and set $P_e = 0.11$, based on the results of Daugman's tests. We choose a value for q, fixing the value of p.

The first bit of the string is determined by this value of p, by using the *rand()* function in C to generate a random number between 0 and $(\frac{1}{p} - 1)$, as described in [PTVF92, p. 277]. If the random number that results equals 1, then we set the first bit position to be equal to 1, thereby inserting an error; otherwise, it is set to be equal to 0. Depending on the value of this first bit, we use either p or q, as appropriate, to determine the value of the second bit, and so on until the N-bit array has been generated.

At this point, we determine the number of bit errors by counting all of the 1s in the array. Because we know that no iris reading has a bit-error rate greater than 32.7%, we reject any sample with more than this number of errors. We go on to calculate the number of byte errors in the string, summing bits together in groups of eight.

The program is run 10^4 times before an average bit-error rate for the chosen parameters is calculated, as Daugman's tests involved a sample of 7070 identical irises, meaning that 10^4 runs should be sufficient. The mean bit-error rate should always be close in value to that of P_e, although it will not be exactly equal to it. With P_e fixed at 0.11, we now perform some statistical analysis to examine whether or not our results are consistent with those of Daugman.

6.2.2 Statistical Comparison with Real Iris Data

To choose the most suitable statistical test to measure how similar our generated data is to Daugman's processed irises, we first use the following:

Theorem 6.2.1 (Central Limit Theorem) *Let X be a random variable with mean μ and variance σ^2. If $\bar{X}_n$ is the mean of a random sample size n chosen from the distribution of X, then the distribution of*

$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \tag{6.5}$$

tends to the standard normal distribution as $n \to \infty$ [CC03, p. 338].

As a direct consequence of the Central Limit Theorem, the mean of a large enough sample from any distribution will be approximately normally distributed [CC03, p. 380]. A "large enough" sample can constitute anything from $n \approx 30$ for an almost symmetrical distribution to $n = 200$ for quite a

skewed distribution. As we are dealing with considerably larger sample sizes, we can use techniques applicable to normal distributions to perform "difference of means" tests between the mean bit-error rate of Daugman's results and that of our generated data for various values of p and q. If these are statistically similar, this justifies our assumption that the errors in iris readings occur in bursts, as the data was generated using a burst-error model.

In order to compare two means, μ_1 and μ_2, the standard practice is to implement a hypothesis test in which

$$\begin{aligned} H_0 : &\quad \mu_1 = \mu_2 \\ H_1 : &\quad \mu_1 \neq \mu_2 \ , \end{aligned} \tag{6.6}$$

where H_0 represents the null hypothesis and H_1 the alternative hypothesis. In our case, we wish to do a two-tailed test because we are interested in deviations from H_0 in both directions; if μ_2 is either greater or less than μ_1, we wish to reject the null hypothesis. We define the normal z-score to be

$$\begin{aligned} z &= \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{(\sigma_1^2/n_1) + (\sigma_2^2/n_2)}} \\ &= \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{(\sigma_1^2/n_1) + (\sigma_2^2/n_2)}} \ , \end{aligned} \tag{6.7}$$

as the null hypothesis is that $(\mu_1 - \mu_2) = 0$. Using tables for the standard normal distribution, we know that

$$P(-1.96 \leq z \leq 1.96) = 0.95 \ . \tag{6.8}$$

This means that, with 95% confidence, we expect the z-value to be between ± 1.96, if the null hypothesis is true. If we wish to increase this confidence level to 99% or even 99.9%, we substitute the values ± 2.576 and ± 3.291 respectively for ± 1.96 in equation (6.8).

Using our chosen values of $N = 1600$ and $N = 1304$, we now test the results obtained for various values of q to ascertain whether or not they

sufficiently match those of Daugman. So we let $\bar{x}_1 = 0.11$, $\sigma_1 = 0.065$ and $n_1 = 7070$ as per Daugman's study. As an example, for $N = 1600$, if we set $q = 0.05$, we have $p = 0.0062$ and $n_2 = 10^4$, which yields the results $\bar{x}_2 = 0.1093$ and $\sigma_2 = 0.0456$. Thus, we have

$$\begin{aligned} z &= \frac{(0.11 - 0.1093)}{\sqrt{((0.065)^2/7070) + ((0.0456)^2/10^4)}} \\ &= 0.4552 \ , \end{aligned} \tag{6.9}$$

which is an excellent match for Daugman's data, as it falls between ± 1.96. To determine the full range of q-values which produce acceptable approximations to Daugman's results for $N = 1600$ and $N = 1304$, we compile Tables 6.1 and 6.2 respectively.

$P_e = 0.11$		Mean BER	Standard Deviation	Normal Score
q	p	$\bar{x}_2$	σ_2	z
0.025	0.0031	0.1075	0.0629	2.5410
0.050	0.0062	0.1093	0.0456	0.4552
0.075	0.0093	0.1101	0.0374	-0.1537
0.100	0.0124	0.1104	0.0314	-0.5261
0.125	0.0154	0.1107	0.0282	-0.8738
0.150	0.0185	0.1108	0.0256	-1.0181
0.175	0.0216	0.1109	0.0236	-1.1209
0.200	0.0247	0.1111	0.0219	-1.2695
0.225	0.0278	0.1117	0.0207	-2.1379

Table 6.1: Normal z-scores for comparisons between Daugman's results (subscript 1) and 1600-bit simulations for various choices of q (subscript 2) are shown, along with the mean bit-error rate, $\bar{x}_2$, and standard deviation, σ_2.

$P_e = 0.11$		Mean BER	Standard Deviation	Normal Score
q	p	$\bar{x}_2$	σ_2	z
0.025	0.0031	0.1054	0.0668	4.5189
0.050	0.0062	0.1086	0.0502	1.5212
0.075	0.0093	0.1105	0.0410	-0.5245
0.100	0.0124	0.1107	0.0355	-0.8299
0.125	0.0154	0.1109	0.0317	-1.0640
0.150	0.0185	0.1108	0.0284	-0.9957
0.175	0.0216	0.1108	0.0264	-0.9341
0.200	0.0247	0.1111	0.0243	-1.3918
0.225	0.0278	0.1117	0.0230	-2.1016

Table 6.2: Normal z-scores for comparisons between Daugman's results (subscript 1) and 1304-bit simulations for various choices of q (subscript 2) are shown, along with the mean bit-error rate, $\bar{x}_2$, and standard deviation, σ_2.

With an approximate confidence level of 95%, we reject any z-values outside ± 1.96. For both $N = 1600$ and $N = 1304$, q-values in the range

$$0.05 \leq q \leq 0.2 \tag{6.10}$$

generate data well within these bounds, but values outside this range deviate excessively from Daugman's results. From Table 6.1, the closest approximation, for $N = 1600$, occurs around $q = 0.075$. We produce a histogram showing the bit-error rates of the 10^4 samples with this value of q, shown in Figure 6.3. The fact that these results are such a good approximation for Daugman's data re-inforces the assumption made in Chapter 4 that errors in iris readings occur in bursts. We now devise a means of measuring the "burstiness" of the data, in order to provide further evidence of the suitability

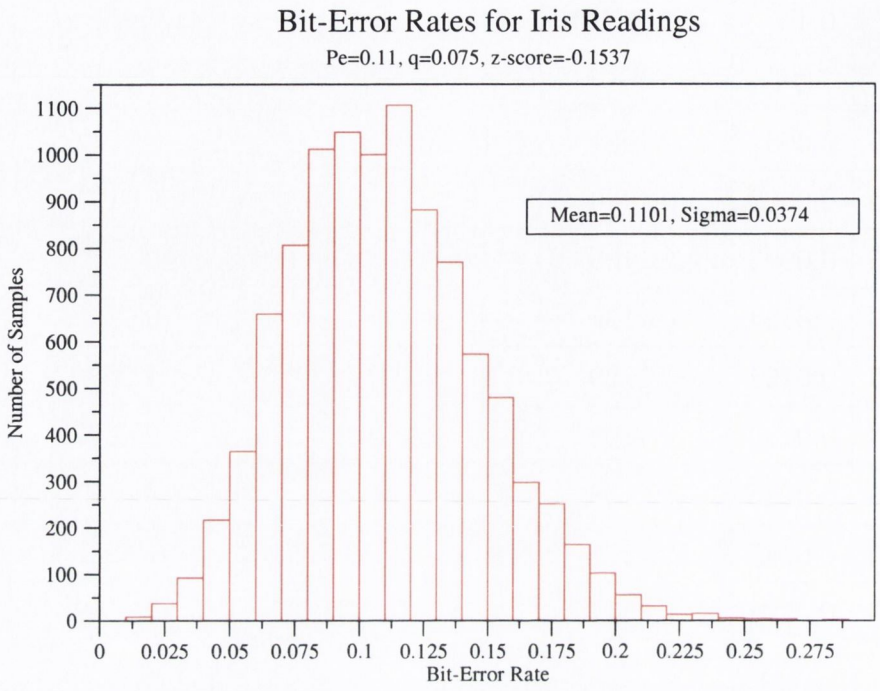


Figure 6.3: Histogram showing the simulated bit-error rates in 10^4 data strings of 1600 bits. With $P_e = 0.11$ and $q = 0.075$, the mean bit-error rate was 0.1101 with a standard deviation of 0.0374. When compared with Daugman's data, the normal z-score of -0.1537 indicates a very close match.

of choosing RS codes to correct these errors.

6.3 Further Evidence of Suitability of RS Codes

In order to measure the "burstiness" of the data, we establish the expected number of bit errors per byte of data. We have proposed the use of an RS code over GF(2^8), in which each byte will be treated as one symbol. Thus, the higher the number of bit errors per byte of data, the more suitable the RS

code is to correct this data. We again use the model we designed in Section 6.2, letting 0 represent an uncorrupted bit and 1 represent a bit error. Recall that we set $P_e = 0.11$, where P_e is the probability of the error state, and $P_n = 0.89$, where P_n is the probability of the non-error state, in agreement with Daugman's results. Once again, when the current bit is not in error, we let p be the probability that the next bit is; when the current bit is in error, we let q be the probability that the next bit is not.

We begin by determining various conditional probabilities, based on whether or not the preceding bit is in error. The formulae below can be most easily understood by referring to the state diagram shown back in Figure 6.2.

$$\begin{aligned} \mathrm{P}(0|0) &= \text{Prob(All 8 bits of byte are 0, given preceding bit is 0)} \\ &= (1-p)^8 \ , \end{aligned} \tag{6.11}$$

$$\begin{aligned} \mathrm{P}(0|1) &= \text{Prob(All 8 bits of byte are 0, given preceding bit is 1)} \\ &= q(1-p)^7 \ , \end{aligned} \tag{6.12}$$

$$\begin{aligned} \mathrm{P}(1|0) &= \text{Prob(Only 1 bit of byte is 1, given preceding bit is 0)} \\ &= pq(1-p)^6 + (1-p)pq(1-p)^5 + (1-p)^2pq(1-p)^4 + \cdots \\ &\quad \cdots + (1-p)^5pq(1-p) + (1-p)^6pq + (1-p)^7p \\ &= p(1-p)^6[7q + (1-p)] \ , \end{aligned} \tag{6.13}$$

$$\begin{aligned} \mathrm{P}(1|1) &= \text{Prob(Only 1 bit of byte is 1, given preceding bit is 1)} \\ &= (1-q)q(1-p)^6 + qpq(1-p)^5 + q(1-p)pq(1-p)^4 + \cdots \\ &\quad \cdots + q(1-p)^4pq(1-p) + q(1-p)^5pq + q(1-p)^6p \\ &= q(1-p)^5[(1-q)(1-p) + 6qp + p(1-p)] \ . \end{aligned} \tag{6.14}$$

Having calculated these, we are now in a position to establish the overall probability of no bit errors in a byte, one bit error in a byte, and so on.

$$\begin{aligned} \mathrm{P}(0) &= \text{Prob(No errors in 8-bit byte)} \\ &= P_n[\ \mathrm{P}(0|0)\ ] + P_e[\ \mathrm{P}(0|1)\ ]\ , \end{aligned} \tag{6.15}$$

$$\begin{aligned} \mathrm{P}(1) &= \text{Prob(Only 1 error in 8-bit byte)} \\ &= P_n[\ \mathrm{P}(1|0)\ ] + P_e[\ \mathrm{P}(1|1)\ ]\ . \end{aligned} \tag{6.16}$$

We can now simply group together the probabilities of two or more bits in error per byte, relying on the fact that the sum of the conditional probabilities for each state must add up to 1:

$$\begin{aligned} \mathrm{P}(\geq 2) &= \text{Prob(2 or more errors in 8-bit byte)} \\ &= P_n[1 - \mathrm{P}(0|0) - \mathrm{P}(1|0)] + P_e[1 - \mathrm{P}(0|1) - \mathrm{P}(1|1)]\ . \end{aligned} \tag{6.17}$$

We define the *expected* number of bit errors per byte of data, $E(\mathrm{err})$, to be

$$\begin{aligned} E(\mathrm{err}) &= \sum_{i=0}^{\infty} iP(i) \\ &\geq 0(P(0)) + 1(P(1)) + 2(1 - P(0) - P(1))\ . \end{aligned}$$

Note that we use greater than or equal to, to allow for the fact that we have given a weighting of 2 to our grouping of two or more errors, although some will have weights between 3 and 8. Now, since

$$\text{Prob}(i \text{ errors}|\text{byte is not error-free}) = \frac{P(i)}{1 - P(0)}\ , \tag{6.18}$$

we get

$$\begin{aligned} E(\mathrm{err}) &\geq \frac{P(1) + 2(1 - P(0) - P(1))}{1 - P(0)} \\ &= \frac{2 - 2P(0) - P(1)}{1 - P(0)}\ . \end{aligned}$$

Prob(Error| Previous Bit Not) p	Prob(No Error| Previous Bit Is) q	Expected Bit Errors Per Byte $E(\text{err})$
0.006	0.05	1.92
0.025	0.20	1.76
0.040	0.32	1.67
0.080	0.64	1.49
0.110	0.89	1.28

Table 6.3: Expected number of bit errors per byte of iris data, for a range of p and q values. We set $P_e = 0.11$ and $P_n = 0.89$, as per Daugman's data. The final row in the table corresponds to a random bit-error distribution.

Table 6.3 shows the expected number of bit errors per byte, $E(\text{err})$, for various values of p and q. Note that the final row corresponds to a random error-distribution, while the first two rows cover the range of q-values for which we found a statistical match between our generated data and Daugman's. The expected number of bit errors per byte that need to be corrected by our code is between 1.76 and 1.92. If a code type other than RS codes is used, each bit error must be corrected individually; whereas, with RS codes, each 1.76 (or 1.92) bit errors will be treated as 1 symbol error, meaning that the overall number of errors to be corrected is reduced by about 43-48%.

6.4 Choice of Code Parameters

Having demonstrated the suitability of RS codes to correct this data, we must now determine the most suitable code parameters. Let $n = 255$, its maximum value in an RS code over GF(2^8). Erasures are included in our

scheme to increase the error-correcting capacity of the code and the IrisMask is used to identify them, as mentioned in Chapter 4. However, due to the lack of information regarding the IrisMask, the percentage of erasures in an iris reading is unknown. Thus, we begin by mirroring a "worst-case" scenario in which no erasures are present and we must deal only with errors. Provided that the parameters selected correct the majority of such cases, they will be more than adequate for a real-life situation in which erasures are included. Table 6.4 shows the number of correctable errors for various k-values.

Code Length $n = 255$		
Information Bits k	Check Digits $(n-k)$	Correctable Errors t
180	75	37
160	95	47
140	115	57
120	135	67

Table 6.4: Table showing possible parameters for an RS code over GF(2^8).

Note that the check digits are effectively unhidden, as they could be accessed by an unscrupulous systems' operator. As such, we must ensure that their number does not exceed k, as this would allow the codeword to be deciphered in the following manner: Recall that RS codes are cyclic. Therefore, if $(n-k) > k$, and these check digits are known to an attacker, he could take k consecutive check digits and insert these as data in the most significant bit positions. By then calculating the corresponding check digits for these bits, and inserting them in the least significant bit positions, he has created a new codeword. This can then be rotated until the original k bits

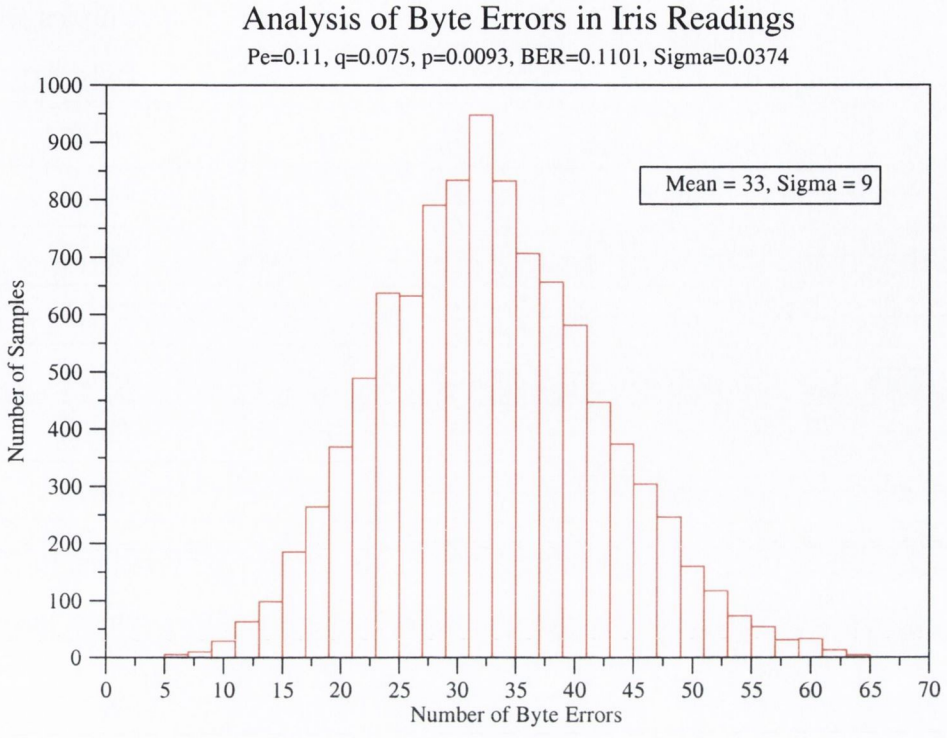


Figure 6.4: Histogram showing the simulated number of byte errors in 10^4 data strings of 1600 bits. With $P_e = 0.11$ and $q = 0.075$, the bit-error rate was 0.1101 with a standard deviation of 0.0374. This resulted in a mean number of 33 byte errors with a standard deviation of 9.

chosen are back in their proper position (so that the new "check digits" have been moved into the data section of the codeword). The attacker has now found the original codeword. To counteract this attack, we must have less than 127 check digits, making it unadvisable to use a lower k-value than 140.

We must now calculate the number of byte errors that occur for the range of acceptable values of q. We again look at $N = 1600$ and $q = 0.075$ and produce a histogram to illustrate the number of byte errors that result, shown in Figure 6.4, with a mean number of 33 byte errors and a standard deviation of 9.

Prob(0:1) q	Mean Byte Errors $\bar{x}_B$	Standard Deviation σ_B	% of Samples with ≤ 57 Byte Errors
0.050	29	11	99.00
0.075	33	9	99.04
0.100	37	9	98.74
0.125	41	8	97.17
0.150	44	8	94.21
0.175	48	8	89.06
0.200	51	8	79.15

Table 6.5: The mean number of byte errors, $\bar{x}_B$, for various q-values, in 1600-bit strings are shown, along with the percentage of samples which have 57 errors or less, and would be corrected by a $(255, 140)$ RS code.

To assess the suitability of a $(255, 140)$ RS code, we use Figure 6.4 to calculate the number of samples containing less than 57 errors. For this simulation, 99.04% of the samples are within the error-correcting capacity of such a code. We see from Table 6.5 that the lower values of q produce impressively high percentages, although the largest acceptable value of $q = 0.2$ results in over one-fifth of the samples exceeding the error-correcting capacity of a $(255, 140)$ RS code. For $N = 1304$ bits, as shown in Table 6.6, the results are far better, with the vast majority of samples across all q-values containing 57 errors or less. Our proposed inclusion of erasures would mean that almost twice the number of errors could be corrected by such a code, depending on the overall number of erasures present. Therefore, a $(255, 140)$ RS code is a suitable choice for this implementation.

Prob(0:1) q	Mean Byte Errors $\bar{x}_B$	Standard Deviation σ_B	% of Samples with ≤ 57 Byte Errors
0.050	23	9	99.88
0.075	27	9	99.93
0.100	30	8	99.90
0.125	33	8	99.87
0.150	36	7	99.69
0.175	39	7	99.28
0.200	41	7	98.52

Table 6.6: The mean number of byte errors, $\bar{x}_B$ for various q-values, in 1304-bit strings are shown, along with the percentage of samples which have 57 errors or less and would be corrected by a $(255, 140)$ RS code.

6.5 Alternative Iris Recognition Studies

In addition to Daugman's results for non-ideal conditions similar to verification, he has published data from "ideal" studies undertaken in highly controlled conditions. The error rates in such cases are greatly reduced, as can be seen in Figure 6.5. The mean bit-error rate given here is 1.9%, with a standard deviation of 3.9%. Although these conditions are unrealistic and so cannot be expected to occur in a practical use of this algorithm, it would be instructive to investigate if our model can also re-create these results.

Unfortunately, we only have access to a restricted version of the report on the study which generated these results, so although we know there were 482,600 cross-comparions done in total, we cannot be entirely sure of the sample size used for the same-iris comparisons. However, we know that there were 416 different irises used in the study; it seems extremely likely

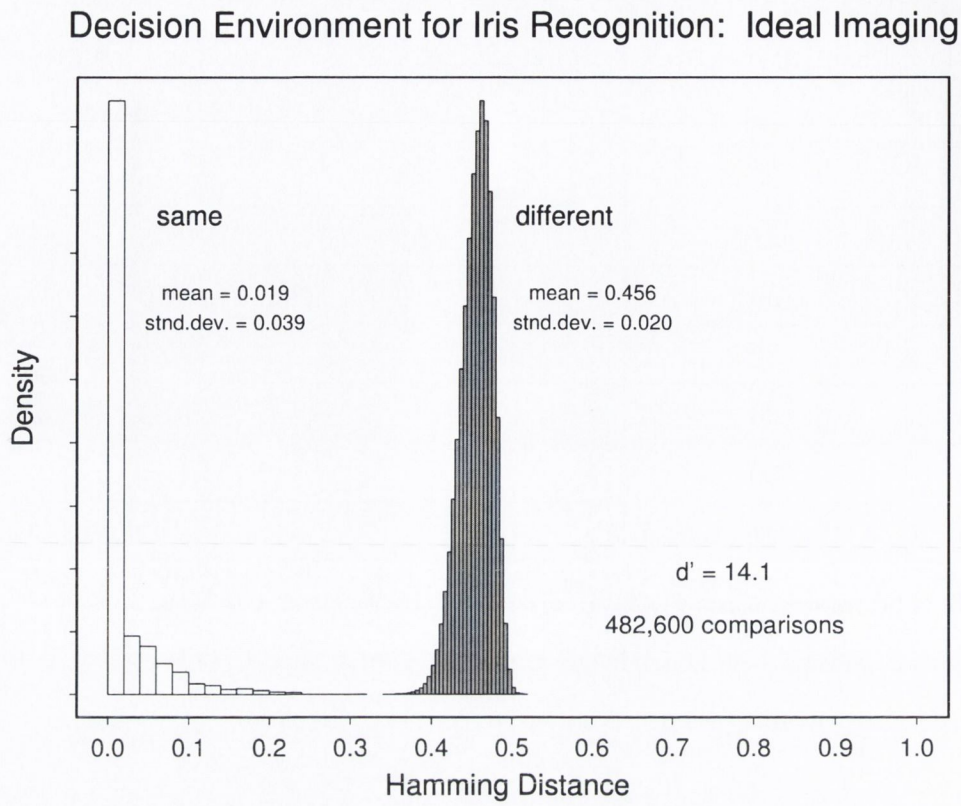


Figure 6.5: Results of Daugman's algorithm using artificially ideal conditions [Dau04]. The left histogram represents identical irises.

that each of these would have been enrolled once using the "ideal" conditions. Therefore, we assume $n_1 = 416$ in this case.

Setting $N = 1600$ bits, we repeat our analysis of the normal z-scores produced for the simulations resulting from various values of q. We find that, once $q > 0.06$, the calculated z-scores are within the bounds of acceptability for approximate confidence level of 95%. For the maximum value of $q = 1.0$, we have a mean number of 29 bytes in errors with a standard deviation of 5. In the 10^4 samples we simulated, 100% fell below the threshold of 57 errors set by the choice of a $(255, 140)$ RS code. Thus, *all* acceptable values of q

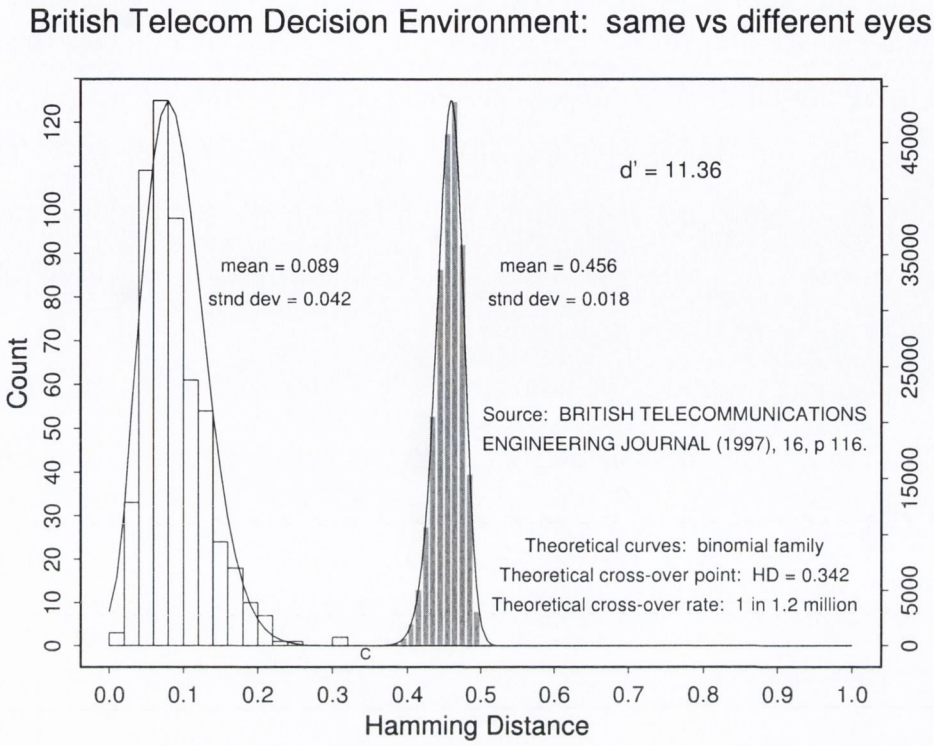


Figure 6.6: Results of independent tests on Daugman's algorithm done by British Telecom Labs [SGM97]. The left histogram represents identical irises.

produce simulations which can be corrected by such a code. Clearly, this will also be the case if N is reduced to 1304 bits.

While these results are undoubtedly unrealistic, it is further evidence of the suitability of our model that it can achieve such accurate approximations for these parameters also. Therefore, we consider one final test of Daugman's data, produced in 1997 by British Telecom Labs [SGM97]. The study involved 222,743 cross-comparisons although only 340 of these consisted of same-same comparisons, creating the left-hand distribution shown in Figure 6.6. (Note that, although a binomial curve has been fitted to this data, we

do not believe that it is strictly binomial, particularly based on the data from other studies. We believe that the right-hand distribution does follow the binomial distribution, but we are not concerned with this.) Our new $n_1 = 340$; the mean of the distribution is 8.9% with a standard deviation of 4.2%. This mean is much lower than that produced by Daugman's comparable study, but the sample size in this case is significantly smaller and so it is likely that the results are considerably less accurate.

Once again, we repeat our analysis, with $N = 1600$ and $N = 1304$ bits. A wide range of q-values again produce desirable z-scores; indeed, for $N = 1600$, any value in the range

$$0.0175 \leq q \leq 0.75 \tag{6.19}$$

should give a z-score between ± 1.96, while for $N = 1304$, a range of

$$0.025 \leq q \leq 0.75 \tag{6.20}$$

is suitable. Tables 6.7 and 6.8 show the percentage of our simulated samples which contain less than 57 errors. Here, our $(255, 140)$ RS code may not as suitable as before, as large proportions of the samples are outside the upper limit of 57 errors. However, it should be remembered that even our highest recorded mean of 96 errors could be corrected by the code, should it prove possible to split these into 80 erasures and 17 errors.

