Cryptographic Hash Functions: Cryptanalysis, Design and Applications

by

Praveen Gauravaram

Bachelor of Technology in Electrical and Electronics Engineering
Sri Venkateswara University College of Engineering, Tirupati, India, 2000
Master of Information Technology
Queensland University of Technology, Brisbane, Australia, 2003

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

Information Security Institute
Faculty of Information Technology
Queensland University of Technology

2007

Keywords

Cryptography, hash functions, cryptanalysis, design, applications, Merkle-Damgård construction, CAVE, 3CG, 3C, GOST-L, F-Hash, NMAC, HMAC, O-NMAC, M-NMAC, NMAC-1, Iterated Halving, digital signatures, side-channel attacks, practical and legal implications.

Abstract

Cryptographic hash functions are an important tool in cryptography to achieve certain security goals such as authenticity, digital signatures, digital time stamping, and entity authentication. They are also strongly related to other important cryptographic tools such as block ciphers and pseudorandom functions. The standard and widely used hash functions such as MD5 and SHA-1 follow the design principle of Merkle-Damgård iterated hash function construction which was presented independently by Ivan Damgård and Ralph Merkle at Crypto'89. It has been established that neither these hash functions nor the Merkle-Damgård construction itself meet certain security requirements. This thesis aims to study the attacks on this popular construction and propose schemes that offer more resistance against these attacks as well as investigating alternative approaches to the Merkle-Damgård style of designing hash functions. This thesis aims at analysing the security of the standard hash function Cellular Authentication and Voice Encryption Algorithm (CAVE) used for authentication and key-derivation in the second generation (2G) North American IS-41 mobile phone system. In addition, this thesis studies the analysis issues of message authentication codes (MACs) designed using hash functions. With the aim to propose some efficient and secure MAC schemes based on hash functions.

This thesis works on three aspects of hash functions: design, cryptanalysis and applications with the following significant contributions:

- Proposes a family of variants to the Damgård-Merkle construction called **3CG** for better protection against specific and generic attacks. Analysis of the linear variant of **3CG** called **3C** is presented including its resistance to some of the known attacks on hash functions.

- Improves the known cryptanalytical techniques to attack **3C** and some other

similar designs including a linear variant of GOST, a Russian standard hash function.

- Proposes a completely novel approach called Iterated Halving, alternative to the standard block iterated hash function construction.
- Analyses provably secure HMAC and NMAC message authentication codes (MACs) based on weaker assumptions than stated in their proofs of security. Proposes an efficient variant for NMAC called NMAC-1 to authenticate short messages. Proposes a variant for NMAC called M-NMAC which offers better protection against the complete key-recovery attacks than NMAC. As well it is shown that M-NMAC with hash functions also resists side-channel attacks against which HMAC and NMAC are vulnerable. Proposes a new MAC scheme called O-NMAC based on hash functions using just one secret key.
- Improves the open cryptanalysis of the CAVE algorithm.
- Analyses the security and legal implications of the latest collision attacks on the widely used MD5 and SHA-1 hash functions.

Contents

Keywords iii

Abstract v

Declaration xvii

Previously Published Material xix

Acknowledgements xxiii

1 Hash Functions in Cryptology 1

1.0.1 Security Goals in Cryptography 2

1.1 Cryptographic Hash Functions 3

1.1.1 Position of Hash Functions in Cryptology 4

1.2 Design, Analysis and Applications of Hash Functions 7

1.2.1 Hash Function Designs 7

1.2.2 Hash Function Analysis 8

1.2.3 Applications of Hash Functions 9

1.3 Aims and Objectives 10

1.4 Research Results 11

2 An Overview of Merkle-Damgård Hash Function Construction **15**

2.1 Definition and Properties 16

2.2 Iterated Hash Functions 17

2.2.1 Merkle-Damgård Construction 18

2.2.2 Hash Functions following the Merkle-Damgård Structure . 20

2.2.3 Differences between Damgård's Design and Practical Hash Functions 22

2.2.4 Compression Functions of Hash Functions 23
2.3 Attacks on Hash Functions 24
2.3.1 Brute force Attacks 24
2.3.2 Cryptanalytical Attacks 25
2.3.3 Collision Attacks on the Compression Functions 27
2.3.4 Collision finding Algorithms 28
2.4 Generic Attacks on the Merkle-Damgård Construction 29
2.4.1 Length Extension Attacks 30
2.4.2 Joux Generic Attacks 31
2.4.3 Generic 2^{nd} preimage Attacks 33
2.4.4 Herding Attack 35
2.5 Multi-block Collision Attacks on Hash Functions 37
2.5.1 New Observations on Multi-block Collision Attacks 37
2.6 Conclusion 40

3 New Modes of Operation for Hash Functions 41
3.1 The 3CG Construction: An Enhancement to the Merkle-Damgård Scheme 43
3.1.1 The 3C construction: A Linear Variant of 3CG 43
3.2 Analysis of 3C against Collision Attacks 44
3.3 Security Analysis of **3C** against Known Generic Attacks 49
3.3.1 Analysis Against Joux's Attacks 49
3.3.2 Analysis against 2^{nd}-preimage Attacks 50
3.3.3 Analysis against Herding Attack 51
3.3.4 Analysis against Length Extension Attacks 51
3.3.5 Analysis against 2^{nd}-collision Attacks 52
3.4 Comparison of **3C** with Related Hash Function Proposals 54
3.5 Hybrid Hash Function Constructions using 3C 57
3.6 The **3C+** Construction 58
3.6.1 Some Variants for 3C and 3C+ 59
3.7 Implementation Issues 61
3.8 Conclusion 61

4 Cryptanalysis of a Class of Cryptographic Hash Functions 65
4.1 Linear checksum variants of Merkle-Damgård 66

4.1.1 GOST and its Variants 66
4.1.2 F-Hash Hash Function 67
4.1.3 3C-GOST-L Design 68
4.2 Inapplicability of Known Generic Attack Techniques on GOST-L, F-Hash and 3C-GOST-L 69
4.2.1 2^{nd}-preimage Attack 69
4.2.2 Herding Attack 70
4.3 New Cryptanalytical Techniques 71
4.3.1 Combining Multi-block Collisions and Multicollisions . . . 71
4.3.2 Building 2^t 2-block multicollisions on **3C** 71
4.3.3 Defeating the Linear Checksum in a 2^t 2-block Multicollision on **3C** 72
4.3.4 Building a 2^b 1-block Multicollision on GOST-L 75
4.3.5 Defeating the Linear Checksum in 2^b 1-block Multicollision on GOST-L 76
4.3.6 Defeating the Checksum in 2^t Multicollision on F-Hash . . 77
4.3.7 Defeating the Linear Checksum in **3C-GOST-L** 79
4.4 Generic Attacks on 3C, GOST-L, F-Hash and 3C-GOST-L 82
4.4.1 Long-message 2^{nd}-preimage Attack on **3C** 82
4.4.2 Herding Attack on **3C** 85
4.4.3 The Generic Attacks on Better Linear Checksums 87
4.4.4 Making Meaningful Messages in the Attacks 88
4.4.5 Some Other Techniques to Perform the Generic Attacks . . 88
4.5 Collision Attacks on GOST-L, **3C** and F-Hash 89
4.6 Finding Multiple 2^{nd} preimages for Hash Functions in less than 2^t Work 91
4.6.1 Multi-2^{nd} preimage Attack on the Cascade Constructions . 94
4.7 Conclusion 95

5 Iterated Halving- A New Approach for Hash Function Design 97
5.1 Iterated Halving Structure 99
5.2 Working details of an IH hash function 102
5.3 Performance and Security of the IH Process 103
5.4 Security Analysis 105
5.5 CRUSH: A Hash Function Proposal based on IH Technique 108

5.5.1 Notation and Definitions 108
5.5.2 Working Details of CRUSH 111
5.6 Conclusion 112

6 Improved Cryptanalysis of the CAVE Algorithm 117
6.1 Notation and Definitions: 118
6.2 The CAVE Algorithm Description 121
6.3 Practical Usage of CAVE 124
6.4 Properties of the CAVE Algorithm 126
6.5 Previous Analysis of CAVE 130
6.5.1 Millan's Reconstruction Attack on CAVE 131
6.6 Improved Cryptanalysis of CAVE 134
6.7 Conclusion 138

7 An Update on MACs based on Hash Functions 141
7.1 A Review of MACs based on Hash Functions 143
7.1.1 NMAC and HMAC Functions 150
7.1.2 Analysis of NMAC and HMAC 152
7.1.3 Summary of the Review on MACs 154
7.2 Analysis of NMAC using Weaker Assumptions on the Hash Functions 156
7.2.1 Security Analysis 156
7.3 An Efficient Variant for NMAC 159
7.3.1 Specification of NMAC-1 159
7.3.2 Performance Aspects of NMAC-1 160
7.3.3 Security Analysis of NMAC-1 161
7.3.4 Comparison of NMAC-1 with Other Efficient MACs based on Hash Functions 163
7.4 M-NMAC: A New Variant of NMAC 166
7.4.1 Security Analysis of M-NMAC 168
7.5 Pseudorandomness of NMAC and HMAC Functions 168
7.6 On designing a MAC using a Hash Function with a Single Secret Key 172
7.6.1 The O-NMAC Function 173
7.6.2 Analysis of O-NMAC 174
7.6.3 Comparison with NMAC, HMAC and ACSC Constructions 178

7.7 Conclusion 179

8 Side-Channel Cryptanalysis of MACs using Hash Functions 183

8.1 Side-channel Attacks on NMAC and HMAC 185

8.1.1 DPA and Reverse DPA Settings 185

8.1.2 Side-channel Attacks on NMAC and HMAC with 12 PGV Schemes 186

8.2 Side-channel Attacks on M-NMAC 187

8.2.1 Target Compression Functions in M-NMAC 187

8.2.2 Key Recovery Attacks on M-NMAC with 12 PGV Schemes 188

8.2.3 Extending Side-channel Analysis on M-NMAC to Forgery Attacks 190

8.2.4 Comparison with NMAC and HMAC 190

8.3 Conclusion 191

9 Collision Attacks on MD5 and SHA-1: Is this the "Sword of Damocles" for Electronic Commerce? 195

9.1 Practical and Legal Implication of Collision Attacks on MD5 and SHA-1 196

9.1.1 Digital Signatures on Messages 197

9.1.2 Digital Signatures on Digital Certificates 203

9.1.3 Message Authentication 207

9.1.4 Software Protection 208

9.2 Conclusion 211

10 Conclusion and Future Research 213

10.1 Summary of Thesis Contributions 214

10.2 Future Directions 216

A Finding Expandable Messages for Hash Functions 221

A.1 Fixed Point Expandable Message Algorithm 221

A.2 Finding Expandable Messages using Generic Multicollision 222

A.2.1 Building a Full $(d, d + 2^d - 1)$-expandable Message 223

A.3 Long Message 2nd preimage Attack using Expandable Messages . 224

B Evolution of MD4 Family of Hash Functions **227**

B.0.1 Balanced and Unbalanced Feistel Networks 228

B.0.2 Hash Functions in the MD4 Family 229

B.0.3 Some Hash Functions outside the MD4 family 237

C Differential Cryptanalysis of Hash Functions **239**

C.1 2-block Collision Attack on MD5 240

D Generic Attacks on GOST-L, F-Hash and 3C-GOST-L **243**

D.1 Long Message 2nd-preimage Attack on GOST-L 243

D.1.1 Long Message 2nd-preimage Attack on F-Hash 245

D.1.2 Long-message 2nd-preimage Attack on **3C-GOST-L** . . . 247

D.1.3 Herding Attack on GOST-L 250

D.1.4 Herding Attack on F-Hash 251

D.1.5 Herding Attack on **3C-GOST-L** 252

E CAVE Tables **255**

F Application based on HMAC-SHA-1 **257**

G Implementation Issues **259**

H 12 Provably Secure PGV Compression Functions **261**

Bibliography **263**

List of Figures

1.1 Cryptographic Hash Function 3

2.1 The Merkle-Damgård construction 21
2.2 Davies-Meyer compression function 24
2.3 Attacks on Cryptographic Hash Functions 24
2.4 Multicollisions on the Merkle-Damgård hash function 32
2.5 Diamond structure with $d = 3$ 36
2.6 Finding a linking message and producing the suffix 37

3.1 The generic **3CG**-hash function 43
3.2 The **3C**-hash function 44
3.3 Creating an internal collision for **3C** 45
3.4 The wide-pipe hash function 55
3.5 The **3CWP** hybrid construction 58
3.6 The **3C+** hash construction 58
3.7 The **3CCW+** Construction 60

4.1 Hash function with a general linear checksum 66
4.2 GOST hash function 67
4.3 GOST-L hash function 67
4.4 The F-Hash hash function 67
4.5 The compression function (f-function) of F-Hash 68
4.6 The **3C-GOST-L** hash function 69

5.1 Standard Hashing using a Block Cipher 100
5.2 Similarities to stream and message encryption 100
5.3 Key Feedback Mode 101
5.4 Iterated Halving Technique 101

5.5 Iterated Halving (IH) process 103
5.6 The F-function of CRUSH 111
5.7 Structure of CRUSH 114

6.1 CAVE Linear Feedback Shift Register Block Diagram 122
6.2 Segment operation in CAVE 123
6.3 Many roles of CAVE 127
6.4 Usage of CAVE in IS-41 and IS-54 phone systems 127

7.1 The slice by slice trail key recovery attack 148
7.2 The NMAC Construction 151
7.3 The HMAC Construction 152
7.4 The M-NMAC Construction 166
7.5 The O-NMAC construction 174

8.1 The DPA and Reverse DPA models 185
8.2 Refined $f_{3(2)}$ (resp. $f_{7(2)}$) from [221] and $f_{3(3)}$ (resp. $f_{7(3)}$) 189

B.1 Balanced (a) and Unbalanced Feistel Networks (b) 228
B.2 Core of the MD4 compression function 230
B.3 Core of the MD5 compression function 231
B.4 The core of the SHA-0/1 compression function 233
B.5 Core of 1-track of RIPEMD-160 compression function 235
B.6 SHA-256 Compression Function 236

H.1 Compression functions based on PGV construction 262

List of Tables

2.1 Joux attacks 33
2.2 Classification for 2-block collision attacks 38
2.3 Resistances of some compression functions 39

3.1 Time in seconds to process one gigabyte of data using MD5 on Pentium-M machine with a CPU speed of 2GHz 62
3.2 Time in seconds to process one gigabyte of data using SHA-1 on Pentium-M machine with a CPU speed of 2GHz 62

5.1 The key-schedule of CRUSH 110
5.2 The constants for the key-schedule algorithm 110
5.3 The Inital state of BSC 110
5.4 The operations in the **B** function 110
5.5 Pseudo code for the IH process in CRUSH 113
5.6 Pseudo code for the stage process in CRUSH 113
5.7 The non-redundant 8*8 S-box for F-function 115

6.1 The pseudo-code of the CAVE algorithm 125
6.2 Initial loading of CAVE 126
6.3 Imbalances in the LOW CAVE table 130
6.4 Imbalances in the HIGH CAVE table 131
6.5 Comparison of two attacks 138

7.1 Efficiency improvement of NMAC-1 over NMAC 161
7.2 Comparison among MAC schemes 178

8.1 Results of side-channel attacks on M-NMAC 191
8.2 Secure Compression Functions for NMAC/HMAC and M-NMAC 191

9.1 Summary of impact of collisions in MD5 on some applications . . 212

B.1 Merkle-Damgård Hash Functions 229

E.1 The High and Low CAVE Tables, In Hex Notation 255

G.1 Time in seconds taken to hash one gigabit of data using MD5 and MACs based on it 260

G.2 Time in seconds taken to hash one gigabit of data using SHA-1 and MACs based on it 260

Declaration

Previously Published Material

The following page contains papers published or presented at conferences containing the material included in this thesis.

Bibliography

[1] Praveen Gauravaram and William Millan. Improved Attack on the Cellular Authentication and Voice Encryption Algorithm. In *Proceedings of the workshop on Cryptographic Algorithms and their uses*, ISBN 1-74107-064-3, pages 1–13, Goldcoast, Australia, July 4–5 2004.

[2] Praveen Gauravaram, William Millan, and Lauren May. CRUSH: A New Cryptographic Hash Function using Iterated Halving Technique. In *Proceedings of the workshop on Cryptographic Algorithms and their uses*, ISBN 1-74107-064-3, pages 28–39, Goldcoast, Australia, July 4–5 2004.

[3] William Millan and Praveen Gauravaram. Cryptanalysis of the Cellular Authentication and Voice Encryption Algorithm. *IEICE Electronic Express*, 1(15):453–459, 2004.

[4] Praveen Gauravaram, William Millan, Ed Dawson, and Kapali Viswanathan. Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction. In *Australasian Conference on Information Security and Privacy (ACISP)*, volume 4058 of *Lecture Notes in Computer Science*, pages 407–420, 2006.

[5] Praveen Gauravaram, William Millan, Ed Dawson, Matt Henricksen, Juanma González Nieto and Kapali Viswanathan. Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction (full version). Technical Report QUT-ISI-TR-2006-013, Information Security Institute (ISI), Queensland University of Technology (QUT), July 2006. This technical report is available at `http://www.isi.qut.edu.au/research/publications/technical/qut-isi-tr-2006-013.pdf`. Last access date: 22nd of January 2007.

[6] John Kelsey and Praveen Gauravaram. Linear checksums Don't Block Generic Damgård-Merkle Attacks. Presented at Crypto'06 Rump Session, 2006.

[7] Praveen Gauravaram, Adrian McCullagh, and Ed Dawson. The legal and practical implications of recent attacks on 128-bit cryptographic hash functions. *First Monday*, 11(1), 2006.

[8] Praveen Gauravaram, Shoichi Hirose, and Suganya Annadurai. An Update on the Analysis and Design of NMAC and HMAC functions. To be published in the International Journal of Network Security (IJNS), 2007.

[9] Praveen Gauravaram, Adrian McCullagh, and Ed Dawson. Collision Attacks on MD5 and SHA-1: Is this the "Sword of Damocles" for Electronic Commerce? In Andrew Clark and Mark McPherson and George Mohay, editor, *AusCERT Asia Pacific Information Technology Security Conference Refereed R & D Stream*, pages 1–13, May 2006.

[10] Praveen Gauravaram and Katsuyuki Okeya. Security Analysis on Keyed Hash Functions from the Viewpoint of Side Channel Attacks. Presented at the Symposium on Cryptography and Information Security (SCIS), Sasebo, Japan, Jan. 23-26, 2007.

Acknowledgements

First of all, I wish to express my deepest, evergreen gratitude to my father, mother, brother, grand parents and all other members of the family. Their love, support and unshakable belief in me since my childhood provides an immeasurable strength to succeed in the goals of life. I wish to express my gratitude to all the teachers from my home town Tirupati who contributed to my knowledge and skills development since my childhood.

My interest in cryptology has developed when I was having fun breaking classical ciphers given in the Assignment-1 of the course work unit "Introduction to Cryptology" during my masters programme at QUT. Strong philosophical discussions with my friend Dr. Kapali Viswanathan have motivated me to pursue career in the research studies in cryptology. I wish to thank Kapali. I wish to thank Professor Ed Dawson for admitting me to pursue PhD at the then Information Security Research Center (ISRC) which has now become Information Security Institute (ISI). I wish to thank him for his enthusiastic support all these years. I wish to thank my principal supervisor Dr. William Millan for accepting me as his student and suggesting cryptographic hash functions as the area of research. His friendly discussions on the subject are of immense help. I thank him for his support, collaboration and encouragement. I wish to thank my associate supervisor Dr. Lauren May and other lecturers Mr. Michael Adams, Dr. Ernest Foo, Mr. Cecil Goldstein, Miss Vicky Liu, Dr. Greg Maitland, Mr. Richard Thomas for supporting my living by providing me with an opportunity to teach their respective courses.

I sincerely wish to thank ISI and Society of Electronic Transactions and Security (SETS, India) in 2004 November-2005 January for their valuable financial support and encouragement which has been responsible for obtaining some significant results in my thesis in 2006. I wish to thank Dr. Vijaya Raghavan, the executive director of SETS, the researchers Ms. Suganya Annadurai, Mr. Aswin Gopalan, Mr. Harish Kumar, Ms. Jayasri Kunchithapatham, Dr. Vijay Rangan and other employees at SETS for their immense support in Chennai during that period. I wish to thank Professor Lars Knudsen, the visiting scientist to ISI during the period 2005 December-2006 June for his time, many fruitful discussions on hash functions and valuable encouragement. It is my pleasure to thank Dr. Jiri Tuma, the visiting scientist to the University of Queensland in 2005 for his time and valuable discussions on hash functions.

I personally wish to thank Dr. Matt Henricksen, Adjunct Professor Adrian McCullagh and Dr. Juanma Gonzalez Nieto for many discussions on hash functions, valuable support, collaboration and encouragement all these years. I personally wish to thank Dr. John Kelsey for giving me an opportunity to work with him and his good guidance. I wish to sincerely thank Dr. Shoichi Hirose for collaborating with me and sharing many fruitful discussions. I wish to sincerely thank Dr. Katsuyuki Okeya for his collaboration and immense insights and discussions which has improved my knowledge in this area of research. It is my sincere thanks to Ms. Suganya Annadurai and Dr. Kapali Viswanathan for providing some good research problems to solve. It is my pleasure to thank all the researchers across the globe for answering my questions and clarifying several questions. Their input is acknowledged at appropriate places in the thesis.

I thank some researchers at ISI for their friendly and research interactions and support. They include Dr. Riza Aditya, Mr. Sultan Al Hinai, Prof. Colin Boyd, Prof. William Caelli, Dr. Gary Carter, Mr. Kevin Chen, Dr. Kim-Kwang Choo, Dr. Andrew Clark, Dr. Yvonne Cliff, Mr. Robert Dawson, Dr. Jiang Du, Dr. Ernest Foo, Associate Professor Audun Josang, Dr. Byoungcheon Lee, Dr. Sang Gon Lee, Dr. Greg Maitland, Professor George Mohay, Dr. Havård Molland, Mr. Sorot Panichprecha, Dr. Kun Peng, Mr. Jason Reid, Mr. Bradley Schatz, Dr. Leonie Simpson, Mr. Jason Smith, Mr. Kevin Tham, Dr. HariKrishna Vasanta and Dr. Christian Wullems. Many thanks to many other students and staff members at ISI and FIT for their friendly interactions and support. They include Mr. Saleh Almotairi, Dr. Wai Au, Ms. Jaimee Brown, Mr. Mark Branagan, Mr. Malakondaya Choudary, Professor Sharon Christensen, Ms. Rong Du, Professor Bill Duncan, Mr. Rupinder Gill, Miss. Marianne Hirschbichler, Mr. Huseyin Hisil, Dr. John Holford, Mr. Mehdi Kiani, Ms. Susanna Leisten, Dr. Sang Gon Lee, Dr. Mark Looi, Ms. Judith McNamara, Dr. Dennis Longley, Mr. Andrew

Marrington, Ms. Kathryn O'Shea, Mr. David Ross, Dr. Selwyn Russell, Mr. Loo Tang Seet, Ms. Debbie Smit, Mr. Les Smith, Mr. Suriadi Suriadi, Mr. Zeeshan Ali Syed, Dr. Terry Tin, Mr. Suratose Tritilanunt, Mr. Chaw Seng Woo, Mr. Kenneth Wong and Dr. Jacob Zimmerman.

I thank all the technical staff at the institute for their support and technical services, they are: Mr. Matt Bradford, Mr. James Mackie, Mr. Mark Huth and Mr. Sam Lor.

I thank all the administrative staff at the institute and the IT Faculty for their support and administrative services, they are: Ms. Christine Orme, Ms. Elizabeth Hansford, Ms. Gina Farrington and Ms. Anne Hamburger.

Finally, it is a great pleasure for me to sincerely thank many close family friends in Brisbane who shared my happiness and sorrow all these years. Their evergreen family support at all times is very much appreciated. They are Smt & Sri Viswanath, Smt & Sri Visvanathan, Smt & Sri Gunashekar, Smt & Sri Sobhanadri, Smt & Sri Sivanathan, Smt & Sri Venkat, Smt & Sri Satish, Smt & Sri Sridha, Smt & Sri Sachin, Smt & Sri Anand, Anil, Balaji, Dhakshmi, Ghajan, Harish, Harshita, Menaka, Pavan, Pravin, Sarada, Sreekanth, Vidya and many others.

Chapter 1

Hash Functions in Cryptology

Cryptography is the art and science of information security. Its opposite term cryptanalysis refers to the art and science of information disclosure. These two terms are often encompassed by the term cryptology, the science of study of secrets: either by aiming to hide or reveal. While a cryptographer studies the techniques needed to protect the privacy of the information communicated between or among people via any communication channel, a cryptanalyst studies the techniques needed to understand the meaning of the secret information communicated among others via any communication channel. Until mid 1970s, the terms cryptography and cryptanalysis referred to the protection and violation of the confidentiality of information respectively. It was then widely believed that by protecting the privacy of the information, other important security goals such as authenticity of the information would be automatically achieved.

However, Whitfield Diffie and Martin Hellman, in their seminal paper on "New Directions in Cryptography" [66] have given birth to many concepts that are widely used these days by clearly distinguishing the problem of privacy from the problem of authentication. Their definition of cryptography is "Cryptography is the study of mathematical systems for solving two kinds of security problems: privacy and authentication." Roughly, fifteen years since that paper was published, Ronald Rivest has noted in the chapter "Cryptography" in the Handbook of Theoretical Computer Science [251] that cryptography is about communication in the presence of adversaries.

The adversary could be really anyone who wants to do something against the wishes of the communicating parties. Adversaries are enemies and they aim to disrupt the means used by the communicators to thwart the aims and goals of the communication. They do always exist in the form of humans or computers operated or programmed by humans. It is the task of the communicators to develop appropriate tools to protect themselves from the tricks of the adversaries making sure that their aims and goals are achieved[1].

Now the question is "what are the security goals that one would expect cryptography to provide for protection from the adversaries?". In many cases, the goals of adversaries originate before the goals of cryptography and these two issues are like two sides of the same coin. The security goals of cryptography are answered to some extent in the following section using [61, 62].

1.0.1 Security Goals in Cryptography

Before 1980, privacy or confidentiality which is the oldest security goal, authenticity [94] as introduced by Gilbert-MacWilliams-Sloane (following a suggestion of Simmons), digital signatures [66] introduced by Diffie-Hellman, secure private computation [257] introduced by Rivest-Adleman-Dertouzos and secret sharing [36] as described by Blakely and Shamir [269] were the primary security goals of cryptography. There was also simultaneous evolution of cryptographic tools to meet these goals. For example, block ciphers, notably the Data Encryption Standard (DES) [204] by National Bureau of Standards (NBS) to provide privacy in communications, one-way functions [76, 77, 293] for authentication and trapdoor one-way permutations to achieve digital signatures [66].

Between 1980 and 1994, some more cryptographic goals that include oblivious-transfer [241], anonymity [43], poker [268], entity-authentication [80], key escrow [21], time-stamping [99] and traitor tracing [44] were identified. Some new tools such as interactive proofs were originated during this period to realize these security goals [62]. On the other hand, during this period there was also development of cryptanalytical tools to analyse the tools used to achieve the security goals in cryptography. A few of the most significant tools developed and further improved during this period are the lattice reduction algorithm (LLL scheme) [168] [2]

[1]Note that the adversaries can try to make the resources of the communicating parties unavailable and this goal of the adversary is not handled by cryptography.

[2]LLL has replaced the original, now outdated, polynomial time algorithm [171] used to

to analyse public key cryptosystems and differential cryptanalysis [30] and linear cryptanalysis [177] to analyse block ciphers. These days one would see other significant goals such as digital steganography [39] which originated after 1994.

There has been an explosion of security goals in cryptography in the past 31 years, design tools have been invented to achieve those goals and analytical tools have been developed to analyse the security of those designs. The design and analytical tools in order to achieve various goals and invent new security goals are like two sides of the same coin and they develop together.

1.1 Cryptographic Hash Functions

This thesis studies cryptographic hash functions. A cryptographic hash function H is an algorithm which processes an arbitrary length message into a fixed length digest or hash code as shown in Figure 1.1. Hash functions are one of the most important tools in cryptography, achieving many security goals including authenticity, digital signatures, digital time stamping, entity authentication, poker, digital steganography. They are technically related to other important cryptographic tools such as block ciphers and pseudorandom functions. In addition, hash functions form an integral component in the functionality of other cryptographic tools such as key distribution protocols and zero-knowledge proofs. Their usage in several information processing applications to achieve various security goals is much more widespread than the application of block ciphers and stream ciphers.

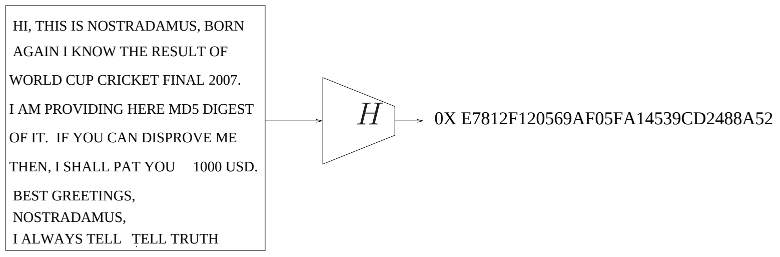

HI, THIS IS NOSTRADAMUS, BORN
AGAIN I KNOW THE RESULT OF
WORLD CUP CRICKET FINAL 2007.
I AM PROVIDING HERE MD5 DIGEST
OF IT. IF YOU CAN DISPROVE ME
THEN, I SHALL PAT YOU 1000 USD.
BEST GREETINGS,
NOSTRADAMUS,
I ALWAYS TELL TELL TRUTH
H
0X E7812F120569AF05FA14539CD2488A52

Figure 1.1: Cryptographic Hash Function

A secure hash function must possess three fundamental security properties: (1) collision resistance, (2) pre-image resistance and (3) 2nd pre-image resis-

analyse the knapsack public key cryptosystem [270, 271].

tance [186]. The relative importance of these properties is application dependent. These fundamental properties of hash functions are discussed in Chapter 2.

1.1.1 Position of Hash Functions in Cryptology

The science of cryptology can be classified into two divisions: symmetric key cryptology and public key cryptology. In the former case, the communicating parties share the same secret key in order to achieve their security goals whereas in the latter case they use two keys: public key used by others who wish to communicate with the main party and private key possessed only by the main party. Now it is interesting to see to which branch of cryptology hash functions fit.

Many researchers might consider hash functions as one of the objects of symmetric key cryptology due to their close inter connection with the two other objects: block ciphers and stream ciphers. One reason for this classification could be due to the fact that the internal structure of the widely deployed, the so called, dedicated hash functions is basically the same as that of some block ciphers [267]. This concept is explored in depth in Chapter 2. In addition, one could easily convert block ciphers and stream cipher structures into hash functions and vice-versa. Even the well established techniques to analyse block ciphers such as differential cryptanalysis are used to analyse practical hash functions.

However, it might not be correct to classify these objects as part of one particular branch of cryptology considering their wide usage in both public key cryptology and symmetric key cryptology. Here it is justified by showing how hash functions are used in the setting of both branches of cryptology to achieve one of the important security goals in cryptology, called authentication. This justification also helps in providing a classification for hash functions from the functional perspective.

Information authentication or authentication refers to protecting the information in communication and the source or sources of the information in the communication from the modification by the third party members or unauthorized parties [228]. In the terminology of International Standards Organization (ISO), the former goal refers to integrity of the information and the latter goal refers to the data origin authentication [118].

- Information authentication is achieved by linking a secret key to the source of information. If the secret key is used to compress the information using a hash function, then the obtained digest is called the message authentication code (MAC) or authentication tag [228]. Note that hashing is done on both the secret key and information and hence MAC schemes are considered to be a subclass of keyed hash functions [186]. The term MAC is also used for the mechanism which generates the hash codes. In this thesis the term MAC scheme is used to refer the mechanism, and tag is used to refer the output hash code. The MAC mechanism can use block ciphers, stream ciphers or hash functions. MAC schemes based on hash functions are covered in depth in Chapter 7.

- If the secret key is used only to protect the digest and/or the information but not in the compression process then the obtained digest is called the modification detection code (MDC) [228]. Note that hashing is done only on the information without the involvement of any secret key and hence MDCs are considered to be a subclass of un-keyed hash functions [186]. Noted that by default, the term hash function refers to un-keyed hash functions. MDCs are further classified into one-way hash functions (OWHFs) and collision resistant hash functions (CRHFs) and these are discussed in Chapter 2.

Hence in the symmetric key setting, hash functions are used in the form of MAC schemes to ensure the authenticity of the information. However, MAC schemes do not prevent either the sender or receiver of the communication denying their involvement in the communication or source of authentication. For example, in a communication between two parties, either party can generate tags at their own discretion using the shared secret key. Therefore MAC schemes do not provide the so called security property or service of non-repudiation of origin [118] or non-repudiation. Diffie and Hellman [66] first identified the need for a “purely digital, unforgeable, message dependent signature” to avoid disputes between the sender and receiver and they have proposed a solution based on trapdoor one-way permutations. Rivest, Shamir and Adleman [257] have proposed the first practical public key cryptosystem called RSA based on the integer factorization with digital signature capability. Digital signature is a security goal of a cryptosystem which intends to achieve the goal of authenticity and a security service or property of non-repudiation.

Without hashing, the signature is same size as the message. Hash functions are used to optimize the digital signature schemes. The fundamental concept here is instead of generating the signature for the whole message which is to be authenticated, the sender of the message only signs the digest of the message using a signature generation algorithm. The sender then transmits the message and the signature to the intended receiver. The receiver verifies the signature of the sender by computing the digest of the message using the same hash function as the sender and comparing it with the output of the signature verification algorithm. It is obvious that this approach saves a lot of computational overhead involved in signing and verifying the messages in the absence of hash functions. In some cases, hash functions are incorporated as part of the signature algorithm. For example, see Federal Information Processing Standards Publication (FIPS PUB 186-2) [217] which has specified the usage of SHA-1 hash function [211] with the Digital Signature Algorithm (DSA).

Hash functions are also used to efficiently authenticate users of computer systems by storing the digests of the passwords in the password file of the system instead of passwords themselves. Whenever users try to log-in to the computer system they need to authenticate by entering their password. The system hashes the entered password and compares it with the stored password digest and allows user access to the system only if these two passwords correctly match. Since passwords must be processed using hash function once per login, its computation time must be small. In addition to their use to achieve efficient digital signature and authentication security goals, hash functions are also used to improve the efficiency of other goals of cryptography such as digital time stamping [99], entity authentication [80] and many more.

Of course, care must be taken in using hash functions with any application as the security of hash functions, in many cases, affects the security of the application in use. This issue on the security requirements of hash functions when they are used in various applications is discussed in great detail in Chapter 9 of the thesis. In Chapter 9, the legal and practical implications that could arise when the applications that use hash functions come under scrutiny due to the violation of security requirements of hash functions are dealt in depth.

Considering this wide spread use of hash functions in many ways in cryptology to achieve various security goals, it is concluded that it is not correct to say that they belong to one particular cryptographic mechanism. These cryptographic tools deserve a separate status for themselves. They are used in almost all places in cryptology where efficient information processing is required. However, the strength of the application which uses hash functions partly or completely depends on the protection of at least one of the security features of hash functions.

1.2 Design, Analysis and Applications of Hash Functions

1.2.1 Hash Function Designs

As pointed out in Section 1.1.1 in the context of MAC schemes and digital signatures, a hash function performs the task of compression. This is one of the minimum general requirements of a hash function the other being the ease of computation of arbitrary length information [186]. The digital signature and MAC examples provided in Section 1.1.1 emphasize the necessity of this property for a hash function. This property of ease of computation and optimization of information processing applications through the use of hash functions is the reason for most of the existing hash function proposals to be in the iterative mode of operation. Such hash functions are called iterated hash functions.

Based on their structure and operational features, iterated hash functions are classified into hash functions based on block ciphers, dedicated or customized hash functions and hash functions based on modular arithmetic [186, p.338]. It is not in the scope of this thesis to survey hash functions based on block ciphers and modular arithmetic. Refer [33, 35, 108, 146, 164, 228, 232] for some literature on the analysis and design of hash functions based on block ciphers. Refer [47, 51, 228] for some literature on the hash functions based on the modular arithmetic. One remark worth making here is while the main motivation for designing hash functions using block ciphers is to use the already available efficient implementations of block ciphers to achieve the hash function functionality at little additional effort [186], recent studies in this area shows that it is difficult to design efficient hash functions from block ciphers [33]. Wherever it is essential in this thesis, the concepts related to hash functions based on block ciphers are emphasized.

Handbook of Applied Cryptography [186, p.343] defines customized hash functions as “those which are specifically designed ‘from scratch’ for the explicit purpose of hashing, with optimized performance in mind, and without being constrained to reusing existing system components such as block ciphers or modular arithmetic". Notably these hash functions are designed following the principles of the popular Merkle-Damgård iterative hash function construction [53, 187]. The first hash function designed following this iterative principle was MD4 by Rivest [252]. After that many hash functions were designed following MD4 and they are considered as part of the MD4 family of hash functions. In this thesis, unless stated otherwise, the term "hash function(s)" refer to a(the) hash function(s) designed following the Merkle-Damgård iterative structure.

It seems that this definition of customized hashing is partly misleading. Bruce Schneier and John Kelsey in their classical paper on "Unbalanced Feistel Networks and Block Cipher Design" [267] show that the internal structure of so called customized hash functions of the MD4 family are basically block-cipher based unbalanced Feistel structures. The SHACAL block cipher [101] designed from the compression function of SHA-1 [211] proves the point of [267]. In fact, SHACAL could have been proposed as soon as SHA-1 was released. In Appendix B, this point is further emphasized by showing that the compression functions of MD4 family of hash functions are unbalanced Feistel networks (UFNs) operating as word-based non-linear feedback shift registers (NLFSRs) along with the known cryptanalytic results on these hash functions from the literature. Chapter 5 of this thesis proposes a new hash function design called Iterated Halving (IH) as an alternative for block-iterated mode of hash functions.

1.2.2 Hash Function Analysis

The security of hash functions primarily depends on the size of the hash function digest. The brute force attacks that are applicable on all hash functions aim to violate the fundamental security properties of hash functions and their computational effort is influenced only by the digest size and are independent of the working details of the hash function. These attacks are similar to brute-force key recovery attacks used to recover the secret key of block ciphers. They are discussed in Chapter 2.

There are also attacks that aim to violate security properties of hash functions working on specific hash function constructions such as iterated hash functions in the style of Merkle-Damgård construction. These attacks are called *generic* *attacks.* The work effort involved in these attacks apart from depending on the size of the hash code also depend on some other parameters (for example, internal state size) of the generic hash function structure. The practical feasibility of these attacks on specific hash functions designed from those generic constructions depends on the sizes of those parameters that influence the work factor of the attacks. The known generic attacks on the Merkle-Damgård construction are discussed in detail in Chapter 2 and the new variants that resist the known techniques of carrying out these attacks on the Merkle-Damgård construction are discussed in Chapter 3. New cryptanalytical techniques to carry out the known generic attacks on a class of cryptographic hash functions (including those proposed in Chapter 3) are covered in Chapter 4.

On the other hand, specific hash functions such as MD4 and MD5 can be analysed individually to learn about their security properties. The tools used to analyse specific hash functions depends on the internal characteristics of those specific hash functions and in many cases differ for each hash function unless the hash functions are similar to each other in many operational aspects. Differential cryptanalysis is an example of a tool used to violate the collision resistance property of some specific hash functions. It is not in the scope of this thesis to provide the detailed explanation of such tools except that they are mentioned and pointed to the literature wherever it is essential. However, in Chapter 2, a new cryptanalytical tool called multi-block differential collision finding tool, used to analyse standard hash functions MD5 [285,291], SHA-0 [27,289] and SHA-1 [290] is explained in detail. New hash function constructions proposed in Chapter 3 are analysed with respect to this tool. It is observed in chapters 3 and 4 of the thesis that specific cryptanalytical tools used to analyse specific hash functions may be used to perform generic attacks on the Merkle-Damgård hash function construction.

1.2.3 Applications of Hash Functions

As pointed out in Section 1.1.1, hash functions have many security goals in cryptography. Analysing an application or a protocol based on hash functions explains the security of the application based on the strengths of the hash function used in it. For example, see Chapter 9 which explains the legal and practical implications of collision attacks on the MD5 and SHA-1 hash functions [288,290,291] in some applications where these hash functions are used.

As pointed out in Section 1.1.1, information authentication is one of the security goals of hash functions and in the symmetric key setting it is achieved using MAC functions. The design of MAC functions using hash functions is a very complicated issue as shown in Chapter 7. Chapter 7 studies analysis, design and performance issues of MAC schemes based on hash functions.

The classical analysis of a MAC function views the MAC function as a mathematical object, namely, a function parameterized by the secret key mapping an input to output. The results of such analysis are valid on any concrete implementation of the MAC function. On the other hand, one can view a MAC function as a concrete implementation of a mathematical object. Side channel analysis looks at the implementation specific characteristics of the MAC function and such an analysis is specific to an implementation. Chapter 8 studies side channel attacks on the popular MAC schemes hash based MAC (HMAC) and nested MAC (NMAC) based on hash functions such as MD5 and SHA-1. In addition, it performs side channel analysis of a variant of MAC scheme proposed in Chapter 7.

1.3 Aims and Objectives

This thesis studies analysis, design and applications of cryptographic hash functions.

1. The main objective of this thesis is to examine the analysis of the popular Merkle-Damgård construction against known generic attacks and against some known cryptanalytical attacks that work on some hash functions based on it. Similar to encryption schemes, the security aspect of hash functions deals with both the design of hash functions and their cryptanalysis. It is difficult to design generic hash function constructions that resist all known attacks on generic constructions and specific attacks on the hash functions derived from those generic constructions. Conversely, being well versed in contemporary hash function design points a cryptanalyst to potential weak spots within a hash function design. The thesis aims to design and analyze the efficient variants to the Merkle-Damgård construction. In addition, it also aims at improving the known cryptanalytical results on some hash

function proposals. It also aims to design hash functions that do not involve a block iterated paradigm of the Merkle-Damgård construction.

2. The other important objective of this thesis is to examine the design, analysis and efficiency issues of the MAC schemes based on cryptographic hash functions. Again, both the cryptanalysis and design of MAC schemes are equally important to develop the state of art of MACs based on hash functions. This thesis seeks to meet this challenge and aims at carrying out further studies in this area of research. It aims to design MAC schemes that offer improved efficiency and security over popular MAC schemes based on hash functions, notably NMAC and HMAC [13].

1.4 Research Results

This thesis contributes the following research results to the current state of art in the analysis, design and applications of hash functions:

1. In Chapter 2, generic attacks on the Merkle-Damgård hash functions are reviewed. The new contributions of this chapter are classification for length extension generic attacks on hash functions and classification of multi-block collision attacks on hash functions. Part of these new results are published in [88, 89].

2. In Chapter 3, a new hash function construction family called **3CG** is proposed as a variant for the Merkle-Damgård hash function construction providing better security against the known generic attacks and multi-block differential collision attacks on the Merkle-Damgård hash functions. Linear checksum variants of **3CG** called **3C** and **3C+** are designed and analysed with respect to both kinds of attacks. The results show that these linear variants provide protection against most of the known generic attacks on Merkle-Damgård hash functions. These results are published in [88, 89].