A further detailed test was carried out by the Dutch company Joh. Enschedé in 2000, resulting in a mean bit-error rates of 13.2% with a standard deviation of 7.2%. However, although we know there were 19,900 cross-comparisons done in this study, we do not know how many of these affected the same-same comparison and so are unable to test our model against this data in the above manner.

q	Mean Byte Errors	Stan Dev	% Samples $\leq$ 57 Byte Errors
0.0175	19	14	98.66
0.1000	30	8	99.89
0.2000	41	7	98.21
0.3000	52	7	76.98
0.4000	63	7	21.15
0.5000	73	7	01.79
0.6000	81	7	00.04
0.7500	96	7	00.00

Table 6.7: The mean number of byte errors, $\bar{x}_B$, for various values of q, in 1600-bit strings are shown, along with the percentage of samples with 57 errors or less, as these would be corrected by a $(255, 140)$ RS code.

q	Mean Byte Errors	Stan Dev	% Samples $\leq$ 57 Byte Errors
0.025	16	11	99.98
0.100	24	7	100.00
0.200	34	7	99.95
0.300	42	7	98.74
0.400	52	7	81.42
0.500	60	7	38.92
0.600	66	7	10.09
0.750	79	6	00.08

Table 6.8: The mean number of byte errors, $\bar{x}_B$, for various values of q, in 1304-bit strings are shown, along with the percentage of samples with 57 errors or less, as these would be corrected by a $(255, 140)$ RS code.

6.6 Chosen Parameters

We have now generated a series of data strings, based on the published results of various studies on Daugman's algorithm. Clearly the most accurate of these studies must be assumed to be Daugman's "non-ideal" test - the first that we carried out. This study involved the largest database by far; the results were garnered using different cameras and distances; therefore, it may be presumed to be the most indicative of a real-life scenario. The choice of a $(255, 140)$ RS code satisfied all of the 10^4 samples generated for the appropriate q-values for the shorter string-length of 1304 bits, and over 80% of the suitable samples for the longest possible string-length of 1600 bits. In a real-life scenario, it will be possible to classify a number of the errors as erasures, allowing us to correct even more discrepancies with these parameters. Therefore, this code should be suitable for use in a practical day-to-day implementation of this algorithm. We examine how this might be done in the next section, describing a computer simulation for our suggested iris recognition technique using Reed-Solomon codes.

6.7 RS Iris Matching Algorithm

As mentioned in our first design chapter, there are two main stages in our iris matching algorithm: enrolment and verification. To begin, the user must enrol in the system by submitting to an iris reading. This reading is then processed in the usual manner. Check digits are calculated for the reading and appended to it to form a codeword. These check digits are then stored together with a hash of the codeword in the user's file.

For the verification procedure, the user again provides an iris reading, together with some claim of identity. This claim retrieves the check digits

and the hashed enrolment reading from the relevant file. The verification reading is treated as a corrupted codeword and the check digits are used to attempt to correct this. If the hashed result of the corrected codeword is found to be equal to the hashed value of the original codeword stored for the user, then the verification is considered to be valid.

One major advantage of our approach is that it is no longer necessary to store the individual's biometric template explicitly in some kind of file or database. As a result, the biometric is not put at unnecessary risk from dishonest systems operators.

For our proposed scheme, assume a partial implementation of Daugman's algorithm [Dau93], which provides an IrisCode and corresponding mask, but does not employ his technique for iris matching. In Chapter 3, his algorithm was outlined in some detail. The 256-byte mask contains the locations of bits to be omitted from the reading because of occlusions by eyelids, eyelashes and so on. Information as to the formation or exact structure of the mask is not available. However, prior to the invention of this IrisMask, Daugman's patent[Dau94] suggested methods for eliminating such bits:

1. Assuming that the iris is split into rings with successively larger radii, let the smallest such ring have a radius of 1.1 times the average radius of the pupil, as the pupil may have a slightly irregular boundary.

2. Similarly, let the largest ring have a radius that is only 80% of the estimated full radius of the iris.

3. In order to exclude the eyelids, let the larger rings be restricted to two cones which exclude the upper and lowermost sections of the iris.

4. To counteract specular reflection, which frequently occurs as a result

of an oblique illumination source, exclude a narrow wedge at the "6-o'clock position" of the iris.

These measures are not explicitly mentioned in any of Daugman's more recent papers, suggesting that they have now been incorporated into the IrisMask. Therefore, it is reasonable to assume that the IrisCode is now created from a fuller ring, and the IrisMask contains the locations of all such sections that should be omitted. This is further reinforced by the fact that Daugman makes a direct link between the percentage of the iris that is visible and the number of bits that are compared in the IrisCode, implying that the IrisCode contains all such corrupted bits.

6.7.1 Program for RS Iris Algorithm

We produced two separate programs written in C: one for enrolment and one for verification. A user must enrol in the system prior to attempting the verification process. To begin, he types "Enrol" and follows the on-screen instructions, as outlined in the next section. Data relevant to this user will be stored in a file under the name he has chosen. When this stage is completed, he may proceed to the verification procedure by typing "Verify". Having done so, he will be informed if his attempt has failed or been successful. Examples of the output from each of these programs can be found in Appendix G.

6.7.2 Enrolment

When a user types "Enrol", he must identify himself by name, and is prompted to select one of a series of 256-byte data-strings to represent his iris reading. The name of this data-string is stored in his file for use in verification.

Once the data has been successfully read in and interpreted in the desired

hexadecimal format, the user must choose 116 bytes to be excluded from his reading. These are the "masking bytes", which represent areas of the iris occluded by eyelashes, eyelids and so on. They are permanently deleted from the reading, reducing it to 140 bytes of data. These 116 byte positions are stored in the user's file to enable the program to recreate identical initial data during verification. The user is also given the option of modifying an arbitrary number of data bytes, to ensure that his reading is unique.

At this stage, check digits are calculated for the modified data, by dividing the data by the generating polynomial, $g(x)$, of our $(255, 140)$ RS code over GF(2^8). To obtain $g(x)$, a program was written to multiply polynomials in GF(2^8), and then used to calculate:

$$g(x) = \prod_{i=0}^{114}(x - \alpha^i) \ , \tag{6.21}$$

where α^i is a root in GF(2^8) with minimum polynomial $(x - \alpha^i)$. The data is then organised into a polynomial array and divided by $g(x)$, as required. The remainder obtained gives the 115 check digits. These are written to the user's file, to be used to correct errors during the verification process.

In the final part of enrolment, the user's data is hashed and stored in his file for comparison purposes later. To accomplish this, we use GNU's Libgcrypt [GNU03] library of cryptographic functions to implement the SHA-1 hash algorithm (which should now be replaced by SHA-256 for greater security). The user has been successfully enrolled and the program terminates.

6.7.3 Verification

Once a user has enrolled, it is important to ensure that the system recognises him by verifying that he is who he claims to be. Thus, the user types "Verify" and again, is asked for his name. If this does not match a name previously on

file, he is prompted to enrol in the system; otherwise, his file is opened and the data-string he chose on enrolment is accessed. The 116 bytes corresponding to the enrolment mask are again deleted from the string. This time, the user is also given the option of choosing to treat certain remaining bytes as "erasures", errors whose locations, but not values, are known.

The number of errors which can be corrected depends on the number of erasures entered. The distance of the code is 116, and so if we let s be the number of erasures and r the number of errors, we must have

$$2r + s \leq 115 \tag{6.22}$$

if we wish to be able to correct these. We have already modified x bytes of data in the enrolment procedure. These values are not modified in the verification data string and so will be treated as errors in the verification process. Thus, we must in fact have

$$2r + s \leq 115 - 2x \ . \tag{6.23}$$

The user can modify a certain number of bytes, once this relationship is upheld, to simulate errors in the verification reading, either from system errors or a slight natural difference between two biometric readings.

The next stage of the program is to create a corrupted codeword and attempt to correct it. We form the codeword by extracting the check digits calculated at enrolment from the user's file and appending them to the data polynomial we have just produced. At this stage, we must implement the eight steps outlined previously for correcting errors and erasures in RS codes.

1. To find the values of f_j, we use the fact that we know that the erasure locators $\tilde{X}_i$ are the roots of the equation

$$f_0 x^s + f_1 x^{s-1} + \cdots + f_{s-1} x + f_s = 0 \ , \tag{6.24}$$

where $f_0 = 1$. We use our function for multiplying polynomials to get

$$\prod_{j=0}^{s}(x - \tilde{X}_j) \ . \tag{6.25}$$

The polynomial produced gives us the values for f_j.

2. Calculating the syndromes, S, is straightforward, as we can determine these from the corrupted codeword c, using the formula

$$S = c \times H^T \ , \tag{6.26}$$

where H^T is the parity-check matrix of size 255×115 and S is a 115-element vector. If c has not been corrupted, all syndromes are zero.

3. We must now calculate

$$T_i = \sum_{j=0}^{s} S_{s+i-j} f_j, \text{ for } i = 1, \ldots, d - s \ , \tag{6.27}$$

which is quickly accomplished by multiplying the syndromes by the f_j. Again, all T_i are zero if there are no errors to be corrected.

4. We need to find the values of p_i, the coefficients of the polynomial

$$p(x) = p_0 x^r + p_1 x^{r-1} + \cdots + p_r \ , \tag{6.28}$$

with $p_0 = 1$, and the error locators, X_k, as roots. These are ultimately used to find X_k, so this step is unnecessary if there are no errors. Otherwise, we use the Massey-Berlekamp [Mas69] algorithm to solve for p_i, using the equation

$$T_{i+r} + p_1 T_{i+r-1} + \cdots + p_r T_i \ . \tag{6.29}$$

As well as finding the values of p_i, this process will simultaneously calculate r, the number of errors in the corrupted codeword.

5. Now, we solve the polynomial $p(x)$ to find its roots, the error locators, X_k. We have found a root when a value of x results in the polynomial going to zero. Thus, we try values of α less than α^{255} (the largest possible value in our code) until we have found all the roots.

6. At this stage, we employ Gaussian Elimination with partial pivoting and back substitution in order to find the E_k from the equation

$$T_i = \sum_{k=1}^{r} E_k X_k^i \ . \tag{6.30}$$

Gaussian Elimination is a well-known technique for solving matrix equations of the form $Ax = B$. We choose an element in the diagonal of the matrix and divide the other elements by this "pivot". Partial pivoting involves interchanging the rows of the matrices to place a "good" element in the diagonal position prior to a certain operation [Res06].

7. The error values, Y_k, can be immediately calculated from E_k, using

$$E_k = Y_k f(X_k) \ . \tag{6.31}$$

8. Finally, to calculate the erasure values, we again use Gaussian Elimination to find $\tilde{Y}_k$. Using our formula

$$S_k = \sum_{i=1}^{s} \tilde{Y}_i \tilde{X}_i^k + \sum_{j=1}^{r} Y_j X_j^k \ , \tag{6.32a}$$

we firstly create the matrix

$$M_k = S_k + \sum_{j=1}^{r} Y_j X_j^k \ , \tag{6.32b}$$

and then use this quantity to calculate $\tilde{Y}_k$ from the equation

$$M_k = \sum_{i=1}^{s} \tilde{Y}_i \tilde{X}_i^k \ . \tag{6.32c}$$

Once these stages have been completed, we correct the corrupted codeword by modifying the relevant values as necessary, using our error and erasure locators to ascertain the positions, and Y_k and $\tilde{Y}_k$ for the new values. We form the hash of this newly restored codeword, again using GNU's Libgcrypt function for SHA-1, and compare this value with the hash stored in the user's file. If these match, the user has been successfully verified.

6.7.4 Programming Challenges

As our calculations all take place over the ground field GF(2^8), some modifications to normal arithmetic must be implemented. As mentioned previously, the characteristic of the field is two, so addition and subtraction are equivalent, allowing us to use the exclusive-OR (XOR) function.

Using the primitive polynomial

$$x^8 + x^5 + x^3 + x + 1 \ , \tag{6.33}$$

it was necessary to generate look-up tables, in which each possible power of α corresponds to a different eight-bit representation, which we store as a 2-digit hexadecimal number for simplicity. The usual 8-bit binary representation of each power of α is firstly determined. If this is less than $(80)_{16}$, there will be a zero in the first position. In this case, we simply bit shift the number one place to the left, and store the resulting value. For example, if we begin with α^1, we know that $(1)_{10} = (00000001)_2 = (2)_{16}$. Therefore, we store 2 in the column opposite α^1 in the look-up table. Then, in order to calculate α^2, we bit-shift the last binary value one place to the left to get $(2)_{10} = (00000010)_2 = (4)_{16}$, and we store 4 in the column opposite α^2 in the table. Continuing thus, we get $(3)_{10} = (00000100)_2 = (8)_{16}$ and so on, as illustrated in Table 6.9. If the binary representation is greater than or equal to $(80)_{16}$ however, there

is a 1 in the first position of the 8-bit binary representation. This means that we cannot simply bit-shift the number as before. Thus, we AND the number with $(01111111)_2$ to remove the leading 1, then bit shift it one place to the left, and finally XOR the result with $(00101011)_2$, which corresponds to our primitive polynomial. For example, α^{14} in our table corresponds to $(e5)_{16} = (11100101)_2$. If we AND this with $(01111111)_2$, we get

$$(11100101)_2 \cap (01111111)_2 = (01100101)_2 \ . \tag{6.34}$$

We then bit-shift this one place to the left to get $(11001010)_2$, which we then XOR with our primitive polynomial to get

$$(11001010)_2 \otimes (00101011)_2 = (11100001)_2 \ , \tag{6.35}$$

which is equal to $(e1)_{16}$. This value corresponds to α^{15} in our table.

We also generate a second look-up table, ordered by hexadecimal value, rather than by power by α. These look-up tables are used throughout the algorithm to return the correct result for addition. The code used to generate these tables, and the full output, can be found in Appendix E. If we let α represent the basic root in this field and all other roots are powers of α, then the operation of multiplication is implemented by adding the relevant powers of α modulo 255 and the operation of addition is achieved by XORing the bit patterns corresponding to the powers of α.

α^x	16	α^x	16	α^x	16	α^x	16	α^x	16	α^x	16	α^x	16	α^x	16
0	0	32	ee	64	fb	96	b9	128	c1	160	54	192	23	224	cb
1	2	33	f7	65	dd	97	59	129	a9	161	a8	193	46	225	bd
2	4	34	c5	66	91	98	b2	130	79	162	7b	194	8c	226	51
3	8	35	a1	67	9	99	4f	131	f2	163	f6	195	33	227	a2
4	10	36	69	68	12	100	9e	132	cf	164	c7	196	66	228	6f
5	20	37	d2	69	24	101	17	133	b5	165	a5	197	cc	229	de
6	40	38	8f	70	48	102	2e	134	41	166	61	198	b3	230	97
7	80	39	35	71	90	103	5c	135	82	167	c2	199	4d	231	5
8	2b	40	6a	72	b	104	b8	136	2f	168	af	200	9a	232	a
9	56	41	d4	73	16	105	5b	137	5e	169	75	201	1f	233	14
10	ac	42	83	74	2c	106	b6	138	bc	170	ea	202	3e	234	28
11	73	43	2d	75	58	107	47	139	53	171	ff	203	7c	235	50
12	e6	44	5a	76	b0	108	8e	140	a6	172	d5	204	f8	236	a0
13	e7	45	b4	77	4b	109	37	141	67	173	81	205	db	237	6b
14	e5	46	43	78	96	110	6e	142	ce	174	29	206	9d	238	d6
15	e1	47	86	79	7	111	dc	143	b7	175	52	207	11	239	87
16	e9	48	27	80	e	112	93	144	45	176	a4	208	22	240	25
17	f9	49	4e	81	1c	113	d	145	8a	177	63	209	44	241	4a
18	d9	50	9c	82	38	114	1a	146	3f	178	c6	210	88	242	94
19	99	51	13	83	70	115	34	147	7e	179	a7	211	3b	243	3
20	19	52	26	84	e0	116	68	148	fc	180	65	212	76	244	6
21	32	53	4c	85	eb	117	d0	149	d3	181	ca	213	ec	245	c
22	64	54	98	86	fd	118	8b	150	8d	182	bf	214	f3	246	18
23	c8	55	1b	87	d1	119	3d	151	31	183	55	215	cd	247	30
24	bb	56	36	88	89	120	7a	152	62	184	aa	216	b1	248	60
25	5d	57	6c	89	39	121	f4	153	c4	185	7f	217	49	249	c0
26	ba	58	d8	90	72	122	c3	154	a3	186	fe	218	92	250	ab
27	5f	59	9b	91	e4	123	ad	155	6d	187	d7	219	f	251	7d
28	be	60	1d	92	e3	124	71	156	da	188	85	220	1e	252	fa
29	57	61	3a	93	ed	125	e2	157	9f	189	21	221	3c	253	df
30	ae	62	74	94	f1	126	ef	158	15	190	42	222	78	254	95
31	77	63	e8	95	c9	127	f5	159	2a	191	84	223	f0	255	1

Table 6.9: Look-up table for GF(2^8), generated using the primitive polynomial $x^8+x^5+x^3+x+1$, and presented in increasing powers of α. The power of α is in the first column and the hexadecimal value in the second column.

Thus, we have

$$\begin{aligned} \alpha^{149} \times \alpha^{114} &= \alpha^{(149+114)} \\ &= \alpha^{263} \\ &= \alpha^{8} \bmod 255 \ , \end{aligned} \tag{6.36}$$

and in the case of addition

$$\begin{aligned} \alpha^{149} + \alpha^{114} &= 11010011 \otimes 00011010 \\ &= 11001001 \\ &= \alpha^{95} \ . \end{aligned} \tag{6.37}$$

We need to store α^i throughout the program, and we do so by simply letting array$[x] = i$ for the relevant position x. However, this creates a difficulty when representing zero, as no number y exists such that $\alpha^y = 0$. This means that

$$\alpha^{255} = \alpha^0 \bmod 255 \ . \tag{6.38}$$

To overcome this difficulty, and enable us to differentiate between α^0 and $\alpha^i = 0$, we *always* represent α^0 as α^{255}. Thus, if we set array$[x] = 0$, this corresponds to $\alpha^i = 0$, whereas if we set array$[x] = 255$, this represents α^0. We include checks throughout the program to ensure that this is being correctly implemented, setting zero to 255 when appropriate. This can also be seen in Table 6.9, where $\alpha^0 = 0$ and $\alpha^{255} = 1$, as required.

The correction procedure we have programmed corrects both errors and erasures. If no erasures are present, there is no need for the final step in the algorithm. If no errors are present, stages four to seven are not required, reducing our work considerably. We detect this case by checking if all $T_i = 0$; if this is the case, then no errors exist, and we skip directly to stage eight in the program to avoid unnecessary empty calculations.

6.8 Conclusion

In this chapter, we modelled the distribution of errors in iris readings processed using Daugman's algorithm. Having produced simulated readings according to a probabilistic model, we ran tests comparing our results to Daugman's published findings, and found that these were statistically identical. We then measured the byte-error rate in these readings, and found that a $(255, 140)$ RS code over GF(2^8) would best correct these. Finally, we described a practical implementation of this system that we produced in C, and discussed the challenges it posed and how we overcame these effectively.

Chapter 7

Implementation and Analysis: Masek

"If a listener nods his head when you're explaining your program, wake him up." - Alan Perlis

Given the nature of the commercial proprietory issues surrounding Daugman's algorithm, it is natural to consider whether an alternative algorithm would allow us to produce definitive results for our scheme, based on real rather than simulated data. It is important to remember that we wish to assess the viability of using error-correcting codes to match iris readings, and therefore it is logical to rely on the most accurate iris processing algorithm commercially available, namely Daugman's. The drawback of using a different algorithm is that, should our system fail, it will be difficult to ascertain whether this is due to the shortcomings of the iris processing algorithm or whether error-correcting codes are unsuitable for iris matching.

However, results based on real data would strengthen our study, and therefore we deemed it worthwhile to implement the only open-source iris processing algorithm available, produced by Libor Masek [Mas03].

7.1 Masek's Iris Processing Algorithm

Masek's iris processing algorithm is based on a partial implementation of Daugman's algorithm, with several variations, and is written in MATLAB. Firstly, instead of the integrodifferential operator used by Daugman in order to locate the iris within the image, Masek uses Hough transforms, in a manner similar to that originally suggested by Wildes et al [WAH+96]. The outer and inner boundaries of the iris are located using a circular Hough transform; then a linear Hough transform is employed to identify the eyelids. However, problems associated with this approach include the need for threshold values to be set for edge detection (values which will vary depending on the database of iris readings used); and the fact that the process is computationally intensive and may not be suitable for real-time applications. Eyelashes are also detected using thresholding, provided they are sufficiently dark within the image.

Masek then uses a projected polar coordinate system similar to Daugman's in order to normalise the data. The only difference between his creation of an "IrisCode-equivalent" and Daugman's approach is that he favours convolution using 1D Log-Gabor filters in place of Daugman's 2D Gabor filters, but the phase quantisation stage is identical. However, problems were noted with varying amounts of pupil dilation, as the algorithm does not always correctly reconstruct the same pattern in such cases.

Finally, Masek mirrors Daugman's idea of creating an IrisMask consisting of corrupted bits which should be omitted from the matching process, and recommends the use of Hamming Distances to match the readings. He recommends that the verification template be bit-shifted several times to achieve the best match with the enrolment template, as done by Daugman. This combats any rotational invariance in the readings.

7.2 Choice of Iris Database

In order to implement the iris processing algorithm, we need a database of iris readings. Several such databases are openly available for testing purposes; perhaps the most well-known are the three databases produced by the Chinese Academy of Sciences Institute of Automation (CASIA) [Chi06]. The earliest of these databases (which we will term "Ver1") consists of "ideal" images, captured in a highly controlled laboratory setting. The images were taken from 108 different irises. Seven images are available of each iris: three taken initially, and a further four images captured one month later, giving a total of 756 images in the database.

The second and third databases were recently compiled, using an identical pool of 60 irises, but two separate devices to capture the images. We refer to them as "Ver2Dev1" and "Ver2Dev2". Twenty images of each iris were taken using each device, giving a total of 1200 images in each database. These images were not as carefully controlled and are more challenging to identify.

"UBIRIS" [PA05], a "noisy" iris database, has recently become available; it consists of 1877 images from 241 individuals, taken in two distinct sessions. This database is intended to robustly test iris processing algorithms, with images of a poorer quality than CASIA databases. Unless Masek's algorithm performs well for all CASIA databases, we will not consider UBIRIS further.

7.2.1 Choice of Parameters

Our choice of RS code parameters depended on the size of Daugman's IrisCode (2048 bits) and the bit-error rate within the readings. To produce the best comparison, we wish to produce templates of the same size with Masek's algorithm. The size of the iris templates he produces is determined by three

parameters: the number of 1D Log-Gabor filters used and the radial and angular resolutions chosen when the doughnut-shaped iris image is mapped into a template. The radial resolution is the number of data points chosen along each radial line and the angular resolution is the number of radial lines going around the iris region. Masek states that only one filter is needed, and so only the resolutions need concern us.

To determine the optimum resolutions for the CASIA Ver1 database, Masek calculates the "decidability" for a variety of resolutions. Decidability, d, is a metric defined by Daugman as a decision-making power, where

$$d = \frac{|\mu_1 - \mu_2|}{\sqrt{\frac{(\sigma_1^2+\sigma_2^2)}{2}}} \ , \tag{7.1}$$

with μ_1 and σ_1 representing the mean and standard deviation respectively of the "intra-class distribution" and μ_2 and σ_2 those of the "inter-class distribution". The intra-class distribution refers to the error-rates recorded between different readings of the same iris, while the inter-class distribution is formed from error-rates between readings from different irises. The higher the decidability, the greater the separation between these two distributions, resulting in more accurate recognition of individuals.

As can be seen from Table 7.1, the optimum radial and angular resolutions are 32 pixels by 240 pixels respectively, although Masek recommends the use of 20 pixels by 240 pixels for more efficient coding. However, as the filter produces two bits for each phasor inputted, each iris template adhering to these recommendations would consist of 9600 bits (the product of the angular and radial resolutions times two). Therefore, in order to produce templates of size 2048 bits, we choose an angular resolution of 256 pixels and a radial resolution of 4 pixels. This results in a decidability of

$$5.028 \leq d \leq 5.443 \ , \tag{7.2}$$

Angular Resolution	Radial Resolution							
	4	8	12	16	20	24	28	32
200	4.732	5.454	5.680	5.911	6.012	6.094	6.109	6.183
240	5.028	5.520	5.931	6.110	6.199	6.256	6.313	**6.322**
280	5.443	5.532	5.903	6.134	6.210	6.275	6.299	6.311

Table 7.1: Decidability of the CASIA Ver1 iris database, as determined by Masek [Mas03, p. 37] for various values of angular and radial resolution (given in pixels). The optimum value is shown in bold.

which should be sufficiently large to allow for accurate recognition.

We now run the images from each CASIA database through Masek's iris processing algorithm and assess the number of images in which the iris boundaries are correctly identified, segmenting the iris from the overall image. Figure 7.1 shows the output for several different iris readings, with the white circles indicating the iris boundaries, as calculated by the algorithm. As expected, the "ideal" images from the Ver1 database produced considerably better results than the other two databases, as indicated in Table 7.2. In

CASIA Database	No. of Different Irises	Images per Iris	Total No. of Images	Correctly Segmented Images
Ver1	108	7	756	630 (83%)
Ver2Dev1	60	20	1200	448 (37%)
Ver2Dev2	60	20	1200	266 (21%)

Table 7.2: Summary of the results of using Masek's iris processing algorithm on the three different CASIA iris databases. CASIA Ver2 used two different devices (Dev1 and Dev2) to capture the iris images.

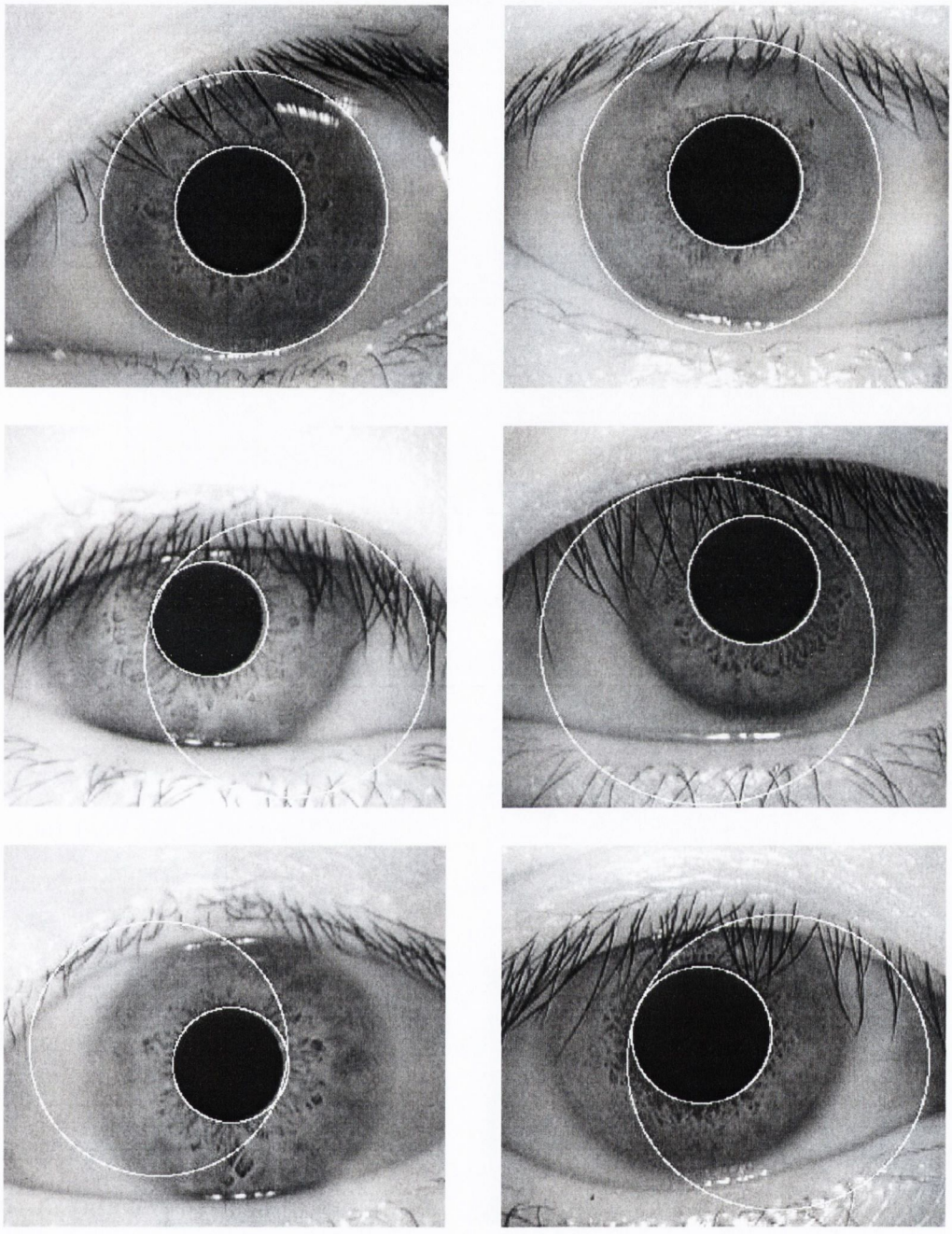


Figure 7.1: Iris images from six different individuals in the CASIA Ver1 database, which have been segmented using Masek's algorithm. The white circles show the iris boundaries, which have been incorrectly identified by the algorithm in the bottom four images.