3. In Chapter 4, new cryptanalytical tools are developed to perform the generic attacks on the **3C** and **3C+** constructions proposed in Chapter 3. In addition, some other linear checksums for the Merkle-Damgård hash functions are proposed and analysed using the new cryptanalytical techniques. This chapter shows that linear checksums of the chaining values or message

blocks or both to the Merkle-Damgård hash do not offer any better protection against multicollision, 2$^{\text{nd}}$ preimage and herding attacks that are possible on the Merkle-Damgård hash functions. In addition, this chapter improves the known generic multi 2$^{\text{nd}}$ preimage attack on the Merkle-Damgård hash functions and cascade constructions. Some results of this chapter were presented at the Rump session of Crypto'06 [136].

4. In Chapter 5, "iterated halving (IH)" approach to design hash functions is proposed and partly analysed. These results are published in [87].

5. In Chapter 6, the known preimage attack on the cellular authentication and voice encryption (CAVE) hash function is improved. These results are published in [90, 195].

6. In Chapter 7, the analysis and design issues of MAC schemes based on hash functions are covered in depth. The new contributions of this chapter are listed below and most of them are published in [84]:

   - The provably secure MAC schemes NMAC and HMAC are analysed using the weaker assumptions on the hash functions than stated in their original proofs of security.
   - New variant to NMAC called NMAC-1 is proposed to attain efficiency for short messages.
   - A new variant to NMAC called modified NMAC (M-NMAC) is proposed and analysed. It is shown that M-NMAC gives better protection than NMAC against complete key recovery attacks.
   - Security proofs for the NMAC and HMAC functions as pseudorandom functions (PRFs) are provided.
   - A new single key MAC function based on hash functions (called one keyed NMAC (O-NMAC)) is proposed and analysed against known attacks. A possible approach that could be adopted as the security proof for this scheme is given.

7. In Chapter 8, side channel cryptanalysis of HMAC and NMAC schemes is studied. This study shows that HMAC and NMAC functions based on the standard hash functions MD5 and SHA-1 are vulnerable against side

channel attacks. The M-NMAC function proposed in Chapter 7 is analysed against side channel attacks. The significant result of this chapter is that M-NMAC with hash functions such as MD5 and SHA-1 is secure against side channel attacks. These results are presented at [91].

8. Chapter 9 considers legal and practical implications that could arise when the security of information processing applications is undermined due to the flaws in the hash functions used in those applications. These results are published at [85, 86].

9. Chapter 10 concludes the thesis with some ideas for potential future work in the field of hash functions.

Chapter 2

An Overview of Merkle-Damgård Hash Function Construction

This chapter defines a hash function and provides its fundamental properties. It introduces the Merkle-Damgård iterated hash function construction along with the hash functions designed using its design principles. It provides a classification of attacks on hash functions. The following are the new contributions in this chapter.

- A high-level classification of attacks on hash functions (Section 2.3).
- New observations on the multi-block collision attacks on hash functions (Section 2.5).
- Classifies length extension attacks on hash functions (Section 2.4)

This chapter is organised as follows: In Section 2.1, definition and properties for hash functions are given. In Section 2.2, iterated hash functions are introduced and Merkle-Damgård iterative structure to design hash functions is explained. Section 2.3 classifies attacks on hash functions. In Section 2.4, generic attacks on hash functions are explored in great depth. Section 2.5 provides new observations on the multi-block collision attacks on hash functions.

2.1 Definition and Properties

Hash functions compress an arbitrary length input message, without the provision of any secret parameter, into a fixed length t-bit output value called a message digest or hash value or hash code. A hash function H is defined as $H : \{0,1\}^* \rightarrow \{0,1\}^t$. A hash computation on the message x is expressed as $H(x) = y$. Given a message x, the computation of y from it must be easy. A t-bit hash function means the output of the hash function is t bits.

The following are the fundamental security properties of cryptographic hash functions [186]:

1. **Pre-image resistance:** A hash function H is said to be pre-image resistant if for any given digest y of H, it is “computationally infeasible” or “hard” to find a message x such that $H(x) = y$. In other words, it must be hard to invert the hash function from y to get x corresponding to x. This property is also referred to as one-wayness.

2. 2nd **pre-image resistance:** A hash function H is said to be 2nd pre-image resistant if for any given message x, it is “computationally infeasible” or “hard” to find another input message x' such that $x' \neq x$ and $H(x) = H(x')$.

3. **Collision resistance:** A hash function H is said to be collision resistant if it is “computationally infeasible” or “hard” to find any two messages x and x' such that $x \neq x'$ and $H(x) = H(x')$.

A hash function which satisfies the first two properties is called a “one-way” hash function (OWHF) whereas one which satisfies all three properties is called a “collision resistant” hash function (CRHF) [228]. For most CRHFs that arise in practice such as MD4 family of hash functions, it is reasonable to assume that collision resistance implies preimage resistance [186, p.330]. However, pathologically this is not true as, for example, an identity function on fixed length inputs is a collision resistant and 2nd-preimage resistant but not preimage resistant. Hence, it is not an obligation to include preimage resistance as part of a CRHF though a CRHF almost has always the additional property of preimage resistance [186, p.325].

In the literature, the collision resistance property is referred to as collision freeness [53] or strong-collision resistance [186], 2^{nd} preimage resistance is called weak collision resistance[1] and preimage resistance is referred to as one-wayness [186]. Unless stated explicitly, this thesis uses the previous terminology while referring to these properties.

Apart from these properties, it is expected that a good hash function will satisfy the so-called certificational properties [186]. A certificational property intuitively appears desirable, although it cannot be shown to be a necessary property of the hash function. Two main certificational properties are given below:

1. **Near-collision resistance:** A hash function is said to be near-collision resistant if it is hard to find any two messages x and x' such that $x \neq x'$ and $H(x) \oplus H(x') = \Delta$ for some small difference Δ.
2. **Partial-preimage resistance:** A hash function satisfies this property when the difficulty of finding a partial preimage for a given digest is the same as that of finding a full preimage using that digest. It must also be hard to recover the whole input even when part of the input is known along with the digest.

Though near-collision resistance is listed as a certificational property, violation of this property could lead to finding full collisions for the hash functions with truncated outputs where only part of the output is used as the digest [134]. For example, a collision resistant 256-bit hash function with chopped 32 right most bits is not a near-collision resistant if near-collisions are found for the 256-bit digest where the left most 224 bits are equal.

2.2 Iterated Hash Functions

Hash functions used in information processing applications such as computation of digital signatures must be efficient. This is achieved by designing them in the iterative mode of operation where a function which accepts fixed length input messages is iterated until an arbitrary length message is processed completely. This is undoubtedly the reason for the popular hash functions such as MD5 and SHA-1 to be in the iterative mode of operation of their respective compression

[1]This is different from the weakly collision resistance property described in [13] and in Chapter 7 in the context of message authentication codes.

functions [2]. In this section, the Merkle-Damgård iterative construction which has been a popular design framework used to design cryptographic hash functions is explained.

2.2.1 Merkle-Damgård Construction

At Crypto'89, Ivan Damgård [53] and Ralph Merkle [187] independently proposed a similar iterative structure to construct a collision resistant cryptographic hash function using a fixed length input collision resistant compression function. Since then, this iterated design has been called the Merkle-Damgård construction which influenced the designs of popular hash functions such as MD4 [252], MD5 [253], SHA-0 [206] and hash functions SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 from the secure hash standard [211] [3].

Damgård and Merkle have independently provided theorems in their papers [53, 187] that show that if there exists a fixed-length input collision resistant compression function $f : \{0,1\}^b \times \{0,1\}^t \rightarrow \{0,1\}^t$ then one can design a variable-length input collision resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^t$ by iterating that compression function. That is, if an attacker finds a collision to the hash function it means that the attacker would have obtained a collision to the compression function some where in the iteration. Hence, if the compression function is vulnerable to any attack then so is the iterated hash function but the converse of this result is not true in general [228]. For example, the generic attacks discussed in Section 2.4 work on all hash functions following the Merkle-Damgård iterative construction without looking at the weaknesses in the compression functions.

While Damgård uses a complexity theoretic approach in proving the result, Merkle uses a more practical approach assuming the random behaviour of a block cipher to construct hash functions. In this section, Damgård's approach is discussed as the practical hash functions [4] such as MD4, MD5 etc. closely follow

[2]In this thesis, the mode of operation of compression function is also referred as mode of operation of hash function.

[3]In the rest of the thesis, unless stated otherwise, the term "hash function" refers to hash functions following the Merkle-Damgård iterative mode of operation of the compression function or hash function.

[4]A practical hash function is a function that is designed for actual practical use. For example, for practical hash functions there will not be any constraints on the format of the messages to be processed unlike in Damgård's construction. MD4 family of hash functions are the examples of practical hash functions.

this approach.

Damgård's Approach

Using this approach, one can construct a collision resistant hash function on variable length inputs defined as $H : \{0,1\}^* \times \{0,1\}^t \rightarrow \{0,1\}^t$ assuming the existence of a fixed length input collision resistant compression function f defined as $f : \{0,1\}^b \times \{0,1\}^t \rightarrow \{0,1\}^t$. The following two cases are observed to process a message M.

- Case-1 [$b - t > 1$:] The message M is split into blocks of size $b - t - 1$ bits and the incomplete last block is padded with 0 bits. Let c be the number of 0 bits needed. The message blocks are denoted by $M_1, M_2, \ldots, M_{L-1}$. To this sequence of blocks, one extra block M_L containing the binary encoded representation of c prefixed with an appropriate number of 0 bits is added. A sequence of t-bit intermediate chaining values H_i are denoted as follows:

$$H_1 = f(0^{t+1}||M_1)$$
$$H_i = f(H_{i-1}||1||M_i) \text{ for } i = 2, 3, \ldots, t.$$

  The hash value $H_t = H(M)$.

- Case-2 [$b - t = 1$:] In this case, the message M to be hashed is processed in blocks M_i each of length 1 bit. The intermediate chaining values are defined by:

$$H_i = f(H_{i-1}||M_i) \text{ where } i = 1, 2, \ldots, t \text{ and } H_0 \text{ is a uniformly distributed } t\text{-bit state.}$$

  The hash value $H_t = H(M)$.

2.2.2 Hash Functions following the Merkle-Damgård Structure

A collision resistant cryptographic hash function H following the Merkle-Damgård structure is a function which processes a message $M \in \{0,1\}^*$ to outputs of fixed length $\{0,1\}^t$. The specification of H includes the description of the compression function $f : \{0,1\}^b \rightarrow \{0,1\}^t$, initial state value (IV) and a padding procedure [186, 228]. Every hash function fixes the IV with an upper bound on the size of the input message M to be processed. This IV is called the fixed IV. Let the upper bound on the length of the message be $2^l - 1$ bits. The message M is padded with a 1 bit followed by 0 bits until the padded message is l bits short of a full block of b bits. The length of the original message in a binary encoded format is filled in those l bits. This compound message M is represented in blocks of b bits each as $M_1, \ldots, M_{L-1}, M_L$. Adding the length of the message in the last block is called Merkle-Damgård strengthening [164][5]. Each block M_i is iterated using the compression function f computing $H_i = f(H_{i-1}, M_i)$ where $i = 1$ to L and producing the hash value $H_{IV}(M) = H_L$ as shown in Figure 2.1. The Merkle-Damgård strengthening prevents trivial attacks on hash functions such as finding collisions for the messages with different initial states as given by $H_{IV}(M_1||M_2) = H_{IV^*}(M_2)$ where $IV^* = H_{IV}(M_1)$.

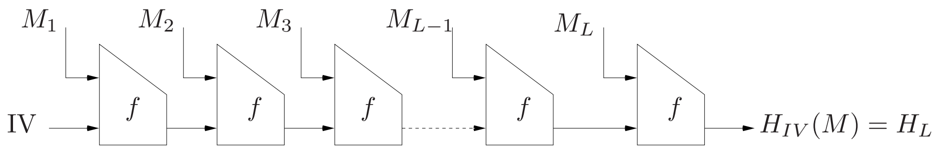


Figure 2.1: The Merkle-Damgård construction

By treating the fixed-length input compression function as a cryptographic primitive, the Merkle-Damgård construction used to design hash functions to process arbitrary length inputs can be viewed as a mode of operation for the compression function [174].

From the performance point of view, the design principle of a Merkle-Damgård hash function H is to iterate f from a fixed initial state H_0 until the whole message M with an upper bound of $2^l - 1$ bits on its length is processed. From the security point of view, its design principle is if f is collision resistant then H is also collision resistant. MD4 is the first hash function designed by Rivest [252, 254] which follows the design principles of Merkle-Damgård iterative structure. All hash functions that have evolved following the design principles of MD4 are considered to be MD4 family of hash functions (see Appendix B).

[5]Some papers (for example, [12,13,109]) mention placing the Merkle-Damgård strengthening always in a separate block. However, implementations of MD4 family of hash functions place the length encoding in the last padded block if there is enough space.

2.2.3 Differences between Damgård's Design and Practical Hash Functions

The original Damgård's construction discussed in Section 2.2.1 differs from the operation of practical hash functions in some aspects. In the first case of Damgård's construction provided in Section 2.2.1, the way the first message block is processed is different from the way the other blocks are processed as the first block contains a 0 bit in the relevant position instead of a 1 bit. This allows the exclusion of impossible situations where the adversary has found a collision for two messages of the format $(M^*||M, M)$ [54]. This condition was considered in the proof of security of the Damgård's scheme [53].

There are other ways to exclude this type of collision. In particular, it cannot occur if the adversary cannot find a preimage for the compression function with the IV of the hash function, which is chosen to be 0^{t+1} in the case-1 of Damgård's construction. One can also choose the IV randomly, then under certain conditions on the compression function f, hardness of finding a preimage follows from the collision resistance of the compression function. Most hash functions such as MD5 and SHA-1 used in practice do not have this extra bit marking at the start of a message, so they are in fact implicitly based on the assumption that an adversary cannot find a preimage for the compression function of the hash function with IV [54].

Damgård's construction also differs from practical hash functions in the way the message is padded. As shown in Section 2.2.2, practical hash functions append the length of the entire message in a binary encoded format to the padded message. This means that the length must be a number small enough to be contained in a block. So, one has to assume that no message in any application is longer than that. For example, MD5, SHA-1 and SHA-256 hash functions impose an upper bound of $2^{64} - 1$ bits on the length of the message to be hashed. Hence, these hash functions represent the message in a 64-bit encoding format. Similarly, hash functions SHA-384 and SHA-512 impose an upper bound of $2^{128} - 1$ bits on the length of the message to be processed. Hence, these hash functions represent the message in a 128-bit encoding format. This is perfectly reasonable in all applications where these hash functions are used, so this limitation is not a practical problem.

However, Damgård's design (discussed in Section 2.2.1) aims at proposing a construction where there is no bound on the message length by appending the message with a block containing a number of padded 0 bits instead of the length encoding of the messages as done in the MD4 family of hash functions.

2.2.4 Compression Functions of Hash Functions

The compression functions of hash functions such as MD5, SHA-1 and SHA-256 use Davies-Meyer design principle [57,178,296][6] of using a block cipher algorithm in a feed-forward mode to achieve one-wayness. A Davies-Meyer compression function is defined as $H_i = g(H_{i-1}, M_i) + H_{i-1}$ where $+$ could be any group operation and g is the block cipher with the message block input M_i working as the secret key and the previous chaining value H_{i-1} as the message input as shown in Figure 2.2. These hash functions use modular addition as the group operation in the Davies-Meyer compression function.

These compression functions are vulnerable against the fixed point attacks where a pair (H_{i-1}, M_i) can be found such that $H_{i-1} = f(H_{i-1}, M_i)$ [199]. There exists *one and only one* fixed point for every message block for these compression functions. The attacker finds a fixed point for the compression function f by choosing any message block M_i and inverting the function g as given by $g^{-1}(0, M_i)$ to get a state H_{i-1} such that $f(H_{i-1}, M_i) = H_{i-1}$. This costs just one inverse operation which is equivalent to the computation of one compression function iteration. Davies-Meyer compression function is one of the 12 provably secure PGV (named with the surnames of Preneel, Govaerts and Vandewalle [232]) compression functions satisfying the security properties of hash functions when the underlying block cipher is assumed as ideal [35].

For hash functions based on block ciphers, the amount of data compressed for each application of the block cipher is measured by hash rate. For an iterated hash function based on a block cipher, it is defined as the number of message blocks processed by one encryption or decryption of the block cipher [146, 150]. The hash rate of the Davies-Meyer scheme is 1. Similarly, if a compression function $f : \{0,1\}^b \rightarrow \{0,1\}^t$ requires s calls to either a block cipher of block size n bits or a smaller compression function with inputs of n bits then the rate r of f is defined as the ratio $r = (b - t)/(s \times n)$ [148].

[6]This design was attributed to Davies in [295, 296] and to Meyer by Davies in [57]. As pointed out in [146], this scheme was never proposed by Davies and as pointed out in [151] it was apparently known to Meyer and Matyas.

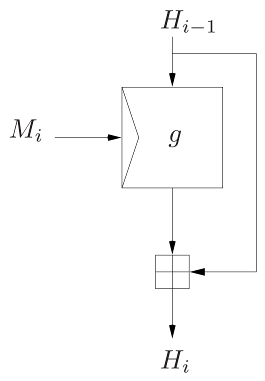


Figure 2.2: Davies-Meyer compression function

2.3 Attacks on Hash Functions

A high-level classification of attacks on hash functions is shown in Figure 2.3.

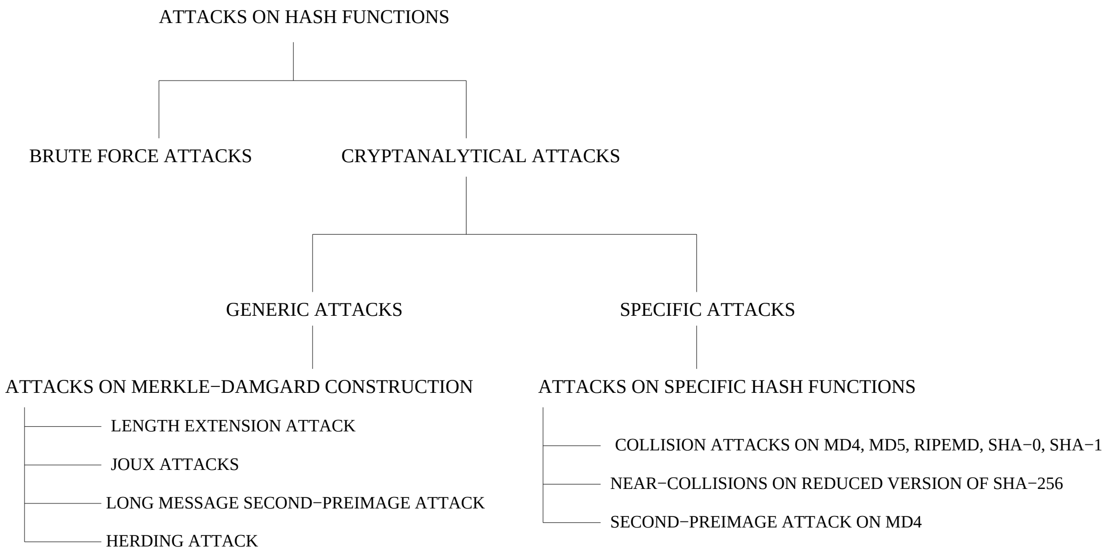

ATTACKS ON HASH FUNCTIONS
BRUTE FORCE ATTACKS
CRYPTANALYTICAL ATTACKS
GENERIC ATTACKS
SPECIFIC ATTACKS
ATTACKS ON MERKLE−DAMGARD CONSTRUCTION
LENGTH EXTENSION ATTACK
JOUX ATTACKS
LONG MESSAGE SECOND−PREIMAGE ATTACK
HERDING ATTACK
ATTACKS ON SPECIFIC HASH FUNCTIONS
COLLISION ATTACKS ON MD4, MD5, RIPEMD, SHA−0, SHA−1
NEAR−COLLISIONS ON REDUCED VERSION OF SHA−256
SECOND−PREIMAGE ATTACK ON MD4

Figure 2.3: Attacks on Cryptographic Hash Functions

Attacks on hash functions can be classified into two broad categories: brute force attacks and cryptanalytical attacks.

2.3.1 Brute force Attacks

Brute force attacks work on all hash functions independent of their structure and any other working details. They are similar to exhaustive search or brute-force key recovery attacks on the encryption schemes to extract the secret key of the encryption scheme. The security of any hash function lies in its output bit size which must be large enough to resist the following attacks:

1. Brute-force preimage attack: In this attack, for a given t-bit digest y of the hash function H, the attacker evaluates H with every possible input message until the attacker obtains the value y. For an ideal hash function, it takes for the attacker about 2^t computations of the hash function to find the preimage with a significant probability.

2. Brute-force 2^{nd} preimage attack: In this attack, for a given message x and the hash function H, the attacker trials H with every possible input message $x' \neq x$ until the attacker obtains the value $H(x)$. For an ideal hash function, it takes for the attacker about 2^t computations of the hash function to find the 2^{nd} preimage with a significant probability.

3. Brute-force collision attack: In this attack, for a given hash function H, the attacker tries to find two messages x and x' such that $x \neq x'$ and $H(x) = H(x')$. For an ideal hash function, it takes about $2^{t/2}$ computations of the hash function to find the pair (x, x') which produces a collision with a probability of 50% due to the birthday paradox[7]. This attack is also called birthday attack. However, for widely used hash functions, this attack cost factor is smaller than that as their outputs are not uniformly distributed [17].

These brute-force attacks can always be attempted, however they are not considered as a break unless the required number of evaluations of the hash function is significantly less than both the strength estimated by the designer of the hash function and that of hash functions of similar parameters with ideal strength [186, p.371]. To perform real attacks on hash functions, they need to be analysed using some specific approaches or methods and such attacks are called cryptanalytical attacks. These attacks are discussed in the following section.

2.3.2 Cryptanalytical Attacks

A cryptanalytical attack (or simply an attack) on a hash function is defined as some analytical approach or method used to violate at least one of its security

[7]Note that in the rest of the thesis, the probability value for a birthday attack is assumed and hence will not be mentioned.

properties. A preimage attack (resp. 2^{nd} preimage attack, collision attack) violates the preimage resistance (resp. 2^{nd} preimage resistance, collision resistance) of a hash function.

Due to fixed size of the hash values compared to much larger size of the messages, collisions must exist in hash functions. However, for the security of the hash function they must be computationally infeasible to find. Note that collisions in hash functions are much easier to find than preimages or 2^{nd} preimages.

Informally, a hash function is said to be "broken" when a reduced number of evaluations of the hash function compared to the brute force attack complexities and the strengths estimated by the designer of the hash function are used to violate at least one of its properties immaterial of the computational feasibility of that effort. For example, assume that it requires 2^{90} evaluations of the hash function to find a collision for a 256-bit hash function. Though it is impractical to generate this amount of computational power today, the hash function is said to be broken as this factor is less than the 2^{128} evaluations of the hash function required by the birthday attack. This theoretical break on the hash function is also termed an "academic break" on the hash function [55].

It should be noted that hash functions are easier to attack practically than encryption schemes because the attacker does not need to assume any secrets and the maximum computational effort required to attack the hash function is only upper bounded by the attacker's resources not users gullibility. This is not the case with block ciphers where the maximum practical count of executions of the block algorithm is limited by how much computational effort the attacker can get the user to do [133].

Cryptanalytical attacks on hash functions are classified into two categories: generic attacks and specific or short-cut attacks as shown in the Figure 2.3. The attacks that work on a general hash function construction are called generic attacks. For example, attacks on the Merkle-Damgård construction that work on all hash functions designed using it are the generic attacks on the Merkle-Damgård hash functions. The generic attacks on Merkle-Damgård construction are discussed in Section 2.4. The attacks that work on specific hash functions designed following a particular hash function design framework are called specific attacks. For example, collision attacks on the specific hash functions MD4 [286], MD5 [285, 291], SHA-0 [27, 289] and SHA-1 [290] that follow the principles of Merkle-Damgård iterative structure come under the category of specific attacks.

An evolution of hash functions following the Merkle-Damgård iterative structure explained in Appendix B provides an overview of attacks on specific hash functions.

2.3.3 Collision Attacks on the Compression Functions

Collision attacks on the compression functions used in the Merkle-Damgård or similar iterated hash functions are classified based on the IV used as follows [186, p.372]:

1. Collision attack: Collisions using an IV specified in the specification of the hash function (fixed IV) for two distinct messages (e.g. [285]). They are called *Type-1* collisions.

2. Semi-free-start collision attack: Collisions using the same random (or arbitrary) IV for two distinct message inputs (e.g. [68]). They are called *Type-2* collisions.

3. Pseudo-collision attack: Collisions using two different IVs for two distinct message inputs (e.g. [164, 186]). They are called *Type-3* collisions. These collisions are also called free-start collisions. The pseudo-collision attack is also called free-start collision attack [146].

4. Special pseudo-collision attack: Collisions using two different IVs on the same message block. They are called *Special Type-3* collisions. Note that the paper [60] uses the term pseudo-collisions on MD5 to describe collisions with two different IVs and the same message block. This contradicts with the definition used in [164, 186]. Hence, this thesis classifies collisions of form described in [60] as *Special Type-3* collisions.

The above collision attacks on the compression functions also apply to their hash function iterative modes. The *Type-1* collisions on the hash functions are considered as strict collisions as they apply to the hash function specification directly. These collision attacks will have practical importance when they are used to attack the applications where the hash functions susceptible to *Type-1* collisions are deployed in those applications. For example see [130, 191] for the practical impact of collision attacks on MD5. Semi-free-start collision (resp. pseudo-collision) attacks on hash functions create doubts on the long-term security of the hash function. For example, as explained in Appendix B, the collision attacks on the MD5 hash function in the early and mid 90s [60, 68] fall under *Type-2* or *Type-3* category. The collision attacks on MD5 found in 2004 [285, 291] fall under *Type-1* category. Recently, it was argued that MD5 should have been in the "must abandon" or "dead" category 10 years ago when it was shown that it has weak collision resistance properties [157]. Hence it is recommended to switch to strong hash functions when *Type-2* ,*Type-3* and *special Type-3* collision attacks are demonstrated on a hash function.

2.3.4 Collision finding Algorithms

A collision finding algorithm for a hash function is an attack algorithm that produces a collision to the hash function for a given state. Namely, when such an algorithm is called with a state, it finds and returns two messages that produce a collision using that state. Similarly, a collision finding algorithm for a compression function produces two distinct message blocks that give a collision when it is called with a state.

A collision finding algorithm used to find collisions in a t-bit hash function or a compression function can be classified into two categories:

1. Brute force collision finding algorithm: This algorithm uses a brute-force attack to find two different messages that produce a collision when it is called with some state. By evaluating a hash function for about $2^{t/2}$ times, such an algorithm finds a collision with a probability of 50%.

2. Cryptanalytic collision finding algorithm: This algorithm finds collisions in a hash function in an effort less than birthday attack. Such algorithms are specific for each t-bit hash function. The existence of such an algorithm depends on the strengths of the hash function. For example, SHA-0 and RIPEMD-160 both are 160-bit hash functions. However, there is a cryptanalytic collision finding algorithm which produces a collision for SHA-0 in 2^{39} SHA-0 operations [289], but so far there is no algorithm that produces a collision for RIPEMD-160 with an effort less than 2^{80} computations of RIPEMD-160. Often, cryptanalytic collision finding algorithms are based on differential cryptanalysis techniques, a well established approach used to analyse block ciphers [30]. For example, the collision attacks on MD4 [286],

MD5 [285, 291], SHA-0 [27, 289] and SHA-1 [290] employ collision finding algorithms.

The collision finding algorithms works on either single message blocks or multiple message blocks. If a collision is produced using single message blocks then it is basically a collision for the compression function. Such an attack is also called single-block collision attack. If a collision is produced for more than 1 block then it is basically a collision for more than one consecutive iteration of the compression function. Such a collision is called multi-block collision. Multi-block collision attacks are explained in great depth in Section 2.5 of this chapter.

2.4 Generic Attacks on the Merkle-Damgård Construction

Generic attacks on hash functions work on all hash functions based on a single design framework or mode of operation of the compression function such as iterated hash function constructions. The computational complexity of a generic attack on iterated hash functions depends on the parameters like hash size and internal state size and are independent of the internal working details of the compression function. The length-extension [79], 2^{nd} preimage [58, 138] and herding [137] attacks are generic in nature as they work on all hash functions following the Merkle-Damgård iterative structure or some other similar iterative structures. A generic attack on a hash function construction also answers the question: "If an attacker can perform a brute force collision attack on the compression function of a t-bit iterated hash function, what other properties can the attacker violate using that collision attack?"

This section covers generic attacks that work on all hash functions following the Merkle-Damgård construction. For some of these attacks, the attackers are probabilistic algorithms and the complexity of the attacks are measured in expected running times. Formally, the symbol O is used for the expected running time and is asymptotically "at most" and Ω is used for the expected running time and is asymptotically "not less than".

2.4.1 Length Extension Attacks

Some applications of iterated hash functions such as naive authentication schemes can be broken by exploiting the iterative structure of hash functions using length extension attacks. For example, length extension attacks can be used to break secret perix MAC scheme where the attacker computes the authentication tags of messages without the knowledge of the secret key [79, p.90]. The secret-prefix MAC scheme is analysed against this attack in the Section 7.1 of Chapter 7. Hence, the study of these attacks on the iterated hash functions is quite significant.

In the cryptographic hash function literature, authors have defined straightforward length extension attacks or extension attacks on iterated hash functions in two ways using the same nomenclature. This section categories extension attacks into two types following the literature closely.

Consider a message M processed by a hash function H. Two broad assumptions on the format of the message M can be made as follows:

1. The message M completely specifies the actual data, meaning all blocks contain the data that is hashed and there is no length encoded padding. If the last block is incomplete, 0 bits would have been appended to make it a data block. The hash result of such a message M is the intermediate or internal hash value and is denoted by $H'(M)$.

2. The message M not only contains the actual data to be hashed, but also the Merkle-Damgård strengthening in the last block. The result $H(M)$ of such an M is the hash value or digest.

The reference [79] uses the first assumption on the format of the message. Here the attacker is given an intermediate or internal hash value $H'(M)$ and $|M|$ but not M itself. With this information, the attacker can process at least one new data block using $H'(M)$ as the starting state. Let this new data block be denoted by M_L. When the hash function uses Merkle-Damgård strengthening, it is straight forward for the attacker to encode the length of the total message in the additional last block M_{L+1} and compute a new hash value for the message $M||M_L||M_{L+1}$ without the need to know the original message M. This attack is a *Type-A* extension attack. The *Type-A* extension attack is discussed in [173, 174].

The second type of extension attack is based on the second assumption where the attacker is given the final hash value $H(M)$ and $|M|$ but not M itself. Note that $H(M)$ is the digest of a message M which was already padded with the Merkle-Damgård strengthening. Now the attacker uses $H(M)$ and the new message block M_L to compute the new hash value for the message $M||M_L$. This compound message is again padded along with the Merkle-Damgård strengthening of the message $M||M_L$. So, in total, padding is performed twice for the hash function. This attack is a *Type-B* extension attack. This form of extension attack is discussed in [124, 174].

2.4.2 Joux Generic Attacks

Multicollision Attack:

Joux [126] described a generic multicollision attack on the Merkle-Damgård hash function where he has shown that constructing 2^d-collisions costs d times as much effort as building ordinary 2-collisions. A 2^d-collision means a collision of 2^d different messages. In this attack, the attacker assumes the existence of a collision finding algorithm which produces a collision for the compression function f with every call to it. Namely, the attacker calls this algorithm with a state H_{i-1} and the algorithm returns two message blocks M_i and N_i such that $M_i \neq N_i$ and $f_{H_{i-1}}(M_i) = f_{H_{i-1}}(N_i)$. Such a collision finding algorithm can either be due to a brute force collision attack or a cryptanalytic collision attack. A brute force collision finding algorithm finds collisions for f in about $2^{t/2}$ computations of the compression function. A cryptanalytic collision finding algorithm finds collisions for f in less than $2^{t/2}$ computations of the compression function.

A 2^4-collision attack is demonstrated in Figure 2.4 where 16 different messages starting from the initial state H_0 of the hash function are mapped to the same digest H_4. The attacker first calls the collision finding algorithm to the compression function f with the state H_0. The algorithm returns two blocks $M_1 \neq N_1$ such that $f_{H_0}(M_1) = f_{H_0}(N_1) = H_1$. Then the attacker calls the algorithm with the state H_1 and the algorithm returns two blocks M_2 and N_2 such that $M_2 \neq N_2$ and $f_{H_1}(M_2) = f_{H_1}(N_2) = H_2$. H_2 is then used as the state and calls the algorithm which returns two blocks M_3 and N_3 such that $M_3 \neq N_3$ such that $f_{H_2}(M_3) = f_{H_2}(N_3) = H_3$. Finally, the collision finding algorithm is called with H_3 and obtains the collision $f_{H_3}(M_4) = f_{H_3}(N_4) = H_4$ and thus a 2^4-collision has occurred where 16 different messages are mapped to the digest H_4. Note that it takes time $O(4 \times 2^{t/2})$ to find a 16-collision for a collision resistant compression function.

Using this technique, 2^d-collisions can be found in time $O(d \times 2^{t/2})$ instead of $\Omega(2^{t(2^d-1)/2^d})$ for a compression function f using a brute-force collision finding algorithm. The complexity of the multicollision attack would be less than this factor if the collision finding algorithm is a cryptanalytic collision finding algorithm.

A characteristic of this multicollision technique is that all the colliding messages are of the same length. Hence, the Merkle-Damgård strengthening employed in the last blocks of two colliding streams does not thwart the multicollision attack due to this feature because a collision propagates even after adding the Merkle-Damgård strengthening.

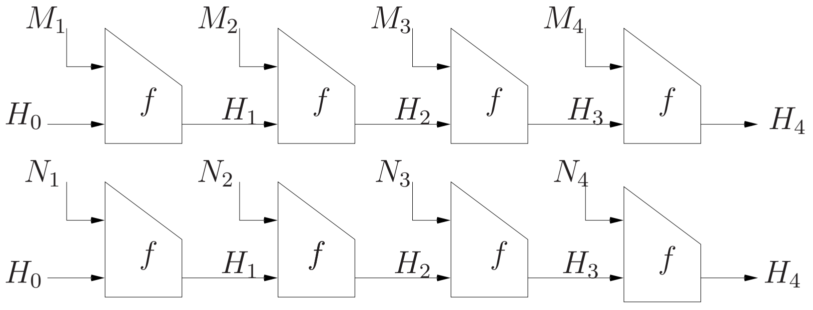


Figure 2.4: Multicollisions on the Merkle-Damgård hash function

Multi (2nd)-preimage Attack:

The multicollision technique can be used as a tool to find multiple (2nd) preimages[8] at a cost less than the brute force complexity of finding multiple (2nd) preimages. In the multi preimage attack, the attacker is given a message digest H_{final} of the hash function H and is required to find multiple preimages for H using H_{final}. To find 2^d preimages for H, the attacker first finds 2^d collisions using the multicollision technique described above to get the multicollision H_d. The multicollision value H_d is then used as the chaining value to find a message block M_{final} such that $f_{H_d}(M_{final}) = H_{final}$. The total cost to find 2^d preimages is $O(d \times 2^{t/2} + 2^t)$ instead of $\Omega(2^d \times 2^t)$.

In the multi 2nd preimage attack, the attacker is given a message M and the task of the attacker is to find multiple second preimages for H using $H(M)$. To

[8]The notation (2nd) preimage indicates preimage and 2nd preimage.

find 2^d second preimages, the attacker first finds a 2^d multicollision value H_d, then tries to find a message block M_{final} such that $f_{H_d}(M_{final}) = H(M)$. The total cost to find 2^d 2^{nd} preimages using this technique is $O(d \times 2^{t/2} + 2^t)$ instead of $\Omega(2^d \times 2^t)$.

The multicollision trick on a single hash function can also be used to find multicollisions for the cascade hash function constructions with an effort less than that of brute force effort [126]. Preneel [228] has proposed cascade hash function designs where hash values of two or more different hash functions or different instances of the same hash function are concatenated to increase the security level of hash functions against collision attacks at the cost of a decreased performance. If H and G are two different hash functions or two instances of the same hash function producing hash values H_t and G_t by processing a message M then their cascade construction $H||G$ produces a large hash value $H_t||G_t$. Refer to [126] for the attack descriptions on the cascade construction. The complexities of Joux attacks are compared with the brute force complexities in Table 2.1.

Table 2.1: Joux attacks

Attack	Brute-force	Joux
2^d-collision on H $(d \geq 2)$	$\Omega(2^{t(2^d-1)/2^d})$	$O(d.2^{t/2})$
2^d-preimages on H $(d \geq 1)$	$\Omega(2^d.2^t)$	$O(d.2^{t/2} + 2^t)$
2^d-second preimages on H $(d \geq 1)$	$\Omega(2^d.2^t)$	$O(d.2^{t/2} + 2^t)$
2-collision on $H\|\|G$ & $H_t \leq G_t$	$\Omega(2^{(H_t+G_t)/2})$	$O(G_t.2^{H_t/2} + 2^{G_t/2})$
2-collision on $H\|\|G$ & $G_t \leq H_t$	$\Omega(2^{(G_t+H_t)/2})$	$O(H_t.2^{G_t/2} + 2^{H_t/2})$
1-preimage on $H\|\|G$ & $H_t \leq G_t$	$\Omega(2^{G_t+H_t})$	$O(G_t.2^{H_t/2} + 2^{G_t} + 2^{H_t})$
1-preimage on $H\|\|G$ & $G_t \leq H_t$	$\Omega(2^{H_t+G_t})$	$O(H_t.2^{G_t/2} + 2^{H_t} + 2^{G_t})$
1-2^{nd} preimage on $H\|\|G$ & $H_t \leq G_t$	$\Omega(2^{G_t+H_t})$	$O(G_t.2^{H_t/2} + 2^{G_t} + 2^{H_t})$
1-2^{nd} preimage on $H\|\|G$ & $G_t \leq H_t$	$\Omega(2^{H_t+G_t})$	$O(H_t.2^{G_t/2} + 2^{H_t} + 2^{G_t})$

2.4.3 Generic 2^{nd} preimage Attacks

In the 2^{nd} preimage attack on a t-bit hash function H, the attacker finds a second preimage N for a given target message M such that $M \neq N$ and $H(M) = H(N)$ with an effort less than 2^t computations of H. There exists a long message 2^{nd}-preimage attack on hash functions without Merkle-Damgård strengthening where the attacker aims to find a second preimage for a long target message M of $2^q + 1$ message blocks. The attacker does this by finding a linking message block

M_{link} such that the digest $f_{IV}(M_{link})$ of M_{link} matches one of the intermediate chaining states H_i obtained in the hashing of M where $i \in [1, 2^q]$. The attack costs about 2^{t-q} calls to the compression function as the attacker tries about that many message blocks M_{link} such that $f_{IV}(M_{link})$ matches the state H_i for some M_{link} and sets the second preimage $N = M_{link}$. However, this attack does not work on the complete hash function as the Merkle-Damgård strengthening used in the processing of both M_{link} and M thwarts the attack on the complete hash function.

Dean [58] and later Kelsey and Schneier [138] have improved this generic attack using the notion of *expandable message* such that it bypasses the defense provided by the Merkle-Damgård strengthening in the above attack. In the following paragraphs, this attack is explained by following the nomenclature provided in [138].

Expandable messages are multicollisions on the messages of different lengths excluding the Merkle-Damgård strengthening used in the last blocks. Following [138], an (a, b)-*expandable message* is a multicollision between messages of lengths a and b blocks. These *expandable messages* are constructed either by finding fixed points for the compression functions as demonstrated by Dean [58] or in a more generic way using Joux multicollision finding technique as demonstrated by Kelsey and Schneier [138].

Dean [58] has shown that one can exploit hash functions with the fixed point compression functions to find second preimages with an effort less than 2^t. The attacker first finds $2^{t/2}$ fixed points for the compression function by testing that many message blocks as given by $g^{-1}(0, M_j) = H_j$ for $j = 0$ to $2^{t/2} - 1$ and collects the pairs (M_j, H_j). Next the hash values $f_{IV}(M_i)$ are computed for the compression function f using random message blocks M_i for $i = 0$ to $2^{t/2} - 1$ and the IV of the hash function and the pairs $(M_i, f_{IV}(M_i))$ are collected. Finally, a match between two sets of hash values H_j and $f_{IV}(M_i)$ for some i and j is found and the colliding messages $(M_j, M_j||M_i)$ are returned. Kelsey and Schneier [138] explained this attack using the notion of *expandable messages* where the above collision pair was termed a $(1, 2)$-*expandable message*. An algorithm to find *expandable messages* using fixed points in the compression function is given in Appendix A.1. Messages of desired length in the multicollision can be produced by concatenating M_i to $M_j||M_i$. For a t-bit hash function H which can process a maximum of 2^d blocks, it costs about $2^{t/2+1}$ compression function computations to construct a $(1, 2^d)$-*expandable message* using this attack [138].

Kelsey and Schneier [138] have expanded this technique for any compression function without the necessity of having fixed points. It was demonstrated in [138] that finding a $(d, d + 2^d - 1)$ *expandable message* for any compression function with a t-bit state takes only $d \times 2^{t/2+1}$ effort. This is possible using a generic *expandable message* algorithm using the Joux multicollision finding technique. The procedure involves first finding a colliding pair of messages, one of one block and the other of $2^{d-1} + 1$ blocks starting from the IV of the hash function. Using this collided state as the starting state, a collision pair of length either 1 or $2^{d-2} + 1$ is found and this process is continued until a collision pair of length 1 or 2 is reached. A detailed algorithm is given in Appendix A.2. Note that the algorithm to build a $(d, d + 2^d - 1)$-*expandable message* in [138] has an error and it is corrected in Appendix A.2.1.

Once the *expandable messages* are found using either of the above techniques, the attacker performs the long message second preimage attack from the end of the *expandable message* [138]. The *expandable message* is then expanded to the desired length so that the length of the second message equals the length of the given target message M such that $H(M) = H(N)$ where N is the 2^{nd} preimage. Thus the total effort required to perform a second preimage attack on a hash function is equal to the sum of the efforts required to find *expandable messages* and linking message blocks. It was shown in [138] that finding the second preimage for a message of $2^d + d + 1$ blocks requires about $d \times 2^{t/2+1} + 2^{t-d+1}$ compression function computations using the generic *expandable message* algorithm and costs about $3 \times 2^{t/2+1} + 2^{t-d+1}$ compression function computations using the fixed point *expandable message* algorithm. The algorithm to find the second preimage is given in Appendix 2.4.3.

2.4.4 Herding Attack

Kelsey and Kohno [137] have shown that an attacker who can find many collisions using birthday attack for the Merkle-Damgård hash functions, can first commit to the digest of a message and some time later "herd" a challenged prefix part of a message to the committed digest by choosing an appropriate suffix. With this attack, they have identified an essential security property for hash functions called chosen-target forced prefix (CTFP) preimage resistance. A hash function is said to be a CTFP preimage resistant if it takes about 2^t evaluations of the hash function to find a message which connects a particular forced prefix message to a previously committed target digest. Kelsey and Kohno have named the attack that violates this property for hash functions a herding attack.