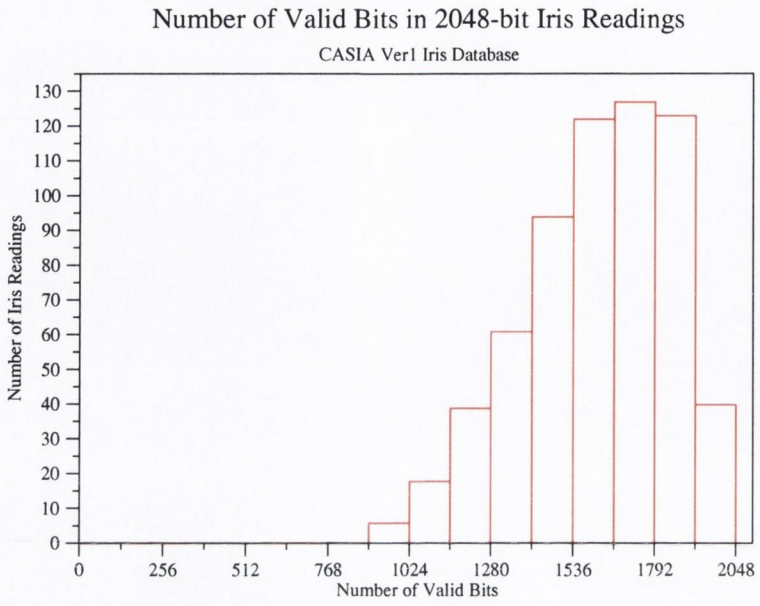


Figure 7.2: Number of valid bits in the correctly segmented 2048-bit iris readings from the CASIA Ver1 database.

83% of the images in Ver1, the iris was correctly segmented, compared with 37% in Ver2Dev1, and a mere 21% in Ver2Dev2. The results for Ver1 match those produced by Masek for the database; however, Ver2 has not previously been tested for his implementation.

However, in addition to the low number of images correctly segmented in the Ver2 databases, the number of bits marked as "corrupted" in the iris masks in these databases is very high. Figures 7.2 - 7.4 show the number of valid bits remaining for the correctly segmented images in each CASIA database. Figure 7.2 indicates that a high number of bits are valid in the Ver1 database, with 80% of these images possessing more than 1400 valid bits, and none dropping lower than 896 valid bits. However, in Figure 7.3, we see that *none* of the readings from the Ver2Dev1 database has more than 1152 valid bits, with about three quarters having less than 512 valid bits.

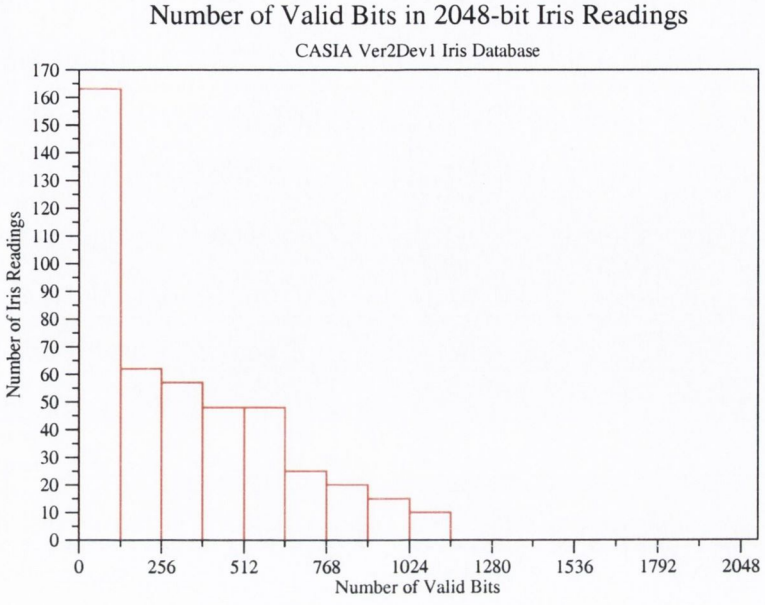


Figure 7.3: Number of valid bits in the correctly segmented 2048-bit iris readings from the CASIA Ver2Dev1 database.

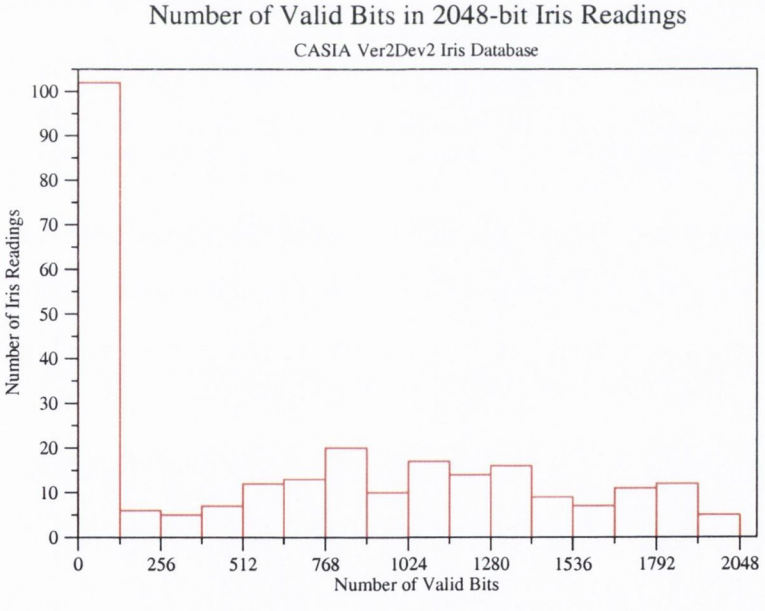


Figure 7.4: Number of valid bits in the correctly segmented 2048-bit iris readings from the CASIA Ver2Dev2 database.

In Figure 7.4, the results are marginally better, with a quarter of the correctly segmented images having more than 1152 valid bits; but this only corresponds to about 60 images from a database of 1200. A combination of a low percentage of correctly identified images with a high number of corrupted bits in these images leads us to reject the idea of completing any further tests on the Ver2 databases. Thus, we do not test the UBIRIS database, as those images are even more challenging and would not yield useful results.

Although the results for the CASIA Ver1 database are too low for real-life implementations, they are sufficiently good to warrant further study. Therefore, we now look at the error-rates within this data, and ascertain if a $(255, 140)$ RS code over GF(2^8) would be sufficient to correct these.

7.3 Implementation

For matching purposes, we treat each of the first three images taken for each iris as enrolment images, and the four images taken a month later as verification images. Thus, we have a maximum of $(3 \times 4) = 12$ comparisons for each iris, although in many cases, this number is reduced as one or more of the images has been incorrectly segmented by the algorithm. As a result, we can carry out a total of 968 same-iris comparisons between valid templates.

To see if our proposed RS code could correct these readings, we must determine the number of errors and erasures in the final templates. If we have r errors and s erasures, we define the *measure of disparity*, m, to be

$$m = 2r + s \ , \tag{7.3}$$

and, to correct all errors and erasures, require that

$$m \leq (n - k) = 115 \ . \tag{7.4}$$

Firstly, any bits deemed to be invalid in the enrolment mask are automatically deleted from both the enrolment and verification readings. Only the first 1120 valid bits (140 bytes) of a reading are used to form the information bits of a codeword in our (255,140) RS code, and so we focus solely on these. Any bits marked in the verification mask which did not appear in the enrolment mask are set to the value 2 in the reduced verification template. By counting the number of bytes with one or more 2s, we determine the number of byte erasures. All other bytes are compared with their corresponding one in the enrolment template to ascertain whether any errors exist.

Masek recommends bit-shifting the verification template eight shifts to the left and eight to the right for optimum results. Thus, having calculated the number of byte errors and byte erasures in a comparison, we bit-shift the verification template and mask and repeat the process. After 16 shifts, we determine the lowest value of m, the measure of disparity, and choose this as the best match between the readings.

7.4 Results and Analysis

Figure 7.5 shows the measure of disparity, m, calculated for 968 comparisons between same-iris images, as described above. As can be seen from the histogram, the mean m-value is 214, with a standard deviation of 16. The minimum value recorded was 156, which is far above the value of $m = 115$ which can be corrected by the (255,140) RS code. Therefore, we conclude that this code would be an unsuitable choice to correct this data.

However, as mentioned earlier, this unsuitability could be due to the fact that Masek's algorithm is not sufficiently accurate. To investigate this further, we look at the bit-error rate for iris readings processed using his

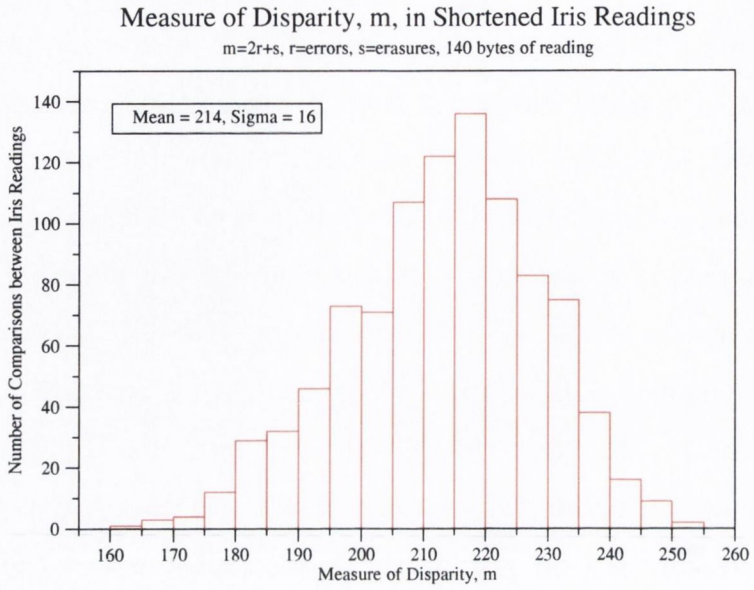


Figure 7.5: Measure of disparity, m, in comparisons between the first 140 bytes of iris readings processed using Masek's algorithm.

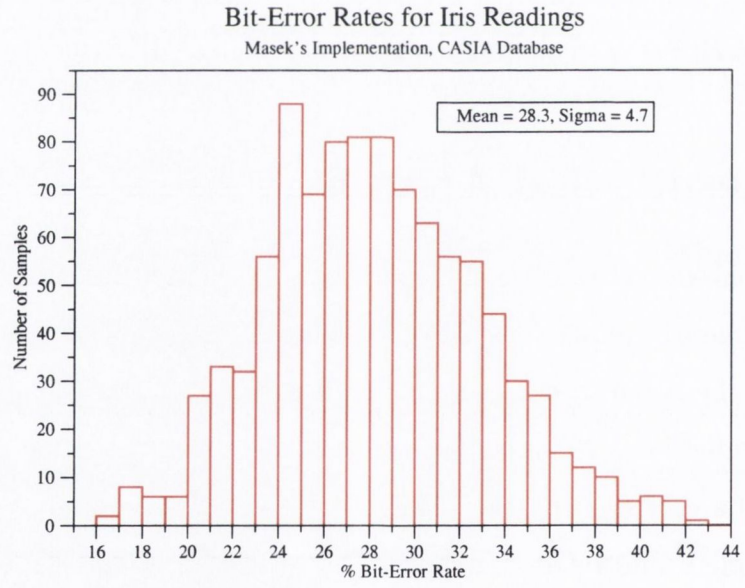


Figure 7.6: Histogram showing the percentage bit-error rate for CASIA Ver1 processed using Masek's algorithm.

algorithm and compare it with that of Daugman's algorithm. To achieve the most accurate comparison, we omit any bits that are included in either the enrolment or verification masks, as done by Daugman. Thus, only bits considered valid in both readings are compared. As shown in Figure 7.6, the mean bit-error rate for readings successfully processed from Ver1 using Masek's algorithm is 28.3%, with a standard deviation of 4.7%.

This value is far higher than the 11% mean bit-error rate (with a standard deviation of 6.5%) reported by Daugman for his iris algorithm. Independent studies on Daugman's algorithm, such as that of British Telecom Labs [SGM97] or Joh. Enschedé [Ens00] in the Netherlands, also show a much lower bit-error rate for iris readings. These values are collated in Table 7.3.

Study	Daugman	British Telecom (Daugman)	Joh. Enschedé (Daugman)	Masek
Mean BER	11%	8.9%	13.2%	28.3%
Stan Dev	6.5%	4.2%	7.2%	4.7%

Table 7.3: Comparison between mean bit-error rates and standard deviations in iris readings using Daugman's implementation and Masek's implementation. The results from British Telecom Labs and Joh. Enschedé are independent, and those for Masek's implementation are based on our calculations.

The spread of this data can be seen by plotting histograms comparing all four studies on the same graph, as done in Figure 7.7. There is considerable overlap between the studies of Daugman's algorithm; while Masek's produces far more erroneous results. With such a large difference between the bit-error rates for these two algorithms, it is no surprise that the RS code suggested for Daugman's algorithm proved unsuitable for Masek's implementation.

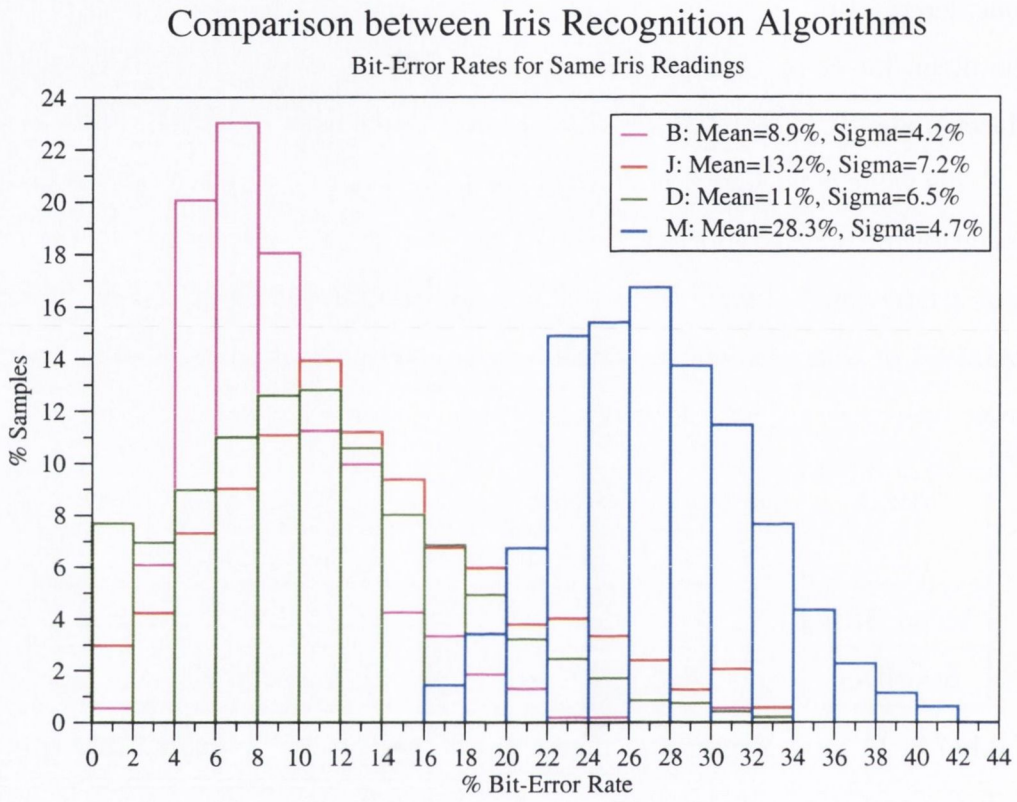


Figure 7.7: Histogram comparing the bit-error rate (%) for iris readings processed using Daugman's and Masek's algorithms. The results of three studies on Daugman's algorithm are shown: "D" refers to his own study [Dau04]; "B" to British Telecom Labs [SGM97]; "J" to Joh. Enschedé [Ens00]. "M" refers to results we obtained for Masek's algorithm.

7.5 Bit-Error Pattern Analysis

Despite the difference in error rates, it may yet prove worthwhile to investigate the pattern of errors within the real iris data we have processed, in order to determine if there is evidence of errors occurring in bursts. Our approach will be to calculate the expected number of error-bursts of length t, assuming that the errors are randomly distributed, and compare these results with the actual number of error-bursts of length t occuring in the data. If these distributions are statistically similar, it indicates that the errors are dispersed in a random manner. We begin by calculating the probability of a burst of length t, using

$$\text{Prob(burst of length } t) = \text{ Prob}(t) = P^{t-1}(1-P) \ , \tag{7.5}$$

where P is the overall probability of a bit-error in an iris reading. Our P-value here will not be equal to that given in Table 7.3, as it was based only on the data considered valid by both the enrolment and verification masks. While our system does exclude bits from the enrolment mask, it attempts to correct those in the verification mask. Therefore, we calculate a new bit-error rate for each comparison, in which the bits in the verification mask *and* any other bits which disagree between the enrolment and verification templates are all treated as errors. The results of this analysis are plotted in Figure 7.8. As seen in the histogram, the mean bit-error rate is 33.91%, with a standard deviation of 7.69%. Thus, the probability of a bit-error, P, is

$$P = 0.3391 \ . \tag{7.6}$$

This allows us to calculate the probability of an error burst of length t. We define the expected number of error bursts of length t, $n(t)$, to be

$$n(t) = N \times \text{ Prob}(t) \ , \tag{7.7}$$

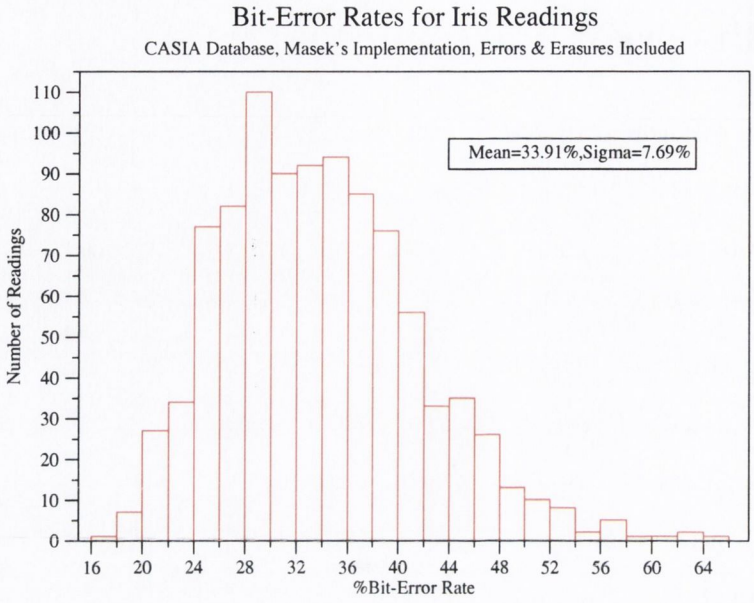


Figure 7.8: This alternative bit-error rate (%) differs from Figure 7.6 as bits in the verification mask *and* bits which differ between templates are "errors".

where N is the total number of error bursts. To determine the value of N, we must count the number of errors bursts in each iris reading, and sum these. For the 968 comparisons possible from the CASIA Ver1 database, we found that $N = 275882$ bursts. This allowed us to calculate values for $n(t)$, as recorded in Table 7.4. By now counting the observed number of error bursts of length t, $o(t)$, we can add the final column to Table 7.4, to contrast this value with our expected value, $n(t)$. Should these final two columns prove to be statistically identical, then the bit-errors may be considered to be randomly distributed and do not occur in bursts. Therefore, we let

$$\begin{aligned} H_0 &: o(t) = n(t) \\ H_1 &: o(t) \neq n(t) \end{aligned} \tag{7.8}$$

where H_0 represents our null hypothesis and H_1 our alternative hypothesis.

Length of Error Burst, t	Probability of Burst of Length t, Prob(t)	Expected Bursts of Length t, $n(t)$	Observed Bursts of Length t, $o(t)$
1	6.61×10^{-1}	182323.79	229819
2	2.24×10^{-1}	61830.34	16975
3	7.60×10^{-2}	20968.13	3993
4	2.58×10^{-2}	7110.79	9321
5	8.74×10^{-3}	2411.44	2265
6	2.96×10^{-3}	817.78	4564
7	1.01×10^{-3}	277.32	1241
8	3.41×10^{-4}	94.05	1936
9	1.16×10^{-4}	31.89	828
10	3.92×10^{-5}	10.82	794
11	1.33×10^{-5}	3.67	545
12	4.51×10^{-6}	1.24	374
13	1.53×10^{-6}	0.42	371
14	5.19×10^{-7}	0.14	252
15	1.76×10^{-7}	0.05	209
20	7.89×10^{-10}	2×10^{-4}	98
30	1.59×10^{-14}	4.38×10^{-9}	38
50	6.42×10^{-24}	1.77×10^{-18}	21

Table 7.4: Comparison between the expected and observed number of error-bursts of length t in the 968 valid iris readings from the Ver1 database, for various values of t. The total number of error-bursts, for $1 \leq t \leq 175$, was found to be $N = 275882$.

The most suitable test to examine whether our observed values equal our expected values is a chi-squared test [Joh05], in which

$$\chi^2 = \sum_t \frac{(o(t) - n(t))^2}{n(t)} \ . \tag{7.9}$$

The chi-squared test requires that all values included are "reasonably" large: the general guideline is that all of the observed and expected values should be five or greater. This causes problems in our expected values, for $t > 10$, and therefore we simply treat all t-values greater than 10 as a single quantity, for which we have $n(11+) = 5.55$ and $o(11+) = 4146$.

We have 10 degrees of freedom, meaning that the corresponding value in the chi-squared table is 29.59, with $p = 0.001$. If the chi-squared value we calculate is larger than 29.59, this indicates a significant difference from our null hypothesis (as there is only a 0.1% chance that this would occur by chance). We conduct a chi-squared test on the data, which gives us a colossal value of

$$\chi^2 = 3281330 \ . \tag{7.10}$$

This provides us with overwhelming evidence that the null hypothesis should be rejected and the observed values do not follow a random error model.

The deviation between the observed and expected values can clearly be seen in Figure 7.9, in which the observed and expected number of error bursts of length t, for $1 \leq t \leq 20$ are plotted. A log-scale is used for the number of bursts, because of the huge range of numbers involved. It is clear that the error bursts are non-random, as these two lines do not agree.

7.6 Bit-Error Burst Model

It would now be instructive to apply the burst-error model we proposed when studying Daugman's data, to see if there is statistical agreement between this

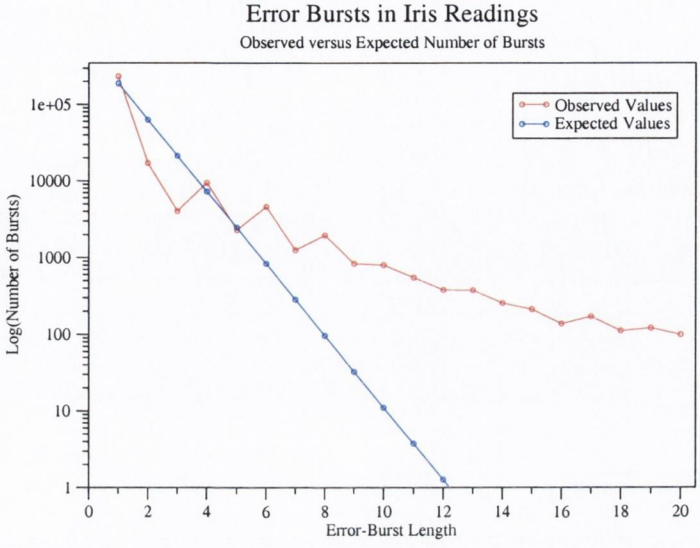


Figure 7.9: Comparison between observed and expected number of error bursts of length t in matches between readings from identical irises. The first twenty values are plotted, with a log scale used for the number of bursts.

model and the real iris readings we have processed using Masek's algorithm. Such an agreement would be strongly indicative of the presence of bursts of errors. Therefore, we apply the model, with $P = 0.283$ being the probability of a bit-error (as again we wish to deal with only errors, omitting masking bits completely from the study). Recall that we vary the value of q (the probability of a correct bit given that the current bit is in error) and then generate data strings containing errors in bursts according to this probability. We fix the string length at 1470 bits, as this approximates the number of uncorrupted bits in the readings processed by Masek, which have a mean length of 1467.24 bits with a standard deviation of 242.25.

As before, we calculate the mean bit-error rate, $\bar{x}_2$, of the simulated data-strings, along with the standard deviation, σ_2. We then perform a difference of means test between the simulated data and the real data from

Prob(Bit Error) $P = 0.283$		Mean BER $\bar{x}_1 = 0.283$; Stan dev $\sigma_1 = 0.047$; Sample size $n_1 = 968$; Sample size $n_2 = 10^3$		
Prob($\bar{e} : e$) q	Prob($e : \bar{e}$) p	Mean BER $\bar{x}_2$	Stan Deviation σ_2	Normal Score z
0.025	0.0099	0.2745	0.0838	2.7908
0.050	0.0197	0.2824	0.0611	0.2381
0.075	0.0296	0.2837	0.0499	-0.3250
0.100	0.0395	0.2843	0.0419	-0.6121
0.125	0.0493	0.2849	0.0384	-0.9582
0.150	0.0592	0.2939	0.0334	-5.8911

Table 7.5: The normal z-scores for comparisons between Masek's data (subscript 1) and the simulations generated by various choices of q (subscript 2) are shown. The simulated bit-string length was 1470 bits, approximately the mean length in Masek's data. Prob($\bar{e} : e$) denotes the probability of a bit not being in error, given that the previous bit is, and vice versa. With a confidence level of 95%, we reject any z-scores outside the range $-1.96 \leq z \leq 1.96$.

the Ver1 database, and calculate a normal z-score. Using an approximate confidence level of 95%, we reject any z-scores that fall outside the range $-1.96 \leq z \leq 1.96$. As shown in Table 7.5, any q-values in the range

$$0.05 \leq q \leq 0.125 \tag{7.11}$$

generate data well within these bounds, and so the simulated data is consistent with readings processed by Masek for these q-values. Not only does this confirm that errors occur in bursts within Masek's readings, it also provides further evidence for the accuracy of our analysis of Daugman's data, given that we applied our model to real data and found it gives a successful match.

7.7 Conclusion

We have analysed real iris data from the CASIA iris database, using Masek's open-source iris recognition algorithm. We found that the bit-error rate was far higher than the results given for Daugman's algorithm. This is due to the inferior performance of Masek's algorithm. As a result, the $(255, 140)$ RS code we proposed to correct iris readings processed using Daugman's algorithm would not be sufficient to match these readings. However, an analysis of the error-pattern within the readings showed overwhelming evidence of the presence of bursts of errors within iris readings, making RS codes the most suitable choice for use in matching irises. Given that Daugman's algorithm produces templates which are extremely accurate, and that his algorithm is in wide commercial use, we stand by our original recommendations of a $(255, 140)$ RS code as the most suitable to match well-processed iris readings. In the next chapter, we consider alternative approaches which would successfully correct poorly-processed data such as Masek's.

Chapter 8

Discussion: Alternative Approaches

"There may always be another reality to make fiction of the truth we think we've arrived at" - Christopher Fry

Having designed our biometric key retrieval scheme so that it would be suitable for use in all types of public-key cryptosystems, the outstanding issue was to select the parameters of the Reed-Solmon code to correct the iris readings used to protect and retrieve the key. We produced results based on simulations of iris readings (which we demonstrated to be statistically similar to actual readings processed using Daugman's algorithm) and showed that a $(255, 140)$ RS code would be suitable to correct these, provided erasures are included. We attempted to use real iris readings to provide further verification of this, but the only open-source algorithm available resulted in substantially higher error rates within the readings, placing them outside the error-correcting capacity of the code. In this chapter, we discuss three outstanding issues which remain: firstly, we look at the possibility of using an alternative RS code size; secondly, we compare our biometric key retrieval

scheme to one produced concurrently in Cambridge [HAD06]; and finally, we propose an alternative error-correction scheme which would be suitable for poorer data such as that produced by Masek's algorithm.

8.1 Alternative Code-Size

We propose two different approaches to adjusting the RS code parameters chosen for our system. Our first suggestion is to encrypt the check digits, allowing us to further decrease the value of k and subsequently increase the error-correcting capacity of the code. Secondly, we consider the possibility of increasing the size of the ground field from GF(2^8) to GF(2^9), which would automatically increase the highest number of errors which can be corrected. We focus on the potential gain from each approach and any subsequent changes that would need to be made to the overall system.

8.1.1 Encrypting the Check Digits

As explained in Chapter 6, the check digits in the $(255, 140)$ RS system we proposed are effectively unhidden from a systems operator, meaning that we could not safely reduce k below the value of 140 without jeopardising the security of the codeword. If we were to encrypt these check digits, we could decrease the value of k while still maintaining the overall security. We must bear in mind, however, that any reduction of k, while n is kept constant, will reduce the code-rate, $\frac{k}{n}$, and adversely affect the efficiency of the code.

8.1.1.1 Expected Improvements

Given that decrementing k by 20 only allows our code to correct an extra 10 errors, it is important to firstly ascertain whether encrypting the check

digits gives us a worthwhile increase in error-correcting capacity. To do so, we return to the simulated data-strings generated in Chapter 6 (statistically identical to Daugman's processed iris readings) and study whether they could be corrected by this technique. We generate a histogram for our longest possible string length of 1600 bits, setting q (the probability that the next bit is not in error, given that the current bit is) to its highest acceptable value of 0.2 and letting the probability of a bit error, P_e, be equal to 0.11, as per Daugman's results. This generates the diagram shown in Figure 8.1. Recall that the mean number of byte errors for these parameters was 51 with

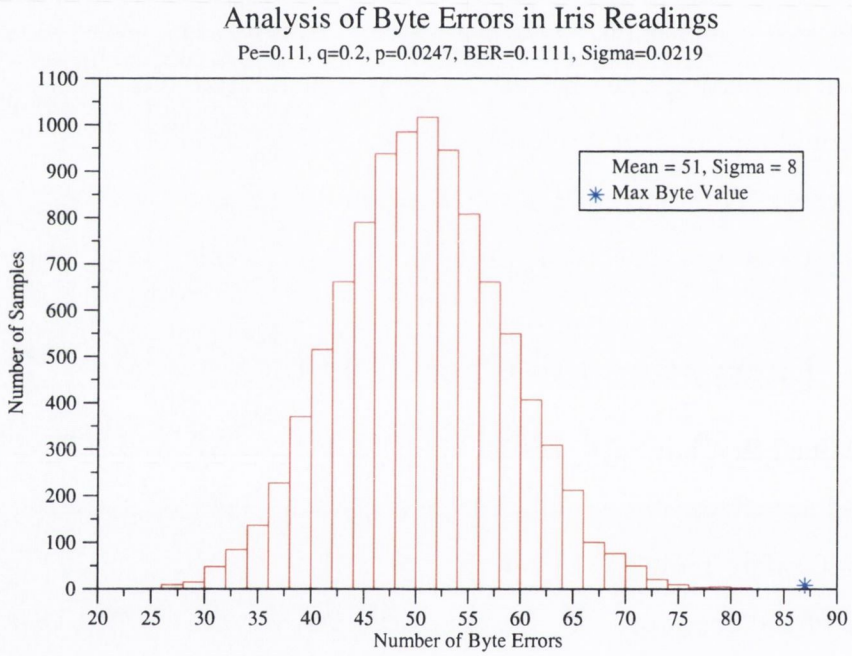


Figure 8.1: Histogram showing the byte errors in 10^4 simulated 1600-bit iris-strings. The mean number of byte errors was 51, with a standard deviation of 8. The maximum value was 87, shown by a star.

a standard deviation of 8. The maximum number of byte errors from this sample was 87, marked with a star in the diagram. We now produce Table

8.1, which compares the number of correctable errors, t, for various k-values with the percentage of samples falling below t. As we can see, encrypting

Prob(Bit Error) $P_e = 0.11$; String-length $N = 1600$ bits Prob(No Error, Given Current Bit in Error) $q = 0.2$			
Information Bits k	Check Digits $(n-k)$	Correctable Errors t	% of 10^4 Samples $\leq t$ Byte Errors
140	115	57	79.15
120	135	67	99.77
100	155	77	99.87
80	175	87	100.00

Table 8.1: Percentage of simulated iris strings within error-correcting capacity of $(255, k)$ RS code over GF(2^8) for decreasing values of k, with $P_e = 0.11$, $N = 1600$ bits and $q = 0.2$ (largest acceptable q-value for these parameters).

the check digits would allow us to correct all possible errors in this sample, although k must be decremented as low as 80, reducing our code-rate to 0.3.

If we briefly look at the results of the British Telecom Labs tests [SGM97] on Daugman's algorithm, we find that, for 1600-bit strings, a bit-error rate of 8.9% and the largest acceptable value of $q = 0.75$, the number of simulated samples corrected by this code (if erasures are not included) has increased from 0% to 11.14%. While this can be expected to further increase when erasures are also taken into account, a substantial number of readings will remain outside the error-correcting capacity of a $(255, 80)$ RS code over GF(2^8).

We now consider the results obtained in the previous chapter, when Masek's algorithm was used to process a database of iris readings. In this case, we can identify the exact number of errors and erasures within each

reading. If r is the number of errors and s the number of erasures, provided

$$2r + s \leq 174 \ , \tag{8.1}$$

we can correct these readings. Only 0.93% of the 968 comparisons produced from the CASIA Ver1 database fall below this upper bound, meaning that data processed by Masek's algorithm is too error-prone to be thus corrected.

8.1.1.2 Choice of Encryption Method

To encrypt the check digits, we need to create a unique secret key for the user that can be regenerated any time they use the system. Symmetric-key encryption is ideally suited to this purpose; the same key can be used to encrypt and decrypt the check digits. To generate this key securely, a combination of an identity claim and a password could be used. However, the introduction of passwords obliterates one of the major advantages of biometric identification: namely, that the implementation does not rely on the user for its security. Using passwords means that we are now depending on the user to remember and reproduce the correct password at each verification, and to refrain from writing it down to avoid forgetting it, adding a new range of vulnerabilities to the system.

Overall, the limited gain from this approach is not worth the decrease in overall security, and so we move on to consider the impact of increasing the size of the ground field of our RS code.

8.1.2 Increasing the Ground Field

Our original choice of GF(2^8) resulted from the natural relationship between the octet-based calculations in such a field and the bytes in the IrisCode. However, if we increased this field to GF(2^9), we would gain considerably

from the larger error-correcting capacity, as we can now choose a maximum code-length, n, of 511 rather than 255. With $n = 511$, if we do not wish to encrypt the check digits, then we must not allow their number to exceed $(511/2) = 256$. Thus, provided $k > 256$, the check digits remain protected. If we let $k = 260$, the distance of the code is

$$d = 511 - 260 + 1 = 252 \ , \tag{8.2}$$

meaning that we can correct up to 125 errors. This is a considerable increase over the upper bound of 57 errors with a $(255, 140)$ code over GF(2^8) and would easily correct the maximum of 87 errors shown in Figure 8.1.

If we again look at the British Telecom Labs tests, for 1600-bit strings, a bit-error rate of 8.9% and $q = 0.75$, we find that *all* errors could be corrected by a $(511, 260)$ RS code over GF(2^9). Although these results are unlikely to be more accurate than Daugman's, given that the sample sizes are several times smaller here, it is nonetheless reassuring to confirm that all such cases could be corrected by this code if necessary.

Finally, of the 968 comparisons we produced from the CASIA Ver1 database using Masek's algorithm, 99.79% fall within the error-correcting capacity of this code, meaning that the overwhelming majority of iris readings processed using Masek's algorithm could be corrected using a $(511, 260)$ RS code over GF(2^9). Thus, it is clearly worthwhile to further develop this approach.

8.1.2.1 Operational Differences

We must now consider the implications of such a change. The characteristic of the field is still 2, meaning that we can use the exclusive-OR operation in place of addition and subtraction, as before. The look-up tables (LUTs) used to perform operations within the field will have to be increased; at the

moment, each entry is simply a byte long, whereas increasing the size of the field means that we will now have to deal with 9-bit values instead, which are not as easily integrated into a data-type. In addition, the number of entries within the LUTs will double in value from 256 to 512.

8.1.2.2 Codeword Generation

However, the area that requires the most adjustment is that of the generation of the codeword. In our $(255, 140)$ code over GF(2^8), it is easily generated, as there are sufficient valid bits within the IrisCode to fill the information bits of the codeword; however, this is no longer possible when we require $260 \times 9 = 2340$ information bits.

One approach is to duplicate the data from the individual's iris reading, to generate the extra symbols needed for the codeword, as shown in Figure 8.2. For example, if we have 1304 bits in an iris reading, we have a maximum

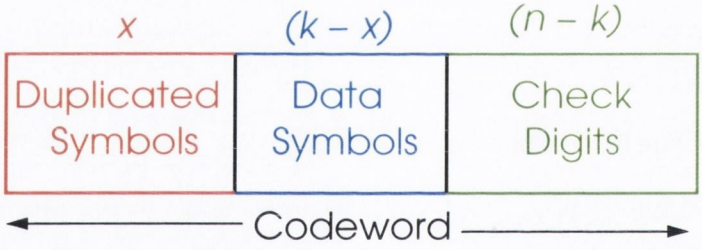


Figure 8.2: Diagram of codeword in RS code. The MSBs are x duplicated data symbols; followed by $(k - x)$ data symbols; and $(n - k)$ check digits.

of about 144 9-bit symbols. We want a total of $k = 260$ information symbols for our RS code over GF(2^9). Therefore, suppose we use 130 of the iris reading symbols and duplicate them so that these symbols are followed by an identical string of $x = 130$ symbols, meaning that $k = 2x$.

The challenge here is that we have automatically doubled the number

of errors in the information bits. However, once we have discovered the erroneous values within the first x symbols, we know that the next x symbols will be similarly affected. We now describe a means of correcting such errors.

Let Y_i be the value of the i^{th} error at location X_i and $\tilde{Y}_i$ the value of the i^{th} erasure at location $\tilde{X}_i$. There are twice the usual number of discrepancies; therefore, assume $2r$ errors and $2s$ erasures, where we have

$$2r + s \leq d - 1 \tag{8.3}$$

as before, with d being the distance of the code. This means that the error-correcting capacity of the code is r errors and s erasures.

With these parameters, the syndromes, S_k, can be expressed as

$$S_k = \sum_{i=1}^{s} (\tilde{Y}_i \tilde{X}_i^k + \tilde{Y}_{i+s} \tilde{X}_{i+s}^k) + \sum_{j=1}^{r} (Y_j X_j^k + Y_{j+r} X_{j+r}^k) \ . \tag{8.4}$$

However, due to the duplication of the discrepancy values, we know that

$$Y_i = Y_{i+r} \text{ and } \tilde{Y}_i = \tilde{Y}_{i+s} \ . \tag{8.5}$$

Thus, equation (8.4) can be re-written as

$$S_k = \sum_{i=1}^{s} \tilde{Y}_i (\tilde{X}_i^k + \tilde{X}_{i+s}^k) + \sum_{j=1}^{r} Y_j (X_j^k + X_{j+r}^k) \ , \tag{8.6}$$

The duplication of discrepancies occurs exactly x positions apart, giving

$$X_{i+r} = X_i \alpha^x \text{ and } \tilde{X}_{i+s} = \tilde{X}_i \alpha^x \ . \tag{8.7}$$

α^x is a fixed, known quantity for each implementation of the algorithm. Thus, we can re-arrange equation (8.6) to give:

$$\begin{aligned} S_k &= \sum_{i=1}^{s} \tilde{Y}_i (\tilde{X}_i^k + (\tilde{X}_i \alpha^x)^k) + \sum_{j=1}^{r} Y_j (X_j^k + (X_j \alpha^x)^k) \\ &= \sum_{i=1}^{s} \tilde{Y}_i \tilde{X}_i^k (1 + \alpha^{kx}) + \sum_{j=1}^{r} Y_j X_j^k (1 + \alpha^{kx}) \\ &= (1 + \alpha^{kx}) \left\{ \sum_{i=1}^{s} \tilde{Y}_i \tilde{X}_i^k + \sum_{j=1}^{r} Y_j X_j^k \right\} \ . \end{aligned} \tag{8.8}$$

We can now correct the errors and erasures present in the codeword using the following technique, adapted from that explained in our background chapter:

1. Evaluate the syndromes, S_k, for $k = 1, \ldots, 2t$ (where $2t = d - 1$), using $S = \vec{r}H^T$, where $\vec{r}$ is the received codeword and H^T the transpose of the parity-check matrix.

2. Scale down each syndrome, S_k, by a factor of $(1 + \alpha^{kx})$, to produce parameters which we term $\tilde{S}_k$. Thus, we can continue with only minor modifications to the correction procedure, as the $\tilde{S}_k$ are identical to the syndromes that would be calculated were no duplications present.

3. The erasure locators, $\tilde{X}_k$, are the roots to the equation

$$f(x) = f_0 x^s + f_1 x^{s-1} + \cdots + f_s \ , \tag{8.9}$$

with $f_0 = 1$, and so, calculate f_i for $i = 0, \ldots, s$, using only the first half of the error locator values to avoid duplication.

4. Find $T_i = \sum_{j=0}^{s} \tilde{S}_{s+i-j-1} f_j$ for $i = 1, \ldots, 2t - s$, using our reduced syndrome values.

5. Assume that the first half of the error locators, X_k, are roots of

$$p(x) = p_0 x^r + p_1 x^{r-1} + \cdots + p_r \ , \tag{8.10}$$

with $p_0 = 1$, and calculate the values of p_i by using the Berlekamp-Massey algorithm (Appendix C) to solve

$$T_{i+r} + p_1 T_{i+r-1} + \cdots + p_r T_i = 0 \ , \tag{8.11}$$

for $i = 1, \ldots, t$ (or less if fewer than 2t discrepancies). This provides the values of p_i as well as *half* the total number of errors.

6. Solve $p(x)$ to find its roots, the first half of the error locators, X_k, for $k = 1, \ldots, r$.

7. Letting $E_k = Y_k f(X_k)$, determine E_k from $T_i = \sum_{k=1}^{r} E_k X_k^i$, using Gaussian Elimination with partial pivoting.

8. Find the first half of the error values, Y_k, for $k = 1, \ldots, r$ from E_k.

9. We now know locations and values for half the errors/erasures; compute the remainder using equations (8.5) and (8.7).

10. Restore the original codeword by adding the values $\tilde{Y}_i$ and Y_i to the corrupted codeword in the appropriate positions.

Thus, we can correct our codeword, even though the number of errors/erasures is twice that of the error-correcting capacity. In this way, we can create codewords for RS codes over GF(2^9), and have shown that a $(511, 260)$ RS code over GF(2^9) could be used for iris matching in our biometric key retrieval scheme.

8.2 Comparison with HAD Report

While the work described in this thesis was being completed, concurrent and independent research was being undertaken in the Computer Laboratory in Cambridge, where John Daugman is based. This research resulted in a preprint [HAD05] by Hao, Anderson and Daugman (HAD) (to appear later this year as [HAD06]), which describes an iris matching scheme similar to that detailed in this thesis and uses a combination of Reed-Solomon and Hadamard codes for iris matching. We shall now review our work in comparison with this latest development, to assess the relative contributions of each.

Key Protection	
HAD Approach	**Our Approach**
1. Generate key for desired cryptosystem in usual fashion.	1. Generate key for desired cryptosystem in usual fashion.
2. Create codeword from key, by including check digits (explicitly for RS codes, implicitly for Hadamard codes).	2. Create codeword from iris reading, storing check digits separately (using RS codes only).
3. Produce token by XORing codeword with iris reading.	3. Produce token by XORing codeword with key.
Key Retrieval	
HAD Approach	**Our Approach**
1. Obtain new iris reading.	1. Obtain new iris reading.
2. XOR iris reading with token to produce corrupted codeword.	2. Attach check digits to produce corrupted codeword and correct this.
3. Correct corrupted codeword using in-built check digits to recover the key.	3. XOR corrected codeword with token to recover the key.

Table 8.2: Summary of the main steps in HAD algorithm and in our system. HAD uses a concatenation of Hadamard and RS codes, whereas we rely solely on RS codes. HAD associates the check digits with the key, rather than the iris reading, meaning they can become corrupted, increasing the overall number of errors.

Table 8.2 outlines the main steps in each algorithm. As can be seen, there are many similarities in the overall approches. Because the HAD system was produced in collaboration with Daugman, they had access to his original iris processing algorithm and processed iris images, allowing them to provide experimental results which are not possible for us at present. However, despite this, our system has numerous advantages over the HAD system, which are summarised below in four main points.

8.2.1 Advantages of our Approach

1. We associate the check digits with the biometric and store them separately; the HAD approach associates them with the key. Therefore, the check digits in their system can become corrupted along with the data, when XORed with the new biometric reading. This results in a far greater number of errors in their system.

2. Because there are more errors in the HAD system, more error-checking information is required, resulting in less space for actual data. Thus, it is only suitable for key sizes of up to 140 bits, which is too small even for elliptic-curve implementations, not to mention RSA. Our approach, in contrast, can deal directly with key sizes up to ten times larger.

3. The inclusion of partial hiding in our biometric key retrieval scheme allows its application for systems such as RSA, which would not be possible with the HAD approach.

4. Our system makes full use of the information stored in the IrisMask and uses this to exploit the error-correcting capacity of RS codes to their full extent, by including erasures in the scheme; the HAD approach ignores the IrisMask entirely.

8.3 RS and Repetition Code Concatenation

The size of the key which can be retrieved using the HAD approach (140 bits) is too small for our purposes. However, the idea of concatenating RS codes with a second code which removes the random noise in the data is an interesting one and could prove effective with poorer data, such as that produced by Masek's non-commercial algorithm. Therefore, we propose the use of a hybrid system, which employs repetition codes in conjuction with RS codes, and we attempt to correct readings from the CASIA Ver1 database, processed using Masek's algorithm. This system will be an improvement on the HAD scheme, in that it will protect keys far larger than the 140-bit threshold imposed by their approach.

Unlike the HAD system, we once again store the check digits separately, meaning that our check digits do not become corrupted. This allows us to protect larger key sizes than 140 bits. The repetition codes are applied first, to deal with the random noise in the data, followed by the RS codes, which deal with the bursts of errors that occur naturally within iris readings. The main steps can be outlined as follows:

Key Protection:

1. Expand the private key, K_P, by a factor of 4, to create K'_P.
2. Create check digits for K'_P as usual, and store these separately.
3. XOR K'_P with the enrolment iris reading, to form a secure entity.

Key Retrieval:

1. XOR the verification iris reading with the secure entity to retrieve the corrupted key.

2. Correct errors, due to noise, within each quartet using a repetition code.

3. Apply the RS check digits to the quartets and correct.

4. Compress the corrected data to form the original key.

These steps are summarised in Figure 8.3, which shows the key protection and key retrieval stages of the process. The repetition code we propose involves an expansion factor of 4. Therefore, in order to remove the random noise from the corrupted key using this repetition code (as in step 2 of the "Key Retrieval" stage), we proceed as follows:

- If a quartet contains no zeros or only one zero, all of the bits in the quartet are set to 1.

- If a quartet contains three or four zeros, all of the bits in the quartet are set to 0.

- If a quartet contains two zeros, all of the bits in the quartet are marked as an erasure (for example, by giving them a "dummy" value of 7).

Including erasures in this way again increases the error-correcting capacity of our RS code, as explained previously. Given that RS codes are not probabilistic, but instead correct a fixed number of errors and erasures, depending on the code capacity, it is sufficient for us to complete the first two steps of the key retrieval process, and determine the measure of disparity of the remaining data. Provided that this is within the capacity of the RS code we suggest, we know that the remaining errors and erasures will be corrected.

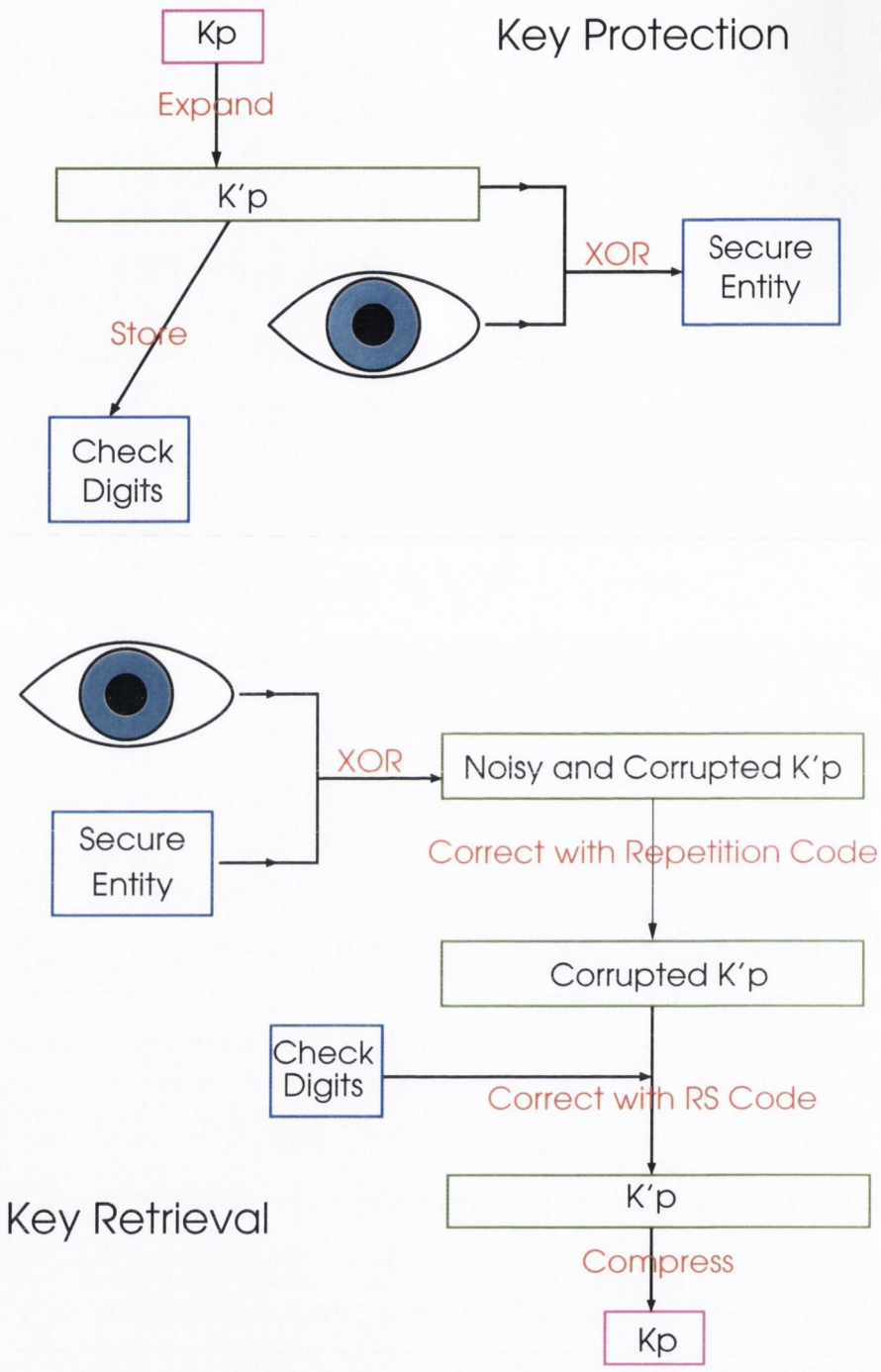


Figure 8.3: Key protection and key retrieval stages in the RS-Repetition code concatenation scheme. K_p is the private key; K'_p the extended private key.

8.3.1 Implementation of RS-Repetition Scheme

We begin by implementing the steps of the key protection scheme outlined above, for the 630 iris readings from the CASIA Ver1 database which were correctly processed using Masek's algorithm. We choose to work with a key of 256 bits, which corresponds to the amount of private-key data that we must keep secret in a partially hidden RSA key. Therefore, if we successfully correct these 630 iris readings, we know that even such poor data could be used in a biometric key retrieval scheme for all types of public-key cryptosystems. To expand a 256-bit private key, K_P, by a factor of 4, we simply repeat each bit four times, to create the 1024-bit K'_P. For each enrolment iris reading in the database, we now calculate

$$K'_P \otimes \text{ Enrolment Reading } \rightarrow \text{ Secure Entity } , \tag{8.12}$$

completing the key protection stage.

Our next step is to XOR the secure entity with a verification reading from the same individual, to retrieve the corrupted key. We then apply the repetition code as outlined above, to correct the random noise in the data. At this stage, we compare this semi-corrected key with K'_P, to ascertain the number of errors and erasures that would need to be corrected by our RS code.

We found that the mean number of byte errors is 13, with a standard deviation of 6, while the mean number of byte erasures is 45, with a standard deviation of 10. If we recall that the measure of disparity, m, is given by $2r + s$, where r is the number of errors and s the number of erasures, we have a mean m-value of 71, with a standard deviation of 18. Figure 8.4 shows the distribution of m for these *intra-class comparisons*, in which each iris is compared against other readings of itself.

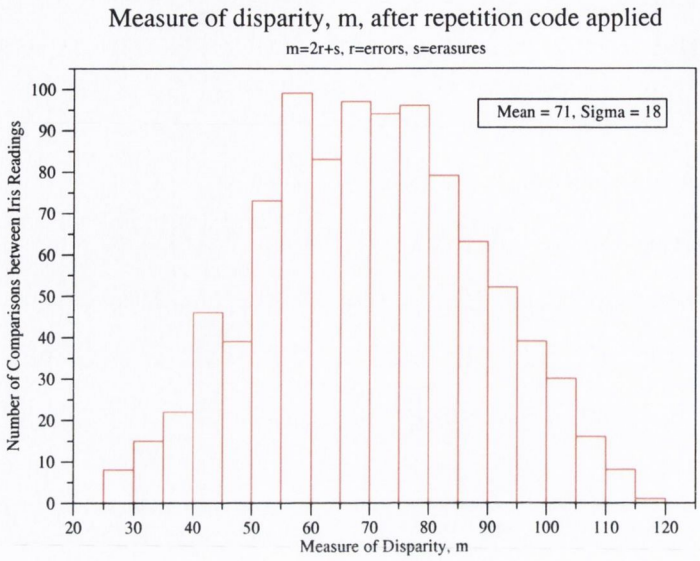


Figure 8.4: The measure of disparity, m, is shown for intra-class comparisons between iris readings, following the application of a 4-bit repetition code.

The expanded 1024-bit key acts as the information bits of the RS code-word. If we again choose to work with an RS code over GF(2^8), we need 128 information symbols, as this gives us the necessary $(8 \times 128) = 1024$ bits.

In order to choose the optimum code-length, n, to correct these errors and erasures, it is advisable to look also at the *inter-class comparisons*, in which an iris reading is only compared against readings from other individuals. By observing the overlap between the measure of disparity for the inter-class comparisons and that of the intra-class comparisons, we can determine how many individuals would be falsely accepted or falsely rejected by the system, for any given code-length.

The mean measure of disparity for the inter-class comparisons is 118, with a standard deviation of 7. While this is substantially higher than the mean of the intra-class comparisons, it is clear from Figure 8.5 that there is also a considerable overlap between these two distributions.

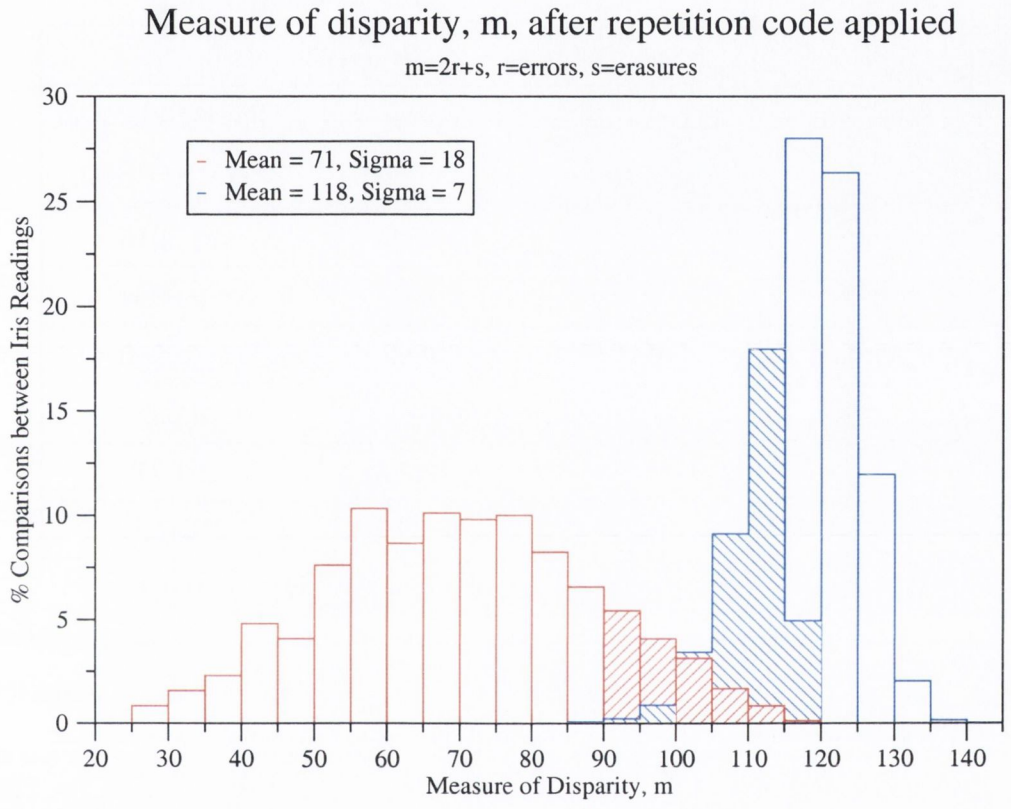


Figure 8.5: Histogram comparing the measures of disparity for the intra-class and inter-class distributions in iris readings. The left distribution (red) represents the measure of disparity, m, for the intra-class comparisons, while the right distribution (blue) represents that of the inter-class comparisons. The shaded areas show the overlap.

Recall, from Chapter 2, that the False Acceptance Rate (FAR) is the probability that an individual will be incorrectly identified, or an impostor will not be rejected. The False Rejection Rate (FRR) is the probability that a genuine enrolee in the system will not be identified or verified. Table 8.3

Information Symbols $k = 128$			
Code Length n	Check Digits $(n-k)$	False Acceptance Rate (FAR)	False Rejection Rate (FRR)
243	115	0.3648	0.0016
238	110	0.1361	0.0094
233	105	0.0451	0.0260
228	100	0.0109	0.0573
223	95	0.0025	0.0979
218	90	0.0005	0.1521

Table 8.3: Table showing the FAR and FRR in iris readings for various values of n, if an $(n, 128)$ RS code is concatenated with a 4-bit repetition code.

presents the FAR and FRR for a variety of code-lengths, n, in an $(n, 128)$ RS code. The optimum choice of n depends on the desired security level. For example, with a $(218, 128)$ RS code, there is only a 0.05% chance that an impostor will be accepted into the system, but a 15.21% chance that a genuine user will be required to re-attempt the verification procedure before being accepted. If a $(243, 128)$ RS code is used, the chance that a genuine user will have to re-attempt falls to 0.2%, but there is then a 36.5% chance of an impostor being accepted.