In the herding attack on a t-bit hash function H, the attacker first performs some precomputation and then outputs a t-bit digest H_{ct} called the chosen target. This is attacker's committed hash value. Later, the attacker herds a forced prefix message P to that chosen target hash value with an effort less than 2^t evaluations of the hash function by finding a suffix message S such that $H(P||S) = H_{ct}$ thus compromising the CTFP preimage resistance property. The attacker does this by first constructing a diamond structure (see Figure 2.5) of $2^{d+1} - 2$ intermediate hash values as part of the precomputation with an effort of $2^{t/2+d/2+2}$ compression function computations and outputs the final hash value H_{ct} as the chosen target. For example, a basic 2^3 diamond structure is shown in Figure 2.5 where the attacker computes $2^{3+1} - 2$ intermediate hash values as part of the precomputation.

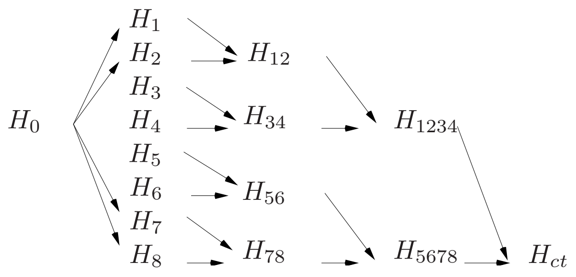


Figure 2.5: Diamond structure with $d = 3$

Later, a challenged prefix message P is herded to that hash value H_{ct} by exhaustively searching for a message S' such that $H(P||S')$ equals one of the intermediate states in the diamond structure. It costs about 2^{t-d} compression function evaluations to perform this task which is equivalent to finding the linking message block M_{link} in order to perform the second preimage attack on hash functions as explained in Section 2.4.3. Finally, the attacker finds the remaining message blocks in the diamond structure with a negligible effort from that intermediate state $H(P||S')$ such that $H(P||S'||R) = H_{ct}$ where R represents the blocks in between the states $H(P||S')$ and H_{ct} in the diamond structure[9]. This

[9]There is no need to process these R blocks as they are already processed as part of the precomputation while constructing the diamond structure.

step is demonstrated in Figure 2.6. The effort required for the herding attack is approximately equal to $2^{t-d} + 2^{t/2+d/2+2}$ evaluations of the compression function assuming that the attacker searches for the linking messages to link the prefix P to one of the hash values in the outermost level of the diamond structure. It takes about $2^{t-d-1} + 2^{t/2+d/2+2} + d \times 2^{t/2+1}$ evaluations of the compression function if the attacker tries to link the prefix P to any of the $2^{d+1} - 2$ intermediate hash values in the diamond structure as the pre-determined *expandable message* must be attached to the end of the suffix.

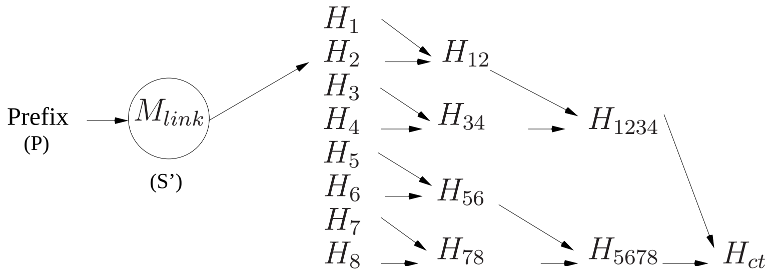


Figure 2.6: Finding a linking message and producing the suffix

2.5 Multi-block Collision Attacks on Hash Functions

A multi-block collision attack (MBCA) technique on an iterated hash function finds two colliding messages each of at least two blocks in length. The recent collision attacks on MD5 [291], SHA-0 [26, 289] and SHA-1 [290] are multi-block collision attacks where collisions are found by processing more than one message block. Since, by far, most of the possible messages are more than a single block and collisions are distributed randomly, it is fair to say that most collisions that could exist for hash functions are in fact multi-block collisions.

2.5.1 New Observations on Multi-block Collision Attacks

This section aims at developing an understanding of the multi-block collision attacks on hash functions by linking several parts of the literature. It is observed that multi-block collision attacks on hash functions can be classified into three categories based on the manner in which the compression functions are attacked. This leads to choosing particular message block formats that result in an MBCA.

For example, 2-block collision attacks can be classified into three types as shown in Table 2.2 based on the message formats chosen.

Table 2.2: Classification for 2-block collision attacks

MBCA Type	Message formats
MBCA-1	(M_1, M_2) and (M_1, N_2)
MBCA-2	(M_1, M_2) and (N_1, M_2)
MBCA-3	(M_1, M_2) and (N_1, N_2)

While the 2-block collision attacks on MD5 [291] and SHA-1 [290] belong to MBCA-3 category, the 2-block collision attack on SHA-0 [289] belongs to an MBCA-1 category. See Appendix C.1, for a brief explanation on the 2-block collision attack on MD5 by Wang *et al.* [291]. The first full 4-block collision attack on SHA-0 [26] also belongs to an MBCA-3 category except that differences in the message blocks span more than two blocks. In MBCA-3, near-collisions found after processing the first message blocks are converted to full collisions as it was demonstrated on MD5 and SHA-1. The *Type-1* collisions were (reportedly) hard to find for the compression functions of these hash algorithms based on their initial states. For example, the attacks on MD5 and SHA-1 use near-collisions obtained after processing the first distinct message blocks (M_1, N_1) as a tool to find collisions for the second distinct message blocks (M_2, N_2). This technique can be generalized to more than two blocks as the 4-block collision attack on SHA-0 [26]. Similarly, 2-block MBCA-1 and MBCA-2 attack techniques can be generalized to more than two blocks. For example, in an MBCA-1 technique [289], a few initial message blocks to be processed can be chosen to be the same to satisfy certain conditions required in the attack followed by the processing of two different message blocks that give a collision.

The collision attacks on MD5 and SHA-1 show that, establishing full collisions in these hash functions by exploiting the Merkle-Damgård iterative structure using near collisions, is easier than attacking their compression functions with fixed IV. Generating collisions in hash functions using this tool is particularly useful when no differential characteristic exists that predicts a full collision in the first block. In addition, this technique reduces the complexity of the attack when the complexity required to find a collision for the first block is really large [26]. For example, it was demonstrated on SHA-1 [290] that due to the freedom available to the attacker in generating first block near-collisions, one can maintain essentially twice the search complexity while converting those near-collisions to full collisions on the second block.

Note that, in a 2-block MBCA, collisions found for the second compression functions are basically one of *Type-2* or *special Type-3* or *Type-3* collisions as these collisions require processing of two equal (MBCA-1) or different blocks (MBCA-2 and MBCA-3) using the fixed IV of the hash function. For example, a 2-block MBCA-3 on a hash function is a combination of a near-collision and a *Type-3* collision for the compression function. The collision for the second compression function is a *Type-3* collision as it requires a particular nearly collided value to be obtained based on certain conditions essential for the attack as inputs for the second block messages. In addition, near-collisions do not need to begin after processing message blocks based on the fixed IV of the hash function. They can also be due to an arbitrary chaining value when the attacker chooses the same blocks initially and starts an MBCA-3 after processing those same initial blocks.

Table 2.3: Resistances of some compression functions

Compression function	*Type-1*	*Type-2*	*Type-3*	*Special Type-3*
MD4	NO [285]	NO	NO	-
MD5	YES	NO [68]	NO [291]	NO [60]
SHA-0	YES	NO [291]	NO [26]	-
SHA-1	YES	YES	NO [290]	-
RIPEMD	NO [285]	NO	NO	-
HAVAL-128	NO [285]	NO	NO	-
RIPEMD-128/160	YES	YES	YES	YES
SHA-224/256	YES	YES	YES	YES
SHA-384/512	YES	YES	YES	YES

From these observations on MBCA, it is clear that the designers of MD5, SHA-0 and SHA-1 *have not considered security of the compression functions of these hash functions against special Type-3 and Type-3 collisions in their design criteria.* Preneel pointed out more than a decade back [228] that most hash functions are not designed to meet this criteria. Note that SHA-1 did not exist then. Even Damgård's [53] proof of his iterative hash function design implicitly notes the necessity of *Type-3* and *special Type-3* collision resistance for the compression functions. In addition, to attain *Type-3* collisions, the two IVs do not have to be significantly different as suggested in [186, p.372]. For example, the two IVs in the *special Type-3* collision attack on the compression function of MD5 [60] differ in *only* 6 bits. Following the known attacks on some popular hash functions, Table 2.3 was derived assuming that if the compression function is not *Type-1* collision resistant then it is neither a *Type-2* nor *Type-3* collision resistant. In Table 2.3, "YES" means that a particular attack is known and not the non-existence of that attack has been proved, "NO"means that a particular attack exists and the sign "-" indicates that a particular attack is not applicable.

2.6 Conclusion

This chapter has presented some fundamental concepts on the Merkle-Damgård construction including the generic attacks on the construction that apply to all hash functions designed following this construction. In Appendix B, the evolution of the MD4 family of hash functions is provided showing that the compression functions of all these hash functions are basically unbalanced Feistel networks (UFNs) operating as word based non-linear feedback shift registers (NLFSRs). In addition, the results of known cryptanalytical attacks on these hash functions are presented. A terminology for the collision attacks on the compression functions and length extension attacks on the Merkle-Damgård hash functions is presented. Finally, the notion of multi-block collision attacks on the Merkle-Damgård hash functions is explained in great depth.

The valuable insight of this chapter is that the security of a mode of operation of a hash function such as Merkle-Damgård iterative structure depends not only on the strength of the compression function but also on the manner in which that compression function is iterated. This is evident from the generic attacks on hash functions reviewed in Section 4.4 and collision attacks reviewed in Section 2.5 and Appendix B. The multi-block collision attacks and generic attacks on the Merkle-Damgård hash functions poses the question as to what type of variants can offer better resistance against these attacks. This is investigated in Chapter 3 where the above insight is further expanded by providing new modes of operations for hash functions that thwart most of the known generic attacks discussed in Section 2.4.

Chapter 3

New Modes of Operation for Hash Functions

The multi-block collision attacks and the generic attacks on hash functions on the Merkle-Damgård construction based hash functions covered in Chapter 2 leave open the following questions:

1. Is it possible to design simple and efficient variants to the Merkle-Damgård construction that offer more resistance to multi-block collision attacks than the Merkle-Damgård hash functions?
2. Is it possible to design simple and efficient variants to the Merkle-Damgård construction that defeat the known techniques of performing the generic attacks on hash functions?

This chapter attempts to answer these questions. The motivation of this chapter is to show that while consecutive iterations of the compression function are necessary for the implementation efficiency of a hash function, **the way** the compression function is iterated or **used** is quite important for the security of the hash function. This chapter proposes a new generic mode of operation for hash functions called **3CG**. The **3CG** construction processes the intermediate chaining values of the Merkle-Damgård construction by maintaining a second internal chaining variable. An in-depth analysis of the linear variant of the **3CG** construction called **3C** is provided and its resistance to the known generic and specific multi-block collision attacks is provided.

This chapter shows that the **3C** construction is at least as secure as the Merkle-Damgård construction against both the single-block and multi-block collision attacks. That is, if there exists an adversary who finds a multi-block collision on **3C**, based on some compression function, then that adversary would also have found a multi-block collision on the Merkle-Damgård hash function based on the same compression function. In addition, it shows that if there exists an adversary who can perform an MBCA on a t-bit Merkle-Damgård hash function based on a given compression function then the security of **3C** against MBCA with the same compression function is as much as 2^t times the security of Merkle-Damgård hash functions against MBCA, depending on the subtle properties of the compression function. It is conjectured that this security multiplier of **3C** against MBCA is close to 2^t for any compression function which is secure against single-block collision attacks.

The **3C** construction is also analysed against the known generic attacks [58, 126, 137, 138]. These analytical results show that the techniques to perform the generic multicollision, multipreimage and multi 2^{nd} preimage attacks [126] on the Merkle-Damgård hash functions work on **3C**. However, the known techniques to carry out the generic second preimage [58, 138] and herding attacks [137] on the Merkle-Damgård hash functions do not work on **3C**. In addition, **3C** and its variants prevent straight forward length extension attacks [79, 173] and 2^{nd}-collision attacks [174] that work on the Merkle-Damgård hash function.

In addition, hybrid hash function constructions are proposed in this chapter by combining **3C** with the wide-pipe and double-pipe hash functions [174]. Analysis of these hybrid constructions is provided. Finally, a 3-chain variant to the **3C** construction called **3C+** is also analysed against the multi-block collision attacks.

The chapter is organised as follows: Section 3.1 introduces the **3CG** construction and its analysis against MBCA is covered in Section 3.2. In Section 3.3, analysis of **3C** against generic attacks is given and **3C** is compared with some other related hash function proposals in Section 3.4. In Section 3.5, hybrid constructions based on **3C** are introduced and the **3C+** construction is introduced and analysed against MBCA in Section 3.6. The implementation aspects of **3C** and **3C+** designs based on the MD5 and SHA-1 compression functions are covered in the Section 3.7. The chapter concludes with some remarks in Section 3.8.

3.1 The 3CG Construction: An Enhancement to the Merkle-Damgård Scheme

The **3CG** construction is shown in Figure 3.1. This construction is a 2-chain structure: a cascade chain as in the Merkle-Damgård construction and an accumulation chain running on top of the cascade chain. The construction has an *accumulator* function f′ iterated in the *accumulation chain* and a compression function f (f, for example, is the compression function of MD5 or SHA-1) iterated in the *cascade chain* as in the Merkle-Damgård construction. The function ZPAD in Figure 3.1 denotes the padding of the chaining values with 0's or any other formatted bits to use them as a block for the function f′ in the *accumulation chain.* The use of ZPAD depends on the choice of the function used for f′ in **3CG**.

The function f′ in the accumulation chain can be any function and it can be as simple as the XOR operation or as complex as the compression function f itself. If the compression function f, used in the cascade chain of **3CG**, is used as an accumulator f′ then this variant is called **3CF**. A variant to the **3CG** construction called **3C** uses the XOR linear operation as the accumulator function f′.

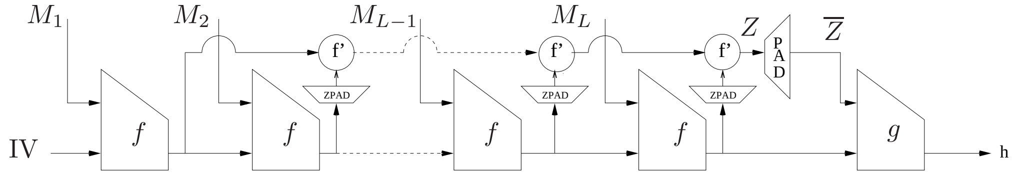


Figure 3.1: The generic **3CG**-hash function

3.1.1 The 3C construction: A Linear Variant of 3CG

The **3C** construction is shown in Fig. 3.2 and a section of the construction is shown in Fig. 3.3. This structure has an *accumulator* XOR function iterated in the *accumulation chain* (whose chaining value is denoted by u_i in Fig. 3.3) and a compression function f (f, for example, is the compression function of SHA-1 or SHA-256) iterated in the *cascade chain* (whose chaining value is denoted by w_i in Fig. 3.3) exactly as in the Merkle-Damgård construction. The **3C** design is a simple and efficient modification to the Merkle-Damgård construction. One economic benefit of this design is that any software currently implementing the

Merkle-Damgård-style hash function can be easily adapted to the **3C** structure, without altering the compression function.

3C Hashing Process: The following procedure outlines the hashing process of any hash function derived from the **3C** construction.

For $i = 1$ to L, let w_i and u_i be the chaining values in the *cascade* and *accumulation chains* respectively. Then, as in the hash functions based on the Merkle-Damgård construction, for $i = 1$ to L, $w_i = f(w_{i-1}, M_i)$ where $w_0 = IV$ and $u_1 = w_1$. In the *accumulation chain*, for $i = 1$ to L, $u_i = u_{i-1} \oplus w_i$ where $u_0 = 0$. The result u_L in the *accumulation chain* is the linear checksum of **3C** and is denoted with Z. An extra compression function, denoted by g, is added at the end and the digest of **3C** is $g(\overline{Z}, w_L) = H_{L'}$ as indicated in Figure 3.2. To process one block or less of data, the compression function is executed three times; first to process the data block, next to process the padded block containing the Merkle-Damgård strengthening[1] and finally the block $\overline{Z}$ formed in the *accumulation chain* as shown in Figure 3.2. If the size of the data to be filled within a block is less than the block size b of f then zeros are appended to the data to make it a b-bit data block. The linear checksum Z is converted to a full b-bit block $\overline{Z}$ by concatenating Z with required number of 0 bits. For example, if f is the compression function of SHA-1 then 352 0 bits are appended to Z to obtain a 512-bit block for the last compression function.

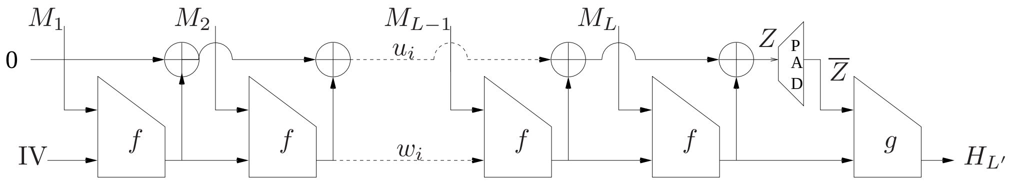


Figure 3.2: The **3C**-hash function

3.2 Analysis of 3C against Collision Attacks

This section investigates the security of **3C** against single-block and multi-block collision attacks. It is concluded that the security of **3C** against single-block collision attacks is upper bounded by the security of the compression function against collision attacks, and its security against MBCA is not less than that

[1]Unlike in the practical implementations of MD4 family of hash functions, length encoded information is always processed in a separate block in **3C** and its variants.

of the Merkle-Damgård construction based hash function. Figure 3.3 is used to explain the analysis.

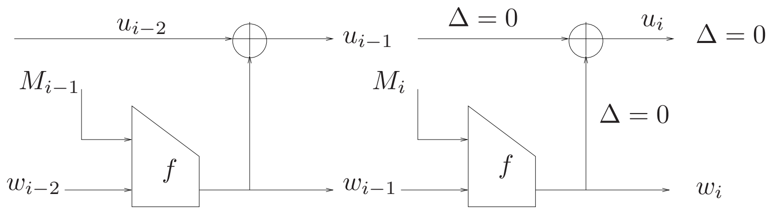


Figure 3.3: Creating an internal collision for **3C**

Consider the **3C** hash function H. Consider two distinct messages $M \neq N$ of the same length L including padding such that $H(M) = H(N)$. The messages M and N are expanded to sequences $(M_1, \ldots, M_L) \neq (N_1, \ldots, N_L)$ where the last data blocks are the padded blocks containing the length L of the messages. Let (H_i^M, H_i^N) and (u_i, v_i) (for $i = 1$ to L) denote the internal hash values obtained on the *cascade* and *accumulation chains* while computing $H(M)$ and $H(N)$ respectively. Let (u_L, v_L) be represented by (Z_M, Z_N) and let $\overline{Z}_M = \text{PAD}(Z_M)$ and $\overline{Z}_N = \text{PAD}(Z_N)$. All possible types of collisions on H are defined below:

Definition 1 *Every collision on H takes one of the following forms:*

1. *Terminal/Final collisions on H: They involve one of the following cases:*
   - $H_L^M \neq H_L^N$ *and* $\overline{Z}_M \neq \overline{Z}_N$ *with* $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$
   - $H_L^M = H_L^N$ *and* $\overline{Z}_M \neq \overline{Z}_N$ *with* $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$
   - $H_L^M \neq H_L^N$ *and* $\overline{Z}_M = \overline{Z}_N$ *with* $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$
2. *Internal collisions on H:* $H_L^M = H_L^N$ *and* $\overline{Z}_M = \overline{Z}_N$ *implies* $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$.

The analysis uses Definition 2 of the collision resistance.

Definition 2 *A compression function $f : \{0,1\}^b \to \{0,1\}^t$ is Type-1* (resp. *Type-2, Type-3*) *collision resistant if the best possible collision attack on it using fixed IV* (resp. *arbitrary IV, different IVs*) *is the birthday attack which takes about $2^{t/2}$ operations of f. For sufficiently large t, it is computationally infeasible to perform this attack.*

Lemma 1 *Against single block collision attacks, the security of* **3C** *is slightly more than the security of* Merkle-Damgård *when both constructions use the same* f.

Proof: By inspection of the **3C** structure in Figure 3.3, it is obvious that it contains a Merkle-Damgård construction within it. Hence, an adversary that is able to find a single block collision for the f-function is also able to find a collision for the Merkle-Damgård construction based on f with no additional effort. Such an adversary can also find a collision for the **3C** hash function based on f with a minimal additional effort of two computations of f as **3C** requires at least three computations of f to process an arbitrary length message. □

Apart from single block collision attacks, the only other approach to find collisions for **3C** is to use multi-block messages. This invites the opportunity to use messages with different lengths. For two messages with the same length, an internal collision for **3C** gives an actual collision for **3C**. In general, for two messages of different lengths, this is not the case due to the different padding strings used as a virtual message block in the second last iteration of the compression function. Thus the security analysis of **3C** can be restricted to considering internal collisions generated by pairs of messages with the same length. The nature of internal collisions for **3C** are examined in Lemma 2.

Lemma 2 *To get an internal collision in* **3C** *at iteration* i*, it is required that a collision in the accumulation chain exists at iteration* $i-1$.

Proof: An internal collision in **3C** at iteration i is a simultaneous collision in the *accumulation chain* and *cascade chain* at iteration i. For messages M and N to collide on the *cascade chain* at iteration i, the condition that $H_i^M \oplus H_i^N = 0$ must be satisfied. Now for an internal collision on **3C**, the condition that $(H_i^M \oplus H_i^N) \oplus (u_{i-1} \oplus v_{i-1}) = 0$ must be satisfied. This condition will occur only when $u_{i-1} \oplus v_{i-1} = 0$, which is a collision in the *accumulation chain* at iteration $i-1$. □

Remark: One possible way in which a collision in the accumulation chain at iteration $i-1$ is obtained is by creating a sequence of *cascade chain* differences where the chaining difference in the *cascade chain* at iteration $i-1$ is the XOR sum of all the previous differences in the *cascade chain* up to iteration $i-2$.

Lemma 3 *Assuming the existence of a collision in the accumulation chain at iteration $i-1$, it requires a single-block special Type-3 collision attack on f to create an internal collision in* **3C** *at iteration i.*

Proof: By the inspection of the **3C** structure in Figure 3.3, a special Type-3 collision attack must be performed on the f-function at iteration i. It is a *special Type-3* collision attack as the attacker must use the internal chaining values of the *cascade chain* at iteration $i-1$ as inputs to create a collision in the *accumulation chain* at iteration $i-1$. This is equivalent to performing a single-block *special Type-3* collision attack on the f function at iteration i. □

The internal collisions in **3C** using multi-block collisions can be found in either of the two ways:

1. Assume the existence of a multi-block collision on the *cascade chain* given some given compression function f. The multi-block collision on the *cascade chain* must then be repeated until the internal chaining differences on the *cascade chain* happen to produce a simultaneous collision on both the *accumulation* and *cascade chains*. This option requires repeating this multi-block collision attack at most 2^t times under the assumption that the internal chaining differences are uniformly distributed.

2. Devise an entirely new MBCA for the **3C** construction instantiated with some given compression function.

The above two cases result in two theorems:

Theorem 1 *If there is a multi-block collision attack on the* Merkle-Damgård *construction based hash function with a given f then the security of* **3C** *with the same f against a multi-block collision attack is at most 2^t times the security of the* Merkle-Damgård *based hash function against a multi-block collision attack.*

Proof: To obtain a collision in the *accumulation chain*, by Lemma 2, an MBCA on the *cascade chain* must be repeated, where each attempt succeeds with probability 2^{-t}. That is, the security of **3C** against MBCA is some multiple ($[1, 2^t]$) of the security against MBCA for the Merkle-Damgård based hash function.

It is difficult to provide a tight quantitative analysis for **3C** against MBCA as the feasibility of the multi-block collision attack on the *cascade chain* depends on the subtle properties of the compression function of **3C**. This is clear from the multi-block collision attack techniques on the Merkle-Damgård based hash functions MD5, SHA-0 and SHA-1. This prevents a more precise formal proof for the practical security of **3C** against collision attacks.

Hence, it is conjectured that the security of **3C** against MBCA is improved by a factor of close to 2^t over the security of Merkle-Damgård hash function against MBCA[2]. □

Theorem 2 *The security of* **3C** *against an MBCA is not less than the security of a* Merkle-Damgård *based hash functions against MBCA.*

Proof: Every internal collision for **3C** contains within it a collision for the Merkle-Damgård based hash function. There exist collisions for Merkle-Damgård hash functions that are not internal collisions for **3C**. Thus the security of **3C** against MBCA is lower bounded by the security of Merkle-Damgård hash functions against MBCA. □

Remarks:

1. While at least two blocks must be processed to find a multi-block collision on the Merkle-Damgård hash function based on some compression function, at least three blocks must be processed to create a multi-block collision on **3C** based on the same compression function. For example, the difference pattern $(0, \Delta, \Delta, 0)$ which creates a collision on the Merkle-Damgård hash function based on a given f, will also create a collision on **3C** based on the same f. However, its reduced pattern $(0, \Delta)$ would create a collision for the Merkle-Damgård hash function but not for **3C**. Hence in this case, the complexity of performing an MBCA on **3C** based on some f might be more than the complexity of performing an MBCA on the Merkle-Damgård hash function based on the same f.

2. As another example, if a special difference pattern for the chaining values like $(0, \Delta_1, \Delta_2, \Delta_1 + \Delta_2, 0)$ is required to create collision on the Merkle-Damgård hash function based on some f, then this pattern would also create a collision for **3C** based on the same f [147]. In this case, the complexity of performing MBCA on both hash functions are the same.

[2]A few months after **3C** was accepted in [89], Kelsey [135], Lucks [175] and Tuma and Joscak [279] have independently found a multi-block collision attack on **3C** based hash functions showing that this conjecture is incorrect. These results are shown in Section 4.5 of Chapter 4.

3. Note that the MBCA-1 type collisions on the Merkle-Damgård hash function with a compression function f work on **3C** with the same compression function f justifying Theorem 2. For example, the two block collision attack which works on SHA-0 [289] also works on **3C** with the compression function of SHA-0. Thus, **3C** is susceptible to MBCA-1 type collision attacks[3].

3.3 Security Analysis of 3C against Known Generic Attacks

3.3.1 Analysis Against Joux's Attacks

The multicollision attack described in Chapter 2 works on the **3C** construction. In this attack, the attacker finds collisions on every compression function f in the *cascade chain* (for example using brute-force collision search techniques) that would result in a collision at the subsequent point of the XOR operation in the *accumulation chain.* If the function f in the *cascade chain* of **3C** is modeled as a random oracle, as an upper bound, the total complexity to find 2^d-collisions on **3C** is $O(d \times 2^{t/2})$.

The attack technique used to find $D = 2^d$ preimages on the Merkle-Damgård hash function for a given hash value works on **3C** as well. An algorithm used to find D preimages for the **3C** hash function is:

Algorithm: Finding D-way preimages on 3C

1. The attacker is given a hash value Y computed using the **3C** hash function.
2. The attacker first finds D-collisions on d-block messages $M^1, \ldots, M^{2^d}$ with $H_d = H(M^1) = \ldots = H(M^{2^d})$. This requires an effort of $O(d \times 2^{t/2})$.
3. The attacker now finds the message block M_{d+1} such that the execution of the compression function processing the block M_{d+1} and the compression function processing the accumulated data from the accumulation chain would result in the given digest Y. Here the last two compression functions are treated as a single component. This task takes time $O(2^t)$.

[3]Tuma and Joscak [279] have implemented this attack on **3C** using the compression function of SHA-0 and provided test vectors.

Work: The total cost to find D-preimages for **3C** is the work required to find $D = 2^d$ multicollisions and the work required to find one preimage. This equals $O(d \times 2^{t/2} + 2^t)$.

Similarly, the attack used to find D-2^{nd} preimages for the Merkle-Damgård hash function also works on **3C**. An algorithm used to find D-2^{nd} preimages for **3C** is:

Algorithm: Finding D-2^{nd} preimages on 3C

1. The attacker is given a message M and has to find a second preimage. The attacker processes the message M using the **3C** hash function H and gets the digest, $H(M) = Y$.

2. Steps 2 and 3 are as for finding D-way preimages for **3C**.

Work: The total cost to find D-2^{nd} preimages for **3C** is the work required to find $D = 2^d$ multicollisions and the work required to find one second preimage. This equals $O(d \times 2^{t/2} + 2^t)$.

3.3.2 Analysis against 2^{nd}-preimage Attacks

Since compression functions used in the Merkle-Damgård hash functions are also used in the **3C** design, these compression functions follow the Davies-Meyer design principle of using the block cipher in the feed-forward mode as shown in Figure 2.2 in Chapter 2.

In the **3C** design, since the chaining state is twice as large as the hash value, a fixed point is defined for both the chains and this is obtained for any message block M_i, *only when* $f(0, M_i) = 0$. That is, the attacker has to find a message block M_i such that $g^{-1}(0, M_i) = 0$ and this occurs with a probability of 2^{-t}. The effort required to find the message block M_i such that $f(0, M_i) = 0$ is about 2^t. Hence, having fixed points for the compression functions will not assist in developing expandable messages to find second preimages in less than 2^t work on the **3C** design.

When the generic expandable message algorithm to find second preimages is applied on **3C**, a collision for both the chains is required and this costs an effort of 2^t at every stage as the size of the internal state is twice that of the hash size.

In addition, note that the expandable messages found using either of the methods (even if expandable messages are given for free) require finding a linking message block M_{link} from the end of the expandable message, producing a simultaneous match to both the chaining values at some stage of the target message M_{target}. This costs about 2^t work. Hence the known techniques to find second preimages on the Merkle-Damgård hash functions do not work on the **3C** design.

3.3.3 Analysis against Herding Attack

The detailed description of the herding attack on the Merkle-Damgård construction based hash functions is given in Section 2.4.4 of Chapter 2. For a t-bit **3C** hash function H, the attacker first constructs a 2^d hash value wide diamond structure by choosing 2^d arbitrary values as the starting hash values for the cascade chain. Assume that the starting hash values for the corresponding accumulation chains are all zero. This search structure with $d+1$ message blocks as the suffix would have 2^{d-1} possible checksum values, any of which can be used as the final padded data block to produce the final hash value H_{ct}. When the attacker is challenged with the prefix message P, P is processed using the initial state H_0 of the hash function H. Now the task is to find a linking message M_{link} to match the cascade chaining value $H(H_0, P)$ and the corresponding checksum value to one of the $2^{d+1}-2$ intermediate hash values and the corresponding checksum values in the diamond structure. This requires an effort of about $2^{2(t-d)}$. Even when $d = t/2$, which is an unrealistic assumption, it takes about 2^t effort to find a match. Hence it takes at least 2^t effort to violate the CTFP preimage resistance property for the hash functions based on the **3C** construction.

3.3.4 Analysis against Length Extension Attacks

In Chapter 2, two types of extension attacks on hash functions are defined. In *Type-A* extension attacks on a hash function H, the attacker is given an intermediate hash value $H(M)$ and $|M|$ but not M itself. The task is to process at least one new data block using $H(M)$ as the chaining value. In *Type-B* extension attacks, the attacker is given the final hash value $H(M)$ and $|M|$ but not M where $H(M)$ is the digest of the message M with Merkle-Damgård strengthening. The task is to use $H(M)$ and the new message block M_L to compute the new hash value for $M||M_L$. An additional message block M_{L+1} containing the Merkle-Damgård strengthening of the message $M||M_L$ is also processed to obtain $H(M||M_L||M_{L+1})$.

It should be noted that both forms of extension attacks do not work on **3C**. In the application of *Type-A* extension attack on **3C**, the attacker is given an intermediate hash value of the message M and $|M|$ with no padding on M. Assume that the attacker is given the hash value after the i^{th} iteration of the compression function in **3C**. This hash value represents the value only on the *cascade chain*. To perform the *Type-A* extension attack, the attacker must know the data in the *accumulation chain* after the iteration $i-1$. The latter requires the knowledge of all the *cascade* chaining state values prior to iteration i. Of course, if the attacker is provided with the values of both the chains after iteration i (i.e complete internal state information) then an extension attack can be performed on **3C**. However, it does not make sense to provide the attacker with the values of both the chains as this information is twice ($2t$) the size of the hash value which is only t bits and the attacker is provided only with the internal hash values of the cascade chain.

Similarly, the application of the extra compression function at the end prevents the *Type-B* length extension attack on **3C**. In this case, the attacker is given the final hash value $H(M)$ of the message M. The task is to use this final hash value in processing at least one new message block to obtain a new hash value. Since the attacker does not know the cascade and accumulation chaining values before the last iteration of the compression function, it would be difficult to perform this extension attack. Note that this analysis is applicable even if the message M is processed using **3C** without Merkle-Damgård strengthening to obtain the digest $H(M)$.

3.3.5 Analysis against 2^{nd}-collision Attacks

One can use the principles of length extension attacks on the Merkle-Damgård hash function H in performing 2^{nd}-collision attacks. If $H(M) = H(N)$ for any two different messages M and N, then $H(M||X) = H(N||X)$ [174]. That is, given a collision for free for the hash function H, the attacker finds a second collision by exploiting the Merkle-Damgård iterative structure. To obtain meaningful 2^{nd}-collisions [56,170], the first collision must be an internal collision with no padding. Hence, if the attacker is provided with a simultaneous internal collision on both the chains of **3C** then the attacker can perform 2^{nd}-collision attack on **3C**. As pointed out in Section 3.3.4, it does not make sense to provide the *accumulation* *chain* values to the attacker as it is giving the attacker twice the amount of the hash size.

The other case is where the attacker is given a final or terminal collision $H(M) = H(N)$ for $M \neq N$ (see Definition 1 for terminal collisions on **3C**). Since the given free collision is a terminal collision, the attacker must first get a simultaneous collision on both the chains of **3C** before performing the 2nd-collision attack by appending the same or different message blocks to the messages that result in a simultaneous collision. Even to find a simultaneous collision, the attacker has to find additional message blocks that would give a simultaneous collision on both the chains. This is equivalent to performing an MBCA on the **3C** hash function.

If the compression function f (also g) in **3C** is modeled as a random oracle, finding 2nd-collisions for **3C** for a given final collision would take time $\Omega(2^{t/2})$. This is analysed by considering all the possible cases of final collisions on **3C** given in Definition 1.

- $H_L^M \neq H_L^N$ and $Z_M \neq Z_N$ with $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$. In this case, for f modeled as a random oracle, the attacker has to get a simultaneous collision on both the chains to get 2nd collisions. This would take time about 2^t as the attacker has to get a collision on the *cascade chain* which costs $\Omega(2^{t/2})$ provided a collision occurs on the *accumulation chain* which also costs $\Omega(2^{t/2})$.

- $H_L^M = H_L^N$ and $\overline{Z}_M \neq \overline{Z}_N$ with $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$. Like the above case, this case also requires time about 2^t to get a simultaneous collision on both the chains.

- $H_L^M \neq H_L^N$ and $\overline{Z}_M = \overline{Z}_N$ with $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$. Since there is a collision on the *accumulation chain* the attacker has to find a collision on the *cascade chain* which costs $\Omega(2^{t/2})$ for f modeled as a random oracle.

In summary, it would take time $\Omega(2^{t/2})$ to perform a 2nd-collision attack on **3C** for the compression function modeled as a random oracle.

3.4 Comparison of 3C with Related Hash Function Proposals

Ferguson and Schneier [79] proposed a double-hashing scheme $H_{IV}(H_{IV}(x))$ to prevent straight-forward length extension attacks discussed in Section 3.3.4. Let this double-hashing scheme be called DHASH. The DHASH is a key-less or fixed IV variant of the nested MAC (NMAC) and hash based MAC (HMAC) constructions proposed by Bellare, Canetti and Krawczyk [13]. It is obvious that multi block collision attacks on this nested construction work as effectively as they do on the Merkle-Damgård hash functions. That is, the attacker finds multi-block collisions on the inner hash function first and the application of the outer function does not prevent the propagation of collisions through the whole hash function. As on the Merkle-Damgård hash function, 2^d-collisions can be found on DHASH with a complexity of $O(d \times 2^{t/2})$. Finding 2^d-(2^{nd}) preimages would take time $O(d \times 2^{t/2} + 2^t)$.

Chapter 5 of this thesis proposes a new technique called iterated halving to design rate 1 hash functions that can make use of any secure 128-bit block cipher reduced to half the number of rounds, as an alternative to the Merkle-Damgård hash functions. The 2^{nd} preimage, herding and length extension attacks do not apply to the iterated halving structures. Unlike Merkle-Damgård hash functions, they require buffering of data to be hashed which may not be needed for some applications that use hash functions. While the iterated halving structure is different to the Merkle-Damgård iterative structure, **3C** and its variants are far superior to the Merkle-Damgård hash functions.

Lucks [173, 174] has proposed the wide-pipe (Figure 3.4) and its special case double-pipe hash designs as failure-tolerant designs showing that they are more resistant to the generic attacks explained in Section 3.3 than the Merkle-Damgård hash functions. While the wide-pipe hash function maintains more internal state than the hash size t using larger compression functions, the double-pipe hash function maintains twice the hash size of the internal state size by employing one single t-bit compression function used twice in parallel for each message block. In contrast, one could see the **3C** structure as a special case of the wide-pipe hash function. In addition, **3C** is optimally efficient as no new large compression function needs to be designed for its execution. The wide-pipe, double-pipe, **3C** and DHASH proposals resist the straight-forward length extension and 2^{nd}-

collision attacks. All these hash functions provide $t/2$-bit level of security against these attacks as long as their design criteria are satisfied; for example, a wide-pipe hash function requires processing of the compression function with an internal state at least twice the size of the hash value, **3C** requires at least three calls to the compression function.

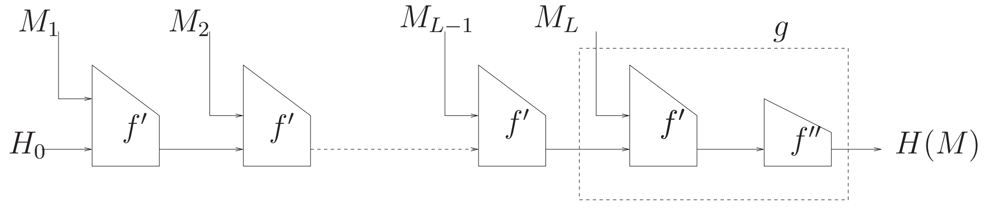


Figure 3.4: The wide-pipe hash function

Significantly, **3C** provides the same security levels as the wide-pipe and double-pipe hash functions against most of the known generic attacks that work on Merkle-Damgård hash functions in a more economical way. That is, one does not have to use large compression functions to prevent these attacks. Unlike the wide-pipe and double-pipe hash functions, **3C** does not provide more resistance but is as good as the Merkle-Damgård hash functions against Joux's generic attacks. While the wide-pipe and double-pipe hash functions are designed for more protection against only known generic attacks than the Merkle-Damgård hash functions, the **3C** construction has also been designed to provide an enhanced protection against not only generic attacks but also from multi-block collision attacks on the Merkle-Damgård based hash functions. However, it will be shown in chapter 4 that under very mild assumptions, a multi-block collision attack on a Merkle-Damgård hash function also works on the **3C** hash function based on the same compression function. The generic 2^{nd} preimage and herding attacks that work Merkle-Damgård construction also work on **3C** with not much additional effort.

In the wake of differential collision attacks on MD5, SHA-0 and SHA-1 [24,27, 42,290,291] due to poor message expansion of the compression functions of these hash functions, Jutla and Pathak [128] proposed a new provably secure collision resistant compression function for SHA-1. In a related work [127], the same authors have shown that their new SHA-1 compression function also resists the natural extensions to the differential collision attacks. While their work focuses on improving the collision resistance of the compression function of the SHA-1 against known collision attacks, this work aims at providing efficient modifications to the Merkle-Damgård mode of operation of hash functions for an improved protection against both the MBCA and some known generic attacks. Note that one can also accommodate the new compression function [128] in **3C** and its variants for more resistance to any future collision attacks.

Coron *et al.* [50] have proposed four hash function modifications to the plain Merkle-Damgård construction that work as random oracles when the underlying compression functions work as random oracles. Two of those constructions are un-keyed NMAC and HMAC [13] constructions. Note that **3C** and its variants are reminiscent of NMAC and HMAC functions as they also have an extra compression function at the end. It is conjectured that **3C** construction might be proven to be working as a random oracle, assuming that the compression function as a random oracle, following the proof techniques on the random oracle behaviour of un-keyed NMAC and HMAC functions shown in [50].

Szydlo and Yin [274] have shown that when some simple message pre-processing techniques are combined with MD5 or SHA-1, the applications based on these hash functions are no longer vulnerable to known collision attacks. The **3C**, its variants and the message pre-processing solutions [274] all attempt to defeat the multi-block collision attacks by removing control of message differentials from the attacker. This occurs either through chaining (**3C** and its variants) or through message redundancy caused by interleaving message blocks with fixed strings or duplicates [274]. Both solutions are minimally invasive to existing deployments of the affected hash functions. However, Szydlo and Yin have the slight advantage in that their solutions do not require hardware implementations of the hash functions to be changed; only the inputs require additional processing. **3C** and its variants are only effective when the hash function can be modified to incorporate the chaining between compression functions, so may not be applicable in legacy hardware environments. For long messages, both solutions are exceptionally efficient, although **3C** and **3C+** (proposed in Section 3.6) have the advantage that their overheads diminish proportionally to the length of the message being hashed. After one hundred blocks have been hashed, the overhead of **3C/3C+** has fallen to less than 1%. Conversely, the overhead of the Szydlo-Yin schemes remains constant in the length of the message, chosen by the user as a parameter of between 12% and 50% reduction in throughput to match the security expectations.

From the performance point of view, **3C** is slightly more expensive than the Merkle-Damgård hash functions, especially when it is used to process short messages as **3C** requires at least three iterations of the compression function to process an arbitrary length message. To process 1-block (resp. 2-block) messages, the running time of **3C** is twice (resp. 1.5 times) that of the Merkle-Damgård hash function. **3C** requires an extra iteration of the compression function similar to DHASH [79] and is as efficient as this scheme for the processing of long messages. Unlike DHASH, **3C** is a single hashing scheme, which means that IV and padding of the message are employed only once.

3.5 Hybrid Hash Function Constructions using 3C

As far we know, in the current state of the art of hash functions, the only way to increase the security of hash functions against Joux's generic attacks discussed in Section 2.4.2 is by using the wide-pipe or double-pipe hash functions [174]. For example, SHA-224 is a wide-pipe hash of SHA-256 [211] and SHA-384 is a wide-pipe hash of SHA-512 [211]. So far, there are no full collision attacks on any of these standard hash functions. Nevertheless, considering some active research in the analysis of SHA-256 [95, 184, 226, 298], it should be noted that it could be applied to its variant SHA-224 with no additional effort. For example, a near-collision on the reduced version of SHA-256 resulted in a full collision on the reduced version of SHA-224 [184].