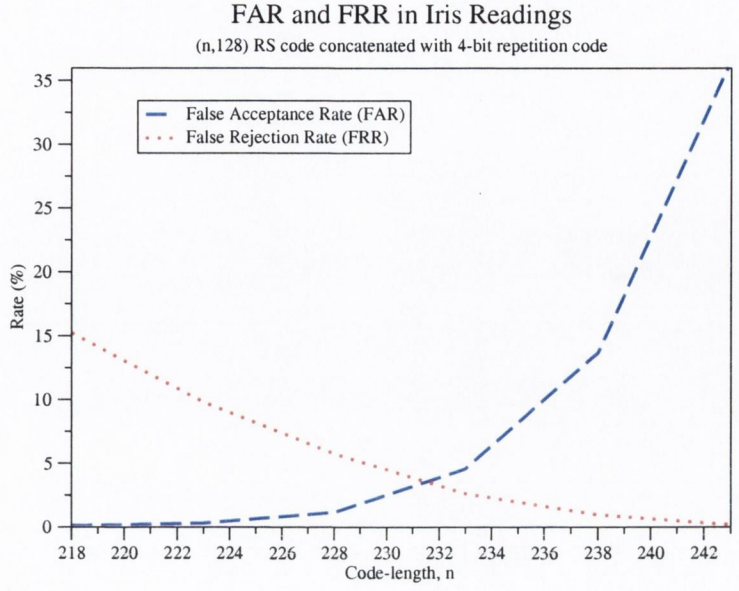


Figure 8.6: Cross-over point of FAR and FRR in poorly-processed iris readings, for an $(n, 128)$ RS code, concatenated with a 4-bit repetition code.

As can be seen in Figure 8.6, the cross-over point between the FAR and FRR occurs just below $n = 232$, where both rates are equal to 3.5%. Ideally, in a biometric system, this cross-over point would occur at 0%, and accurate recognition would be assured, but due to the fact that these iris images were poorly processed, this is not possible here, and so an appropriate n-value should be chosen based on the security requirements of the application.

Thus, we have shown that it is even possible to correct poorly-processed iris readings using a concatenation of RS and repetition codes. We suggest using a 4-bit repetition code with an $(n, 128)$ RS code over GF(2^8), where $218 \leq n \leq 228$. Such iris readings can be used to protect and retrieve private keys in all types of PK cryptosystems, as described, creating a biometric key retrieval scheme. This is a huge improvement on the HAD approach, as we have greatly increased the number of bits that can be securely protected.

Chapter 9

Conclusion

"There are two possible outcomes:
if the result confirms the hypothesis, then you've made a discovery;
if the result is contrary to the hypothesis, then you've made a discovery."
- Enrico Fermi

With the invention of public-key cryptography over thirty years ago, a solution had been found to the problem of securely transporting the secret key between the sender and receiver of an encrypted message. Since then, work has been ongoing to discover the optimum way of securely storing the private key of PKC systems. It is vital that the private key be protected from attack at all times, while still allowing a legitimate user to easily access it. Passwords or PINs are usually used to protect the key, which may be stored on a smart-card, a key-card or even the hard drive of a computer.

By protecting private keys using biometrics, the overall security of the scheme can be greatly enhanced. Biometrics cannot be guessed, nor can they be lost or forgotten, making it substantially more difficult to forge someone's identity. However, for maximum security, the system must be implemented in such a way as to make it extremely difficult for a stolen biometric reading

to be used. Ideally, biometrics should be used in combination with a claim of identity, as in our scheme, to protect against this.

Biometric key-retrieval schemes already exist for symmetric systems; however, none of these are capable of protecting private keys in PKC. This is due to the fact that private keys are generally much larger than symmetric keys, and the maximum number of bits protected in current biometric key-retrieval schemes is 140; too low for PKC.

In this thesis, we set out to address the challenge of designing a biometric key retrieval scheme capable of protecting keys from *all* types of PK cryptosystems. Iris readings are our biometric of choice, due to their stability and reliability. In order to process iris images into data-strings, we chose to work with the most accurate iris processing algorithm currently available, created by Daugman [Dau93]. However, to further increase the security of our scheme, we introduced an alternative means of matching iris readings, using error-correcting codes in place of the statistical independence test used by Daugman. The advantages to this approach are twofold: firstly, only a hashed version of the iris reading is stored, which is more secure than storing the entire reading; and secondly, there is less data to be stored overall. The enrolment iris reading is reduced to a fixed-length codeword, by deleting any bits judged by Daugman's algorithm to be corrupt. Verification iris readings are similarly shortened; any bits considered corrupt in the verification reading which were valid in the enrolment reading are treated as erasures, while any other differences which occur between the readings are errors.

We then considered the key lengths that need to be protected by an iris reading, and found that these fell into two distinct categories: key-lengths of approximately 160-224 bits, suitable for systems whose security relies on the discrete-log problem, and for elliptic-curve-based systems; and key-lengths of

2048 bits, needed for systems such as RSA which depend on the intractability of integer factorisation. The shorter key-lengths could be directly XORed with the iris reading; however, the RSA key sizes are simply too large for this. Therefore, we developed the novel concept of "partial hiding", in which specific segments of the RSA private key are made publicly available, reducing the amount of "secret" data to be XORed with the iris reading. We drew on research investigating attacks on RSA to show that, with carefully chosen parameters, this can be done without adversely affecting security.

Having outlined the design of our system, all that remained was to select the most suitable code-type and code-parameters to correct the iris readings. Although we considered several more recent possibilities, we chose to work with Reed-Solomon codes, as the errors in iris readings occur in bursts, and are still best dealt with by this type of code. To decide the optimum code-parameters, we generated a database of bit-strings with similar error-patterns and statistical properties to irises processed with Daugman's algorithm, and studied the number of errors which would need to be corrected. This led us to propose a (255,140) RS code over GF(2^8) to correct processed iris readings. We subsequently used an open-source algorithm to process a database of real iris readings, and found that the error-rates within these were far higher, and required an alternative approach to be corrected. We looked at ways of increasing the ground field, and also developed a concatenated system using repetition codes and RS codes to successfully correct these iris images.

Thus, we developed a scheme in which iris readings, corrected by error-correcting codes, could be used to protect and retrieve private keys in all types of public-key systems. We will now specify the main conclusions at which we arrived during the course of this work.

9.1 Summary of Main Conclusions

1. Our problem statement has been successfully resolved, as we have created a biometric key retrieval scheme capable of protecting private keys from all types of public-key cryptosystems: those based on the discrete-log problem; elliptic-curve-based systems; and those which rely on the intractability of integer factorisation for their security.

2. Our scheme can also be applied to symmetric-key algorithms; most significantly, to high security-level symmetric algorithms, such as AES-192 or AES-256 [DR02], which could not previously have been used for biometric key retrieval.

3. Key sizes for PKC vary from about 160 bits for SDL and elliptic-curve systems to 2048 bits for RSA. The shorter keys are protected by XORing with the biometric, but for larger keys, partial hiding is recommended. Thus, for a 2048-bit RSA key, the 768 LSBs and 1024 MSBs can be made publicly available, once the 256 "middle" bits are securely protected. Provided that the 384 LSBs of p and q are equal (where the modulus $N = pq$), this system is secure against all known attacks.

4. The iris is highly suitable for biometric key retrieval, being stable, repeatable and not overly intrusive. Daugman's [Dau93] iris processing algorithm produces the highest quality bit-strings and should be used to process iris images into templates.

5. For matching purposes in iris recognition, the use of error-correcting codes is both effective and secure. It is unnecessary to store the iris reading explicitly; a hashed version of the template (along with check digits) is sufficient.

6. The errors in iris readings occur in bursts. Thus, for iris templates produced by Daugman's algorithm, a (255,140) Reed-Solomon code over GF(2^8) is the most suitable code to correct these errors.

7. Erasures should be included in the RS correction procedure to increase the error-correcting capacity of the code. Bits contained in the verification IrisMask which were not included in the enrolment IrisMask should be treated as erasures, and all other differences as errors.

8. Should a larger ground field be required, to correct more errors, the number of information bits required exceeds the valid iris bits available. By duplicating iris bits and adapting the Berlekamp-Massey algorithm [Mas69] for error-correction, this problem can be successfully overcome.

9. If an open-source iris-processing algorithm such as Masek's [Mas03] is used, error-rates within processed iris templates are far higher than in those produced by Daugman. A concatenation of a 4-bit repetition code and an $(n, 128)$ RS code over GF(2^8) (where $218 \leq n \leq 228$) effectively corrects all errors produced when the CASIA [Chi06] database is processed using this algorithm. The repetition code corrects random noise, while the RS code again corrects bursts of errors within the data.

9.2 Summary of Contributions to Knowledge

Having stated the main conclusions of our work, we now summarise the contributions to knowledge contained therein, as outlined in Chapter 1.

1. The most important contribution made by this thesis is in the development of a new means of protecting private keys in PKC, through our biometric key-retrieval scheme, which uses iris readings, corrected by

RS codes, to protect and retrieve keys up to 2048 bits long. Such a scheme provides greater security than PIN/password-based approaches.

2. We achieved a huge increase in the key-length which can be protected, successfully retrieving keys fourteen times larger than previously. This showed for the first time that high security symmetric systems, such as AES-192 and AES-256 could be used in biometric key retrieval schemes. Given that AES is an international standard, it is a significant advance to produce a scheme that can protect its keys in a more secure manner.

3. Drawing on work done on partial key exposure, we suggest the first useful application of this, in the form of partial hiding, to protect 2048-bit RSA keys. This enables far larger keys to be protected by biometrics.

4. We have successfully implemented an original method of matching iris templates, by using RS codes to correct and match iris readings. The inclusion of erasures is a vital new addition which increases in the error-correcting capacity of the code to the necessary level.

5. We have shown that poorly-processed iris images, with high error-rates, from an open-source algorithm such as Masek's, can be corrected using a concatenated scheme involving repetition codes and RS codes. This is an improvement on a recent concatenation [HAD06], as check digits are not corrupted in our scheme, allowing far more bits to be retrieved.

9.3 Future Work

Arising from the multi-disciplinary research undertaken to create our biometric key retrieval scheme, several possible avenues for future work emerged, as outlined below.

1. **Alternative Biometric Identifier:** With the constant improvements that are occurring in biometric acquisition schemes, it is likely that other biometrics apart from iris scans may soon prove suitable for a key retrieval system such as we have proposed. There is much work being done in the area of voice authentication, for example, as a result of the immense advantage posed by its ability for remote identification. As of yet, such schemes are not nearly reliable enough to be used for key retrieval, but this may change in the near future, at which stage a key retrieval scheme using voice authentication could be explored.

2. **Multiple Biometric Identifiers:** The possibility of using multiple biometric identifiers to protect the private key could also be investigated. The advantage of such an approach is that it is more difficult for an attacker to impersonate an individual when more than one biometric is in use. As against that, the false rejection rate of valid users is likely to be higher when using multiple biometric identifiers, so it is probable that work in this area may be centred around making the system efficient.

3. **Further Applications of Partial Hiding:** Our exploration of partially hiding the private key in RSA led us to question whether such a technique might be applicable to other areas of cryptography. Initial investigations have suggested that it could be of particular use in multiple signatory schemes. It may also have uses in areas such as watermarks or protection against copying. It is proposed to study and further develop such alternatives at a future date.

"Great is the art of beginning, but greater is the art of ending."

- Lazurus Long

Appendix A

Publications from Thesis

The work contained in this thesis has already resulted in a number of peer-reviewed publications, which are listed below in order of most recently published.

1. E. Ní Fhloinn and M. Purser. Biometric Key Retrieval using Error-Correcting Codes. In *ENMA 2006 - International Conference on Engineering and Mathematics, Bilbao, Spain.* July, 2006. To appear.

2. E. Ní Fhloinn and M. Purser. Iris Matching using Error-Correcting Codes. In *ISSC 2006 - IET Irish Signals and Systems Conference, Dublin, Ireland (Poster Session)*, pages 179-184. June, 2006.

3. E. Ní Fhloinn and M. Purser. Applications of Partial Hiding in RSA. In C. Wolf, S. Lucks, and P.W. Yau, editors, *WEWoRC 2005 - Western European Workshop on Research in Cryptology*, volume 74 of *Lecture Notes in Informatics*, pages 86-97, 2005.

4. E. Ní Fhloinn. Biometric Identification. *Trinity College Dublin Journal of Postgraduate Research*, 4(1-2):173-183, 2004-2005.

In addition, it should be noted that:

- 50% of the papers submitted to (1) were rejected.
- 20% of those submitted to (2) were rejected.
- 77% of the submissions to the conference in (3) were *not* selected for publication in the post-proceedings.
- A Best Paper Award was received by the author for publication (4).

Appendix B

Gathering Biometric Data

In this appendix, we describe a trial in which biometric data, in the form of fingerprints, was collected from third-level student volunteers in Trinity College Dublin, over a one-month period in 2002. Although our focus moved to iris recognition soon after this trial was completed, it was, nonetheless, an instructive exercise, highlighting some of the issues which may be encountered when implementing biometric identification.

The study was undertaken in conjunction with Daon [Dao05], a biometric authentication company based in Dublin. The data was processed using the DaonEngine [Dao04], a product that was being developed by the company at that time in order to match fingerprint templates and identify individuals. The fingerprint sensor used was a touch-chip sensor by ST Microelectronics [Mic06], which has since been replaced by a more advanced model, and the images and templates obtained were stored in a database. Here we lay out the aims of the study, explain the structure and implementation, and report practical observations based on everyday use of the software.

B.1 Aims of Fingerprint Study

Prior to commencing this study, seven specific aims were identified:

1. Enrol at least four hundred students in the system.
2. Verify at least two hundred of these students once.
3. Verify some percentage of the students a second time.
4. Test the DaonEngine on an extensive population.
5. Gather a large number of fingerprint images and templates.
6. Introduce users to the idea of biometrics.
7. Obtain feedback from users on their attitudes to biometric identification, specifically the fingerprint sensor used in the trial.

B.2 Structure of Biometric Trial

The trial was initially scheduled to take place over a three-week period, from 15^{th} April to 3^{rd} May, 2002. It was hoped that the system would be operated for six hours a day, four days a week, from Monday to Thursday. By setting up in a different location each day of the week and returning there exactly one week later, we hoped to maximise return of students and the period between readings for each individual. Therefore, the trial was based in the Science Block on Mondays and Wednesdays, the Arts Block on Tuesdays and the Sports Hall on Thursdays. Although the population using the Sports Hall was considerably smaller than that of either the Science Block or the Arts Block, it provided us with a unique opportunity to fingerprint people directly after sports or showers.

An assurance of confidentiality was given by all parties involved in order to allay any privacy concerns. No names were stored at any time. Each student supplied their student number, which was stored in a spreadsheet and linked to a four-digit PIN. This PIN was the only means of identifying the individual's fingerprint in the database. Other personal details such as gender, age group and ethnic origin were cited as optional, but no objections were noted to supplying this information. Each user was required to sign a consent form, co-signed by a representative from the company, which stated that personal information could be stored for up to two months, and the fingerprint image itself for up to five years.

B.3 Implementation of Biometric Trial

Unfortunately, due to software issues, it was necessary to postpone the first day of the trial and so, it took place from Tuesday to Friday of the first week and Monday to Thursday of the final two weeks. Because of the large number of students that enrolled in the second and third weeks, it was decided to extend the trial to a fourth week in order to maximise the number of verifications and re-verifications obtained. However it was only possible to work three days (Tuesday to Thursday) during the final week, due to college restrictions. Thus, we specified that students must leave at least one day between readings to allow us to observe changes in the biometric over time. They were also encouraged to leave more days between readings if at all possible. Despite the extension of the trial to an extra week, a student could return a maximum of three times.

Because one of the primary aims was to ensure that the students returned at least once, it was necessary to offer incentives to achieve this. Each student

received a free lollipop upon enrolment. If they returned to verify, they received a free cinema ticket, which proved to be a huge success. If they returned a third time, they were entered into a draw for a Microsoft X-Box or travel vouchers. These incentives were advertised extensively on posters around the college campus and contributed greatly to students' willingness to participate in the trial.

E-mail addresses were collected from students in order to contact the winner of the draw. These were also used to send weekly reminders to students to return to the stand, emphasising the remaining incentives and specifying times and locations for verification. The addresses were also stored in the spreadsheet separate to the fingerprint database.

B.4 Problems Experienced by Users

Quite a number of people had difficulty telling their right hand from their left. As a result, they attempted to use the wrong hand until instructed otherwise. A substantial number were unsure as to which finger was their index finger, even though this finger was highlighted on the screen when selected. Several still had this problem at verification, despite having previously used the system. A smaller number didn't know which was their middle finger.

Some users, most of whom were female with small fingers, had difficulty placing their fingers correctly on the sensor, and were instructed by the system to adjust the position of their fingers frequently. The sensor needed to be cleaned very frequently; for some users, particularly those with naturally sweaty hands, the sensor had to be cleaned after each individual attempt. On the few occasions that someone with substantial amounts of ink or mud attempted to enrol, the sensor registered nothing. Having cleaned the sensor

and the finger, enrolment proceeded as normal. If the user did not remove the finger completely from the sensor, an image was sometimes detected even without them having replaced their finger on the pad.

There were some problems noted with dry skin towards the end of the trial, with users having to breathe on their fingers before the sensor would detect their presence. However none of these users failed to verify completely although the process was sometimes very slow.

B.5 Results of Fingerprint Matching Trial

Overall, interest in the project was widespread and the response to the technology was very positive. As expected, the population sample gathered was almost exclusively under thirty years old and Caucasian. Thus, it is hardly representative of the population at large and yet gives an idea of the best possible operating results of the system, given that this portion of the population seem to have the best defined fingerprints. The final results of the trial are summarised in Table B.1. The total number of enrolments was 689,

	Total Individuals	% Male	% Female	% Return
Enrolment	689	48.2	51.8	n/a
Verification	534	47.9	52.1	77.5
Re-Verification	319	49.8	50.2	46.2

Table B.1: Final results of Trinity College fingerprint trial.

72.25% greater than our initial aim. The number of verifications was more than twice what we had hoped, resulting in a 77.5% return rate for verification and a 46.2% return rate for reverification. The trial was deemed a success, as we far exceeded our initial aims.

Appendix C

Berlekamp-Massey Algorithm

The Berlekamp-Massey algorithm [Mas69] provides a simpler and speedier means of solving a system of linear equations. It can be used in the decoding of Reed-Solomon codes to solve, for p_i, the equations

$$T_{j+r} + p_1 T_{j+r-1} + p_2 T_{j+r-2} + \cdots + p_{r-1} T_{j+1} + p_r T_j = 0 \qquad \text{(C.1)}$$

for $j = 1$ to $(2t - r)$ with $r \leq t$. These equations are known as the "Newton identities". The value of t is known but not the value of r, where t is the maximum number of correctable errors and r is the actual number of errors. This algorithm produces the value of r and the corresponding values of the p_i. The steps of the algorithm are [Pur95, pp 125-126]:

1. Initialise $g(x) = 1$, $c(x) = x$, $n = 1$ and $r = 0$,
   where $g(x) = x^r p(1/x)$, $c(x)$=*correction polynomial,* n=*count variable.*

2. Calculate the prediction error: $e = T_n + \sum_{i=1}^{r} p_i T_{n-i}$

3. If $e = 0$, go to step 8.
   Else, do steps 4 to 6 to make adjustments to remove the error.

4. Let $g^*(x) = g(x) - ec(x)$.

   This forms a new predictor polynomial that satisfies the equations up to the current value corresponding to n, namely T_n.

5. If $2r \geq n$, go to step 7.

   There are $(n - r)$ *equations in* r *unknowns, so if* $2r \geq n$ *there is no problem of inconsistency.*

6. Let $r = (n - r)$ and $c(x) = g(x)/e$.

   There are too few unknowns for $(n - r)$ *equations; increase them to accommodate* $g^*(x)$ *and establish a new correction polynomial that leaves all equations up to* $(n - 1)$ *satisfied and normalises the error for the last equation.*

7. Let $g(x) = g^*(x)$.

   Set g(x) to the value of the new predictor polynomial.

8. Let $c(x) = xc(x)$.

   Increment powers of x in correction polynomial for next loop iteration.

9. Let $n = n + 1$.

   Increment the counter for the next iteration of the loop.

10. If $n \geq 2t$, go to step 2. Else, terminate.

Appendix D

Code to Generate Byte Errors in Simulated Iris Readings

The C code of the model (Chapter 6) to determine the number of byte errors in a simulated iris reading is below. Let 0 represent a correct bit and 1 an error. R is the bit-error rate; p is the probability of a 1, given the previous bit is a 0; Q is the opposite. We gradually increase the size of Q; p is determined from R and Q. The first bit is set using p; the other $(N-1)$ bits use either p or Q, as appropriate. X is used in random number generation; A is the total number of bytes; B the number of program runs. The number of 1s gives the number of bit errors; the number of byte errors is calculated from this.

```
#include<stdio.h>       #include<math.h>        #include<stdlib.h>
#define R 0.11          #define N 1600          #define A 200
#define Q 0.05          #define X 1000          #define B 10000
int main()
{       int i, j, x, count, err, ran1, ran2, temp8, *byte, *array, *res;
        float temp1, temp2;
        double p, rate, temp3, *acrate, temp9, btot;
        double temp5, temp6, tot, stan, meanrate, temp7, meanbyte, bstan;
        FILE *file_b, *file_e;
        array = (int *) malloc((N+1) * sizeof(int));
        res = (int *) malloc((A+1) * sizeof(int));
        byte = (int *) malloc((B+1) * sizeof(int));
```

```
acrate = (double *) malloc((B+1) * sizeof(double));

srand(time(NULL));
file_b = fopen("Calcs05", "w");
file_e = fopen("Byte05", "w");
p = ((R*Q)/(1-R));

for(i=0; i<B; i++)
{       acrate[i] = 0;
        byte[i] = 0;  }
tot = 0;        stan = 0;       btot = 0;
temp6 = 0;      temp9 = 0;      bstan = 0;

for(x=0; x<B; x++)
{for(i=0; i<N; i++)
{       array[i] = 0; }
for(i=0; i<A; i++)
{       res[i] = 0; }          rate = 0;

/* Calculating first bit in array, using p */
temp1 = (float)rand()/RAND_MAX;
temp3 = (1/p);
ran1 = (int) temp3*temp1;
if(ran1 == 1)
{       array[0] = 1; }
/* Calculating remainder of array bits, using p and Q */
for(i=0; i<N; i++)
{       if(array[i] == 0)
        {       temp1 = (float)rand()/RAND_MAX;
                ran1 = (int) temp3*temp1;
                if(ran1 == 1)
                {        array[i+1] = 1; }}
        else{   temp2 = (float)rand()/RAND_MAX;
                ran2 = (int) X*temp2;
                temp7 = X*Q;
                if(ran2 < temp7)
                {        array[i+1] = 0; }
                else{    array[i+1] = 1; }}}
/* Counting number of bit errors */
count = 0;
for(i=0; i<N; i++)
{       if(array[i] == 1)
        {       count++; }}
/* No iris reading has greater than 32.7% bit-error rate -
ensure that this does not occur in model */
if(count>(N*0.327))
{       x--; }
else{ /* Calculating actual bit error rate */
```

```
rate = (double) count/N;
acrate[x] = rate;

j=0;
for(i=1; i<(N+1); i++)
{       res[j] += array[i-1];
        if(i%8 == 0)
        {       j++; }}
/* Calculating number of byte errors */
err = 0;
for(i=0; i<A; i++)
{       if(res[i] != 0)
        {       err++; }}
byte[x] = err;
fprintf(file_e, "%d\n", err);
fclose(file_e);

/* Calculating mean bit-error rate */
for(i=0; i<B; i++)
{       temp5 += acrate[i]; }
meanrate = temp5/B;

/* Calculating standard deviation of bit-error rate */
for(i=0; i<B; i++)
{       if(acrate[i] > meanrate)
        {       temp6 = acrate[i] - meanrate; }
        else{   temp6 = meanrate - acrate[i]; }
        temp6 = (temp6 * temp6);
        tot += temp6; }
stan = sqrt(tot/B);
fprintf(file_b, "%0.2f %0.3f %lf %lf %lf\n", R, Q, p,meanrate,stan)
fclose(file_b);

/* Calculating mean byte errors */
temp8=0;
for(i=0; i<B; i++)
{       temp8 += byte[i]; }
meanbyte = (double)temp8/B;

/* Calculating standard deviation of byte errors */
for(i=0; i<B; i++)
{       if(byte[i] > meanbyte)
        {       temp9 = byte[i] - meanbyte; }
        else {  temp9 = meanbyte - byte[i]; }
        temp9 = (temp9 * temp9);
        btot += temp9; }
bstan = sqrt(btot/B);
exit(0);}
```

Appendix E

Generating Look-Up Tables for RS Codes

In this appendix, we give extracts of the C code used to generate two look-up tables used for arithmetic in our iris recognition program. Each look-up table is also given so that calculations may be independently verified. The ground field is GF(2^8), and the primitive polynomial used is

$$x^8 + x^5 + x^3 + x + 1 \ . \tag{E.1}$$

E.1 Look-Up Table LUT

Look-up table LUT has the α-power as its first value and the hex value linked to that power as its second. The output is in increasing power of α.

```
int x=0;
unsigned char y=1;

/* x<256 as 2^8=256 and all calculations done mod 255 */
while(x<256)
{       x++;

        /* Bit shift y if less than 10000000 (0 in first pos) */
        if(y<128)
        {       y = y << 1; }
        /* Else AND y with 01111111 to remove leading 1, bit shift
           and XOR y with 00101011 (corresponds to primitive poly) */
```

```
else
{       y = y&127;
        y = y << 1;
        y = y^43;  }}
```

Note that $\alpha^0 = 0$ and $\alpha^{255} = 1$, as explained in the main body of the thesis. The code generates the following output for look-up table LUT:

```
0 0     39 35   78 96   117 d0   156 da   195 33
1 2     40 6a   79 7    118 8b   157 9f   196 66
2 4     41 d4   80 e    119 3d   158 15   197 cc
3 8     42 83   81 1c   120 7a   159 2a   198 b3
4 10    43 2d   82 38   121 f4   160 54   199 4d
5 20    44 5a   83 70   122 c3   161 a8   200 9a
6 40    45 b4   84 e0   123 ad   162 7b   201 1f
7 80    46 43   85 eb   124 71   163 f6   202 3e
8 2b    47 86   86 fd   125 e2   164 c7   203 7c
9 56    48 27   87 d1   126 ef   165 a5   204 f8
10 ac   49 4e   88 89   127 f5   166 61   205 db
11 73   50 9c   89 39   128 c1   167 c2   206 9d
12 e6   51 13   90 72   129 a9   168 af   207 11
13 e7   52 26   91 e4   130 79   169 75   208 22
14 e5   53 4c   92 e3   131 f2   170 ea   209 44
15 e1   54 98   93 ed   132 cf   171 ff   210 88
16 e9   55 1b   94 f1   133 b5   172 d5   211 3b
17 f9   56 36   95 c9   134 41   173 81   212 76
18 d9   57 6c   96 b9   135 82   174 29   213 ec
19 99   58 d8   97 59   136 2f   175 52   214 f3
20 19   59 9b   98 b2   137 5e   176 a4   215 cd
21 32   60 1d   99 4f   138 bc   177 63   216 b1
22 64   61 3a   100 9e  139 53   178 c6   217 49
23 c8   62 74   101 17  140 a6   179 a7   218 92
24 bb   63 e8   102 2e  141 67   180 65   219 f
25 5d   64 fb   103 5c  142 ce   181 ca   220 1e
26 ba   65 dd   104 b8  143 b7   182 bf   221 3c
27 5f   66 91   105 5b  144 45   183 55   222 78
28 be   67 9    106 b6  145 8a   184 aa   223 f0
29 57   68 12   107 47  146 3f   185 7f   224 cb
30 ae   69 24   108 8e  147 7e   186 fe   225 bd
31 77   70 48   109 37  148 fc   187 d7   226 51
32 ee   71 90   110 6e  149 d3   188 85   227 a2
33 f7   72 b    111 dc  150 8d   189 21   228 6f
34 c5   73 16   112 93  151 31   190 42   229 de
35 a1   74 2c   113 d   152 62   191 84   230 97
36 69   75 58   114 1a  153 c4   192 23   231 5
37 d2   76 b0   115 34  154 a3   193 46   232 a
38 8f   77 4b   116 68  155 6d   194 8c   233 14
```

```
234 28      238 d6      242 94      246 18      250 ab      254 95
235 50      239 87      243 3       247 30      251 7d      255 1
236 a0      240 25      244 6       248 60      252 fa
237 6b      241 4a      245 c       249 c0      253 df
```

E.2 Look-Up Table LUT2

Look-up table LUT2 has the hex value as its first value and the α-power as its second. The output is in increasing hex byte value.

```
int x=0, i, x_array[256];
unsigned char y=1, y_array[256];
x_array[0] = 0;
y_array[0] = 1;

/* x<255 as 2^8=256 and all calculations done mod 255 */
while(x<255)
{       x++;

        /* Bit shift y if less than 10000000 (0 in first pos) */
        if(y<128)
        {        y = y << 1; }
        /* Else AND y with 01111111 to remove leading 1, bit shift
           and XOR y with 00101011 (corresponds to primitive poly) */
        else
        {       y = y&127;
                y = y << 1;
                y = y^43;}
        /* y-value in relevant part of y_array; x-value in corresponding
           part of x_array so order is in increasing y */
        y_array[y] = y;
        x_array[y] = x; }
```