Due to the above reasons, hybrid variants of **3C** are proposed that offer additional protection against both the generic attacks and multi-block differential collision attacks. The wide-pipe hash function and the **3C** construction are combined to attain a hybrid construction called 3CWP (see Figure 3.5) attaining additional protection against both the generic attacks and multi-block collision attacks. The security analysis of 3CWP against the known generic attack techniques follows from [174] and against multi-block collision attacks follows from Section 3.2. Similarly, one can combine **3C** with the double-pipe hash function to obtain one more hybrid construction called **3CDP**.

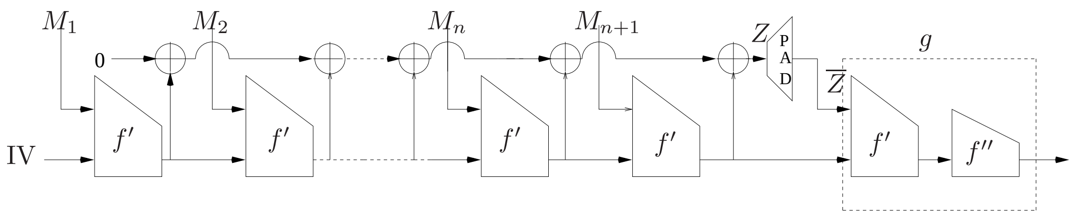


Figure 3.5: The **3CWP** hybrid construction

3.6 The 3C+ Construction

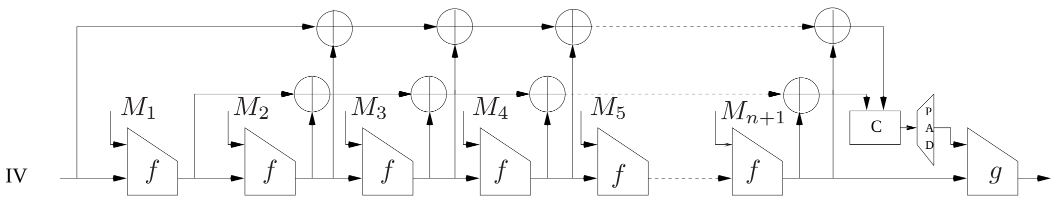


Figure 3.6: The **3C+** hash construction

This section analyses the behaviour of the **3C** construction with an additional chain running on top of the **3C** construction. The **3C+** construction is shown in Figure 3.6 where a third internal chain called the *final chain* has been added on top of the *cascade* and *accumulation chains* of **3C**. In the analysis of **3C+**, the *accumulation chain* is called the *middle chain*. The *final chain* in **3C+** slightly differs in the way it accumulates data from the *accumulation chain* of **3C** as it accumulates data from the *cascade chain* but the accumulation starts after processing the second message block. The final compression function f (denoted by g in Figure 3.6) takes appropriately padded concatenation of the accumulated data from the *middle* and *final* chains as input to the compression function g. The concatenation of the data in the *middle* and *final* chains is shown by the C function and the padding of that result is shown by the PAD function in Figure 3.6.

To find a multi-block collision on **3C+**, the attacker has to get collisions simultaneously on all the three chains. To create a simultaneous collision on all the three chains at iteration i, the chaining difference on the *cascade chain* at iteration $i-1$ (say Δ_{i-1}) must cancel the differences accumulated in the *middle* and *final* chains until iteration $i-2$. That is, the *middle* and *final* chains must maintain an equal difference Δ_{i-1} until iteration $i-2$ of the function f for a cancellation at iteration $i-1$. This is impossible if the *middle* and *final* chains do not start with the same difference before iteration $i-2$.

For example, consider the pattern $(0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5)$ of differences on the *cascade chain* obtained from the first to fifth iterations of f in **3C+**. This creates the pattern $(\Delta_1, \sum_{j=1}^{2} \Delta_j, \sum_{j=1}^{3} \Delta_j, \sum_{j=1}^{4} \Delta_j, \sum_{j=1}^{5} \Delta_j)$ on the *middle chain* until the fifth iteration and the pattern $(\Delta_2, \sum_{j=2}^{3} \Delta_j, \sum_{j=2}^{4} \Delta_j, \sum_{j=2}^{5} \Delta_j)$ on the *final chain* until the fifth iteration. Now the difference $\sum_{j=2}^{5} \Delta_j$ in the *cascade chain* after the sixth iteration of f cancels the difference accumulated in the *final chain* until the fifth iteration and creates a difference Δ_1 on the *middle chain*. To cancel the difference Δ_1 on the *middle chain*, the attacker must get the same difference Δ_1 on the *cascade chain* after the seventh iteration of f and again creates a difference of Δ_1 on the *final chain*. So canceling a difference in the *final chain* creates a difference in the *middle chain* and this process repeats.

Now, if an attacker finds two colliding messages that have identical first message blocks, then these two chains begin with a difference of zero. This implies that a minimum of four message blocks need to be processed to find an MBCA on **3C+**. For example, the pattern $(0, 0, \Delta, \Delta, 0)$ creates a simultaneous collision on all the chains after processing four blocks. From this discussion, it is clear that the **3C+** structure demands crafting and maintaining the same difference in both the *final* and *middle* chains until a multi-block collision is found.

Finally, note that if the input to the *final chain* is taken from the *middle chain* rather than from the *cascade chain*, the difference pattern $(0, \Delta, \Delta, 0)$ that creates a collision on the Merkle-Damgård hash function and **3C** will also create a collision on this modified construction of **3C+**. The security of this modified construction against MBCA is lower bounded by the security offered by the **3C** and Merkle-Damgård hash functions against MBCA. Note that this pattern does not create a collision on **3C+**. The analysis given for **3C** against generic attacks in Section 3.3 also applies to **3C+**.

3.6.1 Some Variants for 3C and 3C+

Note that one can construct many variants for **3C** and **3C+** designs by replacing XOR functions with any function in such a way that the new construction is at least as secure as the hash functions based on the Merkle-Damgård construction.

In this section, two other variants are provided for the **3CG** and **3C+** constructions. As pointed out in Section 3.1, if the function f′ in **3CG** is replaced with the compression function f then this modified construction is called **3CF**. The *cascade chain values* after the second iteration of f will be used as data blocks by applying the ZPAD function on them and the padded blocks are the inputs for the f functions in the accumulation chain. Clearly the amount of control that an attacker can have on these blocks to create an MBCA is much less than the one on **3C**. Alternatively, one can use the reduced versions of the compression function f in the *accumulation chain* as the substitution for f′. In such a case, an appropriate number of padded bits need to be included in the ZPAD function to meet the block size requirements of f′.

A slight variant of **3C+** can be designed by interpreting the t-bit chaining value in the *final chain* as an element of GF(2^t) and multiplying it by 2 at each step [10]. For this variant, the *final chain* accumulation process starts after the first iteration of f unlike in **3C+** and this structure results in a *final chain* accumulation equation that resembles Galois-Carter-Wegman structure of GHASH in [182]. This construction is called **3CCW+** and is shown in Figure 3.7.

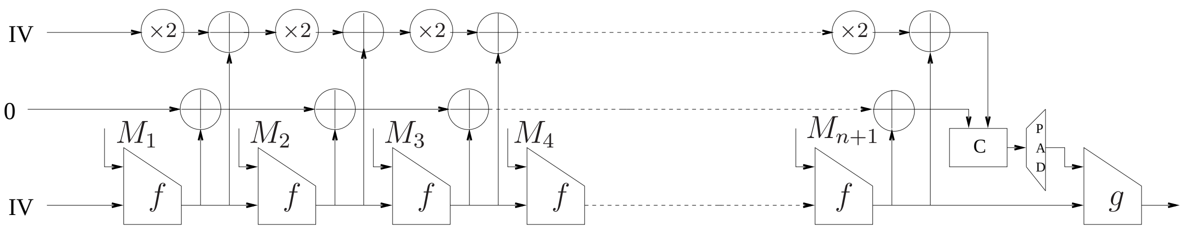


Figure 3.7: The **3CCW+** Construction

If w_i, u_i and v_i are the chaining values in the three chains at any state i, then for **3CCW+**, $w_0 = IV$, $u_0 = 0$ and $v_0 = IV$. At any state i, $w_i = f(w_{i-1}, m_i)$, $u_i = u_{i-1} \oplus w_i$ and $v_i = (2 \times v_{i-1}) \oplus w_i$. In this manner, the second chain as in **3C**, would have chaining states $0, w_i, w_1 \oplus w_2, w_1 \oplus w_2 \oplus w_3, \ldots$, starting from $i = 0$ and the third chain would have states $IV, (2 \times IV) \oplus w_1, 2 \times ((2 \times IV) \oplus w_1) \oplus w_2, 2 \times (2 \times ((2 \times IV) \oplus w_1) \oplus w_2) \oplus w_3, \ldots$. So, the accumulation equation due to the third chain can be represented as a polynomial of form $2^j \times IV \oplus 2^{j-1} \oplus w_1 \oplus 2^{j-2} \times w_2 \oplus \ldots \oplus w_j$ where $j \geq 3$ is the number of iterations of the compression function f.

Assuming that the compression function f in **3CCW+** is the SHA-1 compression function, the computation of the third chain is interpreted as multiplying an element of GF(2^{160}), represented as GF(2[x]/$\mathrm{x}^{160}+\mathrm{x}^5+\mathrm{x}^3+\mathrm{x}^2+1$), by the polynomial x. This can be implemented as a shift of the 160-bit chaining value towards the most significant bit by one bit position. If the plain polynomial has a degree 160 (i.e the final carry bit is 1) it must be reduced modulo $x^{160} + x^5 + x^3 + x^2 + 1$, which means ignoring the carry bit and adding $x^5 + x^3 + x^2 + 1$, which fits exactly into one byte, namely 0x2D, affecting the least significant byte in the representation of the third chain. Note that this variant, like **3C** does not protect MBCA-1 type multi-block collisions. Further analysis of this linear variant of **3C+** will be considered in future.

3.7 Implementation Issues

Tables 3.1 and 3.2 compare the throughputs of MD5 and SHA1 respectively, against their DHASH, **3C** and **3C+** versions. The metrics are obtained on a Pentium-M machine with a CPU speed of 2 GHz, by hashing messages of the given payload repeatedly until one gigabit of digest material has been acquired.

The results are unsurprising. It is well known that SHA-1 is less efficient than MD5 because it executes more iterations of the compression function per message block and has a more complex message expansion than MD5. For each compression function, DHASH outperforms **3C** and **3C+** for short messages. **3C** and **3C+** suffer from the requirement that a minimum of three blocks (eg. 192 bytes) be hashed, irrespective of the payload. For payloads in excess of this amount, none of the schemes shown in the tables are especially distinguishable via their throughputs.

An increase in the state size of **3C**, **3C+** and their associated hash functions is required to hold the additional chaining variables. For MD5, these increases in size amount to 17% and 34% respectively. For SHA-1, the chain represents an overhead of 4% and 8% respectively.

3.8 Conclusion

Cryptanalysis of the hash functions MD5, SHA-0 and SHA-1 [26, 144, 145, 172, 264, 287–291] exploited the Merkle-Damgård iterative structure of these hash functions using multi-block collision search techniques discussed in Chapter 2. As mentioned earlier, while the Merkle-Damgård iterative structure provides implementation efficiency for a hash function designed from it, the way it is iterated is essential for the security of the hash function against both generic and specific

Table 3.1: Time in seconds to process one gigabyte of data using MD5 on Pentium-M machine with a CPU speed of 2GHz

Payload (bytes)	MD5	MD5-DHASH	MD5-3C	MD5-3C+
16	4.53	8.91	11.72	11.67
32	2.11	4.32	5.33	5.56
64	1.95	3.05	2.99	3.17
128	1.61	2.17	1.85	1.97
256	1.43	1.71	1.55	1.64
1K	1.30	1.37	1.33	1.40
1M	1.25	1.26	1.25	1.33
1G	1.25	1.25	1.25	1.34

Table 3.2: Time in seconds to process one gigabyte of data using SHA-1 on Pentium-M machine with a CPU speed of 2GHz

Payload (bytes)	SHA-1	SHA-1-DHASH	SHA1-3C	SHA1-3C+
16	7.73	18.13	24.34	26.55
32	3.61	8.51	11.99	12.67
64	3.37	5.81	6.51	6.79
128	2.67	3.92	3.7	3.82
256	2.33	2.95	2.58	2.63
1K	2.08	2.22	2.14	2.17
1M	1.99	1.99	1.99	2.02
1G	1.99	2.00	2.02	2.03

or short-cut attacks. The **3C** construction is a fundamental design aimed at improving the security of the Merkle-Damgård construction against these two forms of attacks. Hence, it should be taken as a first step in improving the general hash function design.

Inspired by **3C** and its variants, Filho *et al.* [81] have proposed a variant for the **3C+** construction called **3CM** as a replacement for the Merkle-Damgård construction as the underlying module in the Whirlpool hash function [248] for better protection against the multi-block collision attacks. The **3CM** construction differs from **3CCW+** as it shifts the t-bit accumulation value in the third chain by 8 bits instead of by 1 bit. This choice in the shift is motivated by the fact that 8-bit smart cards process data byte wise and SSE2 128-bit registers use byte shift instructions rather than 1-bit shifts for efficient software implementation. Note that so far there are no known multi-block collision attacks on the Whirlpool hash function. This precautionary measure indicates the effectiveness of multi-block collision attacks on the Merkle-Damgård construction based hash functions. The analysis of **3C** against the generic attacks is also applicable to **3CM** and **3CCW+** designs. Similarly, one can construct several variants for both **3CG** and **3C+** designs. Further analysis of **3CM** and **3CCW+** are considered in future.

It is important to construct variants for the **3C** and **3C+** constructions as done in the form of **3CM** that become instances of **3CG** by replacing the XOR function with some efficient non-linear functions that provide more security against the known collision attacks. Such designs would take away the control from the attacker in constructing message blocks with specific differentials that produce a simultaneous collision on all the chains used in the construction.

Chapter 4 works on improving the known techniques to perform the two known powerful generic attacks: (2^{nd} preimage and herding attacks) on the **3C** construction and some other similar designs. The new cryptanalytical techniques show that the linear checksum of chaining values (as in **3C**) or message blocks or both does not make generic attacks on these constructions much harder as compared to the Merkle-Damgård construction. In addition, that chapter shows that if one can obtain a single block collision for the compression function then the linear checksum of chaining values or message blocks or both do not offer any better protection against multi-block collision attacks on these constructions.

Chapter 4

Cryptanalysis of a Class of Cryptographic Hash Functions

Chapter 3 shows that the **3C** hash function construction resists the known techniques of performing the generic 2$^{\text{nd}}$ preimage and herding attacks. This chapter introduces new cryptanalytical techniques required to implement these attacks on **3C** and some other variants of **3C**. These new cryptanalytical results show that the linear checksum of chaining values or message blocks or both for the Merkle-Damgård based hash functions do not make generic attacks and several other collision attacks that work on Merkle-Damgård hash functions significantly harder.

This chapter is organised as follows: In Section 4.1, some new hash function designs that use linear checksums are introduced. In Section 4.2, the known cryptanalytical techniques to perform generic attacks on these new designs are shown to be inapplicable. In Section 4.3, new cryptanalytical techniques to perform the generic attacks are discussed. In Section 4.4, the generic attacks are performed. In Section 4.5 the techniques to find collisions for hash functions with linear checksums if there is a multi-block collision attack on the underlying Merkle-Damgård construction are explored. In Section 4.6, complexities of some of the Joux attacks discussed in Chapter 2 are improved. Section 4.7 concludes the chapter.

4.1 Linear checksum variants of Merkle-Damgård

A number of variant constructions have been proposed, which augment the Merkle-Damgård construction by computing some kind of linear checksum on the message bits and/or intermediate chaining values, and providing the linear checksum as a final block for the hash function as shown in Figure 4.1. Note that **3C** differs from this generic scheme as it calculates checksum as a final block using only the intermediate chaining values.

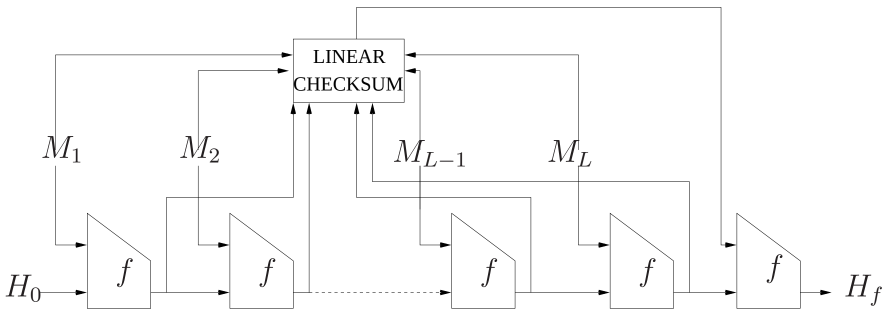


Figure 4.1: Hash function with a general linear checksum

4.1.1 GOST and its Variants

GOST is a block cipher based hash function specified in the standard GOST R 34.11-94 [220]. It is a 256-bit hash function and uses a compression function f which takes two inputs: a message block and chaining state each of 256 bits. The functionality of its compression function is derived from the functionality of the block cipher GOST specified in the standard GOST R 34.10-89 [219]. In this chapter, GOST design is explained from a high-level point of view only, ignoring the working details of the compression function specified in the standard GOST R 34.11-94 [220]. In addition, it is assumed that the compression function f in GOST and its variants use a Davies-Meyer structure as in the hash functions MD5 and SHA-1.

An arbitrary length message M to be hashed using GOST is split into b-bit blocks $M_1, \ldots, M_{L-1}$. If the last block M_{L-1} is incomplete, it is padded by prepending it with 0 bits. The binary encoded representation of the length of the true information is processed in a separate block. GOST performs addition modulo 2^b of all the message blocks[1].

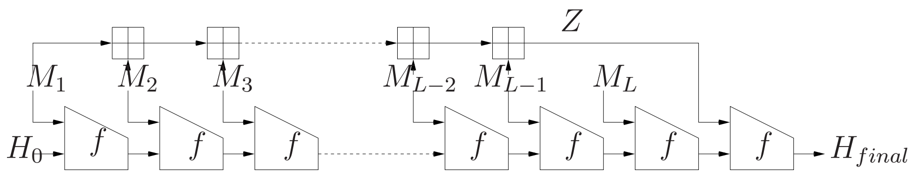


Figure 4.2: GOST hash function

The IV of GOST is represented with H_0 as shown in Figure 4.2. The compression function f is defined as $H_i = f(H_{i-1}, M_i)$ where $1 \leq i \leq L$. The checksum Z is computed over the message blocks. $Z = (M_1+M_2+\ldots+M_{L-2}+M_{L-1}) \bmod 2^b$. The chain used to accumulate the data blocks and compute the checksum Z is called the accumulation chain. Now, the final digest H_{final} of M is derived using the equation $H_{final} = f(Z, f(H_{L-1}, M_L))$.

A linear variant of GOST called GOST-L is defined by replacing the modulo 2^b checksum with a linear checksum based on modulo 2 addition of data blocks as shown in Figure 4.3. For GOTS-L, it is assumed that $b = t$.

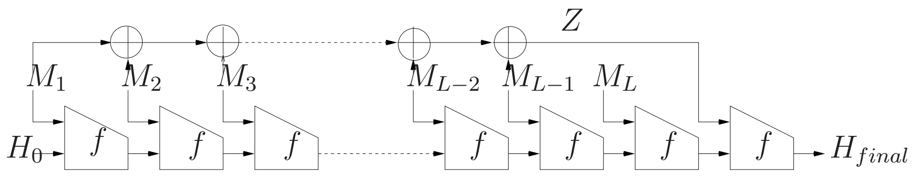


Figure 4.3: GOST-L hash function

4.1.2 F-Hash Hash Function

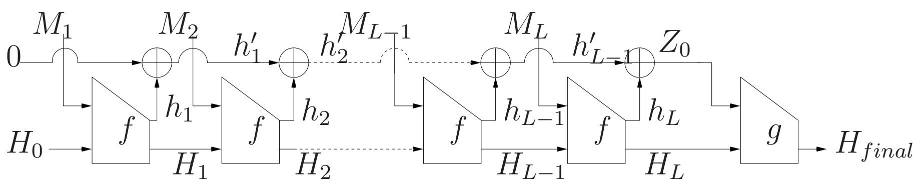


Figure 4.4: The F-Hash hash function

Lei [165, 166] has designed the F-Hash hash function based on a Feistel compression function as shown in Figure 4.4. The initial state of a t-bit F-Hash is H_0 and the compression function f is a Feistel structure based on a round function F

[1]For the GOST hash function $b = 256$ and $t = 256$.

of r rounds as shown in Figure 4.5. The message M to be processed using F-Hash is padded following the same method as defined for Merkle-Damgård hash functions. The padded message contains equal length message blocks and they are represented as $M = M_1, M_2, \ldots, M_L$. Each block M_i is expanded into sub-blocks $M_i^{(1)}, M_i^{(2)}, \ldots, M_i^{(R)}$ and all these sub-blocks are processed using the chaining state H_i and the round function F of f as shown in Figure 4.5. Every t-bit chaining state H_i is padded by prepending it with 0 bits according to the state input of the F-function. The two chaining output states (h_i, H_i) each of t bits for every iteration of the compression function f are given by $H_i = f_r(M_i, H_{i-1})$ and $h_i = f_{r-1}(M_i, H_{i-1})$ where $i = 1, 2, \ldots, L$; f_r is the r-round execution of F and f_{r-1} is the r-1-round execution of F. The accumulation value at any state i is given by $h'_i = \bigoplus_{j=1}^{i} h_j$ or $h'_i = h'_{i-1} \oplus h_i$. The chaining values (H_i, h_i) at any state i are considered as cascade chaining values.

F-Hash computes the checksum $Z_0 = \bigoplus_{i=1}^{L}(h_i)$ using cascade chaining values. The digest is $H_{final} = g(H_L, Z_0)$. The function g is the continuous iteration of F for r rounds with the chaining state $Z_i = F(H_L^i, Z_{i-1})$ where $i = 1, 2, \ldots, r$ and $H_L^1, H_L^2, \ldots, H_L^r$ is the expansion of H_L and the digest is $H_{final} = Z_r$.

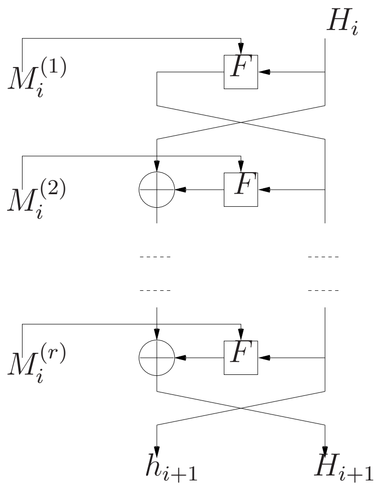


Figure 4.5: The compression function (f-function) of F-Hash

4.1.3 3C-GOST-L Design

Here **3C** and GOST-L structures are combined to obtain a new design called **3C-GOST-L** as shown in Figure 4.6. In this 3-chain structure, the linear checksum is obtained by XORing the checksum Z_1 obtained using the chaining values with the checksum Z_2 obtained using the message blocks as shown in Figure 4.6. The combined checksum $Z_1 \oplus Z_2$ is padded with 0 bits if needed and then processed using the final compression function f to obtain the message digest H_f. Assuming that the block size b of the message is the same as the size t of the intermediate chaining values in each of the other two chains, this design maintains three times the hash size as the internal state size. However, this design could be implemented with a memory that is only 2 times larger than the hash size. The chain used to process the cascade chaining values (resp. message blocks) to obtain the checksum Z_1 (resp. Z_2) is called the second chain (resp. third chain).

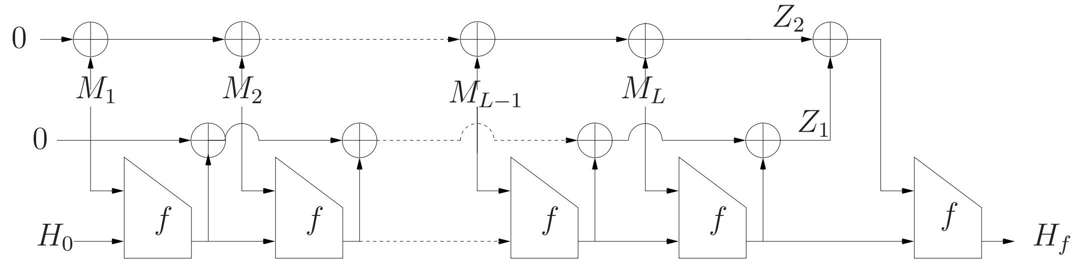


Figure 4.6: The **3C-GOST-L** hash function

4.2 Inapplicability of Known Generic Attack Techniques on GOST-L, F-Hash and 3C-GOST-L

The inapplicability of known generic attack techniques on **3C** were shown in Section 3.3.2 of Chapter 3. This section shows the inapplicability of performing known techniques of generic 2nd-preimage and herding attacks on the other linear checksum variants of Merkle-Damgård hash functions.

4.2.1 2nd-preimage Attack

The difficulty in finding the expandable messages using either of the techniques discussed in Chapter 1 for the GOST-L, F-Hash and **3C-GOST-L** hash functions is the same as that on **3C** explained in Section 3.3.2 of the Chapter 3.

Even if the attacker finds an expandable message using either of the techniques for GOST-L, the attacker still has to find a linking message block M_{link} from the end of the expandable message which results in a state matching some state in both the cascade chain and the corresponding accumulated linear checksum of data blocks in the target message M_{target}. This costs about at least 2^b. Similar reasons hold for the inapplicability of the known techniques in performing the generic second preimage attack on **3C-GOST-L**.

In the case of F-Hash, a collision for the compression function f at iteration i is defined as finding two message blocks M_i and N_i such that $M_i \neq N_i$, $f(H_{i-1}, M_i) = f(H'_{i-1}, N_i) = (H_i, h_i)$ where either $H_{i-1} = H'_{i-1}$ or $H_{i-1} \neq H'_{i-1}$. Once the attacker finds an expandable message for F-Hash using either of the techniques, the attacker has to find a linking message block M_{link} which results in a pair of values $(H_j, h_{j'})$ where $h_{j'} = \bigoplus_{i=1}^{j} h_j$ that match both the chaining states obtained after the j^{th} iteration simultaneously in the processing of the target message M_{target}. This costs about at least 2^t. Hence finding expandable messages for F-Hash would not assist the attacker in finding second preimages in effort less than 2^t.

4.2.2 Herding Attack

The analysis of F-Hash against the herding attack is similar to that on **3C** given in Section 3.3.3 of Chapter 3. Note that while computing the diamond structure, the attacker can either aim for collisions for the complete cascade chaining output (H_i, h_i) of every intermediate compression function or only H_i part of it which is used in the iteration of the compression function f. It might be sufficient if f produces near-collisions at any intermediate state i where all H_i values are the same and h_i values differ in some or all the bits.

The analysis of GOST-L with respect to the herding attack is similar to **3C** except that in the construction of 2^d diamond structure for GOST-L the starting accumulation chaining values are 2^d distinct message blocks used to obtain 2^d distinct intermediate hash values at the widest point of the diamond structure. Note that the diamond search structure with $d+1$ message blocks as suffix would have 2^d possible checksum data blocks, any of which can be used as the final message block to produce the final committed hash value H_{ct}.

4.3 New Cryptanalytical Techniques

This section explores new techniques required to perform the generic multicollision, 2^{nd}-preimage and herding attacks on the linear checksum constructions presented in Section 4.1 and **3C** proposed in Chapter 3. Since the techniques employed on the different designs differ slightly, they are explained for all the designs.

4.3.1 Combining Multi-block Collisions and Multicollisions

Let $C(s, n)$ be a collision finding algorithm for a hash function where s denotes the state and n denotes the number of message blocks on which it is applied. The algorithm $C(s, n)$ can either be a single block or a multi-block collision finder. Joux multicollision attack can be applied to messages of multiple blocks, with $n \geq 2$, leading to a multicollision attack over n-block messages. For example, $C(s, n)$ returns a 2-block collision for a hash function when it is called with the parameters $n = 2$ and $s = H_0$ and this is repeated leading to multicollisions over 2-block messages.

The algorithm $C(s, n)$ can either be a brute force or a cryptanalytic collision finding algorithm. A brute force algorithm spends about $2^{t/2}$ effort to find a collision with significant probability whereas a cryptanalytic collision finding algorithm requires less than that effort as noted in Section 2.3.4 of the Chapter 2.

4.3.2 Building 2^t 2-block multicollisions on 3C

Consider a t-bit **3C** hash function with an initial state H_0 and compression function f. Consider a 2-block multicollision-finding algorithm $C(s, 2)$ which returns a collision on the cascade chain only after processing every two blocks. Now a 2^t 2-block multicollision is obtained by calling $C(s, 2)$ t times.

Work: The computational effort required to perform a 2^t 2-block multicollision on **3C** depends on the characteristics of $C(s, 2)$. If $C(s, 2)$ is a brute-force collision finding algorithm then the work to perform a 2^t 2-block collision attack is $t \times 2^{t/2} + t$. It will be less than this factor if $C(s, 2)$ is a cryptanalytic collision-finding algorithm.

Characteristics of the collision-finding algorithm $C(s, 2)$

The collision-finding algorithm $C(s, 2)$ used to build a 2^t 2-block multicollision on **3C** can either be a cryptanalytic collision-finding algorithm or a brute force collision-finding algorithm.

1. If $C(s, 2)$ is a brute-force collision-finding algorithm then it always returns random message blocks with every call to it with no particular XOR difference between the first chaining values of every 2-block collision. In this case, the checksum at the end of the 2^t 2-block collision on **3C** is random.

2. If $C(s, 2)$ is a cryptanalytic collision-finding algorithm, as pointed out in Chapter 2, it returns two 2-block collision pairs with one of the formats $(M_1, M_2; N_1, N_2)$, $(M_1, M_2; M_1, N_2)$ or $(M_1, M_2; N_1, M_2)$ with every call to it.

   (a) If the format of the blocks is $(M_1, M_2; N_1, N_2)$ then the blocks M_1 and N_1 would either result in a fixed XOR chaining difference ($f(M_1) \oplus f(N_1) = \Delta$) or a XOR chaining difference that is fixed in most of the bits but varying randomly in a few bits or very tightly constrained (for example, the 2-block collision attack on SHA-1 [290] uses additive difference at this point). This is also applicable for blocks of the format $(M_1, M_2; N_1, M_2)$.

   (b) If the format of the blocks is $(M_1, M_2; M_1, N_2)$ then a collision on the cascade chain would also result in a collision on the accumulation chain and the **3C** hash function itself.

4.3.3 Defeating the Linear Checksum in a 2^t 2-block Multicollision on 3C

Case 1:

Assume that a 2^t 2-block multicollision on **3C** is performed with the initial state H_0 and compression function f using a brute-force collision finding algorithm $C(s, 2)$. Now there are t independent choices of chaining values in the accumulation chain after every 2-block collision on the cascade chain where each choice imposes a random XOR difference on the t-bit checksum at the end of the multicollision. These t choices let us control the checksum value at the end of the 2^t 2-block multicollision on **3C** or the checksum value obtained after processing any additional message blocks after the multicollision, while still keeping the same chaining values on the cascade chain. These t choices are termed as checksum control sequence (CCS).

The following algorithm is used to build a prefix from the CCS to adjust the final checksum value at the end of the multicollision to the desired checksum value. The CCS contains t different choices and the checksum value has t bits and this is set up as a system of t linear equations in t unknowns to find a prefix which gives the desired checksum.

ALGORITHM: Defeat checksum in a 2^t 2-block multicollision on 3C Variables:

1. (e_i^0,e_i^1) = A pair of independent choices of random values after every 2-block collision in the 2^t 2-block multi collision on **3C** and $e_i^0 \neq e_i^1$ for $i = 1, 2, \ldots, t$.
2. $a = a[1], a[2], \ldots, a[t]$ = Any t-bit string.
3. $D = D[1], D[2], \ldots, D[t]$ = The desired t-bit checksum to be imposed.
4. i, j = Temporary variables.

Steps:

1. Each of the parts of the CCS gives one choice e_i^0 or e_i^1 for $i = 1, 2, \ldots, t$ to determine some random t-bit value that either is or is not XORed into the final checksum value at the end of 2^t multicollision. Note that $e_i^0 = H_{2i-1}^0 \oplus H_{2i}^0$ for $i = 1, 2, \ldots, t$ and $e_i^1 = H_{2i-1}^1 \oplus H_{2i}^1$ for $i = 1, 2, \ldots, t$. That is, e_i^0 and e_i^1 are the XOR differences of two consecutive cascade chaining values on either side of the collision.
2. For any t-bit string $a = a[1], a[2], \ldots, a[t]$, let $e^a = e_1^a, \ldots, e_t^a$.
3. D is the desired checksum to be imposed at the end of 2^t 2-block multicollision.
4. Find $a = a[1], a[2], \ldots, a[t]$ such that $e_1^{a[1]} \oplus e_2^{a[2]} \oplus \ldots \oplus \ldots e_t^{a[t]} = D$. By treating $a[1], a[2], \ldots, a[t]$ as variables, we now solve the equation
$$\bigoplus_{i=1}^{t} e_i^0 \times a[i] \oplus e_i^1 \times (1 - a[i]) = D.$$
5. Each bit position of $e_i^{a[i]}$ gives us one equation and we turn the above into t equations, one for each bit. Let $\overline{a}[i] = 1 - a[i]$.

6. The resulting system is:

$$\bigoplus_{i=1}^{t} e_i^0[j] \times a[i] \oplus e_i^1[j] \times \overline{a}[i] = D[j] \ (j = 1, \ldots, t)$$

   Here there are t equations in t unknowns over modulo 2 addition which need to be solved for the solution $a[1], a[2], \ldots, a[t]$.

7. The solution $a[1], a[2], \ldots, a[t]$ allows us to determine the blocks in the 2^t 2-block multicollision that give the desired checksum D.

Work: It would take at most t^3 bit-XOR operations work to solve the above system of $t \times t$ equations using Gaussian elimination to find a solution with a probability of 0.5 [18, Appendix A], [49,282]. For **3C** based on SHA-1 ($t = 160 \approx 2^{7.322}$), this costs about $2^{21.966}$.

Case 2: Assume that a cryptanalytic collision finding algorithm $C(s, 2)$ is used to perform a 2^t 2-block multicollision on **3C**. Now, whether the system of linear equations due to the CCS obtained from this $C(s, 2)$ can be solved depends on the type of 2-block collision. Here this is analysed using three possible 2-block collision formats as pointed out in Section 4.3.2.

1. Consider the format of message blocks $(M_{2.i-1}, M_{2.i})$, $(N_{2.i-1}, N_{2.i})$ for $i = 1, \ldots, t$ producing 2-block collisions. The 2-block collisions due to near-collisions for the first blocks will have XOR differences after processing the first compression function in every collision [290, 291]. In many cases, these differences are either fixed or very tightly constrained and it would be difficult to find a solution for the system of equations as the attacker would not be able to control fixed or constrained bits of the XORed-together chaining states. This type of collision for the second compression function falls under the category of *Type-3* collision discussed in Section 2.3.3 of Chapter 2.

2. Similarly, it is also difficult to solve the system of equations from the CCS due to collisions of the messages of format $(M_{2.i-1}, M_{2.i})$,$(N_{2.i-1}, M_{2.i})$ for $i = 1, \ldots, t$ [60]. This type of collision for the second compression function falls under the category of *Special Type-3* collision discussed in Section 2.3.3 of Chapter 2.

3. It is not possible to control the checksum due to 2-block collisions of the format $(M_{2.i-1}, M_{2.i})$, $(M_{2.i-1}, N_{2.i})$ for $i = 1, \ldots, t$ [289]. The reason is every 2-block collision of this format produces a zero checksum difference. Then the final checksum difference in the 2^t multicollision is also zero. Hence, the attacker cannot choose the message blocks from either side of the 2^t 2-block multicollision of this format to force the checksum to a desired checksum.

4. Unlike above, assume that two random and different message blocks are processed initially using f to get two different random chaining states s_1 and s_2. Then a cryptanalytic collision finding algorithm $C(s_1, s_2, 1)$ is called with s_1 and s_2 as parameters which produces either the same or different message blocks that collide. In effect, the XORed-together chaining states after every 2-block collision in the 2^t 2-block multicollision is random. Hence, a prefix can be constructed from the CCS which forces the checksum at the end of 2^t 2-block multicollision to the desired checksum.

4.3.4 Building a 2^b 1-block Multicollision on GOST-L

Consider a t-bit digest and b-bit block GOST-L hash function H with an initial state H_0 and compression function f. Now, the CCS is built using a multicollision-finding algorithm $C(s, 1)$ on one block messages. The attacker starts with the initial state H_0 and calls $C(s, 1)$ b times. The algorithm $C(s, 1)$ returns a colliding pair (M_i, N_i) with every call to it such that $f(H_{i-1}, M_i) = f(H_{i-1}, N_i)$ for $i =$ 1 to b and $H_1, H_2, \ldots, H_{b-1}$ are the intermediate chaining state collisions and H_b is the final multicollision.

Characteristics of the collision-finding algorithm $C(s, 1)$

1. If $C(s, 1)$ is a brute-force collision finding algorithm then it always returns random message blocks with every call to it with no specific XOR difference of the message blocks. In such a case, the linear checksum at the end of 2^b 1-block multicollision on GOST-L is random.

2. If $C(s, 1)$ is a cryptanalytic collision-finding algorithm then one may assume that in many cases, the message blocks (M_i, N_i) where $i = 1, 2, \ldots, b$ returned by $C(s, 1)$ may either have a fixed XOR difference (i.e, $M_i \oplus N_i = \Delta$ and $f(H_{i-1}, M_i) = f(H_{i-1}, N_i)$) or a XOR difference that is fixed in most of the bits, but varying randomly in a few bits.

4.3.5 Defeating the Linear Checksum in 2^b 1-block Multicollision on GOST-L

Case: 1 Assume that a 2^b 1-block multicollision on GOST-L has been carried out using a brute-force collision-finding algorithm $C(s, 1)$. Let $M = M_1||M_2||\ldots||M_b$ and $N = N_1||N_2||\ldots||N_b$ be two b-bit block sequences with each block returned by $C(s, 1)$ such that $f(H_{i-1}, M_i) = f(H_{i-1}, N_i)$.

With that 2^b multicollision, there are b choices of 1-block messages where each choice imposes a random XOR difference on the b-bit linear checksum block. These b choices form the CCS and give the prefix which lets us control the final checksum block of b random bits or any checksum block obtained after processing any additional message blocks after the multicollision, while still keeping the same intermediate chaining values on the cascade chain. Now CCS contains b different choices and the checksum has b bits which yields a system of b linear equations in b unknowns. The solution to this system of equations will give us the checksum control prefix which is the actual choice of message blocks that give us the desired checksum.

The following attack algorithm is used to defeat the checksum in 2^b 1-block multicollision on GOST-L.

ALGORITHM: Defeat checksum in 2^b multicollision on GOST-L
Variables:

1. (M_i, N_i) = A pair of independent and random choices of message blocks in the 2^b 1-block multicollision on GOST-L and $M_i \neq N_i$ for $i = 1, 2, \ldots, b$.

2. $a = a[1], a[2], \ldots, a[b]$ = Any b-bit string.

3. $D = D[1], D[2], \ldots, D[b]$ = The desired b-bit checksum to be imposed.

4. i, j = Temporary variables.

Steps:

1. Each of the parts of the prefix gives one choice (M_i or N_i) for $i = 1$ to b to determine some random b-bit value that either is or is not XORed into the final checksum of data blocks at the end of 2^b multicollision.

2. D is the desired checksum to be imposed at the end of 2^b 1-block multicollision.

3. Now find $a = a[1], a[2], \ldots, a[b]$ such that the choice of message blocks that provide D will be known. By treating $a[1], a[2], \ldots, a[b]$ as variables, we solve the equation

$$\bigoplus_{i=1}^{b} M_i \times a[i] \oplus N_i \times (1 - a[i]) = D$$

4. Having $\overline{a}[i] = 1 - a[i]$, turn the above into a system of b equations in b unknowns over modulo 2 addition as below:

$$\bigoplus_{i=1}^{b} M_i[j] \times a[i] \oplus N_i[j] \times \overline{a}[i] = D[j] \ (j = 1, \ldots, b)$$

5. Solve the above system of equations for a unique solution $a[1], a[2], \ldots, a[b]$ which lets us determine the blocks in the 2^b 1-block multicollision that give the desired checksum D.

Work: It would take at most b^3 work to solve the above system of $b \times b$ equations using Gaussian elimination to find a solution with a probability of 0.5 [18, Appendix A], [49, 282]. When $b = 2^8$ as in GOST-L, this costs about 2^{24}.
Case: 2 Assume that $C(s, 1)$ is a cryptanalytic collision finding algorithm maintaining a fixed XOR difference of message blocks. When most of the bits in the XOR difference of the blocks are fixed except a few bits that vary randomly, the CCS can be built for the bits that vary and the system of equations due to those variable bits can be solved but the fixed bits in the difference cannot be used to control the checksum.

4.3.6 Defeating the Checksum in 2^t Multicollision on F-Hash

Case 1:

Consider a t-bit digest F-Hash with the initial state H_0 and compression functions f and g. Similar to **3C**, a 2^t 2-block multicollision is constructed by calling $C(s, 2)$ t times. Assume that $C(s, 2)$ is a brute-force collision finding algorithm. Now there are t independent choices of chaining values in the accumulation chain after every 2-block collision for f where each choice imposes a random XOR difference on the t-bit checksum at the end of multicollision. These t choices form the CCS and let us control the checksum value at the end of the 2^t 2-block multicollision on F-Hash or checksum value obtained after processing any additional message blocks after the multicollision. The CCS contains t different choices and the checksum value has t bits and this produces a system of t linear equations in t unknowns to find the prefix that give us the desired checksum.

The following algorithm is used to bypass the checksum in the 2^t 2-block multicollision on F-Hash.

ALGORITHM: Defeat checksum in the 2^t multicollision on F-Hash
Variables:

1. (e_i^0, e_i^1) = A pair of independent choices of random values after every 2-block collision in the 2^t 2-block collision on F-Hash and $e_i^0 \neq e_i^1$ for $i = 1, 2, \ldots, t$.
2. $a = a[1], a[2], \ldots, a[t]$ = Any t-bit string.
3. $D = D[1], D[2], \ldots, D[t]$ = The desired t-bit checksum to be imposed.
4. i, j = Temporary variables.

Steps:

1. Each of the parts of the CCS gives us one choice (e_i^0 or e_i^1) for $i = 1, 2, \ldots, t$ to determine some random t-bit value that either is or is not XORed into the final checksum value at the end of 2^t multicollision. Note that $e_i^0 = h_{2i-1}^0 \oplus h_{2i}^0$ and $e_i^1 = h_{2i-1}^1 \oplus h_{2i}^1$ for $i = 1, 2, \ldots, t$ where $e_0^0 = 0$ and $e_0^1 = 0$.
2. For any t-bit string $a = a[1], a[2], \ldots, a[t]$, let $e^a = e_1^a, \ldots, e_t^a$.
3. D is the desired checksum to be imposed at the end of 2^t 2-block multicollision.
4. Find $a = a[1], a[2], \ldots, a[t]$ such that $e_1^{a[1]} \oplus e_2^{a[2]} \oplus \ldots \oplus \ldots e_t^{a[t]} = D$. By treating $a[1], a[2], \ldots, a[t]$ as variables, solve the equation
$$\bigoplus_{i=1}^{t} e_i^0 \times a[i] \oplus e_i^1 \times (1 - a[i]) = D$$
5. Each bit position of $e_i^{a[i]}$ gives us one equation and turn the above into t equations, one for each bit. Let $\overline{a}[i] = 1 - a[i]$.