The code generates the following output for look-up table LUT2:

```
0 0         9 67        12 68       1b 55       24 69       2d 43
1 255       a 232       13 51       1c 81       25 240      2e 102
2 1         b 72        14 233      1d 60       26 52       2f 136
3 243       c 245       15 158      1e 220      27 48       30 247
4 2         d 113       16 73       1f 201      28 234      31 151
5 231       e 80        17 101      20 5        29 174      32 21
6 244       f 219       18 246      21 189      2a 159      33 195
7 79        10 4        19 20       22 208      2b 8        34 115
8 3         11 207      1a 114      23 192      2c 74       35 39
```

36 56	58 75	7a 120	9c 50	be 28	e0 84
37 109	59 97	7b 162	9d 206	bf 182	e1 15
38 82	5a 44	7c 203	9e 100	c0 249	e2 125
39 89	5b 105	7d 251	9f 157	c1 128	e3 92
3a 61	5c 103	7e 147	a0 236	c2 167	e4 91
3b 211	5d 25	7f 185	a1 35	c3 122	e5 14
3c 221	5e 137	80 7	a2 227	c4 153	e6 12
3d 119	5f 27	81 173	a3 154	c5 34	e7 13
3e 202	60 248	82 135	a4 176	c6 178	e8 63
3f 146	61 166	83 42	a5 165	c7 164	e9 16
40 6	62 152	84 191	a6 140	c8 23	ea 170
41 134	63 177	85 188	a7 179	c9 95	eb 85
42 190	64 22	86 47	a8 161	ca 181	ec 213
43 46	65 180	87 239	a9 129	cb 224	ed 93
44 209	66 196	88 210	aa 184	cc 197	ee 32
45 144	67 141	89 88	ab 250	cd 215	ef 126
46 193	68 116	8a 145	ac 10	ce 142	f0 223
47 107	69 36	8b 118	ad 123	cf 132	f1 94
48 70	6a 40	8c 194	ae 30	d0 117	f2 131
49 217	6b 237	8d 150	af 168	d1 87	f3 214
4a 241	6c 57	8e 108	b0 76	d2 37	f4 121
4b 77	6d 155	8f 38	b1 216	d3 149	f5 127
4c 53	6e 110	90 71	b2 98	d4 41	f6 163
4d 199	6f 228	91 66	b3 198	d5 172	f7 33
4e 49	70 83	92 218	b4 45	d6 238	f8 204
4f 99	71 124	93 112	b5 133	d7 187	f9 17
50 235	72 90	94 242	b6 106	d8 58	fa 252
51 226	73 11	95 254	b7 143	d9 18	fb 64
52 175	74 62	96 78	b8 104	da 156	fc 148
53 139	75 169	97 230	b9 96	db 205	fd 86
54 160	76 212	98 54	ba 26	dc 111	fe 186
55 183	77 31	99 19	bb 24	dd 65	ff 171
56 9	78 222	9a 200	bc 138	de 229	
57 29	79 130	9b 59	bd 225	df 253	

Appendix F

Code for Iris Authentication

F.1 Enrolment Program

This C program simulates the enrolment of a new user in an iris authentication system. The user types "Enrol" and follows the instructions. He identifies himself by name, and his details are stored in a file under this name. He chooses a 256-byte character string from a selection provided to represent his iris reading. He picks 56 bytes to omit from the string, as masking bytes. Finally, he modifies some further bytes, to simulate a unique reading. The program calculates check digits for the data, hashes the iris bytes and stores this value for verification purposes.

```
#include<stdio.h>       #include<string.h>      #define P 200
#include<math.h>        #include<gcrypt.h>      #define PP 256
#include<ctype.h>       #define M 254           #define QQ 512
#include<stdlib.h>      #define N 55

int main()
{ FILE *file_iris, *file_data, *file_datapoly, *file_lut, *file_lut2;
FILE *file_code, *file_id, *file_genpoly;
int posit[N+1], data[PP], data_temp[PP], test[QQ];
int datapoly[M], gen[N], quot[P], lut[2][256], lut2[2][256];
int buff[P], poschange[P], numbytes, temp, temp2, count, i, j, k, l, m, n;
unsigned char *results, irisname[QQ], iriscode[QQ], array[QQ], id[N];
```

```
GcryMDHd H;

/************** PREPARING AND SCANNING IN DATA FILE ********************/
/* To prepare incoming data file, convert string of uppercase hex character
to file of lowercase hex, organised in bytes (hex symbols must be taken in
pairs from string). Store in file Data for later use. */

for(i=0;i<QQ;i++)
{       array[i]=0;
        test[i]=0; }
for(i=0;i<PP;i++)
{       data_temp[i]=0;
        data[i]=0; }
for(i=0;i<M;i++)
{        datapoly[i]=0; }

printf("Please enter your name: ");
scanf("%s", &id);
printf("\nWelcome %s!\n", id);

printf("This is a program to simulate enrolment in a system which identifie
printf("\npeople by their iris.\n We use a string of data to represent the
printf("iris reading.\n Choose one of the following data-strings: ");
printf("Iris1, Iris2, Iris3, Iris4\n%s", iriscode);
scanf("%s", &iriscode);
printf("\nThe basic iris file that will be used is %s\n", iriscode);

/* To read in files: LookUpTable by power of alpha, LookUpTable2 in hex ord
file_iris = fopen(iriscode, "r");
file_lut = fopen("LookUpTable", "r");
file_lut2 = fopen("LookUpTable2", "r");
/* In case no suitable filename is specified */
if(file_iris == NULL)
{       printf("ERROR: Iris file cannot be read in \n");
        exit(3); }
/* Copy irisfile name to new, unmodified string, so can be re-written later
else{   printf("This file has been successfully read in \n\n");
        strcpy(irisname, iriscode); }

/* To write contents of file to an array */
j=0;
while(fscanf(file_iris, "%c", &array[j]) == 1)
{        j++; }
fclose(file_iris);

/* j represents the number of items in the file */
j--;
printf("There are %d hex digits in the file\n", j);
```

```
printf("This means we have %d numbers in total\n\n", j/2);

/* i becomes the number of items in the file */
i=j;

/* To write contents of LookUpTable to array lut */
j=0;
while(fscanf(file_lut, "%d %x", &lut[0][j], &lut[1][j]) != EOF)
{       j++; }
fclose(file_lut);

/* To write contents of LookUpTable2 to array lut2 */
j=0;
while(fscanf(file_lut2, "%x %d", &lut2[0][j], &lut2[1][j]) != EOF)
{       j++; }
fclose(file_lut2);

/* To convert all letters to lower-case */
for(j=0;j<i;j++)
{       array[j]=tolower(array[j]); }
m=0;

/* k is half number of items in original file; items can be taken in pairs */
k=i/2;

/* To convert hex to dec & put byte hex in temp and powers of alpha in temp2 */
for(j=0;j<i;j=j+2)
{       if(array[j]=='a' || array[j]=='b' || array[j]=='c' ||
           array[j]=='d' || array[j]=='e' || array[j]=='f')
        {       test[j] = array[j] - 'a' + 10; }
        if(array[j+1]=='a' || array[j+1]=='b' || array[j+1]=='c' ||
           array[j+1]=='d' || array[j+1]=='e' || array[j+1]=='f')
        {       test[j+1] = array[j+1] - 'a' + 10; }
        if(array[j] <= '9')
        {       test[j] = array[j] - 48; }
        if(array[j+1] <='9')
        {       test[j+1] = array[j+1] - 48; }

        /* To ensure data is calculated in pairs */
        data_temp[m] = (test[j]*16)+test[j+1];

        /* To obtain decimal data */
        data[m] = lut2[1][data_temp[m]];
        m++; }
/* To write data in new form back to file */
file_data = fopen("Data", "w");
for(j=0;j<PP;j++)
{       fprintf(file_data, "%d\n", data[j]); }
```

```
fclose(file_data);

for(i=0; i<N+1; i++)
{       posit[i] = 0; }
/********************** MASKING OUT BYTES ********************************
/* Choose bytes to be masked out. Also multiply data by x^{54} and store in
DataPoly to be divided by generating poly later to form check digits. */

printf("Now you must select certain bytes to be excluded from your reading.
printf("These represent areas obscured by eyelashes, eyelids etc.\n");

printf("Choose 56 bytes in total to be masked out.\n");
printf("Enter any number from 0-255,but do not enter the same number twice.
for(i=0;i<N+1;i++)
{       printf("Byte %d located at position: ", i+1);
        if(scanf("%d", &posit[i]) != 1)
        {       printf("ERROR: Not a number.");
                printf(" Exiting program. Please start again.\n");
                exit(10); }
        printf("\n");

        /* To ensure that no number greater than 256 is accepted */
        if(posit[i] >= PP)
        {       printf("ERROR: Number must be less than 256.");
                printf("Please try again.\n");
                i--; }
        /* To ensure that 56 different numbers are inputted */
        for(j=0; j<i; j++)
        {       if(posit[i] == posit[j])
                {       printf("ERROR: This number has been entered");
                       printf(" previously. Please try again.\n");
                       i--; }}}
/* Now we must omit these bits from array data */
k=0;
for(i=0; i<PP; i++)
{      count = 0;

       /* Check if this bit position has been chosen for omission */
       for(j=0; j<N+1; j++)
       {       if(i == posit[j])
               {       count++; }}
       /* If not, write it to datapoly (data multiplied by x^54) */
       if(count == 0)
       {       datapoly[k+54] = data[i];
               k++; }}
printf("56 bytes chosen and masked out from calculations.\n\n");

for(i=0; i<P; i++)
```

```
{       poschange[i] = 0; }
/********************** MODIFYING BYTES ******************************/
printf("Now you may choose to modify some further bytes in your file, in");
printf(" order to make your reading unique.\n");
printf("Enter the number of bytes (at most 8) you wish to modify: ");
if(scanf("%d", &numbytes) != 1)
{       printf("ERROR: Not a number.  Exiting program. Please start again.\n");
        exit(12); }
while(numbytes>8)
{       printf("ERROR: Number of modified bytes must be less than or equal to");
        printf(" 8.\nPlease try again.\n");
        printf("Enter the number of bytes you wish to modify: ");
        if(scanf("%d", &numbytes) != 1)
        {       printf("ERROR: Not a number.");
                printf(" Exiting program. Please start again.\n");
                exit(12); }}
if(numbytes > 0)
{       printf("Enter 1st byte position (between 0 and 199): "); }
for(i=0; i<numbytes; i++)
{       temp2 = 0;
        if(scanf("%d", &poschange[i]) != 1)
        {       printf("ERROR: Not a number.");
                printf(" Exiting program. Please start again.\n");
                exit(11); }
        /* To ensure that the number entered is less than 200 */
        if(poschange[i] >= P)
        {       printf("ERROR: Number must be less than 200.");
                printf(" Please try again.\n");
                i--;
                temp2++; }
        /* To ensure that no number is entered twice */
        for(j=0; j<i; j++)
        {       if(poschange[i] == poschange[j])
                {       printf("ERROR: This byte has already been ");
                        printf("changed. Please choose another.\n");
                        i--;
                        temp2++; }}
        /* To locate contents of byte selected & prompt user to input new
        value for byte & write new value to datapoly */
        if(temp2 == 0)
        {       /* Because first 54 positions in DataPoly are set to zero */
                temp = poschange[i]+54;
                printf("Current value of byte %d is %d.\n", poschange[i],
                                                        datapoly[temp]);
                printf("Enter new value for byte %d ", poschange[i]);
                printf("(between 0 and 255): ");

                /* To ensure a number is entered */
```

```
                if(scanf("%d", &datapoly[temp]) != 1)
                {       printf("ERROR: Not a number. Exiting program. ");
                        printf("Please start again.\n");
                        exit(12); }
                /* To ensure value is less than 255, use mod 255 if needed */
                if(datapoly[temp] >= PP)
                {       printf("Number is greater than 255.");
                        printf(" Converting mod 255...\n");
                        datapoly[temp] = (datapoly[temp]%255); }
                printf("Value for byte %d has been changed to %d.\n\n",
                                                poschange[i], datapoly[tem
        /* To print out the next byte position required */
        if((i==0)&&(numbytes>1))
        {        printf("Enter 2nd byte position (between 0 and 199): "); }
        if((i==1)&&(numbytes>2))
        {        printf("Enter 3rd byte position (between 0 and 199): "); }
        if((i>1)&&(i+1<numbytes))
        {       printf("Enter %dth byte position (between 0 and 199): ", i+
/* To write data by x^54 in new form back to file */
file_datapoly = fopen("DataPoly", "w");
for(j=0;j<M;j++)
{       fprintf(file_datapoly, "%d\n", datapoly[j]); }
fclose(file_datapoly);

for(i=0; i<P; i++)
{       buff[i]=0; }

/* To save data for hashing later on */
for(i=0; i<P; i++)
{       buff[i] = datapoly[i+54]; }
/************************ CREATING CHECK DIGITS ***************************
*/ Divide DataPoly be generating poly and store remainder in file Remainder
These are check digits. Loop must be iterated 200 times for these parameter
(difference between max x powers in two polys in question). */

for(j=0; j<P; j++)
{       quot[j]=0; }
for(j=0; j<N;j++)
{       gen[j]=0; }
count = P;

/* k represents highest power of x in DataPoly */
k=M-1;

/* To write contents of GenPoly file to array */
file_genpoly = fopen("GenPoly", "r");
j=0;
while(fscanf(file_genpoly, "%d", &gen[j]) != EOF)
```

```
{       j++; }
fclose(file_genpoly);

/* n represents highest power of x in GenPoly */
n=j-1;

/* Iterate loop 200 times, successively dividing */
while(count>0)
{       /* To set leading term of quotient */
        quot[k-n]=datapoly[k];

        /* Have already dealt with leading term...*/
        for(m=k-1; m>=k-n; m--)
        {       i=n-k+m;
                /* Eliminate highest power of x, as this has been divided */
                datapoly[k]=0;

                /* Equivalent to multiplying leading coeff by divisor element*/
                temp2 = (gen[i]+quot[k-n])%255;

                if(temp2 == 0)
                {       temp2 = 255; }
                if(quot[k-n] == 0)
                {       temp2 = 0; }
                /* Equivalent to subtracting els to get new remainder */
                if(temp2 != 0)
                {       temp = lut[1][datapoly[m]]^lut[1][temp2];
                       datapoly[m] = lut2[1][temp]; }
                i--; }
        k--;
        count--; }
/* To write the filename back to the user's file */
file_id = fopen(id, "w");
fprintf(file_id, "%s\n", irisname);
fclose(file_id);
printf("Filename has been written to your file.\n");

/* To write the checkdigits back to a file */
file_id = fopen(id, "a");
for(l=0;l<=k;l++)
{       fprintf(file_id, "%d\n", datapoly[l]); }
fclose(file_id);
printf("Check digits have been written to your file.\n");

/* To write the masked byte positions back to the user's file */
file_id = fopen(id, "a");
for(l=0;l<=N;l++)
{       fprintf(file_id, "%d\n", posit[l]); }
```

```
fclose(file_id);
printf("Masked byte positions have been written to your file.\n");

/* To hash user's data (using sha1 functions from Libgrcypt by GNU) */
H = gcry_md_open(GCRY_MD_SHA1,0);
gcry_md_write(H, &buff, 200);
results = gcry_md_read(H, 0);
gcry_md_close(H);

/* To write the hashed bits back to the user's file */
file_id = fopen(id, "a");
for(l=0; l<20; l++)
{       fprintf(file_id, "%x\n", results[l]); }
fclose(file_id);
printf("Hashed data bits have been written to your file.\n");

printf("You have been successfully enrolled!\n");
printf("See you next time, %s!\n", id); }
```