6. The resulting system is:

$$\bigoplus_{i=1}^{t} e_i^0[j] \times a[i] \oplus e_i^1[j] \times \overline{a}[i] = D[j] \ (j = 1, \ldots, t)$$

   Here there are t equations in t unknowns over modulo 2 addition which can be solved for the solution $a[1], a[2], \ldots, a[t]$.

7. The solution $a[1], a[2], \ldots, a[t]$ lets us determine the blocks in the 2^t 2-block multicollision that give the desired checksum D.

Work: It would take at most t^3 work to solve the above system of $t \times t$ equations using Gaussian elimination to find a solution with a significant probability.
Case 2: Assume that a cryptanalytic collision finding algorithm $C(s, 2)$ is used to perform a 2^t 2-block multicollision on F-Hash. Now, whether the system of linear equations due to the CCS obtained from this $C(s, 2)$ can be solved depends on the type of the 2-block collision. The analysis of this case is similar to the analysis of the corresponding case given for **3C** in Section 4.3.3 and hence it is ommited from discussion here. Whether the checksum can be bypassed or not using a cryptanalytic collision finding algorithm depends on the type of the 2-block collision as in **3C**.

4.3.7 Defeating the Linear Checksum in 3C-GOST-L

Case 1:

Assume that a 2^t 2-block multicollision on **3C-GOST-L** was performed with the initial state H_0 and compression function f using a brute-force collision finding algorithm $C(s, 2)$. Now there are t independent choices of chaining values (resp. pair of message blocks) in the second chain (resp. third chain) after every 2-block collision where each choice imposes a random XOR difference on the t-bit checksum in the second chain (resp. third chain) at the end of the multicollision. The following algorithm is used to bypass the linear checksum in **3C-GOST-L**. These t choices on the two chains let us control the respective checksum values at the end of the 2^t 2-block multicollision on **3C-GOST-L** or checksum value obtained after processing any additional message blocks after the multicollision while still keeping the same chaining values on the cascade chain. Let these t choices on each of the chains be the CCS of that chain. Each CCS contains t

different choices and each checksum value has t bits and this yields a system of t linear equations in t unknowns for each chain to find a prefix that gives the desired checksum in each chain.

The following algorithm is used to bypass the checksum due to a multicollision in both the chains.

ALGORITHM: Defeat checksum in the 2^t 2-block multicollision on 3C-GOST-L

Variables:

1. (e_i^0,e_i^1) = A pair of independent choices of random values after every collision in the 2^t 2-block collision on **3C-GOST-L** and $e_i^0 \neq e_i^1$ for $i = 1, 2, \ldots, t$.

2. $(M_i, M_i^*); (N_i, N_i^*)$ = A pair of independent choices of a pair of message blocks in the 2^t 2-block multicollision on **3C-GOST-L** and $(M_i, M_i^*) \neq (N_i, N_i^*)$ for $i = 1, 2, \ldots, t$.

3. $a = a[1], a[2], \ldots, a[t]$ = and $c = c[1], c[2], \ldots, c[t]$ = are any two t-bit strings.

4. $D_1 = D_1[1], D_1[2], \ldots, D_1[t]$ = The desired t-bit checksum to be imposed on the second chain.

5. $D_2 = D_2[1], D_2[2], \ldots, D_2[t]$ = The desired t-bit checksum to be imposed on the third chain.

6. i, j = Temporary variables.

Steps:

1. Each of the parts of the CCS gives one choice (e_i^0 or e_i^1) (resp. (M_i, M_i^*) or (N_i, N_i^*)) for $i = 1, 2, \ldots, t$ on the second chain (resp. third chain) to determine some random t-bit value that either is or is not XORed into the final checksum value in the second chain (resp. third chain) at the end of 2^t 2-block multicollision. Note that $e_i^j = H_{2i-1}^j \oplus H_{2i}^j$ for $i = 1, 2, \ldots, t$ where j is either 0 or 1 and $e_0^j = 0$.

2. For any t-bit string $a = a[1], a[2], \ldots, a[t]$, let $e^a = e_1^a, \ldots, e_t^a$.

3. Now D_1 (resp. D_2) is the desired checksum to be imposed on the second chain (resp. third chain) at the end of 2^t 2-block multicollision.

4. Find $a = a[1], a[2], \ldots, a[t]$ such that $e_1^{a[1]} \oplus e_2^{a[2]} \oplus \ldots \oplus \ldots e_t^{a[t]} = D_1$. Similarly, find $c = c[1], c[2], \ldots, c[t]$ such that the choice of a pair of message blocks that provide the desired checksum D_2 on the third chain will be known.

   By treating $a[1], a[2], \ldots, a[t]$ as variables, we solve the equation

   $$\bigoplus_{i=1}^{t} e_i^0 \times a[i] \oplus e_i^1 \times (1 - a[i]) = D_1.$$

   Similarly, by treating $c[1], c[2], \ldots, c[t]$ as variables, we solve the equation

   $$\bigoplus_{i=1}^{t} (M_i \oplus M_i^*) \times c[i] \oplus (N_i \oplus N_i^*) \times (1 - c[i]) = D_2.$$

5. Force $\overline{a}[i] = 1 - a[i]$ and $\overline{c}[i] = 1 - c[i]$.

6. Now turn the above into two systems of t equations in t unknowns as below:

   The resulting two system of equations are:

   $$\bigoplus_{i=1}^{t} e_i^0[j] \times a[i] \oplus e_i^1[j] \times \overline{a}[i] = D_1[j] \ (j = 1, \ldots, t)$$

   $$\bigoplus_{i=1}^{t} (M_i \oplus M_i^*)[j] \times c[i] \oplus (N_i \oplus N_i^*)[j] \times \overline{c}[i] = D_2[j] \ (j = 1, \ldots, t)$$

   Here there are two sets of t equations in t unknowns over the field GF(2) which can be solved for two solutions $a[1], a[2], \ldots, a[t]$ and $c[1], c[2], \ldots, c[t]$.

7. The solution $a[1], a[2], \ldots, a[t]$ (resp. $c[1], c[2], \ldots, c[t]$) lets us determine the chaining values (resp. pair of message blocks) in the 2^t 2-block multicollision that gives the desired checksum D_1 (resp. D_2).

Work: It would take at most $2 \times t^3$ work to solve the above two systems of $t \times t$ equations using Gaussian elimination to find a solution with a significant probability [18, Appendix A] [49, 282].

Case 2: The analysis on bypassing the linear checksums in the two chains using a cryptanalytic collision finding algorithm $C(s, 2)$ to perform a 2^t 2-block multi-collision on **3C-GOST-L** is similar to the case on **3C** and GOST-L. Hence, it is omitted from discussion here.

4.4 Generic Attacks on 3C, GOST-L, F-Hash and 3C-GOST-L

The method used to perform the generic attacks on all the linear checksum variants is the same. Hence it is discussed for only **3C** here and the attack explanation for other designs is placed in Appendix D.

4.4.1 Long-message 2^{nd}-preimage Attack on 3C

The 2^{nd}-preimage attack on a t-bit **3C** hash function H starts with a long target message for which a 2^{nd} preimage needs to be found. First build the CCS by constructing a 2^t 2-block multicollision to control the checksum, build an expandable message and append it to the CCS and then perform the long message second preimage attack from the end of the expandable message. Then use the CCS to adjust the accumulation chaining value at that point to match the desired value which is equivalent to adjusting the checksum of the 2^t 2-block multicollision. Finally, expand the expandable messages to make up for all the message blocks skipped in the long message second preimage attack resulting in a new message which hashes to the same digest as the long target message.

ALGORITHM: LongMessageAttack(M_{target}) on 3C

Find the second preimage for a message of $2^d + d + 2t + 1$ *blocks.*

Variables:

1. M_{target} = the target long message for which a second preimage is to be found.

2. M_{link} = linking message block used to connect the cascade chaining value at the end of the expandable message to some point in the sequence of the cascade chaining values of the target message.

3. H_{exp} = the intermediate cascade chaining value at the end of the expandable message.

4. H_t = the result of the 2^t 2-block multicollision on H starting from the initial state.

5. M_{final} = the second preimage of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

6. M_{pref} = the checksum control prefix obtained from the CCS to force the linear checksum to the desired checksum.

Steps:

1. Compute the intermediate hash values for M_{target} using H:
   - H_0 and h_0 are the initial states on the cascade and accumulation chains respectively.
   - M_i is the i^{th} message block of M_{target}.
   - $H_i = f(H_{i-1}, M_i)$ and $h_i = H_i \oplus h_{i-1}$ are the i^{th} intermediate hash values on the cascade and accumulation chains respectively.
   - The cascade and accumulation chaining states are organised in some searchable structure for the attack, such as hash table. The elements $H_1, \ldots, H_d$ and the chaining values obtained in the processing of t 2-block messages are excluded from the hash table as the expandable message cannot be made short enough to accommodate them in the attack.

2. Build a CCS by constructing a 2^t 2-block multicollision on H as described in Section 4.3.2 starting from the initial state H_0. Let H_t be the multicollision chaining value. The corresponding checksum value h_t due to the 2^t 2-block multicollision on H is random and its value depends on the choice of the t 2-block messages from the CCS that give the collision H_t.

3. Construct a $(d, d+2^d-1)$-expandable message M_{exp} with H_t as the starting chaining state using either of the expandable message construction methods from sections A.1 and A.2 in Appendix A. Append the expandable message M_{exp} to the CCS. Let H_{exp} be the cascade chaining value at the end of the expandable message M_{exp}.

4. Test message blocks from the end of H_{exp} to find a linking message block M_{link} such that $f(H_{exp}, M_{link})$ matches one of the cascade chaining values stored in the hash table while processing M_{target}. Let this matching value of the target message be H_u and the corresponding accumulation chaining value be h_u where $d + 2t + 1 \leq u \leq 2^d + d + 2t + 1$.

5. Use the CCS built in step 2 to find the checksum control prefix M_{pref} to adjust the accumulation chaining value at that point to match the desired accumulation value h_u in the target message M_{target}. This is equivalent to adjusting the checksum value at the end of the 2^t 2-block multicollision. M_{pref} is obtained by solving a system of $t \times t$ linear equations as outlined in Section 4.3.3.

6. Expand the expandable message to produce a message M^* which is $u-1$ blocks long.

7. Return the second preimage $M_{final} = M_{pref}||M^*||M_{link}||M_{u+1} \ldots$ $M_{2^d+d+1+2t}$ of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

Work: The computational effort required to perform the second preimage attack on **3C** involves the effort in finding a 2^t 2-block multicollision plus the effort in solving a $t \times t$ system of linear equations plus the work in finding the expandable message plus the work to find the linking message. So, the only additional effort in performing the second preimage attack on **3C** over Merkle-Damgård hash function is the work load in solving a $t \times t$ system of equations and producing a 2^t 2-block multicollision.

1. Using the generic expandable-message finding algorithm, this effort equals $t \times 2^{t/2} + d \times 2^{t/2+1} + 2^{t-d+1}$ computations of the compression function and t^3 bit-XOR operations.

2. Using the fixed-point expandable-message finding algorithm, this effort equals $t \times 2^{t/2} + 3 \times 2^{t/2+1} + 2^{t-d+1}$ computations of the compression function and t^3 bit-XOR operations.

Illustration:

Consider a second preimage attack on **3C** with SHA-256 compression function (**3C**-SHA-256). The longest possible message for SHA-256 is $2^{64}-1$ bits, which is just less than 2^{55} blocks. Assume that the target message is $2^{54}+54+512+1$ blocks long.

1. Process the target long message using **3C**-SHA-256 and store all the intermediate hash values of both the chains.

2. Build a 2^{256} 2-block multicollision on **3C**-SHA-256. This requires effort of $2^8 \times 2^{128} = 2^{136}$ compression function evaluations.

3. Construct a $(54, 54 + 2^{54} - 1)$ expandable message from the end of the multicollision. This takes effort 54×2^{129} compression function evaluations.

4. Try about 2^{203} linking message blocks from the end of the expandable message until a match with one of the cascade chaining values from step 1 is obtained. This requires evaluating about 2^{203} compression functions on average.

5. Use the CCS built in step 1 to force the end of the multicollision to a desired value. This is equivalent to solving a system of 256×256 equations and it takes effort of 2^{24}.

6. Produce the second preimage by expanding the expandable message to compensate for the blocks skipped over in the long target message. This takes negligible amount of time.

The total cost to find a second preimage for **3C-SHA-256** using the long message 2nd preimage attack for a message of $2^{54}+54+512+1$ blocks is $2^{136}+54\times 2^{129}+2^{203}$ compression function computations and 2^{24} effort required to solve the system of equations.
Summary of the Attack: The long-message 2nd preimage attack on a t-bit **3C** hash function is summarised as follows: For a target message substantially less than $2^{t/2}$ blocks in length, the effort in performing the attack is dominated by long message attack. Thus, a 2nd preimage attack on a 2^d block message requires about 2^{t-d+1} compression function computations assuming abundant memory.

4.4.2 Herding Attack on 3C

The following steps account for the herding attack on a t-bit **3C** hash function H:

1. Construct a 2^d hash value wide diamond structure for H and output the final hash value H_f. The final hash value H_f, which is the output of the compression function g, is computed using any of the possible 2^{d-1} checksum values or some value chosen arbitrarily. Let h_c be that checksum value.

2. Build the CCS using a 2^t multicollision over 2-block messages. Let H_t be the chaining value on the cascade chain after the 2^t 2-block multicollision on H.

3. When challenged with the prefix message P, process P using H_t as the starting chaining value on the cascade chain. Let $H(H_t, P) = H_p$.

4. Find the linking message M_{link} such that the state $H(H_p, M_{link})$ matches one of the 2^d outermost chaining values on the cascade chain in the diamond structure. If the match is compared against all of the $2^{d+1}-2$ intermediate chaining values then a $(1, d+1)$-expandable message must be produced at the end of the diamond structure ensuring that the final herded message is always a fixed length.

5. Use the CCS computed in step 2 to force the checksum of the herded message P to h_c using the techniques to bypass the checksum in the 2^t multicollision described in Section 4.3.2. Let M_{pref} be the checksum control prefix obtained after solving the system of equations due to the CCS.

6. Finally, output the message $M = M_{pref}||P||M_{link}||M_d$ where M_d are the message blocks contributed to the construction of the diamond structure. The value $H(M)$ will be the same as the chosen target H_f.

Work: The work to perform the herding attack on **3C** is the work required to build the CCS plus the effort to solve the system of equations due to the CCS plus the work required to perform the herding attack from [137]. This equals $t2^{t/2} + 2^{t/2+d/2+2} + 2^{t-d}$ computations of the compression function and t^3 bit-XOR operations assuming that only the outermost 2^d chaining values are used for searching in the diamond structure. If all the $2^{d+1}-2$ intermediate chaining values are used for searching in the diamond structure then the work required equals $t2^{t/2} + 2^{t/2+d/2+2} + d \times 2^{t/2+1} + 2^{t-d-1}$ computations of the compression function and t^3 bit-XOR operations to solve the system of equations.
Illustration: Consider the application of herding attack on **3C-SHA-256** with the width of the diamond structure $d = 84$.

1. Construct a 2^{84} hash value wide diamond structure for **3C-SHA-256** as part of the precomputation. This costs $2^{128+48+2} = 2^{178}$ computations of

the compression function of SHA-256. Let h_c be the checksum value used to compute the committed hash value H_f.

2. Build a CCS using a 2^{256} multicollision for **3C-SHA-256** over 2-block messages from the initial state of SHA-256. Let H_t be the chaining value on the cascade chain of **3C-SHA-256** at the end of this multicollision.

3. Process the challenged prefix P using the state H_t to obtain the state H_p.

4. Try about $2^{256-84} = 2^{172}$ message blocks to link H_p to one of the widest chaining states in the diamond structure computed in step 1. Let M_{link} is the linking message block.

5. Use the CCS developed in step 2, to find the checksum control prefix M_{pref}. This costs 2^{24} effort to solve a system of 256×256 equations.

6. Output the herded message $M_{pref}||P||M_{link}||M_d$ where M_d are the message blocks used in step 1 to develop the diamond structure.

The total work to produce a message hashing to the hash value committed in the past using **3C-SHA-256** with $d = 84$ is $2^{178}+256\times 2^{128}+2^{172}$ computations of the compression function of SHA-256 and 2^{24} work to solve the system of equations.

4.4.3 The Generic Attacks on Better Linear Checksums

The techniques used to bypass the linear checksums described in Section 4.3 can be extended to any linear checksum which is reasonably short. Consider a 512-bit CRC computed over a message and used as the final checksum block. Now we construct the CCS using a 2^{512} Joux multicollision and then append whatever message at the end of the multicollision to perform the generic 2nd-preimage and herding attacks. Each bit of the 512-bit CRC is a linear function of 512 binary variables $a[1], a[2], \ldots, a[512]$ where $a[i]$ selects a message block from one of the sides of the collision in the CCS. One can perform the generic 2nd-preimage and herding attacks ignoring the checksum value, then solve the resultant system of 512×512 equations to force the checksum to the value necessary to make the attack work. Note that these attacks also work on **3CCW+** proposed in Chapter 3 and **3CM** proposed in [81].

4.4.4 Making Meaningful Messages in the Attacks

Using Yuval's trick [300], the attacker can come up with two documents where one is genuine and the other one a forgery and can vary the meaning in both these documents in such a way that at some point of variation they collide when processed using the same hash function. The attackers can use this trick to construct linear checksum control sequences for the hash functions that maintain checksums using message blocks or chaining values or both using meaningful colliding messages based on the brute force collision finding techniques. This is possible when the CCSs to bypass the linear checksums of all these hash functions are constructed from individual collisions that span over multiple message blocks as in (Dear Fred/Freddie,)(Enclosed please find/I have sent you) (a check for \$100.00/a little something) and so on, where the attacker can choose either side of the slash for the next part of the sentence. In that case, any choice for the CCS used to bypass the checksum will be a meaningful message. The impact of this attack is that one can construct not only meaningful collisions for these hash functions but also second preimages and herded messages with genuine meaning.

4.4.5 Some Other Techniques to Perform the Generic Attacks

Independent to this work, Mironov and Narayanan [197] has found an alternative technique to bypass the checksum in GOST-L. While the approach discussed in Section 4.3.5, requires b 1-block collisions to find pairs (M_i, N_i) for $i = 1$ to b, their technique considers repetition of the same message block twice for a collision. That is, their technique aims for b pairs $((M_i, M_i), (N_i, N_i))$ that give a collision after processing every two compression functions. In contrast to the methods presented in this chapter for solving system of equations for the whole message, their approach solves the system of equations once after processing every few message blocks. It should be noted that this constrained choice of messages would result in a zero checksum at the end of the 2^b multicollision on GOST-L.

Note that this constrained choice of message blocks to defeat the linear checksum, thwarts the attempts to carry out the 2^{nd} preimage attack on GOST-L. The reason is that the attacker will lose ability to control the checksum after finding the linking message from the end of the 2^b multicollision for GOST-L which gives a collision with some intermediate chaining value in the target message. That is, the linking message creates a checksum which cannot be adjusted flexibly by the techniques disclosed in Section D.1 to the checksum of the target message. This is due to the constraint on the choice of messages in the multicollision.

However, this technique with a twist is used to perform the herding attack on GOST-L. The attacker chooses the messages for the diamond structure that all have the same effect on the linear checksum. These messages would result in a zero checksum at every stage in the diamond structure. Once the attacker is forced with a prefix, processing the prefix gives a zero checksum to start with and then solving a system of equations will find a set of possible linking messages that will all combine with the prefix to give a zero checksum value.

When the approach of [197] to bypass the checksum is applied to hash functions with linear checksums computed using chaining values such as **3C**, **3CM** and F-Hash, it requires a 2-block collision finding algorithm to output the same pair of message blocks on either side of the collision whenever it is called. In general, in collision attacks such a requirement on the constrained choice of message input is not imposed and in many cases it does not work. However, it is quite capable of defeating linear checksums in many generic attacks. Because it is so different from our technique, some variant of this technique might be useful in cryptanalytic attacks for which our technique does not work.

4.5 Collision Attacks on GOST-L, 3C and F-Hash

As pointed out in sections 4.3.3 and 4.3.4, it would be hard to build the CCS if the collision finding algorithm used to construct multicollisions in GOST-L and **3C** produces structured collisions. Though one cannot perform generic attacks when there are structured collisions, one can still perform collision attacks on GOST-L, **3C** and F-Hash designs as described below.

Consider a collision finding algorithm $C(s, 1)$ with the state $s = H_0$ for the GOST-L hash function H. A call to $C(s, 1)$ results in a pair of message blocks (M_1, N_1) such that $M_1 \oplus N_1 = \Delta$ and $H(H_0, M_1) = H(H_0, N_1) = H_1$. Now a call to $C(s, 1)$ with the state $s = H_1$ results in a pair of blocks M_2 and N_2 such that $M_2 \oplus N_2 = \Delta$ and $H(H_1, M_2) = H(H_1, N_2) = H_2$. Now two messages M and N are constructed where $M = M_1||M_2$ and $N = N_1||N_2$ with the checksums

$M_1 \oplus M_2$ and $N_1 \oplus N_2$ respectively. Since $M_1 \oplus N_1 = \Delta$ and $M_2 \oplus N_2 = \Delta$, $M_1 \oplus M_2 = N_1 \oplus N_2$, a collision for GOST-L. Hence, by just appending two structured collisions from $C(s, 1)$ end to end and get a collision for GOST-L.

Similarly, consider a collision finding algorithm $C(s, n)$ with the state $s = H_0$ for the **3C** hash function H. Let $n = 2$ and a call to $C(s, 2)$ results in a pair of messages (M, N) where $M = M_1||M_2$ and $N = N_1||N_2$ such that $H(H_0, M_1) = H_1$, $H(H_0, N_1) = H_1^*$, $H_1 \oplus H_1^* = \Delta$ and $H(M) = H(N) = H_2$. Now a second call to $C(s, 2)$ with $s = H_2$ results in two pairs of blocks (M_3, M_4) and (N_3, N_4) such that $H(H_2, M_3) = H_3$, $H(H_2, N_3) = H_3^*$, $H_3 \oplus H_3^* = \Delta$ and $H(H_3, M_4) = H(H_3^*, N_4) = H_4$. Since, $H_1 \oplus H_1^* = \Delta$ and $H_3 \oplus H_3^* = \Delta$, $H_1 \oplus H_2 \oplus H_3 \oplus H_4 = H_1^* \oplus H_2 \oplus H_3 \oplus H_4^*$, a collision for **3C**. Hence, concatenation of two structured 2-block collisions due to near-collisions give a collision on **3C**. Similarly, one can construct collisions for **3C** using multi-block collision format (M_1, N_1) and M_1, N_2. These collision attacks on **3C** are independently performed by Kelsey [135], Tuma and Joscak [279] and Lucks [175]. In reality, this attack works when a 2-block collision for a hash function H can be produced using any state and a fixed differential path for the chaining values. Tuma and Joscak [279], in fact, performed the attack using the compression functions of MD5 and SHA-0 and provided the test vectors of their collisions. They explicitly showed that the conjecture on the difficulty in finding multi-block collisions for **3C** made in Section 3.2 of Chapter 3 as incorrect for **3C** with the compression functions of MD5 and SHA-0. The complexity to find collisions for **3C** with MD5 is twice that of on MD5 and for **3C** with SHA-1 is the same as SHA-1. So far, no one has demonstrated collisions for SHA-1 though there is a theoretical 2-block collision attack based on near-collisions on it [287, 290]. Again, if collisions can be produced for SHA-1 using any chaining state, concatenation of such two consecutive 2-block collisions will result in a collision for **3C** with SHA-1.

Consider a collision finding algorithm $C(s, n)$ with the state $s = H_0$ for F-Hash hash function H. Let $n = 2$ and a call to $C(s, 2)$ results in a pair of messages (M, N) where $M = M_1||M_2$ and $N = N_1||N_2$ such that $H(H_0, M_1) = (H_1, h_1)$, $H(H_0, N_1) = (H_1^*, h_1^*)$, $H_1 \oplus H_1^* = \Delta_H$, $h_1 \oplus h_1^* = \Delta_h$ and $H(M) = H(N) = (H_2, h_2)$. Now a second call to $C(s, 2)$ with $s = H_2$ results in two pairs of blocks (M_3, M_4) and (N_3, N_4) such that $H(H_2, M_3) = H_3, h_3$, $H(H_2, N_3) = H_3^*, h_3^*$, $H_3 \oplus H_3^* = \Delta_H$, $h_3 \oplus h_3^* = \Delta_h$ and $H(H_3, M_4) = H(H_3^*, N_4) = H_4, h_4$. Since, $H_1 \oplus H_1^* = \Delta_H$ and $H_3 \oplus H_3^* = \Delta_H$, $H_1 \oplus H_2 \oplus H_3 \oplus H_4 = H_1^* \oplus H_2 \oplus H_3 \oplus H_4^*$, a collision on the cascade chain for F-Hash. In addition, since, $h_1 \oplus h_1^* = \Delta_h$ and $h_3 \oplus h_3^* = \Delta_h$, $h_1 \oplus h_2 \oplus h_3 \oplus h_4 = h_1^* \oplus h_2 \oplus h_3 \oplus h_4^*$, a collision on the accumulation chain for F-Hash.

Hence, concatenation of two structured 2-block collisions due to near-collisions give a collision on F-Hash like on **3C** under the assumption that 2-block collisions are attainable for F-Hash using any state. Similarly, one can construct collisions for F-Hash using multi-block collision format (M_1, N_1) and (M_1, N_2).

4.6 Finding Multiple 2^{nd} preimages for Hash Functions in less than 2^t Work

The generic attacks on the linear checksums for the Merkle-Damgård hash functions have provided some insights to improve the complexities of Joux [126] attacks on hash functions.

It was noted in Chapter 2 the application of Joux multicollision attack on a t-bit hash function to find 2^k 2^{nd} preimages with an effort of $k \times 2^{t/2} + 2^t$. It should be noted that to find 2^k second preimages, the attacker has to at least spend 2^t computational effort needed to find a single 2^{nd} preimage.

Note that the multicollision technique can be combined with the generic 2^{nd} preimage attack technique from [138] to find multiple 2^{nd} preimages for hash functions in less than 2^t effort. To find multiple 2^{nd} preimages for a given long target message, the attacker first finds multiple collisions for the hash function from the IV of the hash function. Then performs the second preimage attack with the multi-collided state as the starting state. Once the attacker finds one 2^{nd} preimage, then the attacker could choose any of the possible messages in the multicollision and they would all be the second preimages for the target message.

The following algorithm is used to find multiple second preimages for the Merkle-Damgård hash function.

ALGORITHM: MultiSecondPreimageAttack(M_{target})

Find 2^k second preimages for a message of $k + 2^d + d + 1$ blocks.

Variables:

1. M_{target} = The message for which 2^k second preimages need to be found.

2. M_{link} = The linking message block to link the expandable message to some point in the target's message's sequence of intermediate hash values.

3. H_{exp} = The intermediate chaining value at the end of the expandable message.

Steps:

1. Compute the intermediate hash values for the target message M_{target} of $k + 2^d + d + 1$ message blocks using the following steps.

    - H_0 = The IV of the hash function.
    - M_i = The i^{th} message block of the target message.
    - $H_i = f(H_{i-1}, M_i)$, the i^{th} intermediate hash value. It is assumed that chaining values are organized in some searchable structure such as hash table for the attack. The chaining values $H_1, \ldots, H_d, H_{d+1}, \ldots, H_{d+k}$ are excluded from the hash table as the expandable message cannot be made short enough to have them in the attack.

2. Find a 2^k-multicollision starting from the initial state H_0. Let H_k be the intermediate chaining value at the end of the 2^k multicollision. Let $M^1, M^2, \ldots, M^{2^k}$ be 2^k different messages in the 2^k multicollision all hashing to the same digest H_k.

3. Construct a $(d, d + 2^d - 1)$-expandable message starting from the chaining state H_k following either of the techniques to find expandable messages from [138]. The intermediate chaining value at the end of the expandable message is H_{exp}.

4. Try linking message blocks M_{link} until $f(H_{exp}, M_{link}) = H_j$ where H_j is the intermediate chaining value in the target message after $d + k$ blocks in the target message for $d + k + 1 \leq j \leq 2^d + d + k + 1$.

5. Expand the expandable message to produce a message M^* of $j - 1$ blocks long.

6. Return 2^k 2nd preimages defined as $M_{sec} = M^i || M^* || M_{link} || M_{j+1} || \ldots M_{2^d+d+1}$ where $i = 1, 2, \ldots, 2^k$ such that $H(M_{sec}) = H(M_{target})$.

Work: The total work done to find multiple second preimages for H is the total work to find multicollision, expandable message and the work to find the linking message.

1. For the generic expandable message-finding algorithm, it costs about $k \times 2^{t/2} + d \times 2^{t/2+1} + 2^{t-d+1}$ calls to the compression function of the hash function.

2. Using a fixed point expandable message-finding algorithm, this effort is $k \times 2^{t/2} + 3 \times 2^{t/2+1} + 2^{t-d+1}$.

It should be noted that for simplicity, it is assumed in the above algorithm that the target message contains an extra k blocks to compensate the length required for the multicollision in order to find multiple second preimages compared to the length of the target message used in finding a 2^{nd} preimage [138].

An Illustration:

Here a 2^4 second preimage attack is illustrated against the SHA-256 [211] hash function. The maximum length of the message that can be hashed using SHA-256 is $2^{64} - 1$ bits, which is just less than 2^{55} blocks. For easy analysis, assume that the target message is $2^{53} + 53 + 1 + 2^4$ message blocks. The following steps are used to find 2^4 second preimages for this target message.

1. Process the target message of length $2^{53} + 53 + 1 + 2^4$ message blocks and store all the intermediate chaining values.

2. Construct a 2^4-multicollision starting from the IV of the hash function SHA-256.

3. Construct a $(1, 53 + 2^{53})$-expandable message from the end of the multi-collision. This requires about $53 \times 2^{128+1}$ compression function iterations. Let the intermediate chaining value at the end of the expandable message is H_{exp}.

4. Starting from the end of the expandable message, test about 2^{202} message blocks until a message block whose intermediate hash value matches one of the last $53 + 2^{53}$ intermediate hash values of the target message.

5. Finally, expand the expandable message to compensate for the message blocks skipped in the target message and produce the second preimage. This requires a negligible computational work.

6. By choosing any of the 2^4 messages that resulted in the multicollision, the other second preimages are found.

Therefore the work required to find 16 2^{nd} preimages for a message of $2^{53} + 53 + 1 + 2^4$ blocks for SHA-256 is about $4 \times 2^{128} + 53 \times 2^{129} + 2^{202}$ computations of SHA-256 compression function compared to Joux complexity of $2^{256} + 4 \times 2^{128}$.

4.6.1 Multi-2^{nd} preimage Attack on the Cascade Constructions

This trick to find multiple 2^{nd} preimages for a given long target message can be used to find multiple 2^{nd} preimages for the cascade hash functions at a cost less than the previously known result of Joux [126]. For a cascade hash function $H||G$ which yields an output of $H_f||G_f$ after processing a message M where $H_f = H(M)$ and $H_g = G(M)$, Joux has shown that it would cost $H_g \times 2^{H_f/2} + 2^{H_f} + 2^{H_g}$ if $H_f \leq H_g$ (resp. $H_f \times 2^{H_g/2} + 2^{H_f} + 2^{H_g}$ if $H_f \geq H_g$) to find a second preimage for a message M. It should be noted that this attack requires more than an effort of finding one second preimage for each of the hash functions in the cascade construction.

However, this attack on the cascade construction can be significantly improved using the long message 2^{nd} preimage attack based on expandable messages. Assume that the target message of each of the hash functions in the cascade construction is of length $2^d + d + 1$ blocks. The 2^{nd} preimage attack can be applied to each of the hash functions H and G separately to get individual 2^{nd} preimages which by combining gives the second preimage for the cascade construction. Since length padding is used in both the hash functions separately, this attack is just an application of the 2^{nd} preimage attack on each of the hash functions. Using generic expandable message finding algorithm, it costs about $d \times 2^{H_f/2+1} + 2^{H_f-d+1} + d \times 2^{H_g/2+1} + 2^{H_g-d+1}$ compression function computations to find a 2^{nd} preimage for the cascade construction.

Now assume that the length of the target message for H is $2^k + 2^d + d + 1$ message blocks and for G is $2^d + d + 1$ message blocks. The trick used to find

multiple 2nd preimages given in the previous section can be employed on H to find 2^k 2nd preimages for H and the generic second preimage attack trick is employed on G to find a single 2nd preimage. This leads to finding 2^k second preimages for the cascade construction for only an additional factor of 2^k multicollision in addition to the work of finding a single 2nd preimage. The total cost is $k \times 2^{H_f/2} + d \times 2^{H_f/2+1} + 2^{H_f-d+1} + d \times 2^{H_g/2+1} + 2^{H_g-d+1}$ compression function computations.
Summary of the attack: The combination of Joux multicollision and the 2nd preimage attack on the Merkle-Damgård construction is quite powerful. This combination can be used in finding multiple second preimages for hash functions in less than 2^t work assuming abundant memory.

4.7 Conclusion

This chapter shows that using linear checksums for the Merkle-Damgård construction do not make several attacks harder than performing them on the Merkle-Damgård hash functions. These results demonstrate that not all forms of maintaining twice the hash size as the internal state size is good enough for the security of the hash function.The valuable insight of this chapter is that the manner in which the largish internal state is maintained is important for the security of the hash function.

Chapter 5

Iterated Halving- A New Approach for Hash Function Design

From chapter 1, it is quite evident that the security goals of hash functions are many and wide spread compared to encryption schemes. Yet, while there is a great diversity of design approaches and many proposed algorithms for *encryption* there is still a quite limited range of design approaches and algorithms for constructing secure and efficient cryptographic hash functions. For example, see the USA's Advanced Encryption Standard (AES) process [218], the European Schemes for Signatures, Integrity, and Encryption (NESSIE) [229] standardisation effort and the current eSTREAM project [209] of European Union (EU) European Network of Excellence for Cryptology (ECRYPT) network for identifying stream ciphers that might become suitable for widespread adoption.

Perhaps this lack of direct research in the design of hash functions is due to its close relationship with the two main objects of symmetric key cryptology: stream ciphers and block ciphers. Given a truly secure instance of one of these primitives, it has been shown possible to derive secure instances of the other algorithm types from it [4, 205][1]. However, these constructions tend to be relatively inef-

[1]This argument is also valid in the design of MAC schemes where MACs are proven secure assuming the secure properties of other primitives. See [13, 15, 16, 33, 34, 122, 160, 161] for examples.

ficient compared to the direct/dedicated designs [200]. So, until the completion of the AES standardisation process, there was a great work in the cryptographic community in the design of block ciphers. In addition, there has been recently a great effort in the cryptographic community in designing new stream ciphers that are both efficient and secure. However, the lack of variety of algorithms proposed for the NESSIE standards processes for hash functions [230, 231, 233] shows clearly that much more expertise has been used in the design of block and stream ciphers [32]. For hash functions, all early efforts were directed towards adapting block ciphers, such as the first encryption standard, Data Encryption Standard (DES) [204].

As the use of hash functions is ubiquitous, being even more prevalent than the applications of symmetric block ciphers and stream ciphers, it is a reasonable expectation that a simultaneous research effort has been directed at the design of hash functions. Enigmatically, this is not the case. Research in hash function design has been sparse and minimalist. The vastly influential customised/dedicated MD4 family of hash functions (see Appendix B) are all structured using an UFN style block cipher surrounded by the usual feed-forward technique. These so called customized “alternatives” to hash functions based on *well-established* block ciphers, are nothing more than hash functions based on other less studied block ciphers (until recently [28, 100, 113, 143, 263] to some extent). Except for the MARS [29, 38, 140] block cipher, this type of internal structure is largely unstudied, as balanced Feistel networks are much more common structures used in block ciphers. For example, see other ciphers such as Camellia [7] which was selected by NESSIE and recommended by the Japanese CRYPTREC project [115, 117] and SEED [156] which was developed by the Korean Information Security Agency (KISA) and used broadly through out South Korean Industry.

With increasing interest in all forms of cryptanalysis on the popular and some widely used hash functions [20, 25, 26, 42, 60, 68, 71, 95, 105, 126, 137, 138, 142, 144, 145, 172, 202, 226, 264, 265, 285–287, 289, 290, 297, 298] this limited choice in the popular hash function design might become a single point of failure for information processing applications that use hash functions for their security, and it is essential to seek alternative paradigms. In addition, the linear checksum variants of the Merkle-Damgård hash functions proposed in chapters 3 and 4 are also shown to be insecure against both generic and specific attacks. There have been some efficient hash function proposals such as PANAMA [52] and more recently SMASH [149] that differ from Merkle-Damgård style of operation, but they have not survived collision attacks [227, 250].

In this chapter, an entirely new approach to the hash function design called iterated halving (IH) is proposed. This class of hash function algorithms can be instantiated with any reduced round block cipher, allowing a subtle security/performance trade-off and a direct performance comparison with the existing approach of hashing. The specification of a new hash function called CRUSH obtained from this IH class is presented. This design is partially analysed with respect to some of the known attacks on iterated hash functions.

This chapter is organised as follows: Section 5.1 introduces the IH structure and some analysis of the IH class of hash functions is presented in Section 5.4. In Section 5.5 a specific instance of IH family of hash functions called CRUSH is proposed.

5.1 Iterated Halving Structure

Firstly, the concepts leading to the definition of IH structure shown in Figure 5.4 are discussed. This is done by recalling tight relations between the feed-forward mode of iterated hash functions based on block ciphers, stream ciphers formed by the output feedback (OFB) mode of a block cipher [73], cipher block chaining (CBC) mode of message encryption [73] and the key feedback mode of the key stream generation [103] as shown in Figures 5.1, 5.2 and 5.3. Note that the feed-forward mode of hash functions based on block ciphers or pseudo random permutation (PRP) networks shown in Figure 5.1 is basically the structure of hash function based on Davies-Meyer compression function discussed in chapter 2. The CBC and OFB modes of encryption are very close to each other. When the plaintext P_i is all zero, the CBC mode becomes identical to the OFB mode. When the nonlinear operation of the OFB mode is relaxed from PRP to a more general Pseudo Random Function (PRF), then it becomes identical to a simple hash function based on block cipher as depicted in 5.1. In Figure 5.1, the initial state of the hash function is denoted by IV, the intermediate chaining value at any state i with H_i, the message block at any state i with M_i and the final digest with H_f.

A generic iterated halving technique for hashing instantiated with any block cipher $\mathbf{B}$ with the key schedule algorithm f is shown in Figure 5.4. The IH structure could be seen as a hybrid of CBC/OFB/KFB, plus other design concepts. Any choice of the "**B**-function", an algebraically complex, highly diffusive and nonlinear bijection on b-bit blocks, could be made for the IH structure. When this choice is made as a possibly reduced round block cipher, then a direct performance comparison can be made with hash functions based on block ciphers. In particular, it can be shown that when the **B**-function has half as many rounds as a complete block cipher, then an equivalent hash rate of 1 is obtained. The IH network is clearly outside the class of some rate 1 circuits that were successfully cryptanalysed [146, 150, 163]. So IH offers an *improved* speed/security trade-off compared to the classic feed-forward approach used in the MD4 family of hash functions.

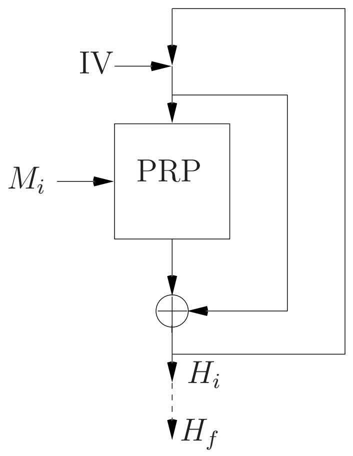
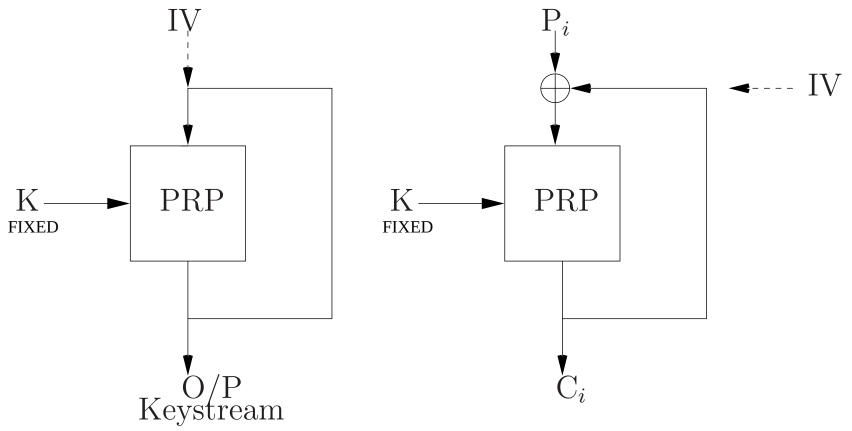


Figure 5.1: Standard Hashing using a Block Cipher



(a)OFB MODE of a Block Cipher (b)CBC MODE of a Block Cipher

Figure 5.2: Similarities to stream and message encryption

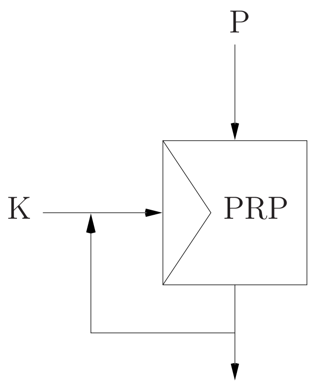


Figure 5.3: Key Feedback Mode

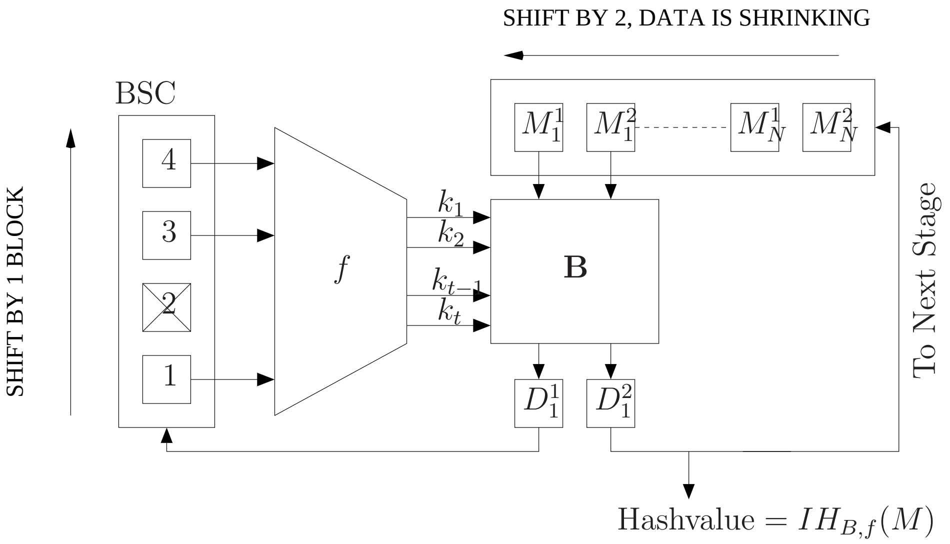


Figure 5.4: Iterated Halving Technique

5.2 Working details of an IH hash function

The IH approach to hashing is motivated by the well known observation that $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 2$. In addition, consider modifying the CBC mode of encryption as follows: In the CBC mode of encryption, a sequence of input data blocks is converted into an equal length sequence of output blocks as shown in Figure 5.2. Now this mode is modified so that only *half* of each output block is kept as data for the next stage processing and the other half is discarded. In the next stage, the accumulated data blocks output from the previous stage are used as input. This process is continued until the desired number of output data blocks are obtained for the digest. If N is the total number of b-bit blocks in the message (including padding), it is clear from these observations that no more than $2N$ block encryptions would be performed in the whole IH process.

By reducing the number of rounds of the bijective nonlinear operation $\mathbf{B}$ down to half the number of rounds of the full sized block cipher, an equivalent complexity of not more than N full encryptions to process N b-bit block message can be obtained. This achieves the desired rate of 1. Finally, to produce the message digest, the iterated halving process is stopped when the state is down to 256 bits or any other desired value. Another change from CBC is to use the other half of each output block as input to a bijection selection component (BSC) which is initially loaded with a fixed public constant represented as four $b/2$-bit values. The BSC structure seeks similar security advantages as KFB mode. The output of BSC works as master key for the non-linear key-scheduling algorithm f of the permutation network $\mathbf{B}$ generating round keys $k_1, k_2 \ldots, k_{t-1}, k_t$. At any stage, the count of input data blocks is not a even number, there will be only a single block (not a pair) remaining for processing at the end of that stage. In this case, the unused $b/2$-bit data block from the BSC (denoted with a cross in Figure 5.4) is copied and used as the second half-block of the final b-bit input for that stage.

The overall idea of IH process is shown in Figure 5.5. The IH process can be seen as a multi stage compression process. The number of halving stages required depends on the size of the initial message with padding and the desired size of the message digest. An appropriate padding, similar to MD4 family of hash functions, is applied to the input message M. In Figure 5.4, each data block of M of size b to be hashed is represented in half blocks of size $b/2$. That is, for

$i = 1$ to N, $M_i = M_i^1 || M_i^2$. Assuming that $b = 128$ bits, $M_i^1 = 64$ and $M_i^2 = 64$ and the last 64-bit block M_N^2 contains the binary encoded representation of the length of the message M as part of the padding. At any stage, the size of the internal state of the IH hash function is the sum of both the amount of data in the BSC section and the amount of data proceeding into the next halving stage.

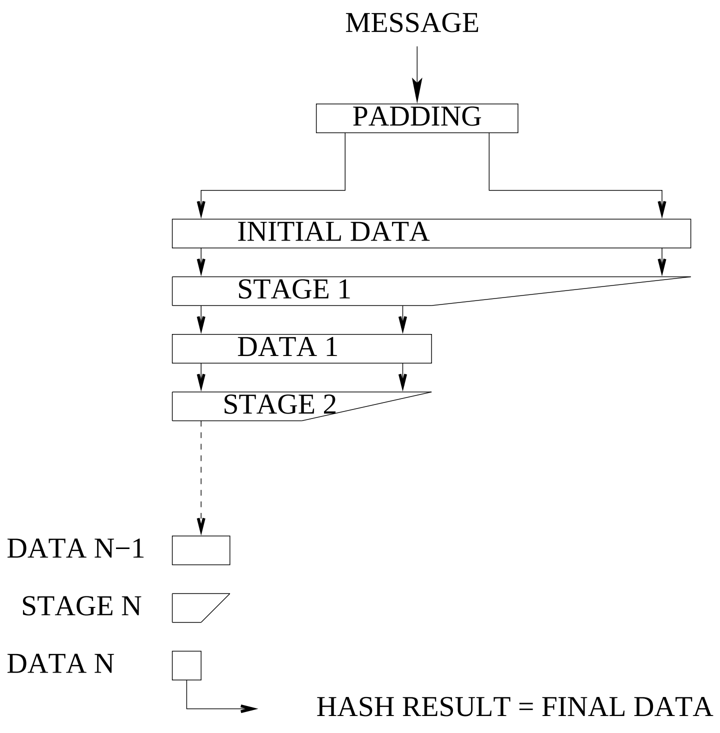


Figure 5.5: Iterated Halving (IH) process

5.3 Performance and Security of the IH Process

- For performance, the total number of executions of the B-function ($\#B$) is counted and the equivalent rate R is calculated using the formula $R = \#B * p$ where the relative performance is expressed as a proportion p of the number of rounds in a full block cipher.
- For security, the minimum number of executions of the B-function that could exist in any path connecting the input and output bits are counted.

> Counting the rounds of block ciphers is a rough but established way [23] to make speed comparisons and to discuss broad margins for error in the security of ciphers. The depth D is the total number of stages in the IH processing of the message and clearly provides a lower bound on the number of executions of the B-function used by any approximation. Let C be the equivalent minimum security complexity, then C is given by $C = D * p$, where $C = 1$ indicates at least the equivalent algebraic complexity expected from a full block cipher. Hence, the minimum depth for security equivalent to a block cipher is $D_{min} = \frac{1}{p}$. In a recommended case of $p = \frac{1}{2}$, the limit is set to $D_{min} = 2$, so at least two halving stages must be performed. Given a minimum fixed length hash output of 256 bits (to obtain a 128-bit security level for a protection from birthday collision attacks), this imposes a lower bound of 1024 bits on the size of the message including padding.

If $len(M)$ is the length of the message with padding and $len(H)$ is the desired size of the hash value then the depth of an instance of IH is calculated using the formula $D \leq log_2 \frac{len(M)}{len(H)}$, rounded up to the next integer, or D_{min}, whichever is larger. A set of parameters $\{len(M), len(H), p\}$ is valid if and only if $1/p \leq log_2 \frac{len(M)}{len(H)}$. A specific instance of IH class of hash functions named "CRUSH" is discussed in Section 5.5.

Note that while Merkle-Damgård hash functions can process communicated messages "on the fly" without having to store the messages in advance, IH hash functions need to store messages before they are processed or process messages that are already buffered. This makes them not suitable for streaming applications. In addition, as explained above, the IH class of hash functions impose a requirement on the minimum length of the message to be processed which is not the case with MD4 family of hash functions. However, unlike MD4 family of hash functions that maintain the same internal state size as the size of the digest, the internal state size of the IH hash functions is at least thrice that of the size of the digest and it is exactly three times that of hash value size during the last stage of compression. For example, when the desired size of the digest of an IH hash function is 256 bits, the internal state during the last stage of compression is 738 bits (512 bits due to the data in the last stage and 256 bits due to the data in the BSC section).

5.4 Security Analysis

One of the difficult issues in hash functions is to provide a precise security analysis for new designs, especially if they differ from the known structures. To some extent, this difficulty to provide analysis for new hash function designs is due to the fact that there is little theory known about the analysis of new designs [149]. In addition, in many cases, the attacks that work on one class of hash functions do not work on the other class of hash functions. For example, the herding attack discussed in chapter 2 has originated due to the iterative nature of the Merkle-Damgård hash functions and this attack might not work for other similar or completely different structures. This implies that apart from the fundamental security properties of the hash functions, the properties that are specific to some structures (for example, CTFP preimage resistance specific to iterative structures) might not be a requirement for other structures. This section provides some analysis for IH hash functions with respect to some attacks that work on Merkle-Damgård hash functions followed by some identified advantages of this structure of hash function design.

- **Length extension attack:** This attack is not applicable to IH class of hash functions. Given the message digest of a message processed using an IH hash function, it is difficult for the attacker to compute a new digest by appending new message blocks to that digest as it is done for the Merkle-Damgård hash functions as shown in chapter 2. The reason is that the attacker can not know the value of the data in the BSC section at the instant the given digest would have been obtained. Moreover, the size of the internal state for an IH hash function is at least three times that of the size of the final digest at any stage of the compression process.

- **Herding attack:** The construction of the diamond structure as for Merkle-Damgård hash functions in order to perform the herding attack is not applicable for IH hash functions. The reason is IH hash functions maintain two different states at any stage: one due to the data in the BSC component and the other one due to the data proceeding to the next stage of the IH process. Once a collision is found for an IH hash function using a birthday attack, the collided hash value will not be part of the chaining input in the processing of next set of message blocks due to the lack of iterative structure for the IH family of hash functions.

This does not mean that herding attack is not possible on the IH class of hash functions. It only means that techniques used to construct the diamond structure in order to carry the herding attack on the Merkle-Damgård hash functions do not apply to the IH class of hash functions. There might be other ways that can be used to violate the CTFP preimage resistance property for this type of structures. In addition, this structure might demand some other security properties that do not apply for Merkle-Damgård hash functions. A similar reason holds for the inapplicability of long message second preimage attack [138] on these type of structures.

- **Preimage attack:** In this attack, the attacker is given the digest of a message and the task of the attacker is to find a message that has produced that digest. Again, for a given digest the attacker would not know the value of the data in the BSC section of the IH structure that has resulted in that digest. This makes it difficult for the attacker to reverse the B function without the knowledge of the inputs to the f function.

- **Second preimage attack:** In this attack, the attacker is given a message M and her task is to find a message $N \neq M$ and producing the same digest as M when processed using an IH hash function. It is assumed that the B function provides all the security properties expected from a good bijective non-linear function. In such a case, it is expected that any difference Δ where $M = N \oplus \Delta$ in the given message M will affect at least one of the BSC data, data proceeding into the next halving stage or probably both. Obviously, when the change in the input data affects the data proceeding to the next stage, it would not result in the same digest as the digest of M. When the change in the input data affects only the data proceeding to the BSC section, it would result in the use of different subkeys in the subsequent stages by the B-function. This makes it difficult to produce a message N having the same digest as the original message M.

- **Collision attack:**

  Consider two distinct message inputs of the same length to any instance of IH, and assume that the difference Δ in the message inputs affects only one half of the output of the B-function in the first halving stage. When two different messages produce a collision only in one half of the output of the B-function at any stage of the IH process, such a collision is called partial collision. Here two cases are considered: a collision in the BSC data and collision in the other data proceeding to the next stage. The significance of this observation is that full collisions for IH must only occur from data that has not partially collided. Let the data output of the B-function proceeding to BSC section is represented with x and the data proceeding to the next stage be denoted by y.

1. Since B is reversible a collision in one half of the output of the B-function ensures a difference Δ in the other half of the output. When the first partial collision is in y, the BSC data x must differ, so the collision in y is unlikely to persist in the following stages as in the following stage, the data input to the non-linear f function from the BSC section is different from the one in the previous stage.
2. Considering the remaining case, where let assume that a partial collision occurs in x. Since the B-function is reversible, there must remain some difference in the other internal IH data for these two messages. In the next halving stage, the situation repeats: in order to keep a collision in x, the differences in y must be accepted. Eventually, in the last halving stage, to get a collision in the hash output, differences in the BSC data must be accepted, which makes subsequent B-functions use different "subkey" values from the f-function. This prevents the other data from producing collisions automatically.

From the above analysis, one can observe the following advantages from the IH class of hash functions.

1. The state size during the operation of IH is much more than the size of the final hash output. This helps to resist guess-and-determine attacks which have been successful on some stream ciphers [41,106] to recover the internal state information.

2. The keying is irregular rather than constant as for standard CBC mode of encryption.

3. The trade-off between speed and security can be easily adjusted by selecting the number of rounds of the block cipher to use as the B-function. This provides flexibility and allows a direct performance comparison with existing hash networks based on block ciphers.

4. Partial collisions do not produce full collisions for same length messages. This property ensures that attacks based on establishing and maintaining a collision in the BSC component must fail.

5. Having a much larger internal state compared to the size of the hash values and a different iterative structure prevents the applicability of generic attacks such as length extension, herding and long message 2nd preimage attacks that work effectively on Merkle-Damgård hash functions and linear checksums to the Merkle-Damgård hash functions discussed in chapters 2 and 4.

5.5 CRUSH: A Hash Function Proposal based on IH Technique

In this section, the specification of an instance of IH class of hash functions called CRUSH is given. The security analysis of CRUSH is out of the scope of this thesis.

5.5.1 Notation and Definitions

Let the total number of halving stages in an instance of CRUSH be denoted by D and let $s = [1, \ldots, D]$. At any stage s of the IH process, one complete execution of the **B**-function is called a step and is denoted by i. At each stage s and step i, the 128-bit input data block as a concatenation of two 64-bit data blocks is denoted by $M^1_{i,s} || M^2_{i,s}$. At any stage s and step i, the 128-bit output of the **B** function is denoted by $D^1_{i,s} || D^2_{i,s}$, where $D^1_{i,s}$ is fed as input to the BSC and $D^2_{i,s}$ is either accumulated for the next stage of the IH process when $s < D$ or collected as part of the hash value when $s = D$.

The three main components of the CRUSH hash function: key scheduling algorithm, the BSC and the **B** function as depicted in Figure 5.7 are defined below.

1. **Key scheduling algorithm:** The key scheduling algorithm f is very simple. It has three Boolean functions f_1, f_2 and f_3 that convert three input values a, c and d from the BSC into three sub keys x, y and z at every step i of every stage s of the IH process. The three Boolean functions of the key scheduling algorithm f working on any 64-bit input values a, c and d are defined in Table 5.1. The 64-bit constants c_1, c_2 and c_3 used for the function f are defined in Table 5.2.

2. **BSC:** At any step i of any stage s in the execution of CRUSH, the 256-bit state of the BSC is denoted by four 64-bit blocks a, b, c and d. The initial state of the BSC is a fixed public constant and is given in Table 5.3. These four 64-bit blocks are taken from some of the fractional parts of the cube roots of the first eighty prime numbers.

3. **B-Function:** The **B**-function of CRUSH is a standard 3-round balanced Feistel structure mapping five 64-bit input data blocks (three 64-bit sub keys and two 64-bit message blocks) to two 64-bit output data blocks. The first 64-bit output data block is used for the keyfeedback and the other 64-bit data block is either used for the next stage or collected as a part of the final hash value depending on the stage count s. The **B**-function contains a 64-bit round function denoted by **F**. The **F**-function is a complex and highly non-linear mapping 64 bits to 64 bits. At any stage s and step i, the operations that take place in the **B**-function are given in Table 5.4.

   The **F**-function of CRUSH is a modified 64-bit substitution permutation network (SPN) with two inner stages as shown in Figure 5.6. The first half has a row of eight bijective 8*8 S-boxes as shown in Table 5.7, specifically chosen as the S-box proposed in [83]. Then a Pseudo-Hadamard Transform (PHT) is used for some quick mixing of the half blocks. The PHT consisting of two 32-bit modular additions is by now well known from its use in the SAFER family of block ciphers [176] as well as in the AES finalist block cipher Twofish [266]. The PHT provides cheap diffusion. The second part of the **F**-function has two parallel 32-bit linear diffusion operations which was chosen as the MIXCOL operations from AES [218], followed by a final PHT to combine the half-blocks.

   The components of the **F**-function in CRUSH have been selected to provide

Table 5.1: The key-schedule of CRUSH

Input $= \{a, c, d\}$
$f_1 = (a \oplus c.d \oplus a.c) + c_1 = x$
$f_2 = (c \oplus d \oplus c.d \oplus a.c \oplus a.d) + c_2 = y$
$f_3 = (1 \oplus a \oplus d \oplus c.d) + c_3 = z$
Output $= \{x, y, z\}$

Table 5.2: The constants for the key-schedule algorithm

$c_1 = 0\text{x}\ B5C0FBCFEC4D3B2F$
$c_2 = 0\text{x}\ E9B5DBA58189DBBC$
$c_3 = 0\text{x}\ 3956C25BF348B538$

Table 5.3: The Inital state of BSC

$a = 0\text{x}12835B0145706FBE$
$b = 0\text{x}243185BE4EE4B28C$
$c = 0\text{x}550C7DC3D5FFB4E2$
$d = 0\text{x}72BE5D74F27B896F$

Table 5.4: The operations in the **B** function

Input $= \{M^1_{i,s}, M^2_{i,s}, x, y, z\}$
$D^1_{i,s} = F(F(M^1_{i,s} \oplus z) \oplus M^2_{i,s}) \oplus y \oplus M^1_{i,s}$
$D^2_{i,s} = F(D^1_{i,s} \oplus x) \oplus F(M^1_{i,s} \oplus z) \oplus M^2_{i,s}$
Output $= \{D^1_{i,s}, D^2_{i,s}\}$

robust resistance to current and future attacks. The S-boxes in Table 5.7 have a better combination of high nonlinearity, high algebraic degree, low autocorrelation and lack of linear redundancy than any other S-box currently available in the literature. It has much better properties than can be obtained with random search techniques. The MIXCOL operation is known to provide a branch number of 5 (with respect to bytes of data), thus maximising the possible diffusion. The PHT is meant for software efficient way of mixing the half-blocks, and it is used to bind the blocks after their separate local mixing operations. It can be shown that the **F**-function is a complete mapping, where every input bit does affect every output bit. It follows that the overall **B**-function shares this security requirement.

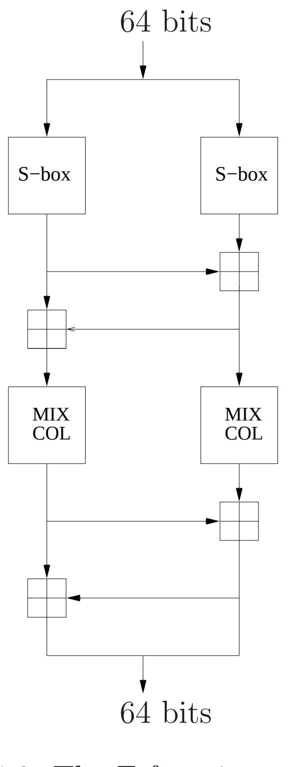


Figure 5.6: The F-function of CRUSH

5.5.2 Working Details of CRUSH

In this section, the operation of CRUSH hash function is explained using the notation introduced in section 5.5.1. In particular, the pseudo code for the IH process of CRUSH is given.

1. **Initialization:** Set the 256-bit initial state of the BSC of CRUSH with the fixed public constants defined in Table 5.3.

2. **Preprocessing:** The standard preprocessing step involves padding the original message M to a multiple of 64-bits, which is *half* the block size of the **B** function of CRUSH. The padding process includes appending a bit 1 and followed by one or more 0 bits until the last before half-block is complete. For short messages, sufficient zero half-blocks are inserted to ensure that the padded message has not less than fifteen half-blocks of 64-bits each at this stage. This padded message is then completed by appending a final half-block of 64 bits of the binary encoded representation of $len(M)$ of the original message M. If $len(M)$ is the length of the total message after padding and $len(H)$ is the desired size of hash output then the depth of an instance of IH is calculated using the formula $D \leq log_2 \frac{len(M)}{len(H)}$, rounded up to the next integer, or D_{min}, whichever is larger.

3. **Iterate Halving:** The message M is processed as a multiple of 64-bit message blocks by the permutation function **B** whose functionality is given Table 5.4. The 128-bit blocks are processed one after the other. At any stage s, the pseudo code for the **B**-function is given in Table 5.5. The next stage process in Table 5.6 indicates using the second half of the output of the current stage as message input for the following stage.

4. **Termination:** After stage D, the last set of half data blocks stored (the most recent 256 bits generated or the last four $D_{i,s}$ half blocks) are used as the final hash output. Note that there is no feed-forward addition.

5.6 Conclusion

The main intention to propose IH class of hash functions is to show a new methodology for hash function design different from iterative hash function structures. The IH class of hash functions are also iterative structures except in a different way compared to Merkle-Damgård hash functions. It is quite clear that the generic attacks that work for Merkle-Damgård hash functions do not apply for IH class of hash functions. The applicability of any other generic attacks that work on this class of hash functions demanding some new security properties is

Table 5.5: Pseudo code for the IH process in CRUSH

```
Input = {$M^1_{i,s}, M^2_{i,s}, x, y, z$}
for ($s = 1$ to $D$)
 {
  for ($i = 1$ to $N$)
  {
   $D^2_{0,s}$ = null
   $x = f1(a, c, d)$
   $y = f2(a, c, d)$
   $z = f3(a, c, d)$
   $D^1_{i,s} = \mathrm{B}(M^1_{i,s}, M^2_{i,s}, x, y, z)$
   $D^2_{i,s} = \mathrm{B}(M^1_{i,s}, M^2_{i,s}, x, y, z)$
   $a = D^1_{i,s}$
   $b = a$
   $c = b$
   $d = c$
   $D^2_{i+1,s} = D^2_{i-1,s}||D^2_{i,s}$
   $N = N/2$
   If $N = 2$ then hash value = $D^2_{i+1,s}$
   else
   Use $D^2_{i+1,s}$ as data input for the next stage
   $s = s + 1$
  }
 }
```

Table 5.6: Pseudo code for the stage process in CRUSH

```
Input = {$D^2_{i+1,s}$, integer $j$}
for($j = 1$ to $N$)
 {
  $M^1_{j,s+1} = D^2_{2j-1,s+1}$
  $M^2_{j,s+1} = D^2_{2j,s+1}$
 }
```

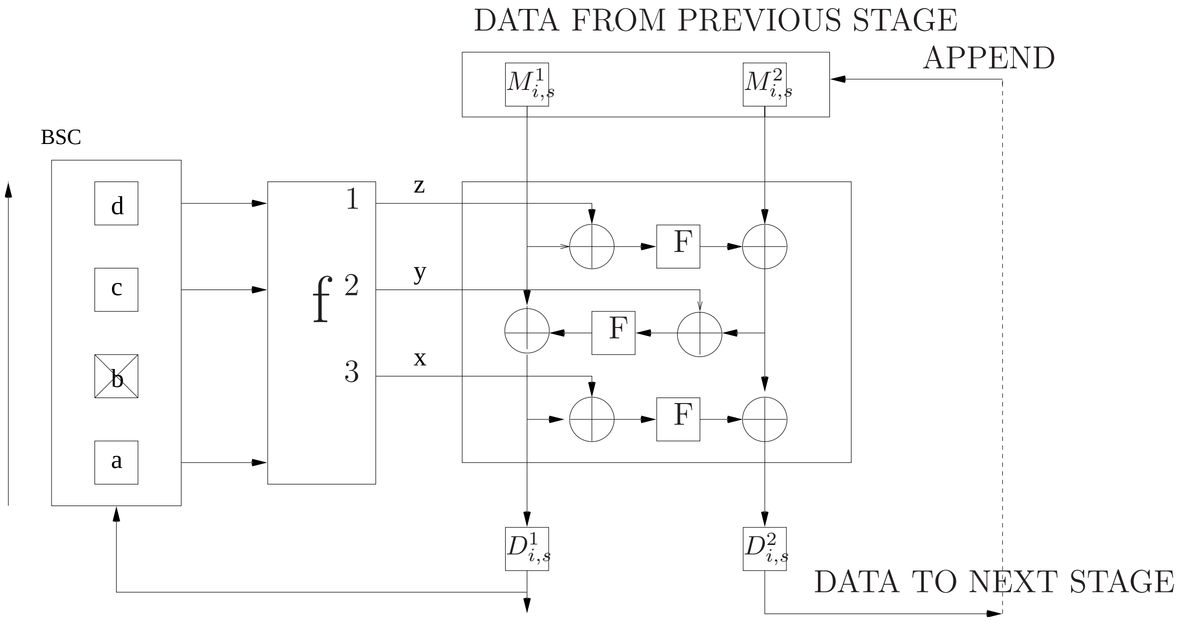


Figure 5.7: Structure of CRUSH

yet to be investigated. The CRUSH algorithm as an example of the IH class of hash functions is proposed. As future work, an indepth analysis of IH class of hash functions using specific B functions such as CRUSH needs to be performed.

Table 5.7: The non-redundant 8*8 S-box for F-function

63	7C	77	DD	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	17
04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
09	83	2C	1A	1B	6E	10	A0	52	3B	D6	B3	29	74	2F	84
53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
D0	EF	AA	FB	43	4D	56	85	45	F9	02	7F	50	3C	9F	A8
51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	15	FF	F3	D2
CD	0C	13	EC	5F	97	44	5A	C4	A7	7E	3D	64	5D	19	73
60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
E7	C8	16	6D	8D	D5	4E	A9	6C	33	F4	EA	65	7A	AE	08
BA	78	25	2E	1C	A6	B4	C6	E8	7B	E3	1F	4B	BD	8B	8A
70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	37

Chapter 6

Improved Cryptanalysis of the CAVE Algorithm

The Cellular Authentication and Voice Encryption Algorithm (CAVE) [277] was proposed in 1992 for authentication and key-derivation for use in the second generation (2G) North American IS-41 mobile phone system. Currently it is used in ANSI-41 wireless networks: Analog Mobile Phone Standard (AMPS)[1], Time Division Multiple Access (TDMA) and Code Division Multiple Access (CDMA One and CDMA 2000) for authentication and key derivation [237, 259]. The CAVE algorithm is discussed in Appendix A [277] of the IS-54 standard approved by the Telecommunications Industry Association (TIA). CAVE is a 128-bit keyed hash function which mixes a given set of input values including the 64-bit secret key. It produces a 128-bit output in such a way that it should be hard to invert the output to recover the secret key. In addition, it should be hard to predict the output without the knowledge of the secret key.

A number of attacks on CAVE were known within the wireless industry [224] for quite some time since its release. Not much was known about the strength of CAVE in the open cryptographic community until Millan [192] independently published a reconstruction attack on the CAVE algorithm in 1998 inverting the 4-round (resp. 8-round) CAVE in about 2^{11} (resp.2^{13}) hashing operations of the algorithm [192]. Millan's attack has demonstrated that it costs about 2^{X+11} (resp.

[1]AMPS is also an abbreviation of Advanced Mobile Phone Service [272].

2^{X+13}) hashing operations of 4-round (resp. 8-round) CAVE to find a single valid preimage where X is the number of bits that contribute to the input redundancy in the CAVE algorithm. This chapter presents a new attack algorithm that can be used to recover all possible valid inputs in about 2^{72} hashing operations of 4-round CAVE algorithm. In addition, this attack is extended to 8-round CAVE in 2^{72} hashing operations of 8-round CAVE algorithm. That is, it takes twice the effort to break 8-round CAVE as it does 4-round CAVE[2] and this contrasts sharply with other primitives that gain a lot more security when the number of rounds is doubled.

The chapter is organised as follows: In Section 6.1, some notation and definitions are introduced that will be used throughout the chapter. In Section 6.2, the working details of CAVE are discussed. In Section 6.3, the practical usage of CAVE is discussed. In Section 6.4, properties of the CAVE algorithm are explained following [192]. In Section 6.5, Millan's reconstruction attack on CAVE is explained in detail. Improved cryptanalysis of CAVE is provided in Section 6.6. The chapter concludes with Section 6.7.

6.1 Notation and Definitions:

In this section, some notation and definitions are introduced following [277] that will be used in the rest of the chapter.

- LSB: Least Significant Bit
- MSB: Most Significant Bit
- $LFSR_A$: The register A of the LFSR for bits 31-24. The bits are represented as $A7, \dots, A0$ or $L_7, \dots, L_0$.
- $LFSR_B$: The register B of the LFSR for bits 23-16. The bits are represented as $B7, \dots, B0$ or $L_{15}, \dots, L_8$.
- $LFSR_C$: The register C of the LFSR for bits 15-8. The bits are represented as $C7, \dots, C0$ or $L_{23}, \dots, L_{16}$.

[2]This factor comes due to the additional four rounds involved in the execution of 8-round CAVE.

- $LFSR_D$: The register D of the LFSR for bits 7-0. The bits are represented as $D7, \ldots, D0$ or $L_{31}, \ldots, L_{24}$.
- LUT: Look-up table.
- $CT\_low[.]$: The low order 4 bits of the 256-byte LUT used in the CAVE algorithm. These low oder bits can be specifically maintained in a separate table called Low CAVE and this is represented as $CT\_low[.]$.
- $CT\_high[.]$: The high order 4 bits of the 256-byte LUT used in the CAVE algorithm. These high oder bits can be specifically maintained in a separate table called High CAVE and this is represented as $CT\_high[.]$.
- *offset_1*: An 8-bit quantity that points to $CT\_low[.]$.
- *offset_2*: An 8-bit quantity that points to $CT\_high[.]$.
- Round: A round is the iterated portion of the CAVE algorithm. A single round is represented with r and the total number of rounds to be executed is represented with R.
- $sreg[r][i]$: This represents the value of register i at the start of a round r. The abbreviation *sreg* stands for start register.
- $ereg[r][i]$: This represents the value of register i at the end of a round r. The abbreviation *ereg* stands for end register.
- A-Key: A 64-bit cryptographic key variable stored in the semi-permanent memory of the mobile station and also known as the home location register (HLR) of the mobile phone. It is entered once from the keypad of the mobile station when the mobile station is first put into service with a particular subscriber and usually remains unchanged unless the operator determines that its value has been compromised.
- SSD: Shared Secret Data. It contains two quantities, SSD-A and SSD-B. A new value of the SSD quantities may be generated in the mobile station when desired by the operator via the SSD update message transaction.
- SSD-A: A 64-bit value in the semi-permanent memory of the mobile station. It is used in the computation of the authentication response.

- SSD-B: A 64-bit value in the semi-permanent memory of the mobile station. It is used in the computation of the voice privacy mask (VPM) and the CMEA-Key.
- RANDSSD: It is a 56-bit random number generated by the HLR.
- $\mathrm{LSB_x}$(RANDSSD): x LSBs of RANDSSD.
- $\mathrm{MSB_x}$(A-key): x MSBs of the A-key.
- $\mathrm{LSB_x}$(A-key): x LSBs of the A-key.
- $\alpha = \mathrm{LSB}_{32}(\mathrm{RANDSSD}) \oplus \mathrm{MSB}_{32}(\text{A-key}) \oplus \mathrm{LSB}_{32}(\text{A-key})$
- CMEA: Cellular Message Encryption Algorithm
- CMEA-Key: Eight distinct 8-bit registers that result from the action of the CAVE algorithm and is used to encrypt messages.
- VPM: Voice Privacy Mask. The name describes two different 260-bit binary data values that are used for voice privacy functions as specified in the wireless system standards.
- GSM: Global System for Mobile communication. GSM is a European digital standard for mobile or cellular telephony.
- TDMA: Time Division Multiple Access. TDMA is a digital wireless telephone transmission technique.
- CDMA: Code Division Multiple Access. CDMA is a digital wireless telephony transmission technique. It is often referred as the IS-95 communications standard.
- BS: Base Station. BS is a radio station in a land mobile radio network. It consists of a transmitter, a receiver and an antenna facility and connects traffic from the fixed networks to the rest of the telecommunications network.
- MSC: Mobile Switching Center. MSC is a device within a wireless telephony network that routes calls to and from base station controllers (BSC) and forwards calls from BSCs to the public telephone system, another MSC, an

Internet Service Provider (ISP), or a private network for connection to the appropriate destination.

- HLR: Home Location Register. The HLR is the main database of permanent subscriber information for a mobile network. It is an integral component of CDMA, TDMA and GSM networks.
- AUTH_SIG: A mobile phone must present a response to a random challenge based on SSD-A (IS-41) or the master key (GSM) before it accesses the network. AUTH_SIG is an 18-bit random authentication signature generated from the CAVE algorithm based on the inputs from the mobile phone number, a broadcast random challenge and SSD-A. AUTH_SIG is verified by the BS allowing the legitimate subscriber to access the network.
- AAV: Authentication Algorithm Version. It is an 8-bit constant equal to 0xC7 used in the CAVE algorithm.
- ESN: Electronic Serial Number. ESN is a unique serial number of a cellular phone that identifies it to the cellular system for the purpose and placing and receiving calls. It is assigned by the phone manufacturer. ESN can be checked electronically to help prevent fraud.
- MIN: Mobile Identification Number. It uniquely identifies a mobile unit within a wireless carrier's network. It often can be dialed from other wireless or wireline networks. MIN can be checked electronically to help prevent fraud.

6.2 The CAVE Algorithm Description

The description of the CAVE algorithm follows from [192,277]. For completeness, a description of the algorithm is provided here, using some of the notation and definitions introduced in Section 6.1 and drawing attention to various aspects that affect security.

The main components of the algorithm are sixteen 8-bit data registers, two 8-bit offsets *offset_1* and *offset_2* and a 32-bit LFSR. The CAVE algorithm hashes 176 bits of fixed length input data to 128 bits of fixed length output which makes it unique compared to hash functions such as MD5 and SHA-1 that can process arbitrary length input messages.

CAVE operates in four or eight rounds as per the requirements of a specific application with each round having 16 register update phases. The round number r in CAVE starts as one less than the desired number of rounds, and is decremented to 0. That is $r = R - 1, R - 2, \ldots, 0$ where R is the total number of rounds to be executed[3]. The LFSR (see Fig 6.1) defines a primitive feedback polynomial whose feedback function at any time t is defined as:

$$L_{t+32} = L_t \oplus L_{t+1} \oplus L_{t+2} \oplus L_{t+22}$$

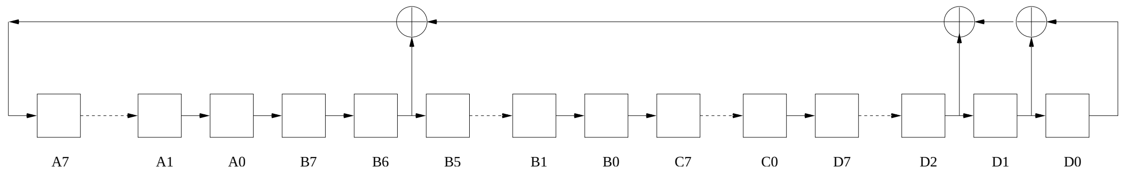


Figure 6.1: CAVE Linear Feedback Shift Register Block Diagram

Application specific input processing determines how various mobile phone data is mapped into the initial contents (see Table 6.2 of Section 6.3). In every case, the LFSR is not allowed to be initialised as all zero.

Each phase of a round of CAVE has four parts: the low nibble segment, the high nibble segment, a single LFSR cycle and the register update. The low and high segments (see Fig 6.2) are identical to each other in operation although they use slightly different data. The segment operations constitutes the main operation of the algorithm.

The current phase of any round r is denoted by i. In the low (resp. high) segment of phase i, $0 \leq i \leq 15$, a 4-bit nibble value, called the low (*temp_low*) (resp. high (*temp_high*)) temp nibble is determined. These two nibble values are concatenated to form a "temp" byte which is finally XORed with $sreg[r][(i + 1) \bmod 16]$ to form $ereg[r][i]$. The offsets *offset_1* and *offset_2* act as pointers into the low and high CAVE tables $CT\_low[\cdot]$ and $CT\_high[\cdot]$. The description of CAVE tables is available in Appendix E .

The steps that take place in the low and high segments of CAVE are expressed

[3]This is in contrast to the more common method of labeling the round number in the increments of one as $r = 1, 2, \ldots, R$.

in the following equations and the segment operation is shown in Fig 6.2

$$\mathit{offset\_1} = \mathit{offset\_1} + (LFSR_A \oplus sreg[i]) \bmod 256 \tag{6.1}$$

$$\mathit{temp\_low} = \mathit{CT\_low}[\mathit{offset\_1}] \tag{6.2}$$

$$\mathit{offset\_2} = \mathit{offset\_2} + (LFSR_B \oplus sreg[i]) \bmod 256 \tag{6.3}$$

$$\mathit{temp\_high} = \mathit{CT\_high}[\mathit{offset\_2}] \tag{6.4}$$

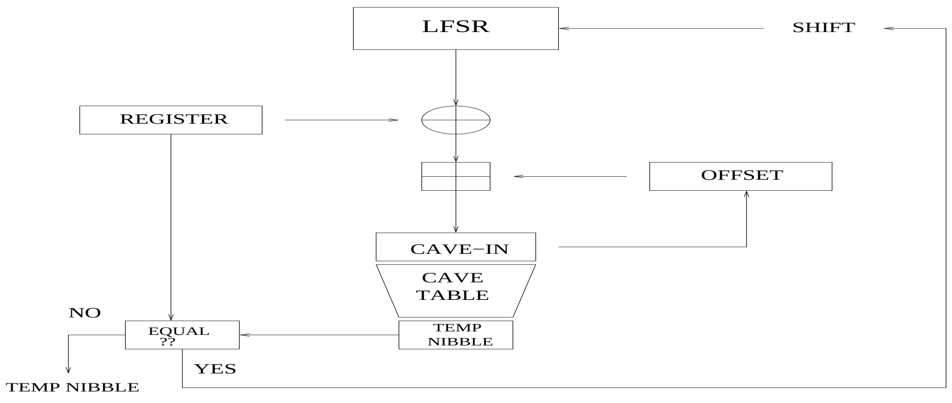


Figure 6.2: Segment operation in CAVE

The LFSR is updated when the nibbles become equal to the corresponding *low*/*high* order bits of $sreg[i]$. When the two nibbles become unequal, the *temp* byte is computed by concatenating the nibbles *temp_low* and *temp_high*. If the compared nibble values become equal, there would be an extra cycle of the LFSR and the above calculation is repeated with the latest LFSR byte and offset values. If the count of these extra cycles reaches thirty-two (for this phase and segment only, it is reset to zero with each new segment), then the byte $LFSR_D$ is incremented modulo 256. This is a very infrequent event (but it does happen (see 6.4)) and cryptanalytical techniques discussed in Section 6.6 ignore this non-linear "jump" due to excessive cycles as the probability of this event happening is 2^{-128}. After the completion of a phase, the LFSR cycles once resulting in a minimum of sixteen LFSR shifts in each round of CAVE. Between rounds, bits in the registers are shuffled by using the low CAVE table to define a byte permutation followed by a 1-bit rotation on the 128-bit register block as a whole. Thus the details of this between round bit permutation (BP) is considered to be always known to the attacker and can easily be reversed during cryptanalysis. The data registers at the end of a round are represented as $ereg[0, 1, \ldots, 15]$. The end round bit permutation is represented as $sreg[r-1][0, 1, \ldots, 15] = BP[ereg[r][0, 1, \ldots, 15]]$, recalling that the round index is being decremented, so that round $r-1$ follows r. Pseudo-code of the 4-round CAVE algorithm is given in Table 6.1.

6.3 Practical Usage of CAVE

The CAVE algorithm is used for five tasks within the IS-41 mobile phone system: A-key verification, shared secret data generation, Challenge-response authentication, the generation of the CMEA key (used for control channel encryption) and Voice Privacy Mask (VPM) generation. See [277] for a detailed explanation on how the CAVE algorithm is used in each of these applications. Each application has data specific to itself that is used in the initialisation of 176 input bits to the CAVE algorithm. The input processing (see Fig 6.4) includes redundancy shrinking the effective input space. The redundant bits are either fixed constants or repeated. For example, the data input to register byte 9 ($sreg[8]$) is always set to the algorithm version number 0xC7 for the applications A-Key verification and SSD generation as shown in Table 6.2. In addition, the offset bytes *offset_1* and *offset_2* are always set to 0x80. Several other bytes are duplicated as shown in Table 6.2 where the LFSR is filled with 32 MSBs of the secret key A-key.

A subset of 128-bit CAVE output register data is taken for use and the remaining bits are discarded. This is quite important for the security of CAVE, since the known attacks [192,224] on CAVE require access to all 128 bits of output and cannot be applied in practice.

The purpose of the CAVE algorithm is to authenticate a legitimate subscriber to the wireless network and protect the network and mobile phones from the cloning fraud [294]. CAVE is intended to deter radio access to the 32-bit Electronic Serial Number (ESN), Mobile Identification Number (MIN) and the 64-bit Authentication key (A-Key) of a mobile phone. CAVE uses ESN, A-Key and a random number (RANDSSD) generated by the Home Location Register (HLR),

Table 6.1: The pseudo-code of the CAVE algorithm

```
Input = $\{sreg[0,\dots,15], offset\_1, offset\_2, LFSR_{A,B,C,D}\}$
for $r = 3$ to $0$ do:
 for $i = 0$ to $15$ do:
  $LFSR\_cycle\_count = 0$
  1. Repeat:
  (a) $offset\_1 = offset\_1 + (LFSR_A \oplus sreg[i]) \bmod 256$
  (b) $temp\_low = CT\_low[offset\_1]$
  (c) if$(temp\_low = sreg[i]$ & $0\text{x}0F)$
  $LFSR\_cycle\_count{+}{+}$
  Until $((temp\_low \neq sreg[i]$ & $0\text{x}0F)$ OR $LFSR\_cycle\_count == 32)$
  $LFSR\_cycle\_count = 0$
  2. Repeat:
  (a)  $offset\_2 = offset\_2 + (LFSR_B \oplus sreg[i]) \bmod 256$
  (b) $temp\_high = CT\_high[offset\_2]$
  (c) if$(temp\_high = sreg[i]$ & $0\text{x}F0)$
  $LFSR\_cycle\_count{+}{+}$
  Until $((temp\_high \neq sreg[i]$ & $0\text{x}F0)$ OR $LFSR\_cycle\_count == 32)$
 3. $temp = temp\_low || temp\_high$
 4. $ereg[r][i] = sreg[r][(i+1) \mod 16] \oplus temp$
 5. Cycle LFSR once.
$sreg[r-1][0,\dots,15] = BP[ereg[r][0,\dots,15]]$
Output = $\{ereg[0][0,\dots,15]\}$
```

Table 6.2: Initial loading of CAVE

CAVE Component	A-key Verification	SSD Generation
LFSR	$\begin{cases} \text{MSB}_{32}(\text{A-key}) \text{ if } \text{MSB}_{32}(\text{A-key}) \neq 0 \\ \text{ESN} \text{ if } \text{MSB}_{32}(\text{A-key}) = 0. \end{cases}$	$\begin{cases} \alpha \text{ if } \alpha \neq 0 \\ \text{LSB}_{32}(\text{RANDSSD}) \text{ if } \alpha = 0. \end{cases}$
$sreg[0,1,\ldots,7]$	A-key	A-key
$sreg[8]$	AAV	AAV
$sreg[9,10,11]$	LSB_{24}(A-key)	MSB_{24}(RANDSSD)
$sreg[12,\ldots,15]$	ESN	ESN
offset_1	0x 80	0x 80
offset_2	0x 80	0x 80

an integral component of a wireless network which permanently stores subscriber information, to generate a 128-bit intermediate key called "shared secret data", SSD-A and SSD-B. The 64-bit SSD-A is used for signature authentication and SSD-B is used for cryptographic key generation.