F.2 Verification Program

This program simulates verification in an iris authentication scheme. The user types "Verify" and is asked for his name, which retrieves the information stored in his file. The 56 bytes previously masked out are automatically excluded. The user chooses to treat up to twenty bytes as erasures, and can modify a number of other bytes, depending on how many erasures are selected. The check digits from the user's file are used to construct a corrupted codeword from the data, and the program attempts to correct this and restore it to its original form. The result is hashed and compared to the stored hash. If these match, the user has been successfully verified.

```
#include<stdio.h>       #include<gcrypt.h>      #define Q 400
#include<math.h>        #define M 254           #define R 27
#include<ctype.h>       #define N 55            #define PP 256
#include<stdlib.h>      #define P 200           #define QQ 512

int main()
{ FILE *file_iris, *file_data, *file_datapoly, *file_lut, *file_lut2;
FILE *file_id, *file_rem, *file_quot, *file_genpoly, *file_nullt;
```

```
FILE *file_g, *file_p, *file_r, *file_x, *file_y, *file_code, *file_syn;
int enrol[2*N], posit[N+1], data[PP], data_temp[PP], test[QQ], i, j, k, l, m, n;
int datapoly[M], gen[N], quot[P], lut[2][256], lut2[2][256], e, r, d, s, T_temp;
int arrr[M+1], HT[M][N-1], code[M], D[N], Syn[N-1], roots[N-1], hashchar[20];
int newc[P], arr[N], gstar[N], f[N], E[R], cor[N], g[N], Y[R], X[R][R], alpha;
int finalprod[N], Mat[N], prod[N], power[2], res[N], store[N], T[N], total, max;
int correr[R], correras[R], numbytes, checkarray[N-1], poschange[P], Syn_temp;
int temp, temp2, temp3, temp4, temp5, temp6, temp7, sum, Mat_temp, check, count;
unsigned char *results, iriscode[QQ], array[QQ], id[N];
GcryMDHd H;

/*************** PREPARING AND SCANNING IN DATA ************************/
/* To prepare incoming data file, convert string of uppercase hex characters to
file of lowercase hex, organised in bytes (hex symbols must be taken in pairs
from string). Store in file Data for later use. */
for(i=0;i<QQ;i++)
{       array[i]=0;
        test[i]=0; }
for(i=0;i<PP;i++)
{       data_temp[i]=0;
        data[i]=0; }
for(i=0;i<M;i++)
{        datapoly[i]=0; }
printf("Please enter your name: ");
scanf("%s", &id);

/* To ensure the user has previously been enrolled */
file_id = fopen(id, "r");
if(file_id == NULL)
{       printf("ERROR: You have not yet successfully enrolled.\n");
        printf("Please type 'Enrol' to do so now...\n");
        exit(10); }
fscanf(file_id, "%s", &iriscode);
printf("\nWelcome %s!\n This is a program to simulate verifying that a", id);
printf(" person is who they claim to be,\nbased on a comparison between the");
printf(" iris reading they provide now,\nand the one previously stored for");
printf(" them at enrolment.\n");

printf("\nYour chosen iris filename was %s\n", iriscode);

/* To read in the files; lut by power of alpha, lut2 in hex order */
file_iris = fopen(iriscode, "r");
file_lut = fopen("LookUpTable", "r");
file_lut2 = fopen("LookUpTable2", "r");

/* In case no suitable filename is specified */
if(file_iris == NULL)
{       printf("ERROR: Iris file cannot be read in \n");
```

```
        exit(3); }
/* To write contents of file to an array */
j=0;
while(fscanf(file_iris, "%c", &array[j]) == 1)
{       j++; }
fclose(file_iris);

/* j represents the number of items in the file */
j--;
printf("There are %d hex digits in the file\n", j);
printf("This means we have %d numbers in total\n\n", j/2);

/* i becomes the number of items in the file */
i=j;

/* To write contents of LookUpTable to array lut */
j=0;
while(fscanf(file_lut, "%d %x", &lut[0][j], &lut[1][j]) != EOF)
{       j++; }
fclose(file_lut);

/* To write contents of LookUpTable2 to array lut2 */
j=0;
while(fscanf(file_lut2, "%x %d", &lut2[0][j], &lut2[1][j]) != EOF)
{       j++; }
fclose(file_lut2);

/* To convert all letters to lower-case */
for(j=0;j<i;j++)
{       array[j]=tolower(array[j]); }
m=0;

/* k is half number of items in original file; items can be taken in pairs
k=i/2;

/* To convert hex to dec & put byte hex in temp and powers of alpha in temp
for(j=0;j<i;j=j+2)
{       if(array[j]=='a' || array[j]=='b' || array[j]=='c' ||
           array[j]=='d' || array[j]=='e' || array[j]=='f')
        {       test[j] = array[j] - 'a' + 10; }
        if(array[j+1]=='a' || array[j+1]=='b' || array[j+1]=='c' ||
           array[j+1]=='d' || array[j+1]=='e' || array[j+1]=='f')
        {       test[j+1] = array[j+1] - 'a' + 10; }
        if(array[j] <= '9')
        {       test[j] = array[j] - 48; }
        if(array[j+1] <='9')
        {       test[j+1] = array[j+1] - 48; }
```

```
        /* To ensure data is calculated in pairs */
        data_temp[m] = (test[j]*16)+test[j+1];

        /* In order to obtain the decimal data */
        data[m] = lut2[1][data_temp[m]];
        m++; }

/* To write data in new form back to file */
file_data = fopen("Data", "w");
for(j=0;j<PP;j++)
{        fprintf(file_data, "%d\n", data[j]); }
fclose(file_data);
/* To read in data from enrolment file into arrays*/
for(i=0;i<2*N;i++)
{        fscanf(file_id, "%d", &enrol[i]); }
for(i=0; i<20; i++)
{        fscanf(file_id, "%x", &hashchar[i]); }
fclose(file_id);
/******************** MASKING OUT BYTES ***********************************/
/* Omit same bits as before from array Data. These were chosen for omission by
user in enrolment and represent masked-out bytes in each subsequent reading. */
k=0;
for(i=0; i<PP; i++)
{       count = 0;
        /* Check if this bit position has been chosen for omission */
        for(j=0; j<N+1; j++)
        {       if(i == enrol[j+54])
                {        count++; }}
        /* If not, write it to datapoly (data multiplied by x^54) */
        if(count == 0)
        {       datapoly[k+54] = data[i];
                k++;}}
printf("The 56 bytes you chose to exclude during enrolment are once again");
printf(" masked out.\n\n");

for(j=0; j<N; j++)
{        store[j] = 255; }
/*************************** CHOOSING ERASURES ***************************/
printf("Now you may choose to have some bytes treated as erasures\n(which ");
printf("are errors whose location is known)\nNote: These MUST be entered in ");
printf("increasing order.\nEnter the number of erasures (no more than 20): ");

/* To ensure a number is entered */
if(scanf("%d", &s)!= 1)
{       printf("ERROR: Not a number. Exiting program. Please start again.\n");
        exit(14); }
/* To ensure the number of erasures does not exceed 20 */
while(s > 20)
```

```
{       printf("Number of erasures must be less than 20.\nPlease try again.\
        printf("Enter the number of erasures: ");
        scanf("%d", &s); }
printf("You have chosen to include %d erasures.\n\n", s);

for(i=0; i<s; i++)
{       temp2 = 0;
        /* To prompt the user to input the next position */
        if(i==0)
        {       printf("Enter 1st byte position (between 0 and 199): "); }
        if(i==1)
        {       printf("Enter 2nd byte position (between 0 and 199): "); }
        if(i==2)
        {       printf("Enter 3rd byte position (between 0 and 199): "); }
        if(i>2)
        {       printf("Enter %dth byte position (between 0 and 199): ", i+
        /* To ensure that a number is entered */
        if(scanf("%d", &store[i]) != 1)
        {       printf("ERROR: Not a number.");
                printf(" Exiting program. Please start again.\n");
                exit(9); }
        /* To ensure that the number entered is less than 200 */
        if(store[i] >= P)
        {       printf("ERROR: Number must be less than 200.");
                printf(" Please try again.\n");
                i--;
                temp2++; }
        /* To ensure that no number is entered twice */
        for(j=0; j<i; j++)
        {       if(store[i] == store[j])
                {       printf("ERROR: This byte has already been ");
                        printf("chosen. Please choose another.\n");
                        i--;
                        temp2++; }}
        /* To ensure that erasure values are entered in increasing order */
        if(store[i] < store[i-1])
        {       printf("ERROR: Number is less than last number entered.\n")
                printf("Erasures must be entered in increasing order.\n");
                printf("Please try again.\n");
                i--;
                temp2++; }
        if(temp2 == 0)
        {       printf("Byte %d is an erasure.\n", store[i]); }}

/* To access correct part of codeword (check digits are in first 54 positio
for(i=0; i<s; i++)
{       store[i] = store[i]+54; }
```

```
for(i=0; i<P; i++)
{       poschange[i] = 0; }
/************************ MODIFYING BYTES ********************************/
/* total is number of correctable errors, based on max of 8 from enrolment
and whatever number (less than 20) of erasures have just been chosen */
total = ((38-s)/2);

printf("\nNow you may choose to modify some further bytes\n Enter the number");
printf(" of bytes (at most %d) you wish to modify: ", total);
if(scanf("%d", &numbytes)!= 1)
{       printf("ERROR: Not a number. Exiting program. Please start again.\n");
        exit(12); }
/* To ensure number of bytes modified does not exceed total number possible */
while(numbytes > total)
{       printf("ERROR: Number of bytes modified must be less than or equal to");
        printf(" %d.\n Please try again.\n", total);
        printf("Enter the number of bytes you wish to modify: ");
        scanf("%d", &numbytes); }
if(numbytes > 0)
{        printf("Enter 1st byte position (between 0 and 199): "); }
for(i=0; i<numbytes; i++)
{       temp2 = 0;

        /* To ensure a number is entered */
        if(scanf("%d", &poschange[i]) != 1)
        {       printf("ERROR: Not a number.");
                printf(" Exiting program. Please start again.\n");
                exit(11); }
        /* To ensure that the number entered is less than 200 */
        if(poschange[i] >= P)
        {       printf("ERROR: Number must be less than 200.");
                printf(" Please try again.\n");
                i--;
                temp2++; }
        /* To ensure that no number is entered twice */
        for(j=0; j<i; j++)
        {       if(poschange[i] == poschange[j])
                {       printf("ERROR: This byte has already been ");
                        printf("changed. Please choose another.\n");
                        i--;
                        temp2++; }}
        /* Locate contents of byte selected, prompt user to input new value
        and write this new value to datapoly */
        if(temp2 == 0)
        {       /* Because first 54 positions in DataPoly are set to zero */
                temp = poschange[i]+54;
                printf("Enter new value for byte %d ", poschange[i]);
                printf("(between 0 and 255): ");
```

```
                /* To ensure a number is entered */
                if(scanf("%d", &datapoly[temp]) != 1)
                {       printf("ERROR: Not a number. Exiting program. ");
                        printf("Please start again.\n");
                        exit(12); }
                /* To ensure value is less than 255, use mod 255 if needed *
                if(datapoly[temp] >= PP)
                {       printf("Number is greater than 255.");
                        printf(" Converting mod 255...\n");
                        datapoly[temp] = (datapoly[temp]%255); }
                printf("Value for byte %d has been changed to %d.\n\n",
                                                poschange[i], datapoly[tem

        /* To print out the next byte position required */
        if((i==0)&&(numbytes>1))
        {        printf("Enter 2nd byte position (between 0 and 199): "); }
        if((i==1)&&(numbytes>2))
        {        printf("Enter 3rd byte position (between 0 and 199): "); }
        if((i>1)&&(i+1<numbytes))
        {       printf("Enter %dth byte position (between 0 and 199): ", i+

/* To write data by x^54 in new form back to file */
file_datapoly = fopen("DataPoly", "w");
for(j=0;j<M;j++)
{        fprintf(file_datapoly, "%d\n", datapoly[j]); }
fclose(file_datapoly);
/********************** FORMING A CODEWORD ******************************/
/* Codeword has first 54 slots of array containing check digits from enrolm
and remaining 200 containing Data file. Values go from lowest powers of x *

file_code = fopen("CodeWord", "w");
for(l=0;l<54;l++)
{        fprintf(file_code, "%d\n", enrol[l]); }
for(l=54;l<PP-2;l++)
{        fprintf(file_code, "%d\n", datapoly[l]); }
fclose(file_code);
************* BEGINNING CORRECTIONS - STAGE 1 - CALCULATE f_j ***********
/* Use erasure locators X_k from store[] and calculate f_i, where f(X_k)=0.
Can solve up to 20 erasures - any more than this causes program to terminat
Determine f_i by repeatedly multiplying (x-X_k); obtain f_i from final poly

for(j=0; j<N; j++)
{       res[j]=255;
        prod[j]=255; }
power[1]=255;

/* k is counter of erasure values, so k-1 is highest power of x in store[]
```

```
k=s;

/*m=num of vals in current poly product, so m-1 is highest power of x in this*/
m = k+1;

/* temp5=2 as only have (x-X_k) in each new poly to be multiplied each time */
temp5 = 2;
count = k-1;
n=1;
prod[0] = store[0];

/* Multiplying polys - only enter loop if sum of power of x's (j+1) equals
current x-value in poly product (i) */
while(count > 0)
{       /*To ensure right val multiplied, put val from store[] into power[]*/
        power[0]=store[n];

        for(i=0;i<temp5;i++)
        { for(j=0;j<k;j++)
          {       for(l=0;l<2;l++)
                  {       if(j+l == i)
                          {       /* add powers of alpha mod 255 */
                                temp3 = (prod[j]+power[l])%255;

                                /* In case mod turns temp to 0 */
                                if(temp3 == 0)
                                {       temp3 = 255; }
                                if((power[l] == 0)||(prod[j] == 0))
                                {       temp3 = 0; }

                                /* res[i] is poly product array */
                                if(res[i] == 255)
                                {       res[i] = temp3; }
                                /* XOR hex values corresponding to current value
                                stored in res[i] and temp3; convert back to
                                power of alpha to store in res[i] */
                                else
                                {       /* If values are equal, XOR to zero */
                                        if(lut[1][res[i]] == lut[1][temp3])
                                        {       res[i]=0; }
                                        else
                                        { temp4 = lut[1][res[i]]^lut[1][temp3];
                                          res[i] = lut2[1][temp4];}}}}}}
        /* To put result back into prod array and restart process */
        for(j=0;j<N;j++)
        {       prod[j]=res[j];
                res[j]=255; }
        n++;
```

```
        k++;
        temp5++;
        count--; }
/* To produce polynomial in desired order */
for(i=0;i<m;i++)
{        finalprod[m-i-1] = prod[i]; }
/*************** STAGE 2 - CALCULATING SYNDROMES ************************/
/* Use Syn = c.HT; HT is tranpose of null matrix (NullTrans) & c is codewor

for(i=0;i<M;i++)
{       for(j=0;j<N-1;j++)
        {        HT[i][j]=0; }}
/* To read in codeword into file (in reverse order) */
file_code = fopen("CodeWord", "r");
i=253;
while(fscanf(file_code, "%d", &code[i]) != EOF)
{        i--; }
fclose(file_code);

/* To read in the transpose of the null matrix */
file_nullt = fopen("NullTrans", "r");
for(i=0;i<M;i++)
{       for(j=0;j<N-1;j++)
        { fscanf(file_nullt, "%d", &HT[i][j]); }}
fclose(file_nullt);

for(i=0;i<N-1;i++)
{        Syn[i]=0; }
for(i=0;i<N-1;i++)
{       for(j=0;j<M;j++)
        {       /* If codeword val is zero, multiplying by H^T is still zer
                if(code[j]==0)
                {        temp3=0; }
                else{   /* To multiply the codeword and HT */
                        temp3 = (code[j]+HT[j][i])%255;

                        /* In case temp3 is set to zero by mod 255 */
                        if(temp3 == 0)
                        {        temp3 = 255; }}
                /* value of temp3 XORed with syn and new value put in */
                temp4 = lut[1][temp3]^lut[1][Syn[i]];
            Syn[i] = lut2[1][temp4];}}
/* To write syndromes to a file */
file_syn = fopen("Syn", "w");
for(i=0;i<N-1;i++)
{        fprintf(file_syn, "%d\n", Syn[i]); }
fclose(file_syn);
/******************** STAGE 3 - CALCULATING T_i ************************
```

```
/*Calculate val of T_i, sum of products of synromes with prod array calculated*/
for(i=0;i<N;i++)
{       T[i]=0; }
/* check sees if any errors present; if not, next section is skipped */
check = 0;

/* To calculate T_i */
for(i=0;i<N-s-1;i++)
{       for(j=0;j<=s;j++)
        {       temp3 = (Syn[s+i-j]+finalprod[j])%255;
                if(temp3 == 0)
                {       temp3 = 255; }
                if((finalprod[j] == 0)||(Syn[s+i-j] == 0))
                {       temp3 = 0; }
                temp4 = lut[1][temp3]^lut[1][T[i]];
                T[i] = lut2[1][temp4]; }
        if(T[i] != 0)
        {       check++; }}
if(check == 0)
{       printf("NOTE: There are no errors to be corrected\n\n"); }
/************ STAGE 4 - FINDING p_i USING MASSEY-BERLEKAMP ***********/
/* Only enter this section if check is non-zero i.e. there are errors.
Determine vals of p_i, which ultimately solve locations of errors.
Use Massey-Berlekamp on the equation T_m+r + p_1 T_m+r-1 + ... + p_r T_m = 0.*/

if(check != 0)
{ for(i=0;i<N;i++)
 {       arr[i]=0;
        g[i]=0;
        cor[i]=0;
        gstar[i]=0; }
n=1;   r=0;
g[0]=255;
cor[1]=255;

while((((2*r)+s)<=54)&&(n<=N-s-1))
{       /* Evaluating the prediction error; if r=0, this is T_(n-1) */
        if(r==0)
        {       e = T[n-1]; }
        else{   e = T[n-1];
                for(i=1;i<=r;i++)
                {       /* If T=0, product=0, so no need to enter loop */
                        if((T[n-i-1] != 0)&&(g[i] != 0))
                        {       /*g[i] contains p(i) value needed for formula */
                                temp3 = (g[i]+T[n-1-i])%255;

                                /* In case mod 255 has changed temp3 to zero */
                                if(temp3 == 0)
```

```
                                {       temp3 = 255; }
                                if((T[n-i-1] == 0)||(g[i] == 0))
                                {       temp3 = 0; }
                                /* Add new quantity to error */
                                temp4 = lut[1][temp3]^lut[1][e];
                                e = lut2[1][temp4]; }}}
        if(e != 0)
        {       for(i=0;i<N;i++)
                {       /* If c(x)=0 then ec(x)=0 and g^*(x)=g(x) */
                        if(cor[i]==0)
                        {        gstar[i]=g[i]; }
                        else{   temp3 = (e+cor[i])%255;
                                /* In case mod 255 has changed temp3 to zero
                                if(temp3 == 0)
                                {       temp3 = 255; }
                                /* Add new quantity to gstar */
                                temp4 = lut[1][temp3]^lut[1][g[i]];
                                gstar[i] = lut2[1][temp4]; }}
                if((2*r)<n)
                {       /* To divide g(x) by e, multiply by inverse of e*/
                        temp5 = (255-e)%255;
                        r = n-r;
                        for(i=0; i<N; i++)
                        {       if(g[i] != 0)
                                {       cor[i]=(g[i]+temp5)%255;
                                        if(cor[i] == 0)
                                        {       cor[i] = 255; }}
                                /* If g(x)=0, then c(x)=0 */
                                else{    cor[i]=0; }}}
                for(i=0;i<N;i++)
                {        g[i]=gstar[i]; }}
        /* To multiply c(x) by x */
        for(i=0;i<N;i++)
        {        arr[i] = cor[i]; }
        cor[0]=0;
        for(i=1;i<N;i++)
        {        cor[i] = arr[i-1]; }
        n++; }
printf("\n");

/* To print value of r to file, where r = number of errors */
file_r = fopen("RVal", "w");
fprintf(file_r, "%d", r);
fclose(file_r);

/* To write values of p[i] to file */
file_p = fopen("PVals", "w");
for(i=0;i<N;i++)
```

```
{       fprintf(file_p, "%d\n", g[i]); }
fclose(file_p);
/************* STAGE 5 - SOLVING p(x) TO FIND ERROR LOCATORS **********/
for(i=0;i<N-1;i++)
{       roots[i]=0; }

/* To guess roots, try diff vals in succession; if val gives zero, is a root*/
temp3 = 0;
for(i=0;i<=r;i++)
{       temp4 = lut[1][temp3]^lut[1][g[i]];
       temp3 = lut2[1][temp4]; }
j=0;
if(temp3 == 0)
{       roots[j]=0;
       j++; }
for(count=1; count<255; count++)
{       arrr[count]=0;  }

/* Trying different values until find roots...*/
for(count=1;count<255;count++)
{       for(i=0;i<=r;i++)
        {      temp5 = (count*i)%255;
               temp6 = (temp5+g[r-i])%255;
               if(temp6 == 0)
               {       temp6 = 255; }
               if(g[r-i] == 0)
               {       temp6 = 0; }
               temp4 = lut[1][arrr[count]]^lut[1][temp6];
               arrr[count] = lut2[1][temp4]; }
       if(arrr[count] == 0)
       {       roots[j]=count;
               j++; }}
/* k represents the number of roots found */
k=j;

/* To write roots X_k to a file */
file_x = fopen("XVals", "w");
for(j=0;j<k;j++)
{       fprintf(file_x, "%d\n", roots[j]); }
fclose(file_x);
/************* STAGE 6 - DETERMINE E_k USING GAUSSIAN ELIMINATION ********/
/* Want to solve the equation (X_j^i)E_j = T_i+1 */
for(i=0;i<R;i++)
{       E[i]=0;
       for(j=0;j<R;j++)
       {       X[i][j]=0; }}
/* Let XMatrix be successive rows of roots, raised to greater power each time*/
for(i=0;i<k;i++)
```

```
{       for(j=0;j<k;j++)
        {       X[i][j]=(roots[j]*(i+1))%255;
                /* Partial pivoting doesn't work if 0 is included */
                if(X[i][j] == 0)
                {       X[i][j] = 255; }}}
/* Gaussian Elimination */
for(i=0; i<k-1; i++)
{       /* Find pivot for partial pivoting; find max val of first col of X[]
        and set this to be first row of matrix */
        max = i;
        for(j=i+1; j<k; j++)
        {       if(X[j][i] > X[max][i])
                {       max = j; }}
        /* Swap rows */
        T_temp = T[i+1];
        T[i+1] = T[max+1];
        T[max+1] = T_temp;

        for(d=i; d<k; d++)
        {       temp3 = X[i][d];
                X[i][d] = X[max][d];
                X[max][d] = temp3; }
        for(j=i+1; j<k; j++)
        {       /* Elimination - Need to divide X[j][i] by X[i][i] */
                temp2 = (255 - X[i][i])%255;
                alpha = (X[j][i]+temp2)%255;
                if(alpha == 0)
                {       alpha = 255; }
                /* To multiply alpha by T[i+1] */
                temp4 = (alpha+T[i+1])%255;
                if(temp4 == 0)
                {       temp4 = 255; }
                if(T[i+1] == 0)
                {       temp4 = 0; }
                /* To add temp5 to T[j+1] */
                temp5 = lut[1][T[j+1]]^lut[1][temp4];
                T[j+1] = lut2[1][temp5];

                /* To add alpha*X[i][d] to X[j][d] */
                for(d=i; d<k; d++)
                {       temp6 = (alpha+X[i][d])%255;
                        if(temp6 == 0)
                        {       temp6 = 255; }
                        if(X[i][d] == 0)
                        {       temp6 = 0; }
                        temp7 = lut[1][X[j][d]]^lut[1][temp6];
                        X[j][d] = lut2[1][temp7]; }}}
/* Start the backsubstitution */
```

```
for(i=k-1; i>=0; i--)
{       sum = 0;
        for(j=i+1; j<k; j++)
        {       temp3 = (X[i][j]+E[j])%255;
                if(temp3 == 0)
                {       temp3 = 255; }
                if((X[i][j] == 0)||(E[j] == 0))
                {       temp3 = 0; }
                temp4 = lut[1][sum]^lut[1][temp3];
                sum = lut2[1][temp4]; }
        temp5 = lut[1][T[i+1]]^lut[1][sum];
        temp6 = lut2[1][temp5];
        E[i] = (temp6+(255-X[i][i]))%255; }
/* To write the E[] to a file */
file_y = fopen("EVals", "w");
for(i=0; i<k; i++)
{        fprintf(file_y, "%d\n", E[i]); }
fclose(file_y);
/***************** STAGE 7 - FIND THE ERROR VALUES Y_k ********************/
/* To find the error values Y_k from E_k = Y_k f(X^_k) */
for(i=0; i<N; i++)
{        f[i] = 0; }

for(i=0;i<k;i++)
{       for(j=0;j<=s;j++)
        {       temp3 = (finalprod[s-j]+(roots[i]*j))%255;
                if(temp3 == 0)
                {       temp3 = 255; }
                if(finalprod[s-j] == 0)
                {       temp3 = 0; }
                temp4 = lut[1][temp3]^lut[1][f[i]];
                f[i] = lut2[1][temp4]; }
        /* To divide by f[i], multiply by its inverse */
        temp5 = (255-f[i])%255;
        Y[i] = (E[i]+temp5)%255; }
/***** STAGE 8 - FIND THE ERASURE VALUES USING GAUSSIAN ELIMINATION ********/
/* Solve the equation S_k = D_iX_i^k + Y_jX^_j^k */
for(i=0;i<N;i++)
{        Mat[i]=0; }
/* First calculate matrix S_k+Y_jX^_j^k */
for(n=1;n<=s;n++)
{       temp5 = 0;
        for(j=0;j<k;j++)
        {       temp3 = (Y[j]+(roots[j]*n))%255;
                if(temp3 == 0)
                {       temp3 = 255; }
                if(Y[j] == 0)
                {       temp3 = 0; }
```

```
                temp4 = lut[1][temp3]^lut[1][temp5];
                temp5 = lut2[1][temp4]; }
        temp6 = lut[1][temp5]^lut[1][Syn[n]];
        Mat[n] = lut2[1][temp6]; }}
/* In case there are no errors */
if(check == 0)
{       for(i=0;i<N;i++)
        {        Mat[i] = Syn[i]; }}
for(i=0;i<R;i++)
{       D[i]=0;
        for(j=0;j<R;j++)
        {        X[i][j]=0; }}
/* Need to create XMatrix from roots */
for(i=0;i<s;i++)
{       for(j=0;j<s;j++)
        {       X[i][j]=(store[j]*(i+1))%255;
                /* If X[][] = 0, partial pivoting won't work */
                if(X[i][j] == 0)
                {        X[i][j] = 255; }}}
/* Gaussian Elimination */
for(i=0; i<s-1; i++)
{       /* Find a pivot, for partial pivoting */
        max = i;
        for(j=i+1; j<s; j++)
        {       if(X[j][i] > X[max][i])
                {        max = j; }}
        /* Swap rows */
        Mat_temp = Mat[i+1];
        Mat[i+1] = Mat[max+1];
        Mat[max+1] = Mat_temp;

        for(d=i; d<s; d++)
        {       temp3 = X[i][d];
                X[i][d] = X[max][d];
                X[max][d] = temp3; }
        for(j=i+1; j<s; j++)
        {       /* Elimination - Need to divide X[j][i] by X[i][i] */
                temp2 = (255 - X[i][i])%255;
                alpha = (X[j][i]+temp2)%255;
                if(alpha == 0)
                {        alpha = 255; }
                /* To multiply alpha by Mat[i+1] */
                temp4 = (alpha+Mat[i+1])%255;
                if(temp4 == 0)
                {        temp4 = 255; }
                if((alpha == 0)||(Mat[i+1] == 0))
                {        temp4 = 0; }
                /* To add temp4 to Mat[j+1] */
```

```
                temp5 = lut[1][Mat[j+1]]^lut[1][temp4];
                Mat[j+1] = lut2[1][temp5];

                /* To add alpha*X[i][d] to X[j][d] */
                for(d=i; d<s; d++)
                {       temp6 = (alpha+X[i][d])%255;
                        if(temp6 == 0)
                        {       temp6 = 255; }
                        if((alpha == 0)||(X[i][d] == 0))
                        {       temp6 = 0; }
                        temp7 = lut[1][X[j][d]]^lut[1][temp6];
                        X[j][d] = lut2[1][temp7];}}}
/* Start the backsubstitution */
for(i=s-1; i>=0; i--)
{       sum = 0;
        for(j=i+1; j<s; j++)
        {       temp3 = (X[i][j]+D[j])%255;
                if(temp3 == 0)
                {       temp3 = 255; }
                if((X[i][j] == 0)||(D[j] == 0))
                {       temp3 = 0; }
                temp4 = lut[1][sum]^lut[1][temp3];
                sum = lut2[1][temp4]; }
        temp5 = lut[1][Mat[i+1]]^lut[1][sum];
        temp6 = lut2[1][temp5];
        /* In case erasure noted falsely - ie no actual error */
        if(temp6 == 0)
        {       D[i] = 0; }
        else{   D[i] = (temp6+(255-X[i][i]))%255; }}
file_y = fopen("DVals", "w");
for(i=0; i<s; i++)
{       fprintf(file_y, "%d\n", D[i]); }
fclose(file_y);

for(i=0; i<R; i++)
{       correr[i] = 0;
        correras[i] = 0; }
/******************** CORRECTING ERASURES ********************************/
printf("The number of erasures located is %d\n", s);
for(i=0;i<s;i++)
{       temp3 = code[M-1-store[i]];
        temp4 = lut[1][D[i]]^lut[1][temp3];
        correras[i] = lut2[1][temp4];
        if(D[i] == 0)
        {       printf("No correction needed for erasure located in ");
                printf("position %d\n", store[i]-54); }
        else{   printf("Erasure corrected in position %d ", store[i]-54);
                printf("to %d\n", correras[i]); }}
```

```
/********************** CORRECTING ERRORS ********************************/
printf("\nThe number of errors located is %d\n", r);
if(check != 0)
{       for(i=0;i<k;i++)
        {       temp3 = code[M-1-roots[i]];
                temp4 = lut[1][Y[i]]^lut[1][temp3];
                correr[i] = lut2[1][temp4];
                printf("Error corrected in position %d ", roots[i]-54);
                printf("to %d\n", correr[i]); }}
printf("\n");
/********************** RESTORING THE CODEWORD ****************************/
j=0;    l=0;
for(i=0;i<M;i++)
{        /* Correcting the erasures...*/
         for(n=0; n<s; n++)
         {      if(i == store[n])
                {       code[M-1-i] = correras[l];
                        l++;}}
         /* Correcting the errors...*/
         for(n=0; n<k; n++)
         {      if(i == roots[n])
                {       code[M-1-i] = correr[j];
                        j++;}}}
for(i=0; i<P; i++)
{        newc[i] = 0; }
/* To write the restored data (minus check digits) to an array */
j=54;
for(i=0; i<P; i++)
{       newc[i] = code[M-1-j];
        j++; }
/***************** HASHING DATA AND COMPARING ********************/
/* Use GNU's Libgcrypt functions to do this */
H = gcry_md_open(GCRY_MD_SHA1,0);
gcry_md_write(H, &newc, 200);
results = gcry_md_read(H, 0);
gcry_md_close(H);

/* Compare hash just formed in results[] with stored hash in hashchar[] */
j=0;
for(i=0; i<20; i++)
{       if(results[i] != hashchar[i])
        {        j++; }}
if(j == 0)
{       printf("Verification was a success!\n");
        printf("You have been recognised as %s\n", id); }
else{   printf("Verification has failed.\nThe system does not recognise you'
        printf(" as %s\nIf this is an error, please try again.\n", id); }
exit(0); }
```

Appendix G

Walk-through of Iris Authentication Program

G.1 Enrolment Program

```
>Enrol
Please enter your name: Eabhnat
Welcome Eabhnat!
This is a program to simulate enrolment in a system which identifies
people by their iris. We use a string of data to represent the iris.
Choose one of the following data-strings: Iris1, Iris2, Iris3, Iris4
>Iris1
The basic iris file that will be used is Iris1
This file has been successfully read in
There are 512 hex digits in the file
This means we have 256 numbers in total

Now you must select certain bytes to be excluded from your reading.
These represent areas obscured by eyelashes, eyelids etc.
Choose 56 bytes in total to be masked out.
Enter any number from 0-255, but do not enter the same number twice.

Byte 1 located at position: 0      Byte 13 located at position: 34
Byte 2 located at position: 1      Byte 14 located at position: 45
Byte 3 located at position: 2      Byte 15 located at position: 56
Byte 4 located at position: 3      Byte 16 located at position: 67
Byte 5 located at position: 4      Byte 17 located at position: 78
Byte 6 located at position: 5      Byte 18 located at position: 89
Byte 7 located at position: 6      Byte 19 located at position: 90
Byte 8 located at position: 7      Byte 20 located at position: 98
Byte 9 located at position: 8      Byte 21 located at position: 87
Byte 10 located at position: 9     Byte 22 located at position: 76
Byte 11 located at position: 12    Byte 23 located at position: 65
Byte 12 located at position: 23    Byte 24 located at position: 54
```

```
Byte 25 located at position: 43    Byte 44 located at position: 25
Byte 26 located at position: 32    Byte 45 located at position: 36
Byte 27 located at position: 21    Byte 46 located at position: 47
Byte 28 located at position: 13    Byte 47 located at position: 58
Byte 29 located at position: 24    Byte 48 located at position: 69
Byte 30 located at position: 35    Byte 49 located at position: 70
Byte 31 located at position: 46    Byte 50 located at position: 96
Byte 32 located at position: 57    Byte 51 located at position: 85
Byte 33 located at position: 68    Byte 52 located at position: 74
Byte 34 located at position: 79    Byte 53 located at position: 63
Byte 35 located at position: 80    Byte 54 located at position: 52
Byte 36 located at position: 97    Byte 55 located at position: 41
Byte 37 located at position: 86    Byte 56 located at position: 41
Byte 38 located at position: 75    ERROR: Number has been entered previously
Byte 39 located at position: 64    Please try again.
Byte 40 located at position: 53    Byte 56 located at position: 321
Byte 41 located at position: 42    ERROR: Number must be less than 256.
Byte 42 located at position: 31    Please try again.
Byte 43 located at position: 14    Byte 56 located at position: 121

56 bytes chosen and masked out from calculations.
Now you may choose to modify some further bytes in your  file.
Enter the number of bytes (at most 8) you wish to modify: 8
Enter 1st byte position (between 0 and 199): 3
Current value of byte 3 is 69.
Enter new value for byte 3 (between 0 and 255): 35
Value for byte 3 has been changed to 35.

Enter 2nd byte position (between 0 and 199): 5
Current value of byte 5 is 113.
Enter new value for byte 5 (between 0 and 255): 57
Value for byte 5 has been changed to 57.

Enter 3rd byte position (between 0 and 199): 5
ERROR: This byte has already been changed. Please choose another.
Enter 3rd byte position (between 0 and 199): 7
Current value of byte 7 is 68.
Enter new value for byte 7 (between 0 and 255): 79
Value for byte 7 has been changed to 79.

Enter 4th byte position (between 0 and 199): 9
Current value of byte 9 is 114.
Enter new value for byte 9 (between 0 and 255): 97
Value for byte 9 has been changed to 97.

Enter 5th byte position (between 0 and 199): 211
ERROR: Number must be less than 200. Please try again.
Enter 5th byte position (between 0 and 199): 12
```

```
Current value of byte 12 is 33.
Enter new value for byte 12 (between 0 and 255): 23
Value for byte 12 has been changed to 23.

Enter 6th byte position (between 0 and 199): 34
Current value of byte 34 is 0.
Enter new value for byte 34 (between 0 and 255): 45
Value for byte 34 has been changed to 45.

Enter 7th byte position (between 0 and 199): 58
Current value of byte 58 is 249.
Enter new value for byte 58 (between 0 and 255): 69
Value for byte 58 has been changed to 69.

Enter 8th byte position (between 0 and 199): 70
Current value of byte 70 is 38.
Enter new value for byte 70 (between 0 and 255): 80
Value for byte 70 has been changed to 80.

Filename has been written to your file.
Check digits have been written to your file.
Masked bit positions have been written to your file.
Hashed data bits have been written to your file.
You have been successfully enrolled! See you next time, Eabhnat!
```

File "Eabhnat" contains:

```
Iris1   146     116     240     1       67      35      25      46      2e
235     32      194     89      2       78      46      36      fc      5a
31      227     118     200     3       89      57      47      50      4c
32      187     16      158     4       90      68      58      fa      de
105     114     155     95      5       98      79      69      ae      fc
109     175     237     117     6       87      80      70      d5
233     33      191     1       7       76      97      96      36
65      112     104     4       8       65      86      85      d6
12      195     228     118     9       54      75      74      b5
181     116     254     29      12      43      64      63      48
184     172     172     142     23      32      53      52      2a
209     13      128     21      34      21      42      41      c5
231     106     142     47      45      13      31      121     5
196     209     183     0       56      24      14      93      c
```

- The first entry, "Iris1", refers to the iris filename chosen.
- The next 54 $(235, \ldots, 47)$ are check digits calculated on enrolment.
- Then, there are 55 byte positions $(0, \ldots, 121)$ the user chose to omit.
- The final 20, in hex rather than decimal, are the hashed iris data.