CAVE uses SSD-A and a broadcast random number (RAND) generated at the Mobile Switch Center (MSC) to produce an 18-bit random authentication signature (AUTH_SIG). The Base station (BS) verifies this signature allowing the legitimate subscriber to access the network. CAVE uses the 64-bit SSD-B data to generate a private long code mask which is used for voice scrambling for data privacy over the network interface of the mobile phone. SSD-B is also used to generate keys for other encryption algorithms like ORYX (32 bits) and CMEA (64 bits)[4]. CAVE is also used to verify the A-Key by truncating the 128-bit hash output to 18 bits and comparing this value with the A-key checksum. The initial loading of CAVE for A-key verification and SSD generation is shown in Table 6.2. Figure 6.3 shows different applications of CAVE and Figure 6.4 shows how CAVE is used in reality in IS-41 and IS-54 wireless communication systems.

6.4 Properties of the CAVE Algorithm

Some security properties of the CAVE algorithm are discussed in [192]. This section discusses some additional properties of CAVE, some of them are used by the improved attack on CAVE in Section 6.6.

[4]ORYX, a stream cipher for data transmissions and CMEA, a block cipher that protects the control channel are broken [283, 284].

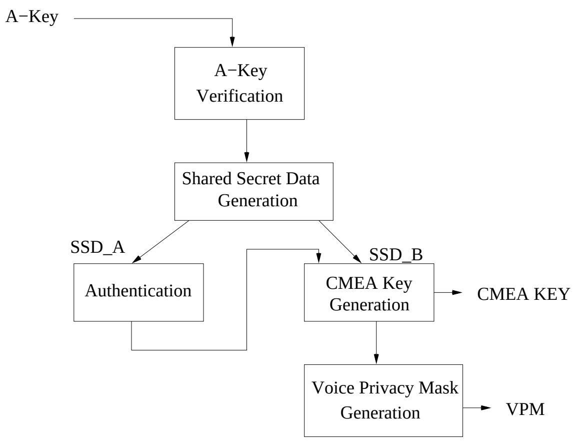


Figure 6.3: Many roles of CAVE

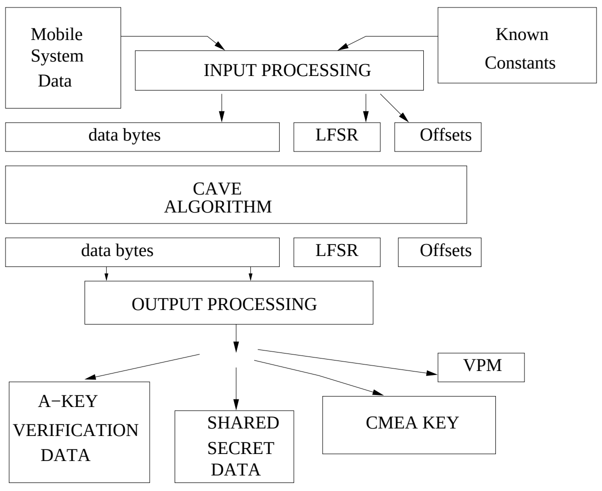


Figure 6.4: Usage of CAVE in IS-41 and IS-54 phone systems

- The bits of $LFSR_A$ ($\{L_0, L_1, \ldots, L_7\}$) before the start of a phase are used again in $LFSR_B$ ($\{L_8, L_9, \ldots, L_{15}\}$) after 8 LFSR cycles as L_7 jumps into the MSB position of $LFSR_B$ for every cycle. The LFSR bytes also do not depend on offsets and registers. So given the set $\{L_0, L_1, \ldots, L_7\}$ at time t, the set $\{L_8, L_9, \ldots, L_{15}\}$ is completely specified after time $t + 8$. These relations are expressed as follows.
  For $\Delta t = 1$, $L_n(t) = L_{n-1}(t-1)$
  For $\Delta t = 4$, $L_n(t) = L_{n-4}(t-4)$
  For $\Delta t = 8$, $L_n(t) = L_{n-8}(t-8)$

  Millan's reconstruction attack [192] on CAVE guesses an LFSR state and so does not need to use this property explicitly. The improved attack on CAVE described in Section 6.6 does make use of this feature in conjunction with the non-randomness inherent in the reverse CAVE table operations used to reconstruct CAVE.

- The number of extra LFSR cycles that occur in a segment during an execution of the CAVE algorithm is usually small. The most common number of extra cycles in a segment is zero, with the probability of larger values decreasing rapidly. It is very rare for more than four or five extra cycles to occur in any round. The algorithm specifies that an addition of 1 mod 256 be applied to one of the LFSR bytes in the event that 32 consecutive extra cycles occur. However, this event is extremely rare[5]. The probability of 32 extra cycles occurring during any particular segment can be estimated by $(1/16)^{32} = 2^{-128}$. During a round of CAVE there are a total of 16 phases in each segment. This makes 32 opportunities for extra cycles in each round of CAVE. The probability that, during 8 round CAVE, a sequence of 32 extra cycles does not occur is given by $(1-2^{-128})^{256} \approx 1$. It is almost certain that the nonlinear LFSR jump will not occur for randomly chosen input data. CAVE uses irregular clocking of the LFSR and this structure has been used in stream ciphers to prevent cryptanalysis. Dealing with extra cycles is a problem for the attacker as it makes the LFSR cycle dependent on data.

- The CAVE tables are constructed as multi permutations, so that holding

[5]One of the test vectors of the CAVE algorithm in the Common Cryptographic Algorithms (CCA) document [238] was chosen so that an extra cycle occurs [239]. Sometimes CAVE algorithm produces 32 extra cycles continuously in a segment and there is evidence of a set of data that has produced it [239].

constant the value of high four bits of the input induces a bijection from the low input bits to the output. This property of the CAVE tables is used as the source for the 16-byte end of round permutation. However, the CAVE tables are not latin squares[6]. The rows of CAVE tables are all permutations but the columns are not. Still, the tables have the *regular* property, this follows from it being a multi-permutation: all possible outputs occur equally frequently given uniformly distributed input.

- Millan's attack [192] only needs to use the CAVE tables forward. The improved attack discussed in Section 6.6 does the reconstruction of CAVE using the following property of the inverse CAVE tables: for any given value of the temp nibble (a CAVE table output), the set of values for the low 4-bits of the input to the CAVE table is not uniformly distributed. Given some input data or a small guess, this property of CAVE assists in determining unknown values with a higher probability than just guessing. This imbalance in the low and high CAVE tables is represented in Tables 6.3 and 6.4. The new attack on CAVE directly uses this property. Table 6.3 shows the frequencies of low order input bits to the low CAVE table giving a particular *temp_low* nibble output.

  Similarly, Table 6.4 shows the frequencies of the high order input bits to the high CAVE table giving a particular *temp_high* nibble output. On average, there are six low order input bits to the low CAVE table, with five being the minimum and nine being the maximum, that do not give a particular *temp_low* nibble output. Similarly, on average, there are six high order inputs to the high CAVE table, with four being the minimum and eight being the maximum not giving a *temp_high* nibble output. These observations indicate a strong non-uniform probability distribution of data in the CAVE tables when they are exploited in the backward direction.

- The way CAVE is used in the authentication application as discussed in Section 6.3 suggests that CAVE must have the security properties of a keyed hash function. For example, the output of CAVE must not leak bits from the first half of the input block or from the input LFSR value. Instead, there is a considerable leakage of information in the inputs to the CAVE table

[6]A latin square is an $n \times n$ table filled with n different symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column.

Table 6.3: Imbalances in the LOW CAVE table

	Frequency of low nibble inputs						
low nibble output	0	1	2	3	4	5	6
0	6	6	3	-	1	-	-
1	9	2	2	2	1	-	-
2	6	6	2	2	-	-	-
3	6	6	2	2	-	-	-
4	5	8	1	2	-	-	-
5	6	7	1	1	1	-	-
6	9	3	2	1	-	-	1
7	6	6	2	2	-	-	-
8	7	3	5	1	-	-	-
9	5	8	1	2	-	-	-
A	5	6	5	-	-	-	-
B	6	6	2	2	-	-	-
C	6	7	2	-	-	1	-
D	6	5	4	1	-	-	-
E	6	7	1	1	1	-	-
F	6	6	3	-	1	-	-

for a given value of the output. This reduction in entropy when working the CAVE tables backwards can interact with the modular addition that is used to calculate CAVE table inputs. This connection along with the strong correlation of LFSR bytes allows the reconstruction attack which works by maintaining lists of all possible values for these nibbles, and using the known relations to exclude the impossible combinations[7]. The new attack on the CAVE algorithm discussed in Section 6.6 uses this information to check the most likely candidate values and relies on less known data.

6.5 Previous Analysis of CAVE

CAVE was designed during the period of slow microprocessors and export restrictions. The 64-bit A-Key is obsolete and Internet stations have attacked 64-bit keys by brute force [183] and it is relatively easier to attack CAVE in this way than RC5 encryption algorithm [256] as the key schedule computation of RC5

[7]This general approach was previously used by Millan *et al.* [194] in the cryptanalysis of a self-synchronous stream cipher with unknown internal structure.

Table 6.4: Imbalances in the HIGH CAVE table

	Frequency of high nibble inputs					
high nibble output	0	1	2	3	4	5
0	4	9	2	1	-	-
1	5	7	3	1	-	-
2	6	6	2	2	-	-
3	7	4	4	-	1	-
4	8	3	3	1	1	-
5	6	5	4	1	-	-
6	7	3	5	1	-	-
7	6	7	1	1	1	-
8	4	9	2	1	-	-
9	5	7	3	1	-	-
A	8	4	1	2	1	-
B	7	6	1	1	-	1
C	5	7	3	1	-	-
D	6	6	2	2	-	-
E	7	4	4	-	1	-
F	5	7	3	1	-	-

requires more computation effort than a single execution of the CAVE algorithm.

The CAVE algorithm as an unkeyed hash function was first broken in 1995 [224]. That attacks shows that given the full 128-bit output, it was possible to find a preimage in 2^{56} hashing operations of the algorithm. Independently, Millan [192] has significantly improved this result by finding a preimage for 4-round (resp. 8-round) CAVE in about 2^{11} (resp. 2^{13}) hashing operations of the respective algorithms. This attack is described below.

6.5.1 Millan's Reconstruction Attack on CAVE

The attacker is assumed to have access to all the 16 output register bytes of the CAVE algorithm and the task is to determine input data that produces the known output. The first step of this forward and backward reconstruction attack is to guess a 32-bit LFSR value and then run the known primitive feedback polynomial for sufficiently many cycles to specify all LFSR bytes at every stage of the CAVE calculation. Generating 200 cycles is enough. Without loss of generality, a cycle number 175 was chosen as the position of the LFSR at the start of the last round.

This gives 25 cycles for each round, which is very probably more than enough. Additional cycles may be calculated as needed.

Given the LFSR values, the attack attempts to discover values for the two offset bytes at the start of the last round allowing "self-consistent" re-construction of the input data registers. In practice, the values of both the offset bytes and the value of $sreg[0]$ at the start of the last round are guessed. Given such a guess, the CAVE algorithm is then worked forward normally for one phase to determine the temp byte $temp[0]$ for the phase $i = 0$. Since the values of $ereg[0, \ldots, 15]$ are all known after reversing the last bit permutation, it is straightforward to calculate $sreg[1] = ereg[0] \oplus temp[0]$. This may be iterated for all phases in the last round, until a value for $temp[15]$ is obtained. Then, the value $ereg[15] \oplus temp[15]$ is calculated and compared with the guess made for $sreg[0]$ back at the start of the last round reconstruction. This is called a "sanity check" condition. If the values are equal, then the attacker's initial guess has led to a self-consistent reconstruction, and the attack may then pass to the last round, and so on.

However, should the sanity check condition fail, the count of failed guesses for that round is incremented, and then a guess is made again. The attack parameter $abort(r)$ can be used to set the maximum number of attempts to reconstruct round r before it is abandoned and the attacker goes back to try an alternative reconstruction of round $r + 1$. One abort parameter is required for each round and one extra to control the maximum number of initial LFSR guesses.

The last round of CAVE is treated differently from other rounds, in that it is reconstructed forward, rather than backwards. This is possible for the last round only, since the final offset values are not already fixed. When the attack proceeds to the second last (and previous) rounds, the final values of the offsets have been fixed, so the attack must work backwards from those values. This process of reverse reconstructing a segment is examined in more detail below.

At the start of the backwards reconstruction of a round $r > 0$, the value for $sreg[15]$ is guessed and the effect of this choice is propagated backwards to obtain self-consistent values for $sreg[14, 13, \ldots, 0]$. Once $sreg[0]$ is available, it is compared to $ereg[15] \oplus temp[15]$. As before, a match means the reconstruction is successful.

When working a segment backwards, the attacker has to choose when to stop adding extra cycles. This choice could be arbitrary or random, but in these cases the distribution of the number of extra cycles would differ to that expected from authentic CAVE data. In a practical attack, this difference would be detectable and might reveal the reconstructed CAVE data as fake. In order to produce CAVE data that has the correct distribution of extra cycles, the attack is conducted in the following manner.

Consider the sequence of temp nibbles that occur during a single segment of an arbitrary phase i of CAVE. Ignoring the possibility that the segment ends after 32 extra cycles, the last value of the temp nibble must be one that does not equate to the corresponding (high or low) nibble of $sreg[i]$, since the only other way a segment ends is when the nibbles differ. Considering that the offset values at the start of the subsequent segment are known, this value is exactly the CAVE table input that generated the last temp nibble. The previous value of the offset byte $\mathit{offset\_1}_{prev}$ for the low segment (resp. for the high segment) can be calculated as

$$\mathit{offset\_1}_{prev} = \mathit{offset\_1} - (LFSR_A \oplus sreg[i]) \bmod 256.$$

Now exactly as in the forward direction, the attacker finds the corresponding previous temp nibble from $CT[\mathit{offset\_1}_{prev}]$. Now, while the temp nibble does equal the corresponding $sreg[i]$ nibble, this process is repeated, producing extra cycles. Finally, when a temp nibble that does not equal the $sreg[i]$ is found, the segment reconstruction stops *without* using the data from the last iteration. This ensures that the number of extra cycles induced by the self-consistent segment data has an identical probability distribution to that of actual CAVE extra cycles. Millan reported results of software experiments that confirmed the attack works and allowed complexity to be directly measured by counting the round reconstruction attempts.

Using this attack procedure, 4-round CAVE can be broken in time equivalent to $1.3 * 2^{10}$ executions of 4-round CAVE and 8-round CAVE can be broken in $1.25 * 2^{12}$ executions of the algorithm. Increasing the number of rounds from 4 to 8 has only increased the workload by about 8 times. This suggests that each extra round of CAVE adds less than 1 bit security, and hence increasing the number of rounds of CAVE is not an effective way to increase its security.

As explained in Section 6.3, the input processing places redundancy in the CAVE initial data. This reconstruction attack does not consider the redundancy placed in the CAVE initial data. The attack would have to be repeated 2^{X} times in order to generate just one input data that has redundancy consistent with the input processing stage of specific CAVE applications where X is the number of redundant bits. The value of X depends on the initial loading of a specific CAVE application. For example, for the A-key verification and SSD generation applications, the values of X are 80 and 88 respectively[8]. Hence, the complexity required to generate a single preimage for a 4-round (resp. 8-round) CAVE that has redundancy consistent with the input processing of these two applications is about 2^{91} and 2^{99} (resp. 2^{93} and 2^{101} for 8-round CAVE) respectively.

6.6 Improved Cryptanalysis of CAVE

In this section, an improved approach to attack CAVE is presented. As in the previous analysis of the CAVE algorithm, the attacker is assumed to have access to all the 16 output register bytes of data. Firstly, a precomputation is used to establish look-up-tables (LUTs) that define the operation of a segment in CAVE. Then, given a 128-bit hash output (the final values of the register bytes), these tables are used to guide a process which maintain lists of *all* data that is self-consistent. These lists are generated across consecutive segments within a phase, then consecutive phases within a round. The resulting data sets after only two phases can specify about half of the unknown LFSR bits. Similarly, the process may be extended across more phases back to the start of the algorithm. Considering that the CAVE algorithm hashes the fixed input of 176 bits down to 128 bits, it is expected that each 128-bit output to have 2^{48} preimages. A 24-bit guess each time leads to an expected list size of 2^{24} elements. To decrease the practical running times, LUTs are precomputed to represent the set of most frequently repeated operations in CAVE. Firstly, the generation of these LUTs is explained below followed by the attack algorithm.

Precomputing the data

Each segment (see Figure 6.2) of CAVE takes 24 input bits; it carries out an exclusive-OR operation on a byte from the input data register and an LFSR byte followed by a mod 256 addition of this result with the respective low/high offset byte. This gives the new offset byte which is the input to the low/high CAVE table giving low/high temp nibble.

[8]This is based on the assumption that $\alpha \neq 0$ for the SSD generation which may be true in many cases; otherwise X = 24.

The LUTs (1 and 2) can be constructed using exhaustive computation on these essential operations in the segments of CAVE. The offset, the LFSR byte and the input data register add up to 24 bits giving 2^{24} different possible values. The computation on these 2^{24} possible input values results in an offset byte in each segment which is input to the CAVE table. The output of the CAVE table is a 4-bit *candidate temp_low* nibble or *temp_high* nibble and an "extra cycle" counter which acts as a flag. A flag value of one indicates the values of input to the CAVE table, for which the low/high *temp* nibble is equal to the low/high order bits of $sreg[i]$. The *temp* nibble value is a *candidate* since, if an extra cycle is indicated, then the LFSR is cycled once, thus changing the respective LFSR byte used in the above calculation by losing the LSB bit, gaining the MSB bit and shifting the other 7 bits by one bit. When the extra cycle count is zero, then the current *temp* nibble is used. Thus these two LUTs consist of 24-bit entries, an output offset byte, a *temp* nibble and an extra cycle counter.

The overall attack algorithm is described as follows:
Pre-computation step: Calculate the high and low LUTs.

- **Init**: Repeat, for all 2^{24} values of $sreg[15]$ and the pair of offset bytes:
- **Step 1**: Use the LUTs to find lists of valid inputs to both segments in two consecutive phases.
- **Step 2**: For each phase, combine the two segment data lists into a list of valid data for that phase.
- **Step 3**: Combine the adjacent phase lists into a single list for the pair of phases.

Final: Combine the remaining lists, filtering for consistency, to determine the list of all possible valid inputs.
Each of these operations are explained in detail below.
Init This step involves choosing a 24-bit value that makes up the two offsets and the start register byte. For a 4-Round CAVE, using the known 128-bit hash value, the values of end registers $ereg[0, 1, 2 \ldots 15]$ of Round 0 are found using the reverse round byte permutation on the 128-bit hash value followed by a left circular shift of the registers. By guessing the value of the input data register of

phase 15 (sreg[15]) of Round 0 to CAVE, the $temp_{14}$ output byte of phase 14 can be determined using the following expression:

$$ereg[14] \oplus sreg[15] = temp_{14}$$

The value $temp_{14}$ is the concatenation of $low/high\ temp_{14}$ nibble outputs of low and high CAVE tables. These nibbles are represented as $temp\_low_{14}$ and $temp\_high_{14}$ respectively. Since these nibbles are the final outputs of the low and high segments of phase 14 of Round 0 of the CAVE algorithm, the offsets $\mathit{offset\_1}_{14}$ and $\mathit{offset\_2}_{14}$ that have given these nibbles could not have produced extra cycles.

Therefore,

$$temp\_low_{14} \neq sreg[14]\ \&\ 0\text{x}0F$$

and

$$temp\_high_{14} \neq sreg[14]\ \&\ 0\text{x}F0$$

There are 16 different possible values of $\mathit{offset\_1}_{14}$ and $\mathit{offset\_2}_{14}$ that can give these nibbles because of the *row permutations* of the CAVE tables. The attack involves testing every value of these offset bytes.

Step 1: The first step is the evaluation of different possible 24-bit values accessed from LUTs 1 and 2 for each value of the data choice in the segments satisfying Equation (6.1). This test is performed simultaneously on both the low and high segments. The considered 24-bit values in the low and high segments should have the same byte of $sreg[14]$. This *important* condition on the selection of the 24-bit values results in a shorter list of 24-bit vectors used in Equation (6.1) to acquire the offsets $\mathit{offset\_1}_{14}$ and $\mathit{offset\_2}_{14}$. This equality condition on the register bytes results in a list of 2^{24} possible values for solving Equation (6.1)) which is a 50% reduction from the original set of $2 * 2^{24}$ values (considering two segments of CAVE). The previous offset bytes $\mathit{offset\_1}_{14prev}$ and $\mathit{offset\_2}_{14prev}$ that resulted in the chosen offsets $\mathit{offset\_1}_{14}$ and $\mathit{offset\_2}_{14}$ can be extracted from the assumed 24-bit values of low and high CAVE segments.

Step 2: The previous offsets are used to calculate the corresponding temp nibbles as follows:

$CT\_low[offset\_1_{14prev}] = temp\_low_{prev}$
$CT\_high[offset\_1_{14prev}] = temp\_high_{prev}$

The *temp_prev* is calculated by concatenating these temp nibbles. The validity of the guess on the 24-bit data obtained from the LUTs is checked using the "sanity check" equation:

$$temp\_13 = ereg[13] \oplus sreg[14].$$

This sanity check equation, in general is represented as:

$$temp\_i = ereg[i] \oplus sreg[i+1].$$

During this step, the lists are reduced to about 2^{16} values.

Step 3: The above steps are repeated for the last two adjacent phases of the last round to get the reduced lists of each phase. The lists are checked for compatibility using the property of correlation between LFSR bytes discussed in Section 6.4 and also the use of final offsets of one phase as the starting offset values in the following phase. The backward reconstruction of the four segments of two phases is enough to establish the values of about *half* the bits of the LFSR and the two offsets used in those particular phases! The attack then proceeds backwards on other phases with much more known information which reduces the complexity for these subsequent iterations. In this process the lists will be reduced until a set of valid data used at the start of Round 3 is obtained.

To assess the complexity of this attack in a way that can be compared with the first attack, the theoretical complexity of each step using units equivalent to (or less than) evaluating a complete phase of CAVE is calculated. Since list processing with pre-computed LUTs is less complex than executing a phase of CAVE, an upper bound for the complexity of the attack is developed using the phase-equivalent complexity as the fundamental unit. Step 1 has complexity less than 2^{24} of these units, for each of the 2 phases in each of 2 adjacent segments making a total effort of 2^{26}. Step 2 requires around 2^{25} effort for each of the 2 phases, so that it takes 2^{26} effort as well, for a running total of 2^{27} phase-equivalent units. For the first time only, Step 3 must consider all pairings from two segment lists each of size 2^{16} elements, for a total complexity of 2^{32} operations. This dominates the complexity from the first two steps. So the complexity of finding all data consistent across two consecutive phases can be safely upper bounded as being clearly less than 2^{33} phase-units. Combining lists of size of 2^{24} costs 2^{48} effort. This is used as an upper bound on the complexity for each phase in the attack (64 phases in CAVE-4). As all these calculations must be performed 2^{24} times (with different initial choices for the pair of offset bytes and the start registers in the Init stage), it is expected that the effort to find all valid data for 4-Round CAVE will be less than $2^{48} * 2^6 * 2^{24} = 2^{78}$ phase-units (which is about 2^{72} calculations of 4-Round CAVE which has 64 phases). This compares favorably with the 2^{91} effort (considering A-key verification) required by the previous attack [192] on the algorithm. The effort to extend this attack to 8-Round CAVE is minor: another $2^{48} * 2^6$ effort for each of the 2^{24} trials is an extra 2^{78} phase-units or double the effort of what was needed to break 4-Round CAVE. To compare, Millan's attack [192] requires eight times the effort. Table 6.5 summarizes the differences between two attacks.

Table 6.5: Comparison of two attacks

Property	Millan's attack	Improved attack
Complexity	2^{91} of CAVE-4 executions	2^{72} of CAVE-4 executions
Complexity	2^{93} of CAVE-8 executions	2^{72} of CAVE-8 executions
Pre-images	Finds just one valid value	Finds all valid values
Information exploited	Less	More
Number of guesses	More	Less
Real Threat	Not possible	Not possible

6.7 Conclusion

In this chapter, a new attack on the CAVE algorithm that can recover all the valid preimages for a given hash value is presented. From the known attacks on CAVE (including the new attack presented in this chapter) as an un-keyed hash function, it is concluded that the CAVE hash function does not satisfy preimage resistance [192, 224] and second preimage resistance [192] security properties of a hash function. So far, there are no known attacks in the literature on finding collisions to CAVE with a complexity less than that of birthday attack on CAVE. It is a known fact that collision resistance of a hash function implies its second preimage resistance [186, p.329]. Since CAVE is not second preimage resistant [192] it can be concluded that it fails to be collision resistant as well.

While the previous attacks [192,224] on the CAVE algorithm do not allow any easy fraud, they do cast serious doubt over the long term security of all CAVE applications. The improved cryptanalysis discussed in the chapter may threaten the security of real CAVE implementations as it was shown that it can recover all the valid preimages of a given hash value.

With regard to the potential for attacking the real system, it may been seen that the VPM which is generated by several successive runs of 4-round CAVE, has 520 bits. This is greater than the 176 bits entropy maximum for any output of the CAVE algorithm, so there must be a considerable redundancy in the VPM. Knowing the VPM (which is easily obtained by frequency analysis of intercepted encrypted speech) should therefore (at least in principle) provide sufficient information to fix *uniquely* the CAVE input that generated it. Thus it might be possible with further improvements, somehow, to break the whole system, recover the shared secret data SSD_A and SSD_B, and so on to recover the master A-key.

As another case, authenticating a legitimate subscriber is the main application of CAVE (Section 6.3). If the different input values that hash to a given digest are found, it is possible to illegally program ESN and MIN into the mobile phone thereby providing a fraudulent customer with access to the wireless network. When the authentication fails, subscriber calls to the network would not be protected even by voice encryption.

Since the only time that all 128 bits of the hash output is available to any attacker is during the SSD calculation, further improvements in the complexity of the improved attack might aid in breaking the security of the A-Key if a roaming partner chooses to derive it from the SSD that was given. In addition, it is not necessary for the roaming partner to know the random challenge used when the SSD was created to perform this attack or for the actual mobile to be present on the network [260]. However, with the current attack complexity, it is infeasible to use this attack to break the security of the A-key.

The decision to replace CAVE with SHA-1 [211] as the preferred cryptographic primitive for 3G algorithms was made in 2000 [242]. This was also noted in [259] that the future functionality of CAVE will be replaced with SHA-1 in IS-41 systems. The new 3G systems (3GPP and 3GPP2) use SHA-1. In the light of collision attacks on SHA-1 [287, 290] it is strongly recommended that it should be replaced with stronger hash functions.

Chapter 7

An Update on MACs based on Hash Functions

A message authentication code (MAC) is a cryptographic algorithm used to validate the integrity and authenticity of information communicated over an insecure channel. In a two party scenario, the communicating parties Alice and Bob share a secret key and use the same MAC algorithm to verify the integrity and authenticity of the information communicated between them. Whenever Alice transmits a message to Bob, she computes an authentication tag for a message using the MAC function with the secret key and message as inputs. She then transmits the tag attached to the message to Bob. Upon the receipt of the message and its tag, Bob computes the tag for the received message using the same MAC function and the secret key used by Alice and verifies whether the computed tag equals the tag received from Alice. Bob confirms that the information has not been altered on its way from Alice only when the value of the received tag matches the value of the computed tag. In this scenario, the security requirement is to prevent any adversary from forging a new message-tag pair that would be accepted by Bob believing that this pair has originated from Alice when in fact it has not. Obviously, an adversary who finds the secret key will easily forge messages of her choice by computing tags of any message.

One of the important applications of cryptographic hash functions is their use in the construction of efficient MAC schemes. Iterated hash functions such as MD5 and SHA-1 are used with minor or no modifications in constructing MAC schemes due to their software efficiency and free availability. Constructing MACs using hash functions is a complicated task as hash functions are not naturally keyed primitives. Analysis of MAC schemes based on dedicated hash functions [13, 228, 234–236, 278] shows that attention must be paid in using the secret key and the hash function while designing a MAC based on the hash function. Attacks on MACs based on hash functions depend on how and where one uses the key and the hash function in the MAC scheme. The Section 7.1 of this chapter provides a survey on the evolution of MACs based on hash functions indicating how difficult it is to construct a MAC scheme from a hash function.

The first formal analysis of MACs based on hash functions was given by Bellare, Canetti and Krawczyk for their MAC proposals nested MAC (NMAC) and hash based MAC (HMAC) [13]. HMAC, which uses hash function as a black box, is a practical variant of the formally analyzed NMAC function which calls the compression function of the hash function as a black box. HMAC is standardized by the bodies NIST FIPS (FIPS PUB 198) [82], IETF (RFC 2104) [158] and ANSI X9.71 [6]. HMAC implementations include SSL, SSH, IPSEC and TLS. This chapter analyses the NMAC function using weaker assumptions on the hash function used in designing NMAC than the assumptions used in the security proof of NMAC [13] and shows that by reducing assumptions on the hash functions, one cannot guarantee a secure MAC function from NMAC.

By revisiting the proof of security of NMAC, it is observed that the security proof and the definitions used in the proof are independent of the padding technique employed for the hash function in NMAC. It is shown why and how the proof of security of NMAC is independent of the padding of the hash function by proposing a variant to the NMAC function called NMAC-1. The NMAC-1 function still observes the design principle of NMAC which is to call the compression function of the hash function as a black box. A formal security analysis for NMAC-1 is provided. In addition, the analysis and performance aspects of the NMAC-1 function are compared with other efficient MACs based on hash functions proposed in the literature. Another variant of NMAC called modified-NMAC (M-NMAC) is proposed in this chapter which has some advantages over NMAC from the perspective of key-recovery attacks.

The HMAC algorithm is often used as a pseudorandom function (PRF) rather than just as a MAC. For example, IPSec's Key Exchange (IKE) protocol [102], TLS [63] and NIST's key establishment draft standard [215] use HMAC as a PRF to derive secret keys. Until recently[1], no explicit analysis of NMAC and HMAC functions as PRFs appeared in the literature though it seems that it is straight forward to use the proof techniques from [13] to prove the pseudorandomness of these functions. This chapter fills this gap by giving the formal analysis of NMAC as a pseudorandom function which applies to HMAC as well.

The design and analysis aspects of MAC schemes based on hash functions studied in Section 7.1 opens a question on the possibility of designing a provably secure MAC scheme based on hash function using a single secret key. The known schemes of MACs based on hash functions with just one secret key are insecure and hence any problem on the design and analysis of MAC schemes using hash functions and one secret key is quite challenging. The Section 7.6 works on this problem and provides a possible solution.

The chapter is organised as follows: Section 7.1 provides the evolution of MACs based on hash functions. Section 7.2 provides the analysis of NMAC based on weaker assumptions on the underlying hash functions than stated in the original proof of security of NMAC in [13]. Section 7.3 proposes an efficient variant for NMAC and provides its formal analysis. Section 7.4 proposes M-NMAC and provides its analysis. Section 7.5 provides the proof of security of NMAC and HMAC functions as PRFs. Section 7.6 proposes a MAC scheme based on hash function with just one secret key and analyses the scheme. Finally concluding remarks to the chapter are given in Section 7.7.

7.1 A Review of MACs based on Hash Functions

A MAC algorithm, represented as **MAC**, takes a secret key K of length k and an arbitrary length message x as inputs and returns the authentication tag defined as $\mathbf{MAC}(K, x) = \mathbf{MAC}_K(x)$. Given a MAC algorithm **MAC** and the inputs x and K, the computation of tag $\mathbf{MAC}_K(x) = \tau$ of fixed size n must be easy. Following generic attacks apply to any MAC function.

- **MAC forgery:** Given a MAC function **MAC**, it must be computationally

[1]Very recently, Bellare [12] has shown the pseudorandomness of NMAC and HMAC functions based on the pseudorandomness of the compression function.

infeasible for the adversary to determine a message-tag pair (x, τ) such that $\mathbf{MAC}_K(x) = \tau$ without the knowledge of the secret key K [228]. For an ideal MAC function, the complexity of the attack is $O(2^{\min(n,k)})$. Otherwise the **MAC** function is said to be existentially forged [234].

The adversary can use any of the following attack techniques to forge the MAC function $\mathbf{MAC}_K$.

1. **Known-message attack**: In this attack, the adversary looks at a sequence of messages $x_1, x_2, \ldots, x_n$ and the corresponding authentication tags $\tau_1, \tau_2, \ldots, \tau_n$ communicated between the legitimate parties in a communication channel. The adversary might observe these messages by intercepting the communication channel in a manner uninfluenced by the communication parties. The MAC scheme is then broken by finding a new un-seen message $x \neq x_i$ and its corresponding tag $\mathbf{MAC}_K(x) = \tau$.
2. **Chosen-message attack**: In this attack, the adversary chooses a sequence of messages $x_1, x_2, \ldots, x_n$ and obtain the corresponding tags $\tau_1, \tau_2, \ldots, \tau_n$ from the party possessing the secret key K to the **MAC** function. Chosen messages are also referred to as "queries" and the corresponding tags as "answers".
3. **Adaptive-chosen message attack**: This is a chosen message attack except that adversary chooses messages as a function of the previously seen messages and their tags.

If the adversary forges a MAC scheme with a message of her choice as in the above three attacks, then that forgery is called *selective forgery* [234]. The adversary, may somehow, possibly by interacting with the sender or receiver of messages, determines the validity of the forged (x, τ) pairs. It is important to note that, in general, the adversary cannot verify the forged pairs even with known message-MAC pairs without interacting with the sender or receiver.

- **Key recovery:** Using a single known message-tag pair, an attacker finds the correct key K used in computing the authentication tag. For an ideal MAC, the complexity of the key recovery attack must be the same as an exhaustive key search attack over the entire key space which is $O(2^k)$. It requires $\lceil k/n \rceil$ message-tag pairs to verify this attack. The key recovery attack allows selective forgery of the MAC function.

- **Collision attack:** In this attack, the attacker tries to find two distinct messages x and x' such that $\mathrm{MAC}_K(x) = \mathrm{MAC}_K(x')$. For an ideal MAC function the complexity of finding a collision is $O(2^{\min(k,n/2)})$ [104]. Collisions are either internal or external [234, 235].

  1. An internal collision for the MAC function $\mathbf{MAC}_K$ is defined as $\mathbf{MAC}_K(x) = \mathbf{MAC}_K(x')$ and $\mathbf{MAC}_K(x||p) = \mathbf{MAC}_K(x'||p)$ where p is any single block (message or key).
  2. An external collision for the MAC function $\mathbf{MAC}_K$ is defined as $\mathbf{MAC}_K(x) \neq \mathbf{MAC}_K(x')$ and $\mathbf{MAC}_K(x||p) = \mathbf{MAC}_K(x'||p)$ where p is any single block (message or key).

- **First Preimage attack:** In this attack, the attacker is provided with an authentication tag τ and she must find a message x such that $\mathbf{MAC}_K(x) = \tau$. For an ideal MAC scheme, the complexity of this attack is $O(2^{\min(n,k)})$ [104].

- **Second preimage attack:** In this attack, the attacker is provided with a message x and she must find a new message $x' \neq x$ such that $\mathbf{MAC}_K(x) = \mathbf{MAC}_K(x')$. For an ideal MAC function, the complexity of this attack is $O(2^{\min(n,k)})$ [104].

The design and analysis of the three earlier MAC function proposals based on the Merkle-Damgård iterated hash function H is given below:

- **Secret prefix method:**

  In this method, the secret key K is prepended to the message x and the authentication tag is computed using the equation $\mathrm{MAC}_K(x) = H(K||x)$ [234, 278]. This scheme is weak against the straight-forward length extension attacks as one can use the authentication tag $H(K||x)$ to compute the MAC value of a new message $x||x'$. This is done by using $H(K||x)$ as the initial chaining value to process the new message x' without the knowledge of the secret key K. The iterative structure of H allows length extension attacks to work. Moreover, any type of padding scheme employed for x initially does not prevent extension attacks as an attacker can cleverly choose x' related to the length of x and its original padding [11].

This forgery attack on the secret prefix method may be prevented by truncating the output of the MAC function and using only truncated output as the authentication tag [234]. It is a well known practice to use only part of the output bits of the MAC function as the authentication tag [5, 158, 190, 277]. In some cases, truncation of the output has its disadvantages too when the size of the tag is short enough for the attacker to predict its value. It is recommended that the output length n of the tag be not less than half the length of the hash function output to match the bound due to the birthday attack. In 1997, this value must not be less than 80 bits which was a suitable lower bound then on the number of bits that need to be predicted by the attacker [158]. Following the recommended security level on the secret key guessing for the block ciphers by the AES process and stream ciphers used in software applications by the ECRYPT process [75], now-a-days this value must not be less than 128 bits.

- **Secret suffix method:**

  In this method, the secret key K is appended to the message x and the authentication tag is computed using the equation $\text{MAC}_K(x) = H(x||K)$ [234, 278]. An off-line collision attack on the hash function H, by any means, results in two messages x and x' such that $x \neq x'$ and $H(x) = H(x')$. For an ideal n-bit hash function H, finding collisions requires at least $2^{n/2}$ off-line computations of H according to the birthday attack. Once a collision for H is found, the adversary queries the MAC_K function with the message x and obtains the tag $\text{MAC}_K(x) = H(x||K)$. $H(x||K)$ is the tag for the message x' without the knowledge of the secret key K.

  Once the adversary finds an off-line collision for H, the adversary can also exploit the length extension weakness of the iterative structure of the hash function by appending a new message y of attacker's choice to x and gets the chosen message $x||y$. Then requests the MAC_K function for the tag of the message $x||y$ and obtains the tag $H(x||y||K)$. Using the tag $H(x||y||K)$, the attacker performs the selective forgery for the message $x'||y||K$ to obtain $H(x'||y||K)$. Again, the attacker does not have to know the secret key K to carry this attack as it is the iterative structure of the MAC function that results in the attack.

This MAC scheme can also be forged by finding internal collisions for the MAC scheme. An internal collision is defined as $H(x||K) = H(x'||K)$ and $H(x) = H(x')$ where $x \neq x'$ and $|x| = |x'|$. It should be noted that an internal collision for the iterated MAC function automatically allows a verifiable MAC forgery, through a chosen-message attack requiring a single chosen message [234, 235].

- **Envelope method:**

  The envelope method [278] combines the prefix and suffix methods to authenticate messages of arbitrary length. In this method, one secret key K_1 is prepended to the message x and the other secret key K_2 is appended to the message x as given by the equation $\text{MAC}_K(x) = H(K_1||x||K_2)$. A variant of this scheme based on the MD5 hash function and a single secret key is specified in the standard RFC 1828 [189] and is defined as $\text{MAC}_K(x) = H(\overline{K}||x||K)$ where $\overline{K} = K||\text{pad}$ denotes the completion of K to the block size by appending some padding bits "pad" to K.

  Divide and conquer key recovery attack:

  The divide and conquer exhaustive search key recovery attack [13, 234–236] on the envelope MAC scheme recovers both the keys K_1 and K_2 in a time around $2^{|K_1|} + 2^{|K_2|}$ where $|K_1| = n$. The attack first recovers the secret key K_1 and then the secret key K_2. This attack requires about $2^{(n+1)/2}$ known message-MAC pairs of equal length to find a collision to the MAC scheme and with significant probability it can be assumed that it is an internal collision. An internal collision occurs before the block containing the key K_2 is processed. The attacker then performs an exhaustive key search to recover the key K_1 with an effort of about $2^{|K_1|}$ which results in a small set of possible keys for K_1 and determines the correct key K_1 by performing a chosen message attack using a few chosen messages. The recovery of the secret key K_1 reduces the security of the envelope scheme to the secret suffix method against the forgery attack. The attacker then finds the key K_2 exhaustively with the effort $2^{|K_2|}$. For reasonable key sizes such as keys of 128 bits each, this attack is impractical.

  Slice by slice trail key recovery attack:

Preneel and van Oorschot [235] have improved the divide and conquer key recovery attack by exploiting the padding procedure of the hash function used in the envelope MAC scheme. This attack is also called slice by slice trail key recovery in the envelope method [236]. They have demonstrated the attack on the variant of the envelope MAC scheme based on the MD5 hash function disclosed in the RFC 1828 by Metzger and Simpson [189] and in [129] by Kaliski and Robshaw where $K_1 = K_2$ and $|K_1| = n$. This attack exploits the padding procedure in the hash functions such as MD5 used in the envelope scheme and shows that the padding part in the hash function must not be used to hide the secret keys. In addition, the attack uses the fact that for an iterated MAC based on the Merkle-Damgård hash function with n-bit tag and chaining state, an internal collision can be found using $\sqrt{2/(s+1)}.2^{n/2}$ known text-tag pairs where $s \geq 0$ is the number of trail blocks in the known texts with the same substring.

The attack relies on the trail key K_2 being split across the blocks and able to be divided into small pieces. For MD4 family of hash functions, if b is the block length of the compression function, then any message y is padded with $1||\alpha$ where α represents 0 bits that are padded until the padded message $y||1||\alpha$ is q bits short of a full block b. The padded message $y||1||\alpha$ is appended with β, which represents a q-bit binary encoded representation of the length of the original message y. For hash functions MD5 and SHA-1, $q = 64$. When this padding scheme is incorporated for the hash function used in the envelope MAC scheme, the trail key K_2 will occupy either the last block or the second last block or split across the last two blocks depending on the length of the message and is represented in Figure 7.1

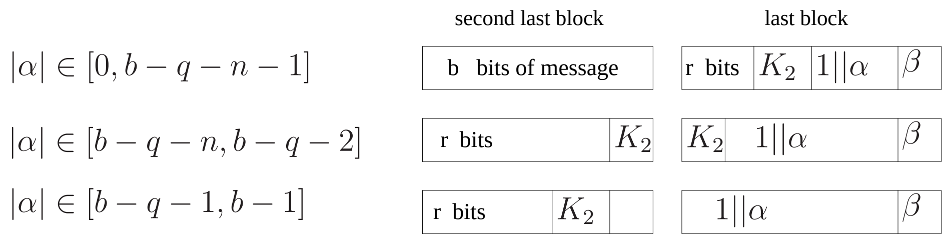


Figure 7.1: The slice by slice trail key recovery attack

An internal collision for this MAC scheme is defined as a collision of internal chaining values just before the block containing the trail key K_2. This collision is identified only through a collision for the MAC and this poses a requirement on the attack that the lengths of the colliding messages are equal and the last r bits of messages are the same. The last r bits of the message occupy either the last block or the second last block as shown in Figure 7.1.

Let $|K_1| = |K_2| = 160$ bits and length of the message x to be authenticated using the envelope MAC scheme be 448 bits. Assume that SHA-1 is the hash function used. For SHA-1, $b = 512$ and $q = 64$. The key K_1 is padded so that the padded key $\overline{K_1}$ is filled in the first block. The second block contains 448 message bits of x and 64 key bits of the trail key K_2. The remaining 96 bits of the key K_2 are filled in with the last block followed by padded bits $1||\alpha$ with $|\alpha| = 351$ and the binary code β representing the length of $\overline{K_1}||x||K_2$ which is 512+448+160 bits.