G.2 Verification Program

```
>Verify
Please enter your name: Eabhnat
Welcome Eabhnat!
This is a program to simulate verifying that a person is who they claim to be
based on a comparison between the iris reading they provide now, and the one
previously stored for them at enrolment. Your chosen iris filename was Iris1.
There are 512 digits in the file. This means we have 256 numbers in total
The 56 bytes you chose to exclude during enrolment are once again masked out.

Now you may choose to have some bytes treated as erasures (errors whose
location is known). Enter the number of erasures (no more than 20): 20
You have chosen to include 20 erasures.

Enter 1st byte position (between 0 and 199): 4
Byte 4 is an erasure.
Enter 2nd byte position (between 0 and 199): 5
Byte 5 is an erasure.
Enter 3rd byte position (between 0 and 199): 7
Byte 7 is an erasure.
Enter 4th byte position (between 0 and 199): 8
Byte 8 is an erasure.
Enter 5th byte position (between 0 and 199): 13
Byte 13 is an erasure.
Enter 6th byte position (between 0 and 199): 24
Byte 24 is an erasure.
Enter 7th byte position (between 0 and 199): 35
Byte 35 is an erasure.
Enter 8th byte position (between 0 and 199): 46
Byte 46 is an erasure.
Enter 9th byte position (between 0 and 199): 57
Byte 57 is an erasure.
Enter 10th byte position (between 0 and 199): 68
Byte 68 is an erasure.
Enter 11th byte position (between 0 and 199): 79
Byte 79 is an erasure.
Enter 12th byte position (between 0 and 199): 80
Byte 80 is an erasure.
Enter 13th byte position (between 0 and 199): 88
Byte 88 is an erasure.
Enter 14th byte position (between 0 and 199): 90
Byte 90 is an erasure.
Enter 15th byte position (between 0 and 199): 99
Byte 99 is an erasure.
Enter 16th byte position (between 0 and 199): 100
Byte 100 is an erasure.
Enter 17th byte position (between 0 and 199): 101
```

```
Byte 101 is an erasure.
Enter 18th byte position (between 0 and 199): 112
Byte 112 is an erasure.
Enter 19th byte position (between 0 and 199): 115
Byte 115 is an erasure.
Enter 20th byte position (between 0 and 199): 150
Byte 150 is an erasure.

Now you may choose to modify some further bytes
Enter the number of bytes (at most 9) you wish to modify: 9

Enter 1st byte position (between 0 and 199): 10
Original value of byte 10 is 197.
Enter new value for byte 10 (between 0 and 255): 20
Value for byte 10 has been changed to 20.

Enter 2nd byte position (between 0 and 199): 20
Original value of byte 20 is 201.
Enter new value for byte 20 (between 0 and 255): 30
Value for byte 20 has been changed to 30.

Enter 3rd byte position (between 0 and 199): 30
Original value of byte 30 is 116.
Enter new value for byte 30 (between 0 and 255): 40
Value for byte 30 has been changed to 40.

Enter 4th byte position (between 0 and 199): 40
Original value of byte 40 is 79.
Enter new value for byte 40 (between 0 and 255): 50
Value for byte 40 has been changed to 50.

Enter 5th byte position (between 0 and 199): 50
Original value of byte 50 is 240.
Enter new value for byte 50 (between 0 and 255): 60
Value for byte 50 has been changed to 60.

Enter 6th byte position (between 0 and 199): 60
Original value of byte 60 is 40.
Enter new value for byte 60 (between 0 and 255): 70
Value for byte 60 has been changed to 70.

Enter 7th byte position (between 0 and 199): 70
Original value of byte 70 is 38.
Enter new value for byte 70 (between 0 and 255): 80
Value for byte 70 has been changed to 80.

Enter 8th byte position (between 0 and 199): 80
Original value of byte 80 is 203.
```

```
Enter new value for byte 80 (between 0 and 255): 90
Value for byte 80 has been changed to 90.

Enter 9th byte position (between 0 and 199): 90
Original value of byte 90 is 113.
Enter new value for byte 90 (between 0 and 255): 100
Value for byte 90 has been changed to 100.
(Hit RETURN to move on to next step throughout program)

STAGE 1:
Calculate f_j
from erasure
locators X_j.
f[0] is 66
f[1] is 24
f[2] is 113
f[3] is 231
f[4] is 227
f[5] is 0
f[6] is 235
f[7] is 246
f[8] is 255
f[9] is 135
f[10] is 22
f[11] is 41
f[12] is 130
f[13] is 179
f[14] is 139
f[15] is 134
f[16] is 23
f[17] is 161
f[18] is 60
f[19] is 198
f[20] is 255

STAGE 2:
Determine
syndromes S_k.
Syn[0] is 179
Syn[1] is 220
Syn[2] is 120
Syn[3] is 163
Syn[4] is 130
Syn[5] is 103
Syn[6] is 255
Syn[7] is 111
Syn[8] is 23
Syn[9] is 208
Syn[10] is 97
Syn[11] is 239
Syn[12] is 129
Syn[13] is 52
Syn[14] is 173
Syn[15] is 2
Syn[16] is 49
Syn[17] is 245
Syn[18] is 186
Syn[19] is 122
Syn[20] is 217
Syn[21] is 163
Syn[22] is 228
Syn[23] is 147
Syn[24] is 243
Syn[25] is 119
Syn[26] is 181
Syn[27] is 155
Syn[28] is 157
Syn[29] is 165
Syn[30] is 77
Syn[31] is 0
Syn[32] is 187
Syn[33] is 223
Syn[34] is 196
Syn[35] is 142
Syn[36] is 249
Syn[37] is 150
Syn[38] is 67
Syn[39] is 131
Syn[40] is 177
Syn[41] is 234
Syn[42] is 142
Syn[43] is 216
Syn[44] is 246
Syn[45] is 182
Syn[46] is 18
Syn[47] is 70
Syn[48] is 241
Syn[49] is 112
Syn[50] is 98
Syn[51] is 173
Syn[52] is 202
Syn[53] is 74

STAGE 3:
Calculate
T_i using S_k
and f_j.
T[0] is 111
T[1] is 236
T[2] is 74
T[3] is 89
T[4] is 107
T[5] is 154
T[6] is 172
T[7] is 209
T[8] is 126
T[9] is 133
T[10] is 179
T[11] is 101
T[12] is 56
T[13] is 74
T[14] is 229
T[15] is 168
T[16] is 22
T[17] is 185
T[18] is 248
T[19] is 50
T[20] is 209
T[21] is 140
T[22] is 210
T[23] is 135
T[24] is 119
T[25] is 183
T[26] is 168
T[27] is 42
T[28] is 119
T[29] is 21
T[30] is 226
T[31] is 110
T[32] is 238
T[33] is 60

STAGE 4:
Find p_i using
Massey-Berlekamp
and T_i.
p[0] is 255
p[1] is 181
p[2] is 125
p[3] is 106
p[4] is 244
p[5] is 156
p[6] is 6
p[7] is 158
p[8] is 18
p[9] is 51
p[10] is 128
p[11] is 184
p[12] is 186
p[13] is 20

Note: Also know
number of errors
at this point.
There are 13
in codeword.

STAGE 5:
Solve p(x) to
find error
locators Z_k.
Z[0] is 57
Z[1] is 59
Z[2] is 61
Z[3] is 63
```

```
Z[4] is 64
Z[5] is 66
Z[6] is 74
Z[7] is 84
Z[8] is 88
Z[9] is 94
Z[10] is 104
Z[11] is 112
Z[12] is 114

STAGE 6:
Use Gaussian
Elimination
to find E_k.
E[12] is 231
E[11] is 47
E[10] is 210
E[9] is 155
E[8] is 231
E[7] is 185
E[6] is 17
E[5] is 211
E[4] is 147
E[3] is 166
E[2] is 250
E[1] is 213
E[0] is 44

STAGE 7:
Calculate
error values
Y_k from E_k
and X_k.
Y[0] is 188
Y[1] is 166
Y[2] is 158
Y[3] is 46
Y[4] is 172
Y[5] is 146
Y[6] is 216
Y[7] is 1
Y[8] is 45
Y[9] is 59
Y[10] is 82
Y[11] is 91
Y[12] is 208

STAGE 8:
Use Gaussian
Elimination
to calculate
erasure
values D_i.
D[0] is 0
D[1] is 0
D[2] is 0
D[3] is 0
D[4] is 0
D[5] is 0
D[6] is 0
D[7] is 0
D[8] is 0
D[9] is 0
D[10] is 0
D[11] is 80
D[12] is 0
D[13] is 112
D[14] is 0
D[15] is 0
D[16] is 0
D[17] is 0
D[18] is 0
D[19] is 0

Total number of erasures to correct is 20

Current value in pos 4 is 35
Erasure value is 0
Thus, no correction needed for
erasure located in position 4

Current value in pos 5 is 113
Erasure value is 0
Thus, no correction needed for
erasure located in position 5

Current value in pos 7 is 68
Erasure value is 0
Thus, no correction needed for
erasure located in position 7

Current value in pos 8 is 15
Erasure value is 0
Thus, no correction needed for
erasure located in position 8

Current value in pos 13 is 53
Erasure value is 0
Thus, no correction needed for
erasure located in position 13

Current value in pos 24 is 127
Erasure value is 0
Thus, no correction needed for
erasure located in position 24

Current value in pos 35 is 0
Erasure value is 0
Thus, no correction needed for
erasure located in position 35

Current value in pos 46 is 118
Erasure value is 0
Thus, no correction needed for
erasure located in position 46

Current value in pos 57 is 114
Erasure value is 0
Thus, no correction needed for
erasure located in position 57

Current value in pos 68 is 3
Erasure value is 0
Thus, no correction needed for
erasure located in position 68

Current value in pos 79 is 12
Erasure value is 0
```

```
Thus, no correction needed for
erasure located in position 79

Current value in pos 80 is 90
Erasure value is 80
From our Look-Up Tables,
90 corresponds to 72
80 corresponds to e
If we XOR 72 with e we get 7c,
which corresponds to 203
Thus, erasure corrected in
position 80 to 203

Current value in pos 88 is 69
Erasure value is 0
Thus, no correction needed for
erasure located in position 88

Current value in pos 90 is 100
Erasure value is 112
From our Look-Up Tables,
100 corresponds to 9e
112 corresponds to 93
If we XOR 9e with 93 we get d,
which corresponds to 113
Thus, erasure corrected in
position 90 to 113

Current value in pos 99 is 71
Erasure value is 0
Thus, no correction needed for
erasure located in position 99

Current value in pos 100 is 84
Erasure value is 0
Thus, no correction needed for
erasure located in position 100

Current value in pos 101 is 114
Erasure value is 0
Thus, no correction needed for
erasure located in position 101

Current value in pos 112 is 80
Erasure value is 0
Thus, no correction needed for
erasure located in position 112

Current value in pos 115 is 183
Erasure value is 0
Thus, no correction needed for
erasure located in position 115

Current value in pos 150 is 13
Erasure value is 0
Thus, no correction needed for
erasure located in position 150

Total number of errors to correct is 13

Current value in pos 3 is 69
Error value is 188
From our Look-Up Tables,
69 corresponds to 24
188 corresponds to 85
If we XOR 24 with 85, we get a1
which corresponds to 35
Error corrected in pos 3 to 35

Current value in pos 5 is 113
Error value is 166
From our Look-Up Tables,
113 corresponds to d
166 corresponds to 61
If we XOR d with 61, we get 6c
which corresponds to 57
Error corrected in pos 5 to 57

Current value in pos 7 is 68
Error value is 158
From our Look-Up Tables,
68 corresponds to 12
158 corresponds to 15
If we XOR 12 with 15, we get 7
which corresponds to 79
Error corrected in pos 7 to 79

Current value in pos 9 is 114
Error value is 46
From our Look-Up Tables,
114 corresponds to 1a
46 corresponds to 43
```

```
If we XOR 1a with 43, we get 59
which corresponds to 97
Error corrected in pos 9 to 97

Current value in pos 10 is 20
Error value is 172
From our Look-Up Tables,
20 corresponds to 19
172 corresponds to d5
If we XOR 19 with d5, we get cc
which corresponds to 197
Error corrected in pos 10 to 197

Current value in pos 12 is 33
Error value is 146
From our Look-Up Tables,
33 corresponds to f7
146 corresponds to 3f
If we XOR f7 with 3f, we get c8
which corresponds to 23
Error corrected in pos 12 to 23

Current value in pos 20 is 30
Error value is 216
From our Look-Up Tables,
30 corresponds to ae
216 corresponds to b1
If we XOR ae with b1, we get 1f
which corresponds to 201
Error corrected in pos 20 to 201

Current value in pos 30 is 40
Error value is 1
From our Look-Up Tables,
40 corresponds to 6a
1 corresponds to 2
If we XOR 6a with 2, we get 68
which corresponds to 116
Error corrected in pos 30 to 116

Current value in pos 34 is 0
Error value is 45
From our Look-Up Tables,
0 corresponds to 0
45 corresponds to b4
If we XOR 0 with b4, we get b4
which corresponds to 45
Error corrected in pos 34 to 45

Current value in pos 40 is 50
Error value is 59
From our Look-Up Tables,
50 corresponds to 9c
59 corresponds to 9b
If we XOR 9c with 9b, we get 7
which corresponds to 79
Error corrected in pos 40 to 79

Current value in pos 50 is 60
Error value is 82
From our Look-Up Tables,
60 corresponds to 1d
82 corresponds to 38
If we XOR 1d with 38, we get 25
which corresponds to 240
Error corrected in pos 50 to 240

Current value in pos 58 is 249
Error value is 91
From our Look-Up Tables,
249 corresponds to c0
91 corresponds to e4
If we XOR c0 with e4, we get 24
which corresponds to 69
Error corrected in pos 58 to 69

Current value in pos 60 is 70
Error value is 208
From our Look-Up Tables,
70 corresponds to 48
208 corresponds to 22
If we XOR 48 with 22, we get 6a
which corresponds to 40
Error corrected in pos 60 to 40

Hash of verification data is: 9346fc50faaed536d6b5482ac55c2e5a4cdefc
Hash stored on enrolment is: 9346fc50faaed536d6b5482ac55c2e5a4cdefc
Hashes are identical, so verification was a success!
You have been recognised as Eabhnat
```

Bibliography

[Adl65] F.H. Adler. *Physiology of the Eye: Clinical Application.* The C.V. Mosby Company, London, fourth edition, 1965.

[BB98] W.W. Boles and B. Boashash. A human identification technique using images of the iris and wavelet transform. *IEEE Trans. on Signal Processing*, 46(4):1185–1188, April 1998.

[BBB+05] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. *Recommendation for Key Management - Part 1: General.* National Institute of Standards and Technology (NIST), August 2005. Special Publication 800-57.

[BDF98] D. Boneh, G. Durfee, and Y. Frankel. An attack on RSA given a small fraction of the private key bits. In K. Ohta and D. Pei, editors, *Advances in Cryptology - Proc. of Asiacrypt '98*, volume 1514 of *LNCS*, pages 25–34. Springer-Verlag, 1998.

[Ber85] A. Bertillon. La couleur de l'iris. *Revue Scientifique*, 36(3):65–73, 1885.

[Ber05] D.J. Bernstein. Cache-timing attacks on AES, 2005. *http://cr.yp.to/papers.html#cachetiming.*

[BM03] J. Blömer and A. May. New partial key exposure attacks on RSA. In *Advances in Cryptology - Proc. of Crypto '03*, volume 2729 of *LNCS*. Springer-Verlag, 2003.

[Bod94] A. Bodo. Method for producing a digital signature with aid of a biometric feature. German patent DE 4243908 A1, 1994.

[Cam99] J. Cambier. *Iris Recognition and Network Security.* Iridian Technologies, Inc., NJ, USA, Oct 1999. Internal White Paper.

[Car06] CardWerk. CardWerk Smarter Card Solutions, January 2006. *http://www.cardwerk.com/smartcards/smartcard_overview.aspx*.

[CC03] G.M. Clarke and D. Cooke. *A Basic Course in Statistics.* Arnold, Hodder Headline Group, fourth edition, 2003.

[CDH+00] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In B. Preneel, editor, *Advances in Cryptology - Proc. of Eurocrypt '00*, volume 1807 of *LNCS*, pages 453–469. Springer-Verlag, 2000.

[Chi06] Chinese Academy of Sciences - Institute of Automation. CASIA iris image database, 2006. *http://www.sinobiometrics.com/Databases.htm.*

[CJJ+03] N. Courtois, R. Johnson, P. Junod, T. Pornin, and M. Scott. Did Filiol break AES? International Association for Cryptologic Research (IACR) eprint archive, 2003. *http://eprint.iacr.org/2003/022.pdf.*

[CKKL04] H.I. Choi, D. Kim, S.-H. Kwon, and S.J. Lee. Modified variable multi-sector method for iris identification. Biometric Consortium Conference, 2004.

[Coc73] C.C. Cocks. *A Note on Non-Secret Encryption.* Communications-Electronics Security Group, GCHQ, Cheltenham, Gloucestershire, UK, Nov 1973.

[Cop97] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[CP02] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology - Proc. of Asiacrypt '02*, volume 2501 of *LNCS*, pages 267–287. Springer-Verlag, 2002.

[Dao04] Daon, 2004. *http://www.daon.com/products/daonengine.htm.*

[Dao05] Daon, 2005. Homepage: *http://www.daon.com.*

[Dau93] J. Daugman. High confidence visual recognition of persons by a test of statistical independence. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(11):1148–1161, November 1993.

[Dau94] J. Daugman. Biometric personal identification system based on iris analysis. U.S. Patent Number 5291560, March 1994. Patent application: July 1991.

[Dau99] J. Daugman. Biometric decision landscapes. Technical report no. tr482, University of Cambridge Computer Laboratory, 1999.

[Dau00] J. Daugman. Wavelet demodulation codes, statistical independence, and pattern recognition. *Institute of Mathematics and its Applications, Proc. 2nd IMA-IP*, pages 244–260, 2000.

[Dau01] J. Daugman. Relationship between number of iriscode bits compared, and search engine maximum hamming distance tolerance. *http://www.cl.cam.ac.uk/users/jgd1000/numbits.pdf*, March 2001.

[Dau02] J. Daugman, July 2002. Private Communication.

[Dau03] J. Daugman. The importance of being random: statistical principles of iris recognition. *Pattern Recognition*, 36(2):279–291, 2003.

[Dau04] J. Daugman. How iris recognition works. *IEEE Trans. on Circuits and Systems for Video Technology*, 14(1):21–30, Jan 2004.

[Dav90] H. Davson. *Davson's Physiology of the Eye.* Macmillan, London, fifth edition, 1990.

[DeL84] J.M. DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8:253–259, 1984.

[DFM99] G.I. Davida, Y. Frankel, and B.J. Matt. On the relation of error correction and cryptography to an offline biometric based identification scheme. In *Proc. of WCC99.* Workshop on Coding and Cryptography, 1999.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, Nov 1976.

[Dog49] J. Doggart. *Ocular signs in slit-lamp microscopy.* London: Kimpton, 1949.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer-Verlag, 2002.

[DRS04] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - Proc. of EuroCrypt '04*, volume 3027 of *LNCS*, pages 523–540. Springer-Verlag, 2004.

[ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - Proc. of Crypto '84*, volume 196 of *LNCS*, pages 10–18. Springer-Verlag, 1985.

[Ell70] J.H. Ellis. *The Possibility of Secure Non-Secret Digital Encryption.* Communications-Electronics Security Group, GCHQ, Cheltenham, Gloucestershire, UK, January 1970.

[Ens00] Joh. Enschedé, September 2000. Unpublished iris study.

[Eur05] European Network of Excellence in Cryptology ECRYPT. *ECRYPT Yearly Report on Algorithms and Keysizes*, 2005. Edited by C. Gehrmann and M. Näslund, Revision 1.1.

[Fhl05] E. Ní Fhloinn. Biometric identification. *Trinity College Dublin Journal of Postgraduate Research*, 4(1-2):173–183, 2004-2005.

[FP05] E. Ní Fhloinn and M. Purser. Applications of partial hiding in RSA. In C. Wolf, S. Lucks, and P.-W. Yau, editors, *WEWoRC 2005 - Western European Workshop on Research in Cryptology*, volume 74 of *Lecture Notes in Informatics*, pages 86–97, 2005.

[FR93] G.L. Feng and T.R.N. Rao. Decoding algebraic geometric codes up to the designed minimum distance. *IEEE Transactions on Information Theory*, 39(1):37–45, 1993.

[FS87] L. Flom and A. Safir. Iris recognition system. U.S. Patent Number 4641349, February 1987. Patent application: Feb 1985.

[Gal62] R.G. Gallger. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8:21–28, January 1962.

[GNU03] GNU. Libgcrypt 1.1.12. *http://www.gnupg.org*, January 2003. Developed by M. Skala, M. Roth, N. Hernaeus, R. Guyomarch, S. Josefsson, W. Koch.

[HAD05] F. Hao, R. Anderson, and J. Daugman. Combining cryptography with biometrics effectively. Technical report no. tr640, University of Cambridge Computer Laboratory, 2005.

[HAD06] F. Hao, R. Anderson, and J. Daugman. Combining cryptography with biometrics effectively. *IEEE Trans. on Computers*, Aug 2006. To appear.

[Hen01] M. Hendry. *Smart Card Security and Applications.* Artech House, second edition, 2001.

[HHL+00] D.R. Hankerson, D.G. Hoffman, D.A. Leonard, C.C. Lindner, K.T. Phelps, C.A. Rodger, and J.R. Wall. *Coding Theory and Cryptography: The Essentials.* Marcel Dekker, 2nd edition, 2000.

[HKQ00] G. Hachez, F. Koeune, and J.J. Quisquater. Biometrics, access control, smart cards: A not so simple combination. In *Proc. of 4th Working Conference on Smart Card Research and Advanced Applications (CARDIS 2000)*, pages 273–288. Kuwer Academic Publishers, September 2000.

[HLT03] M.J. Hinek, M.K. Low, and E. Teske. On some attacks on multi-prime RSA. In *Selected Areas in Cryptography: 9th Annual International Workshop, 2002*, volume 2595 of *LNCS*, pages 385–404. Springer-Verlag, 2003.

[HMV04] D. Hankerson, A.J. Menezes, and S. Vanstone. *Guide to Elliptic-Curve Cryptography.* Springer, 2004.

[HP03] W.C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, 2003.

[HvLP98] T. Høholdt, J.H. van Lint, and R. Pellikaan. Algebraic geometry codes. In V.S. Pless and W.C Huffman, editors, *Handbook of Applied Cryptography*, volume 1, pages 871–961. Elsevier Science B.V., 1998.

[JHP00] A. Jain, L. Hong, and S. Pankanti. Biometric identification. *Communications of the ACM*, 43(2):91–98, February 2000.

[JMV01] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). White paper, Certicom Corporation, 2001.

[Joh05] R.A. Johnson. *Probability and Statistics for Engineers.* Pearson Prentice Hall, New Jersey, seventh edition, 2005.

[JP99] A. Jain and S. Pankanti. Fingerprint classification and matching. Technical Report MSU-CPS-99-5, Department of Computer Science, Michigan State University, January 1999.

[JW99] A. Juels and M. Wattenberg. A fuzzy commitment scheme. *6th ACM Conference on Computer and Communications Security*, pages 28–36, 1999.

[K̈96] R. Kötter. *On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes.* PhD thesis, Linköping University, Sweden, 1996.

[Kal03] B.S. Kaliski, Jr. TWIRL and RSA key size. Technical note, RSA Laboratories, May 2003. Available on RSA Homepage *http://www.rsasecurity.com/rsalabs/technotes/twirl.html.*

[Kni88] P. Knipschild. Looking for gall bladder disease in the patient's iris. *British Medical Journal*, 297:1578–1581, 1988.

[Knu98] D.E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, third edition, 1998.

[Kob87] N. Koblitz. Elliptic curve cryptosytems. *Mathematics of Computation*, 48(177):203–209, 1987.

[Koh78] L.M. Kohnfelder. Towards a practical public-key cryptosystem. B.S. Thesis, supervised by L. Adleman, May 1978.

[KT84] M. Kawagoe and A. Tojo. Fingerprint pattern classification. *Pattern Recognition*, 17(3):295–303, 1984.

[Lan78] S. Lang. *Elliptic Curves: Diophantine Analysis.* Springer-Verlag, 1978.

[Len87] H.W. Lenstra, Jr. Factoring integers with elliptic curves. *Annual of Mathematics*, 126:649–673, 1987.

[LG91] H.C. Lee and R.E. Gaensslen, editors. *Advances in Fingerprint Technology.* CRC Press, New York: Elsevier, 1991.

[LLBK01] S. Lim, K. Lee, O. Byeon, and T. Kim. Efficient iris recognition through improvement of feature vector and classification. *ETRI Journal*, 23(2):61–70, June 2001.

[LMS+97] M. Luby, M. Mitzenmacher, M.A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. *29th Annual ACM Symposium on Theory of Computing*, pages 150–159, 1997.

[LTK+01] S. Lin, H. Tang, Y. Kou, J. Xu, and K. Abdel-Ghaffar. Codes on finite geometries. *Proc. of 2001 IEEE Information Theory Workshop*, pages 14–16, 2001.

[LV01] A.K. Lenstra and E.R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[LW01] S. Liu and M. Wattenberg. A practical guide to biometric security technology. *IEEE IT Professional*, 3(1), January 2001.

[Mac99] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. on Information Theory*, 45(2):399–431, March 1999.

[Mas69] J.L Massey. Shift-register synthesis and BCH recoding. *IEEE Trans. on Information Theory*, IT-15:122–127, January 1969.

[Mas03] L. Masek. Recognition of human iris patterns for biometric identification. The University of Western Australia, *http://www.csse.uwa.edu.au/ pk/studentprojects/libor/*, 2003.

[MB03] U. Martini and S. Beinlich. Virtual PIN: Biometric encryption using coding theory. In A. Brömme and C. Busch, editors, *BIOSIG 2003, Proc. of 1st Conference on Biometrics and Electronic Signatures of GI Working Group BIOSIG*, volume 31 of *LNI*, pages 91–99. GI, 2003.

[Mic06] ST Microelectronics, 2006. Homepage: *http://www.st.com.*

[Mil86] V.S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology - Proc. of Crypto '85*, volume 218 of *LNCS*, pages 417–426. Springer-Verlag, 1986.

[MKCK01] T. Mansfield, G. Kelly, D. Chandler, and J. Kane. *Biometric Product Testing Final Report.* National Physical Laboratory, U.K., March 2001. CESG contract X92A/4009309 *http://www.cesg.gov.uk/technology/ biometrics.*

[MMYH02] T. Matsumoto, H. Matsumoto, K. Yamada, and S. Hoshino. Impact of artificial "gummy" fingers on fingerprint systems. In *Proceedings of SPIE, Optical Security and Counterfeit Deterrence Techniques IV*, volume 4677, January 2002.

[MN96] D.J.C. MacKay and R.M Neal. Near Shannon limit performance of low density parity check codes. *IEEE Electronics Letters*, 32(18):1645–1655, August 1996.

[MR99] T. Mansfield and G. Roethenbaugh. 1999 glossary of biometric terms, 1999. International Association for Biomet-

rics (iAfB) and International Computer Security Association (ICSA). *http://www.iafb.org.uk/DOC/Glossary_Terms.htm.*

[MR03] D.J.C. MacKay and E.A. Ratzer. Gallager codes for high rate applications, 2003. Research note. *http://www.inference.phy.cam.ac.uk/mackay/CodesRegular.html.*

[MTWZ04] L. Ma, T. Tan, Y. Wang, and D. Zhang. Efficient iris recognition by characterising key local variations. *IEEE Trans. on Image Processing*, 13(6):739–750, 2004.

[Mus01] C. Musili. *Algebraic Geometry for Beginners.* Texts and Readings in Mathematics 20. Hindustan Book Agency, 2001.

[MvOV97] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[Nat93] National Bureau of Standards. *FIPS Publication 46-2: Data Encryption Standard (DES)*, December 1993. Supersedes original FIPS Publication 46, 1977.

[Nat00] National Institute of Standards and Technology (NIST). *FIPS Publication 186-2: Digital Signature Standard*, January 2000. First Edition May 1994.

[NBPK05] S. Noh, K. Bae, K. R. Park, and J. Kim. A new feature extraction method using the ICA filters for iris recognition system. In S. Z. Li, Z. Sun, T. Tan, S. Pankanti, G. Chollet, and D. Zhang, editors, *Advances in Biometric Person Authentication: International Workshop on Biometric Recognition Systems, IWBRS*, volume 3781 of *LNCS*, pages 142–149. Springer-Verlag, 2005.

[NSA03] Committee on National Security Systems National Security Agency. National policy on the use of the advanced encryption standard (AES) to protect national security systems and national security information. CNSS Policy No. 15, Fact Sheet 1, June 2003.

[NZM91] I. Niven, H.S. Zuckerman, and H.L Montgomery. *An introduction to the theory of numbers.* John Wiley and Sons, 5th edition, 1991.

[O'S00] M.E. O'Sullivan. Decoding of hermitian codes: The key equation and efficient error evaluation. *IEEE Transactions on Information Theory*, 46(2):512–523, 2000.

[PA05] H. Proena and L. A. Alexandre. Ubiris: A noisy iris image database. In *Proc. of ICIAP 2005 - International Conference on Image Analysis and Processing*, volume 1, pages 970–977, 2005.

[Pol93] J.M. Pollard. Factoring with cubic integers. In A.K. Lenstra and H. W. Lenstra Jr., editors, *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 4–10. Springer-Verlag, 1993.

[Pom85] C. Pomerance. The quadratic sieve factoring algorithm. In *Advances in Cryptology - Proc. of Eurocrypt '84*, volume 209 of *LNCS*, pages 169–182, 1985.

[Pop02] E. Popovici. *Algorithms and Architectures for Decoding Reed-Solomon and Hermitian Codes.* PhD thesis, University College Cork, Ireland, March 2002.

[PT04] J.S. Plank and M.G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applica-

tions. In *DSN-2004: The International Conference on Dependable Systems and Networks*. IEEE, June 2004.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[Pur95] M. Purser. *Introduction to Error-Correcting Codes*. Artech House, 1995.

[QC82] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[RE04] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley and Sons, third edition, 2004.

[Red96] D. Redmond. *Number Theory: An Introduction*. Number 201 in Monographs and Textbooks in Pure and Applied Mathematics. Marcel Dekker, 1996.

[Res06] Wolfram Research. Eric Weisstein's World of Mathematics. *mathworld.wolfram.com*, 2006.

[Riv97] R.L. Rivest. All-or-nothing encryption and the package transform. *Fast Software Encryption*, 1267:210–218, 1997.

[RN04] V. Roca and C. Neumann. Design, evaluation and comparison of four large block FEC codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon small block FEC codec. Rapport de recherche, no. 5225, IN-

RIA Rhône-Alpes, Planète Research Team, France, June 2004. *http://www.inrialpes.fr/planete.*

[RS60] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Maths (SIAM)*, 8:300–304, 1960.

[RS98] R.L. Rivest and R. D. Silverman. Are "strong" primes needed for RSA? Technical report, RSA Laboratories, December 1998.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSU01] T. Richardson, M.A. Shokrallahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. on Information Theory*, 47(2):619–637, Feb 2001.

[Rya04] W.E. Ryan. An introduction to LDPC codes. In B. Vasic, editor, *CRC Handbook for Coding and Signal Processing for Recoding Systems.* CRC Press, 2004.

[Sak90] S. Sakata. Extension of the berlekamp-massey algorithm to n dimensions. *Information and Computation*, 84(2):207–239, Feb 1990.

[Sch99] B. Schneier. Biometrics: Uses and abuses. *Communications of the ACM*, 42(8), August 1999. Inside Risks 110.

[SGM97] C. Seal, M. Gifford, and D. McCartney. Iris recognition for user validation. *British Telecommunications Engineering Journal*, 16(7):113–117, 1997.

[Sha48] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[SKS06] G. Shailaja, K. Phani Kumer, and A. Saxena. Pairing based mutual authentication scheme using smart cards. International Association for Cryptologic Research (IACR) eprint archive, *http://eprint.iacr.org/2006/152.pdf*, 2006.

[Sou01] C. Soutar, December 2001. Private Communication.

[SRS+99] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B.V.K. Vijaya Kumar. Biometric encryption. In R.K. Nichols, editor, *ICSA Guide to Cryptography*. Mc-Graw-Hill, 1999.

[SRS+01] C. Soutar, D.B. Roberge, A. Stoianov, R.M. Gilroy, and V. Bhagavatula. Method for secure key management using a biometric. U.S. Patent No. 6219794, April 2001. Patent application: Oct 1997.

[Sun06] Sun Microsystems. JavaCard homepage, Smart Card Overview, 2006. *http://java.sun.com/products/javacard/smartcards.html*.

[SWM79] A. Simon, D.M. Worthen, and J.A. Mitas. An evaluation of iridology. *Journal of the American Medical Association*, 242:1385–1387, 1979.

[SZ04] R. Steinfeld and Y. Zheng. On the security of RSA with primes sharing least significant bits. *Applicable Algebra in Engineering, Communication and Computing*, 15(3-4):179–200, 2004.

[Tan81] R.M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. on Information Theory*, pages 533–547, Sep 1981.

[TKZ02] L. Thalheim, J. Krissler, and P.M. Ziegler. Body check: Biometric access protection devices and their programs put to the test. *c't (Germany)*, 11:114, May 2002. *http://www/heise.de/ct/english/02/11/114*.

[TMTR02] C. Tisse, L. Martin, L. Torres, and M. Robert. Person identification technique using human iris recognition. In *Proc. of 15th International Conference on Vision Interface*, pages 294–299, 2002.

[vT05] H.C.A. van Tilborg, editor. *Encyclopedia of Cryptography and Security*. Springer, 2005.

[WAH+96] R. Wildes, J. Asmuth, K. Hanna, S. Hsu, R. Kolczynski, J. Matey, and S. McBride. Automated, non-invasive iris recognition system and method. U.S. Patent Number 5572596, November 1996. Patent application made in September 1994.

[Wie90] M.J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Trans. on Information Theory*, 36(3):553–558, May 1990.

[Wil74] M.J. Williamson. *Non-Secret Encryption Using a Finite Field*. Communications-Electronics Security Group, GCHQ, Cheltenham, Gloucestershire, U.K., January 1974.

[WJ97] B.E. Wahlen and J. Jiménez. Performance comparison of hermitian and reed-solomon codes. In *Proceedings of MILCOM 97*, volume 1, pages 15–19, November 1997.

[WS98] A. Worthen and W. Stark. Low-density parity check codes for fading channels with memory. In *Proc. of 36th Allerton Conference on Communications, Control and Computing*, pages 117–125, September 1998.

[YHB+97] N.D. Young, G Harkin, R.M Bunn, D.J. McCulloch, R.W. Wilks, and A.G. Knapp. Novel fingerprint scanning arrays using polysilicon TFTs on glass and polymer substrates. *IEEE Electron Device Letters*, 18(1):19–20, January 1997.

[YK91] K. Yang and P. Vijay Kumar. On the true minimum distance of hermitian codes. In H. Stichtenoth and M.A. Tsfasman, editors, *Coding Theory and Algebraic Geometry*, volume 1518 of *Lecture Notes in Mathematics*, pages 99–107. Springer-Verlag, 1991.

[ZTW00] Y. Zhu, T. Tan, and Y. Wang. Biometric personal identification based on iris patterns. In *Proc. 15th International Conference on Pattern Recognition*, volume 2, pages 805–808, 2000.