The attack assumes that the tags of about $2^{80.5}$ messages x are known in which case an internal collision is expected after the second last block and an external collision after the last block. Let (x, x') be a pair of internal colliding messages and now the attacker constructs 2^{64} messages of the form $(x||K_2^i||z, x'||K_2^i||z)$ where z is an arbitrary block and $i \in [0, 2^{64} - 1]$ and queries the MAC oracle for 2^{65} corresponding tags. The tags for messages x and x' match for the correct partial key K_2^i and with probability of about $1 - 1/2^{96}$ no other tags would match. This reveals the first 64 bits of the key K_2. This process is repeated for messages of length 416 bits which reveals the next 32 bits of the key K_2. The remaining 64 bits of the key K_2 are found exhaustively. Hence, the total complexity of this attack to recover the key K_2 is estimated in two phases. In the first phase, the attack requires $2^{80.5}$ known texts for a collision followed by 2^{65} chosen texts to recover the first 64 bits of key. In the second phase, it requires again $2^{80.5}$ known texts for a collision followed by 2^{33} chosen texts to recover the next 32 bits and 2^{64} to find the last 64 bits exhaustively. Once the trail key K_2 is found, the security of the envelope MAC scheme reduces to the security of the secret prefix MAC scheme against forgery attacks.

Obviously, the recovery of the secret key K_2 is practical even for keys of size 128 bits unlike in the former divide and conquer key-recovery attack. If the trail key K_2 of size 128 bits is split across the last two blocks with 64 bits in each block then it would cost about $2^{64.5}$ known texts and 2^{66} chosen texts to recover the secret key K_2 [235].

- **MDx-MAC:**

  The above analysis of MACs based on hash functions shows that designing a MAC function using a hash function is a very tricky business. One has to pay attention in using the hash function and the secret key when converting the hash function into a MAC scheme. Moreover, the attacks on these MAC schemes except the secret-suffix method are independent of the weaknesses in the hash function. The key-recovery and forgery attacks on the envelope MAC scheme would become impractical if large key sizes are chosen for the two keys.

  With this observation in mind, Preneel and van Oorschot [234] proposed a variant to the envelope MAC scheme called MDx-MAC based on hash functions following the iterative principle of Merkle-Damgård construction. This MAC scheme uses three keys, the first key replaces the initial state of the hash function, the second key exclusive-ored with some constants is appended to the message and a third key influences the internal rounds of the compression function. These three keys are derived from a single master key. Unlike the MAC schemes described above, MDx-MAC does not use a hash function as a black box and requires more changes to the hash functions from the MDx family. In addition, no formal security analysis was provided for this construction.

The first formal security analysis for MACs based on hash functions was given by Bellare, Canetti and Krawczyk [13] for the NMAC and HMAC functions. These functions along with the stated security analysis are given in Section 7.1.1.

7.1.1 NMAC and HMAC Functions

The NMAC algorithm including its security proof was published in [13]. The design goal of the NMAC function is to use the compression function of the hash function "as is" (as a black box).
The NMAC algorithm is defined as follows:

If K_1 and K_2 are two independent and random keys to the hash function H iterated over the compression function f, then the MAC function NMAC on an arbitrary size message x is given by

$$\mathrm{NMAC}_K(x) = H_{K_1}(H_{K_2}(x)). \tag{7.1}$$

The message x to be processed using NMAC is split into blocks $x_1, x_2, \ldots, x_n$ of equal length according to the block length of the compression function f. If the concrete realisation of NMAC uses the iterated hash function H for the inner and outer functions, then the key K_2 would be the IV for the inner keyed iterated hash function and K_1 would be the IV for the outer keyed iterated hash function, which is expected to invoke the compression function f only once. Here the lengths of both the keys is the same which is equal to the length of the IV of the hash function H. For example, if H is SHA-1 then $|K_1| = |K_2| = 160$. Since the two keys K_1 and K_2 are the IVs for the inner and outer functions respectively, it is clear that NMAC calls the compression function f of the hash function H as a black box as shown in Figure 7.2.

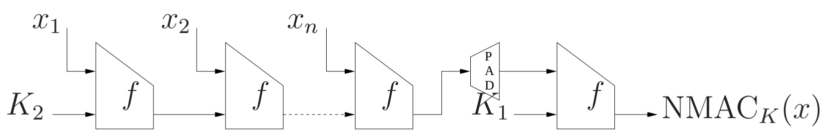


Figure 7.2: The NMAC Construction

Since only one round of the outer function iteration is required, the compression function of the outer hash function is invoked only once. Therefore, the outer function can be renamed f and the above NMAC equation can be written as

$$\mathrm{NMAC}_K(x) = f_{K_1}(H_{K_2}(x)). \tag{7.2}$$

where f_{K_1} is the keyed compression function. In the above NMAC equation, a standard padding technique [120] is defined for the hash function H such that the last block x_n contains the binary encoded representation of the length of the message. The output of the function $H_{K_2}(x)$ is also padded using the same standard padding operation as for the inner function, which is denoted by the PAD function in Figure 7.2. The PAD function is defined as $\mathrm{PAD}(H_{K_2}(x)) = H_{K_2}(x)||1||00\ldots0||\beta$ where β denotes the binary encoded format of the length of the data $H_{K_2}(x)$.

HMAC is a "fixed IV" variant of NMAC and uses the hash function H as a black box. The HMAC function is shown in Figure 7.3. The HMAC function working on an arbitrary length message x is defined as:

$$\mathrm{HMAC}_K(x) = H_{IV}(\overline{K} \oplus \mathrm{opad}||H_{IV}(\overline{K} \oplus \mathrm{ipad}||x)) \tag{7.3}$$

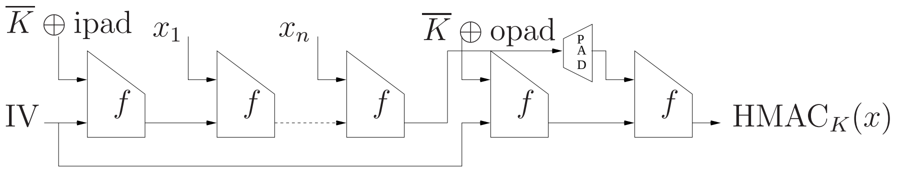


Figure 7.3: The HMAC Construction

HMAC is a particular case of NMAC. $\mathrm{HMAC}_K(x) = H_{K_1}(H_{K_2}(x))$ where $K_1 = f_{IV}(\overline{K} \oplus \mathrm{opad})$, $K_2 = f_{IV}(\overline{K} \oplus \mathrm{ipad})$, f is the compression function of the hash function, **opad** and **ipad** are the repetitions of the bytes 0x36 and 0x5c as many times as needed to get a b-bit block, $\overline{K}$ indicates the completion of the key K to a b-bit block by padding K with 0 bits. Since the outer function in this expression processes only one message block, it can be written as $\mathrm{HMAC}_K(x) = f_{K_1}(H_{K_2}(x)) = \mathrm{NMAC}_{K_1,K_2}(x)$. Precisely speaking, whether this relationship between HMAC and NMAC holds or not depends on the padding employed on the underlying hash function [110]. This fact implicitly suggests that the security of NMAC does not depend on the padding of the hash functions used in NMAC. However, it will be explicitly shown in Section 7.3 that the proof of security of NMAC is independent of the padding employed on the hash function.

7.1.2 Analysis of NMAC and HMAC

NMAC and HMAC algorithms were proved to be secure [13] given some reasonable assumptions on the underlying hash functions. The following definitions are considered in giving the security analysis of the NMAC function.

Definition 3 *A* MAC *based on the keyed compression function f is an (ϵ_f, q, t, b)-secure* MAC *if any attacker, without knowledge of the key K_1 requesting q chosen messages x_i (where $i = 1 \dots q$ and $max(|x_i|) = b$) to the keyed compression function f, cannot break the scheme in a total time t except with probability less than ϵ_f. In other words, ϵ_f is the maximum probability of forging f_{K_1} within time t.*

Definition 4 *A keyed iterated hash function H is an (ϵ_F, q, t, L)- weakly collision resistant hash function if any attacker, without knowledge of the key K_2 requesting*

q chosen messages x_i (where $i = 1 \ldots q$ and $max(|x_i|) = L$) to the keyed iterated hash function H, cannot find two messages x and x' in a total time t such that $H_{K_2}(x) = H_{K_2}(x')$ with probability better than ϵ_F. In other words, ϵ_F is the maximum probability of finding collisions for H_{K_2} within time t.

The main analytical result for NMAC based on the above definitions is given in [13] and is stated as follows:

Theorem 3 *If the keyed compression function f is an (ϵ_f, q, t, b)-secure* MAC *and the keyed iterated hash function H is an (ϵ_F, q, t, L)- weakly collision resistant hash function then the NMAC function is an $(\epsilon_F + \epsilon_f, q, t, L)$-secure* MAC.

The proof of security for NMAC provided by Bellare, Canetti and Krawczyk [13] is constructive in the sense that given an adversary that breaks the NMAC function with some significant probability, one can explicitly construct an algorithm that using the same resources breaks the underlying hash function with at least half of that probability.

The security analysis provided for NMAC works for HMAC as well under the assumption that the compression function used to derive the keys K_1 and K_2 for HMAC works as a pseudorandom function (PRF) [13]. Informally, a compression function f_{IV} is said to be a PRF if it is computationally infeasible for an adversary or a distinguisher to distinguish it from a random function. A formal definition for a PRF is given in the Section 7.5. As the analysis of NMAC assumes that the keys K_1 and K_2 are truly random and independent, to lift the analysis of NMAC on to HMAC, one needs to assume that the keys K_1 and K_2 derived using f cannot be distinguished by the attacker from truly random keys. It should be noted that in practice, the secret keys are generated using the pseudorandom generators and that pseudorandom generator is built within the definition of HMAC using the compression function f and the constant key pads **opad** and **ipad**.

It should be noted that the original design of HMAC uses concatenated pads instead of XOR pads [13]. However, no reasons were given by Bellare, Canetti and Krawczyk for such a replacement. One of the reasons for such a replacement could be for a provision to support long keys. If the key is long enough, the effect of appending the pad is diminished to the point of disappearing entirely when the key size equals the block size of the compression function (for example, 512 bits long for hash functions MD5, SHA-1 and SHA-256). In addition, XORing the pads to the key creates more difference in the blocks than the concatenation of pads to the key [261]. The another goal is to avoid explicitly changing the IVs. While making the keys for the internal and external applications of the hash function, “different-looking” keys must be used as much as possible [40].

7.1.3 Summary of the Review on MACs

The above analysis of MACs based on hash functions shows that the position of the secret key and hash function in the architecture of a MAC function influence the security of the MAC scheme with respect to forgery and key recovery attacks. For example, the slice by slice trail key recovery attack which works on the envelope MAC scheme does not work on the NMAC function due to keying the IV. Similarly, this attack does not work on HMAC as the keys occupy block completely without spanning across the blocks.

From the performance point of view, it makes sense to design MAC schemes (as done for HMAC) that use a hash function as a black box as such schemes directly inherit the performance of the deployed hash functions. The reason for only HMAC but not NMAC being widely deployed and standardised is that apart from its security proof, it calls widely deployed hash functions such as SHA-1 as a black box, which is not the case with NMAC which uses compression function as a black box. From the security perspective, both NMAC and HMAC share the same properties.

Chapters 2, 3 and 4 provide a valuable insight that the security of iterated hash functions depends not only on the strength of the compression function but also on how that compression function is used or iterated to design a hash function. As said above, a similar argument holds for MAC schemes designed using the iterated hash functions. In particular, the side-channel cryptanalysis of HMAC and NMAC schemes [221] shows that not only the way of use of the hash function and secret key but also the internal structure of the compression function influence these attacks on these MAC schemes. For example, HMAC and NMAC functions based on the Davies-Meyer compression function are vulnerable to side channel attacks but not those based on Matyas-Meyer-Oseas compression function as shown in Chapter 8.

From this discussion, one can observe that it may be better to use compression function as a black box in the construction of MAC schemes rather than hash function for a protection from several different attacks[2]. Using compression function as a black box is trivial for hash functions SHA-1 and SHA-256. Compression functions rather than hash functions as black boxes provide a better flexibility in designing MAC schemes. Such MAC schemes can probably be proved secure basing security assumptions on the compression function primitive rather than on the mode of operation of the hash function. These days only Merkle-Damgård hash functions are widely deployed and used. This might not be the case in the future hash function designs due to various attacks on the Merkle-Damgård iterative structure and hash functions such as MD5 and SHA-1 that follow its design principles (see Chapter 2 and Appendix B). For example, one might construct several hash function modes, for example, using the compression function of SHA-256. In such a situation, it makes sense to use compression function "as is" rather than a particular mode of operation of hash function such as Merkle-Damgård construction to design MAC schemes or for that matter any cryptographic protocol. The reason is it is likely that security analysis for different modes require different assumptions and security proofs based on the reduction techniques differ for different modes of operations. This argument is justified first in Section 7.3 where the proof of NMAC-1 differs slightly from NMAC and then in Section 7.6 where a possible approach to prove the security of O-NMAC differs from that on NMAC. In particular, though NMAC and NMAC-1 differ in only padding technique employed in them, the assumptions used in their respective proofs differ.

In addition, if some attack is invented on a particular mode of operation which is widely deployed (e.g. collision attacks on MD5 and SHA-1) then the security proofs of the MAC schemes that base their assumption on such modes, do not provide any guarantee. This is the reason for the provision of the new proof techniques for HMAC and NMAC [12] using strong assumptions on the compression functions as the security guarantee of the old proofs [13] that base their assumptions on the weakly collision resistance property of hash functions is lost due to the collision attacks on the hash functions [12, 109]. However, this does not mean that old proofs are incorrect, it only means that proofs do not provide security guarantee due to the attacks on the hash functions.

[2]In a similar context, Mironov [196] noted that it is wrong, dangerous, reckless and ignorant to use a hash function as a black box.

7.2 Analysis of NMAC using Weaker Assumptions on the Hash Functions

A secure MAC function must be both one-way and collision resistant for someone who does not know the secret key. Whether a MAC function must be one-way or collision resistant for someone who knows the secret key depends on the application [228, p.19], [186, p.327]. Though NMAC and HMAC functions were shown to be secure under the assumption that the adversary who tries to forge them has no knowledge of the secret keys, there may be some applications where one could expect these functions to satisfy some additional properties for protection against insiders who know the secret key. An example application based on HMAC-SHA-1 requiring extra protection from insider attacks is given in Appendix F. Hence, it is interesting and important to analyse the behaviour of the NMAC and HMAC functions with reduced assumptions than given in its original proof of security.

In the following section, security analysis of NMAC and HMAC is given using *weaker assumptions* on the hash functions. The properties that are essential for the protection of these MAC functions from insiders are also given. This analysis shows that theoretically NMAC as a MAC function is not always secure if the assumptions on the hash functions are weaker than the assumptions given in Section 7.1.2.

7.2.1 Security Analysis

Remark 4.9 of Bellare, Canetti and Krawczyk [13] is the motivation to split this analysis on NMAC into three cases. This analysis shows that, theoretically, one cannot further weaken the assumptions in the proof of security of NMAC to obtain a secure NMAC function and that to attain it, both keys *must* be secret. A similar analysis can be applied to HMAC. This analysis also partially solves the open question in Preneel's thesis [228] on the properties required from the MAC function when the adversary has knowledge of the key(s) with respect to the NMAC function.

Case 1: *Only* K_1 is secret

In this case, it is assumed that the adversary knows the key K_2 of NMAC. Hence, one finds offline collisions on H_{K_2} which is easier to perform than finding collisions when the key K_2 is random and secret as the attacker needs to contact the legitimate owners of the key K_2 to get collisions for H_{K_2}.

Once the attacker finds x and x' such that $x \neq x'$ and $H_{K_2}(x) = H_{K_2}(x')$, this MAC function is attackable using the chosen message attack. The attacker sends the message x to the NMAC oracle and gets the tag as response and uses this tag as a forgeable value for the message x'. In addition, once the attacker gets a collision on the internal function, the attacker can extend the collisions on the collided pair of messages x and x' to the format $H_{K_2}(x||y) = H_{K_2}(x'||y)$ where y is an arbitrary text appended to the collided messages. Hence, to attain a secure NMAC function, the internal function H_{K_2} must be collision resistant meaning the complexity of finding collisions for H_{K_2} must be at least $2^{|K_2|/2}$ when the attacker has knowledge of the key K_2.

Case 2: *Only* K_2 is secret

This case assumes that the attacker knows the key K_1 used in NMAC and provides the security analysis of the scheme against straight forward length extension attacks. The analysis also assumes that the attacker can only access the NMAC oracle as a whole though it has knowledge of the key K_1.

To forge the MAC scheme presented in this case, it seems that the attacker either has to invert the function f_{K_1} for the known output of the NMAC function or has to find collisions for NMAC as the attacker cannot get the actual result of $H_{K_2}(x)$ but only its value after applying f_{K_1}. The value $H_{K_2}(x)$ is viewed as a *message dependent secret* input to the function f_{K_1}. Hence the function f_{K_1} must be one-way as it must be hard for the attacker to find $H_{K_2}(x)$ using x and the NMAC output.

Hirose [109] has shown that the weakly collision resistance of H_{K_2} implies collision resistance of f used in H_{K_2} under the assumption that f is a PRF. Assume that the outer compression function f is different from the compression function in the inner iterated hash function H. From this discussion, naturally the *function f_{K_1} must be both one-way and collision resistant for some one who knows the key K_1*. However, the following analysis shows that one cannot always attain a secure MAC function NMAC with a security requirement of weakly collision resistance for the function H_{K_2} even if f_{K_1} is both one-way and collision resistant.

Theorem 4 *A one-way and collision resistant external function f_{K_1} (K_1 is known) and a weakly-collision resistant H_{K_2} inner function do not always imply a secure MAC function NMAC.*

Proof of Theorem:

Let $H_{K_2} : \{0,1\}^* \to \{0,1\}^l$ and $f_{K_1} : \{0,1\}^l \to \{0,1\}^n$.

Let $G_{K_2} : \{0,1\}^* \to \{0,1\}^{l'}$ where $l > l'$.

Let $g_{K_1} : \{0,1\}^l \to \{0,1\}^n$ be one-way and collision resistant.

The function H_{K_2} is defined as $H_{K_2}(x) = G_{K_2}(x)||0^{l-l'}$.

The function f_{K_1} is defined as $f_{K_1}(z'||z'') = z'||g_{K_1}(z'')$ where $z' \in \{0,1\}^{l'}$. Then f_{K_1} is also collision resistant and one-way.

On the other hand, $f_{K_1}(H_{K_2}(x)) = G_{K_2}(x)||g_{K_1}(0^{l-l'})$ and $G_{K_2}(x)$ is obtained from the NMAC oracle. Even if the function f_{K_1} is one-way and collision resistant, it cannot always prevent the straight-forward extension attack as the output of f_{K_1} gives $G_{K_2}(x)$. □

Remarks:

1. The above analysis shows that the function G_{K_2} should be a secure MAC to make NMAC a secure MAC function. The function H_{K_2} is also a secure MAC if G_{K_2} is a secure MAC. The function G_{K_2} will work as a secure MAC function *only* when the input messages are *prefix-free* as extension attacks do not work on the hash functions with *prefix-free* input messages [14, 50].

2. From the theoretical point of view, the above analysis shows that the outer function with no secrecy is not always good enough to prevent straight forward extension attacks. Nevertheless, one can use more “natural” (naturally strong functions like SHA-256) f_{K_1} and H_{K_2} to attain a secure MAC function to protect against insiders who know the key K_1 and also from length extension attacks.

This MAC scheme is weaker than NMAC from the perspective of complete key-recovery as it uses just one key. Note that the NMAC function does not achieve security against the key-recovery over the combined lengths of the keys due to the divide and conquer key recovery attack. The complexity of this attack on NMAC is about $2^{|K_1|} + 2^{|K_2|}$ [13]. Note that this attack is impractical for reasonable key sizes of K_1 and K_2 as stated in Section 7.1. However, once the attacker gets the key K_2 by employing this attack, the security of the NMAC function against forgery reduces to that of secret prefix scheme and there is no need for the attacker to find K_1 to perform a forgery.

Case 3: <u>Both K_1 and K_2 are not secret</u>

When the attacker knows both the keys K_1 and K_2 of NMAC, it is obvious that both functions H_{K_2} and f_{K_1} must be collision resistant for the protection against insider attacks. In such a case, this scheme will provide an $n/2$-bit security level against extension attacks where n is the size of output in bits. This scheme is basically the double hashing scheme proposed in [79] (called DHASH in Section 3.4 of Chapter 3) to obtain a higher security level against length extension attacks.

7.3 An Efficient Variant for NMAC

This section proposes an efficient variant to the NMAC function by altering the padding technique used in the hash function. It will be shown that the proof of security of NMAC does not depend on the padding technique used for the hash function by specifying a new padding technique for the inner and outer functions in NMAC. That is, by proving the security of the NMAC function with the new padding technique, it is demonstrated that the security definitions and proof used by Bellare, Canetti and Krawczyk [13] for establishing the security of NMAC are independent of the specification of padding. That is, Merkle-Damgård strengthening is important for the security of a collision resistant hash function [164, 186], but it is not important to the security of NMAC and HMAC functions[3].

The NMAC function with the new specification for padding shall be called NMAC-1, which shall mean NMAC variant 1. Since padding of the message is not part of the compression function, NMAC-1, like NMAC uses the compression function f of the iterated hash function H "as is".

7.3.1 Specification of NMAC-1

An arbitrary finite length message is denoted by x. The message is considered as short if it fits in one block or less than one block. The block size of the compression function is represented with b which is equal to 512 bits for hash functions such as MD5, SHA-1, SHA-256 and equals 1024 bits for functions such as SHA-384 and SHA-512 [211]. Let n be the size of chaining variable and also the size of the MAC output.

[3]This observation was also made recently in [12].

The NMAC-1 function on an arbitrary length message x is defined as follows:

$$\text{NMAC-1}(x) = f_{K_1}(H_{K_2}(x)) \tag{7.4}$$

where $H_{K_2}(x)$ is defined as follows based on the size $|x|$ in bits of the input message x.

$$H_{K_2}(x) = \begin{cases} f_{K_2}(x) & \text{if } |x| = b \\ f_{K_2}(x') \text{ with the input } x' = x||10\ldots0 & \text{if } |x| < b \\ \text{iteration of} f_{K_2} \text{with the input } x||10\ldots0 & \text{if } |x| > b. \end{cases}$$

For the case, $|x| \neq b$, the message x is padded with a 1 bit followed by 0's (possibly none) to make x a multiple of the block length b of the compression function. For example, when $|x| = b - 1$, only bit 1 is padded to x and when $|x| = b - 2$, x is padded with bits 10.

The padding for the outer function f_{K_1} is based on the size of the input message x. If $|x| = b$, then f_{K_1} is padded with 0's followed by the binary encoded representation of $|x|$ else it is padded with a 1 bit followed by sufficient 0 bits and the binary encoded representation of $|x|$. The reason for having two different padding techniques for the outer function f_{K_1} is due to the following forgery attack. Let x be a binary string such that $|x| = b-2$. That is, the size of x is less than a block. When x is processed using the NMAC-1 function, the inner function appends bits "10" to x and computes $\text{NMAC}_K(x) = a$ (say). Now, without any additional queries, one can forge NMAC-1 with the message $x' = x||10$, where x is the message in the previous query. If one considers the same padding technique, whatever it may be, for the outer function, $a = \text{NMAC-1}(x')$ =NMAC-1(x) which is a forgery on the NMAC-1 scheme. From this discussion, it is obvious that there is no need to use the length encoding twice to pad messages in NMAC.

7.3.2 Performance Aspects of NMAC-1

This section provides a performance comparison of NMAC-1 against NMAC in terms of the number of calls to the compression function for various message sizes when hash functions from the MDx family are chosen as the underlying algorithms for NMAC and NMAC-1. The improvement in the efficiency of NMAC-1 over NMAC is considerably high for short messages.

The efficiency improvement of NMAC-1 over NMAC is calculated using the formula.

Table 7.1: Efficiency improvement of NMAC-1 over NMAC

x in 32 byte increments	$\#(f)$ in NMAC	$\#(f)$ in NMAC-1	%EI
32	2	2	0
64	3	2	33
96	3	3	0
128	4	3	25
160	4	4	0
192	5	4	20
224	5	5	0
256	6	5	17

$$\%\text{Efficiency improvement (EI)} =$$
$$[(\#(\text{f}) \text{ in NMAC} - \#(\text{f}) \text{ in NMAC-1})/\#(\text{f}) \text{ in NMAC}] \times 100$$

From Table 7.3.2, it is clear that for messages of lengths in bits in the set $[448, 512]$, the improvement in the efficiency of NMAC-1 over NMAC is 33% assuming that the block size b of the compression function is 512 bits. The improvement in this efficiency decreases for messages of more than one block size. In comparison to HMAC, the improvement in the efficiency of NMAC-1 over HMAC is 60% for messages that fall in the set $[448, 512]$ as HMAC makes five invocations of the compression function to process messages of lengths in bits in the set $[448, 512]$.

7.3.3 Security Analysis of NMAC-1

The terminology used in the proof of security of NMAC-1 shall be the same as that used Bellare, Canetti and Krawczyk [13] for the sake of clarity. The analytical results of NMAC-1 are based on the chosen or adaptive chosen message attacks. The main analytical result of NMAC-1 uses the definitions of a secure MAC and weakly collision resistant hash function given in Section 7.1.2. In addition, the analysis uses the following definition of weakly collision resistant compression function.

Definition 5 *A keyed compression function f_{K_2} is an (ϵ'_f, q, t, b)- weakly collision resistant compression function if any attacker, without knowledge of the key K_2, requesting q chosen messages x_i (where $i = 1 \ldots q$ and $max(|x_i|) = b$) to the the function f_{K_2} cannot find two messages x and x' in a total time t such that*

$f_{K_2}(x) = f_{K_2}(x')$ with a probability better than ϵ'_f. In other words, ϵ'_f is the maximum probability of finding collisions for f_{K_2} in time t.

Theorem 5 *If the keyed compression function f is an (ϵ_f, q, t, b)-secure MAC and the keyed iterated hash function H is an (ϵ_F, q, t, L)-weakly collision resistant hash function and the fixed length input keyed compression function f is an (ϵ'_f, q, t, b)-weakly collision resistant compression function where $L \geq b$ then the NMAC-1 function is an $(\epsilon_f + \epsilon_F + \epsilon'_f, q, t, L)$ secure MAC.*

Proof of Theorem 5:

The parameters q,t and L are fixed and represent the number of queries to the NMAC-1 oracle, the total attack time t and the maximum length L of each finite length message x_i (where $i = 1, 2, \ldots, q$) to be queried.

The attacker A_N which tries to break NMAC-1 sends each message x_i to the NMAC-1 oracle which gives response NMAC-$1_K(x_i)$ for every queried message x_i. Finally, the attacker A_N outputs the message x and its forged tag y. The forgery is successful if $x \neq x_i$ and NMAC-$1_K(x) = y$. Let ϵ_N be the maximum probability that A_N succeeds in forging NMAC-1.

Using A_N, an attacker A_f is built which aims to forge the MAC function f_{K_1}, on inputs of messages of length b by sending q queries to the oracle f_{K_1} in time t, with a maximum success probability of ϵ_f. The proof model of NMAC [13] applies to NMAC-1 as well because the adversary A_f in NMAC processes each message x_i, block after block, finally padding the last block and computing $H_{K_2}(x_i)$ for every unpadded input message x_i sent by A_N to the NMAC oracle. So, A_f can be simulated to perform $H_{K_2}(x_i)$ for every message x_i using the proposed padding given for the internal function in NMAC-1. Moreover, the proof of NMAC allows the adversary A_f to choose its own messages to query the MAC function f_{K_1}. Therefore, A_f can be simulated to query f_{K_1} using the proposed padding scheme given for the outer function in NMAC-1. Hence, the security of the NMAC function [13] under the given proof model is independent of the padding technique employed for the inner keyed iterated hash function and the outer keyed compression function.

The algorithm of A_f is given below:

Choose random K_2

For $i = 1, \ldots, q$ perform the following steps:

1. $A_N \rightarrow x_i$
2. A_f computes $H_{K_2}(x_i)$ according to the specified padding technique
3. A_f queries f_{K_1} with $\overline{H_{K_2}(x_i)}$ and gets $f_{K_1}\overline{(H_{K_2}(x_i))}$ where $\overline{H_{K_2}(x_i)}$ represents the padding of $H_{K_2}(x_i)$. $\overline{H_{K_2}(x_i)} = H_{K_2}(x_i)||00\ldots0$ if $|x| = b$ and $\overline{H_{K_2}(x_i)} = H_{K_2}(x_i)||10\ldots0$ if $|x| \neq b$.
4. $A_N \leftarrow f_{K_1}\overline{(H_{K_2}(x_i))}$

A_N outputs (x, y) where $x \neq x_i$
A_f outputs $(\overline{H_{K_2}(x)}, y)$

Let ϵ_F and ϵ'_f be the maximum probabilities of any adversary for which $H_{K_2}(x) = H_{K_2}(x_i)$ and $f_{K_2}(x) = f_{K_2}(x_i)$ respectively. The probability with which the adversary A_f fails to forge f_{K_1} is:

$\Pr[A_f fails] \leq \Pr[A_N \text{ fails}] + \Pr[A_N \text{ succeeds } \wedge |x| \leq b\ \exists i(|x_i| \leq b \ \wedge\ f_{K_2}(x) = f_{K_2}(x_i))] + \Pr[A_N \text{ succeeds } \wedge |x| \geq b\ \exists i(|x_i| \geq b \ \wedge\ H_{K_2}(x) = H_{K_2}(x_i))]$.

That is, $1 - \epsilon_f \leq 1 - \epsilon_N + \epsilon'_f + \epsilon_F$

$\Rightarrow \epsilon_f \geq \epsilon_N - \epsilon'_f - \epsilon_F$.
$\Rightarrow \epsilon_N \leq \epsilon_f + \epsilon'_f + \epsilon_F$.

Hence, the probability of forging NMAC-1 is at most the sum of the maximum probabilities of finding collisions for the inner keyed compression function and the keyed iterated hash function and forging the outer keyed compression function.□

7.3.4 Comparison of NMAC-1 with Other Efficient MACs based on Hash Functions

Patel [225] has proposed Enhanced NMAC (ENMAC) to improve the efficiency of NMAC for short messages. In the specification of ENMAC, a short message was defined as a message with size $|x|$ in bits less than or equal to $b - 2$. The ENMAC function is defined below:

$$\text{ENMAC}_K(x) = \begin{cases} f_{K_1}(x) \text{ with the input } x = x||11 & \text{if } |x| = b-2 \\ f_{K_1}(x) \text{ with the input } x = x||10\ldots01 & \text{if } |x| < b-2 \\ f_{K_1}(x_{pref}, F_{K_2}(x_{suff}), 0) & \text{if } |x| > b-2. \end{cases}$$

where x_{pref} contains the bits from 1 to $b-n-1$ and x_{suff} contains bits from $b-n$ to $|x|$. They are represented as $x_{pref} = x^1, \ldots, x^{b-n-1}$ and $x_{suff} = x^{b-n}, \ldots, x^{|x|}$.

When $|x| \leq b-2$, a unique padding technique is employed by appending x with a compulsory 1 bit followed by necessary 0 bits to make the size of x equal to the size of b. In this case, the last bit is always set to 1 and the concatenation of 0 bits depends on whether $|x| < b-2$. The last bit indicates whether or not ENMAC is used to process a single message block. For example, to authenticate a short message of 500 bits, ENMAC based on SHA-1 requires just one call to the compression function f of SHA-1 whereas NMAC requires three calls and NMAC-1 requires two calls to the function f.

When $|x| > b-2$, the message x is split into two parts, prefix (x_{pref}) and suffix (x_{suff}). ENMAC first processes the x_{suff} using the internal function F_{K_2} and then the prefix part x_{pref} using the output of $F_{K_2}(x_{suff})$. For example, see SHA-1-ENMAC discussed in [225]. In this case, if x_{suff} begins at a non-word border, all words in x_{suff} need to be re-aligned. To overcome such problems, a practical variant for ENMAC was proposed in [225] and is defined as follows:

$$\text{ENMAC}_K(x) = \begin{cases} f_{K_1}(x) \text{ with the input } x = x||11 & \text{if } |x| = b-2 \\ f_{K_1}(x) \text{ with the input } x = x||10\ldots01 & \text{if } |x| < b-2 \\ f_{K_1}(F_{K_2}(x_{pref}), x_{suff}, 0) & \text{if } |x| > b-2. \end{cases}$$

where $x_{pref} = x^1, \ldots, x^{|x|-(b-n-1)}$ and $x_{suff} = x^{|x|-(b-n)}, \ldots, x^{|x|}$.

Assume that a standard padding technique has been employed for both x_{pref} and x_{suff} to obtain a secure MAC function ENMAC[4]. Now this variant of ENMAC requires knowledge of the length of x to calculate x_{pref} and x_{suff} as the length of the message x determines the content in the last two blocks of x_{pref} where the last block of x_{pref} is the padded block. The knowledge of the length of the message x is required even if the message x is padded with a bit 1 followed by 0's as the specification for x_{pref} has $|x|$ as an argument. In general, the length of data to be hashed is known ahead of time in many applications. However, in rare situations it is not [123].

In many cases, any MAC function based on the hash function used to protect the authenticity and integrity of the communicated data does not know the length

[4]The padding employed on x_{pref} and x_{suff} is not specified in the specification of ENMAC or its variant in [225].

of the message in advance. However, a machine evaluating the ENMAC function to generate authentication codes for the communicated data, must know the length of the message x in advance to find the content of x_{pref}. The machine may also perform the ENMAC computation as follows: when it receives the authentication tag attached to the message (specifically, to the last block), it has to look at the intermediate MAC value obtained at the previous one or two blocks before the final block of x and has to re-compute x_{pref} accordingly taking into account the padding for x_{pref}. Then it has to evaluate the outer function of ENMAC. In this case, when ENMAC is used to process a large amount of data, there would be a slight performance inefficiency due to the above process. This problem can be solved using a special code [123] to tell the ENMAC routine that the total length of x_{pref} is not known when the processing of the data has begun and that it will be input with the final chunk of data for x_{pref} and the x_{suff} that follows x_{pref}. The special code used for this purpose needs to be clearly different from the valid total length code which appears in the last block so that the correct processing of the ENMAC function can be performed.

Note that the prior knowledge of the message or any special code is not essential to evaluate tags on messages of more than a block using NMAC, HMAC and NMAC-1 functions. They only require knowledge of the length of the encoded message in the last block (if length encoding is used as part of the padding) or the authentication tag attached to the last block to indicate the end of data stream for that particular session in the communication channel.

The security treatment of NMAC and NMAC-1 takes into account the maximum length L of the message to be processed as a security parameter. During the chosen (adaptive) message message attack on any of these functions, their respective oracles return a response to a query of arbitrary length in one step. This is, however, not a realistic measure. It is reasonable to assume that the processing time of a MAC algorithm is proportional to the length of the message and the length of the message to be processed must be a security parameter. The analysis of ENMAC and its variants does not consider the length of the message as a security parameter.

As observed by Bellare, Canetti and Krawczyk [13], one can convert NMAC into a hybrid MAC function using two different functions as long as the assumptions stated in the proof of security of NMAC hold. This also holds for NMAC-1 too. For example, one can use SHA-256 for the internal function of NMAC-1 and a block cipher in the CBC mode for the external function of NMAC-1. In this sense, the result of NMAC-1 is more general than stated in the proof. In addition, one can use the wide-pipe hash [174] (e.g. SHA-512 and truncating half of the output bits) for the inner function and the compression function of SHA-256 for the external function in NMAC and NMAC-1. In comparison, the design of such hybrid schemes using ENMAC imposes a penalty on its performance as the second function used in ENMAC not only uses the output of the first function but also part of the message to be authenticated.

Finally, the ISO/IEC 9797-2 [119] standard specifies a mechanism which is a variant of MDx-MAC [234] that offers high performance for applications that process short messages of up to 256 bits. MDx-MAC, as stated in Section 7.1, unlike NMAC and its variants, invokes the complete hash function once but it makes a small modification to the compression function by adding a secret key to the additive constants in the compression function. In addition, MDx-MAC and its variant were not formally analysed unlike NMAC and its variants.

7.4 M-NMAC: A New Variant of NMAC

As pointed out by Bellare, Canetti and Krawczyk [13], keying the IVs of the hash functions as was done in NMAC and HMAC functions allows for a better modeling of the keyed hash functions with some significant analytical advantages. It is interesting to see how it differs from using the keys as data blocks which is the other common way of designing keyed hash functions. This section will explore this concept further by introducing a new variant to the NMAC function called M-NMAC, which shall mean, modified NMAC.

M-NMAC uses the second key K_1 as a block instead of as a state to the external function f. This function is shown in Figure 7.4 and is defined below:

$$\text{M-NMAC}_K(x) = f_{H_{K_2}(x)}(\overline{K_1}). \tag{7.5}$$

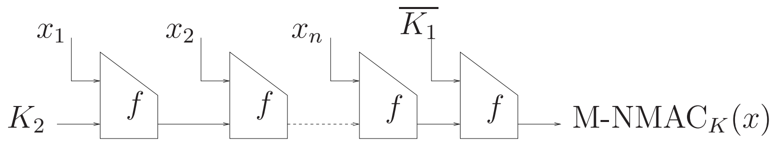


Figure 7.4: The M-NMAC Construction

In the equation 7.5, $\overline{K_1}$ denotes the key K_1 made to a block size b of the compression function f. That is, if the function f is the compression function of SHA-1 then the length $|K_1|$ of the key K_1 is at most 512 bits. It is recommended that $|K_1| \geq |K_2|$ and K_1 is completed to size b by appending 0 bits if $|K_1| < b$. The length of the message x after padding must be a multiple of the block length b of the compression function f as in NMAC. While the maximum lengths of both the keys K_1 and K_2 in NMAC depends on the sizes of the initial state of the keyed iterated hash function H and the external keyed compression function f used in NMAC, *only* the size of the key K_2 is dependent on the initial state of the keyed hash function H.

M-NMAC can also be seen as a kind of envelope MAC scheme discussed in Section 7.1. It differs from the envelope MAC scheme in the following features:

- M-NMAC uses the secret key K_2 as an IV instead of as a block.
- While the position of the trail secret key K_1 depends on the length of the message to be authenticated using the envelope MAC scheme, its position is independent of the length of the message in M-NMAC.
- While the trail secret key K_1 in the envelope MAC scheme can either span across the last two blocks or sit only in the last block, it is always positioned in the last block in M-NMAC.

Note that M-NMAC is also a variant of the variant of the envelope MAC scheme proposed by Preneel and van Oorschot [235, p.22] which uses the trail key in a separate block as M-NMAC. However, while M-NMAC uses the other key as an IV, the variant of the envelope MAC scheme uses the other key as a data block.

In addition, the M-NMAC construction can also be seen as a variant of the *append cascade* (ACSC) construction proposed in [14] with the difference that while the former scheme uses the trail secret key in a separate block, the latter scheme seems to use it immediately after the message as in the envelope MAC scheme. Moreover, the ACSC design was not actually defined in [14] for the purpose of message authentication but as a PRF.

7.4.1 Security Analysis of M-NMAC

It should be noted that the proof of security of M-NMAC follows from the proof of security of NMAC under the assumption that the keyed hash function H_{K_2} is a weakly collision resistant hash function and the external function $f_z(\overline{K_1})$ is a secure MAC where $z = H_{K_2}(x)$. As in NMAC, the birthday attack is the best known forgery attack on M-NMAC. The advantage of M-NMAC over NMAC is the flexibility in using variable key lengths as large as the block size b of the compression function for the trail key K_1 like in HMAC. The total complexity of the divide and conquer complete key recovery attack on M-NMAC is about $2^{|K_1|} + 2^{|K_2|}$. If $|K_1| > |K_2|$, then this total cost approximates to $2^{|K_1|}$. Since the secret key K_1 is used in a separate block independent of the message, the slice by slice trail key recovery attack does not work against M-NMAC as this attack requires the trial key to be split across the blocks. However, due to the divide and conquer key recovery attack on M-NMAC, once the attacker recovers the secret key K_2, the security of M-NMAC against forgery, like NMAC, reduces to the security of the secret prefix MAC scheme.

7.5 Pseudorandomness of NMAC and HMAC Functions

Let $F' : \{0,1\}^b \times \{0,1\}^k \rightarrow \{0,1\}^k$ be a family of keyed compression functions or a fixed-length input pseudorandom function (FI-PRF) family. That is, there is a set of keys and each key K of length k names a function from the family F'. This is denoted by F'_K or f. Let $R : \{0,1\}^b \rightarrow \{0,1\}^k$ be a family of all functions with the distribution being uniform; that is picking a function at random from this family just means drawing a random function of $\{0,1\}^b$ to $\{0,1\}^k$. If S is a probability space then picking a string x from S is denoted by $x \overset{\$}{\leftarrow} S$.

The family F' is said to be a pseudorandom function if the input-output behaviour of a random member of this family is computationally indistinguishable from the behaviour of a function picked at random from the family R. This is formalized using the notion of distinguishers [14] or statistical tests [97]. A distinguisher is given an oracle access to the function f chosen at random from one of the two families (F' or R) and is allowed to decide the family from which f is chosen. Formally, to any such distinguisher D a number between 0 and 1 called *prf-advantage* or *distinguishing probability* is associated and is defined as,

$$\mathrm{Adv}^{\mathrm{prf}}_{F'}(D) = |\Pr_{f \overset{\$}{\leftarrow} F'}[D^f = 1] - \Pr_{f \overset{\$}{\leftarrow} R}[D^f = 1]|$$

with the probabilities taken over the choices of f and the coin tosses of D.
The security of the family F' as a pseudorandom function depends on the resources that D uses. These resources include running-time and number and length of oracle queries. The running-time t includes the time taken to execute $f \overset{\$}{\leftarrow} F'$, time taken to compute responses to oracle queries made by D and the memory which includes the size of the description of D. The distinguisher $D(t, q, b, \epsilon^*)$ distinguishes F' from R if it runs for time t, makes q oracle queries each of block length b bits and $\mathrm{Adv}^{\mathrm{prf}}_{F}(D) \leq \epsilon^*$ where ϵ^* is called the distinguishing probability. This is defined below:

Definition 6

A family of fixed-length input keyed compression functions $\{F'_t\}$ is (ϵ^*, q, t, b)-FI-PRFs if any distinguisher that is not given the key K, is limited to spend total time t and sees the outputs of the given function f computed on q distinct inputs each of size b bits, cannot distinguish the function f from a random function of the family R except with a probability less than ϵ^*.

Now the main analytical result on NMAC as a PRF is given below and the analysis uses Definitions 2 and 6.

Theorem 6

Suppose $F' : \{0,1\}^b \times \{0,1\}^k \rightarrow \{0,1\}^k$ is a fixed-length input function family where k is the length of the key. Suppose F' is an (ϵ^*, q, t, b)-pseudorandom function on inputs of length b bits. Let $\{F_k\}$ be a family of (ϵ_F, q, t, L)-weakly collision resistant keyed hash functions where $L \geq b$. Let $NMAC : \{0,1\}^L \times \{0,1\}^k \rightarrow \{0,1\}^k$ be a function family where k is the length of each key K_1 and K_2 and $L \geq b$. Then the NMAC function is $(\epsilon^* + \epsilon_F, q, t, L)$-pseudorandom.

Proof of Theorem 6:

The parameters q,t and L are fixed and represent the number of queries to the NMAC oracle, the total attack time t and the maximum length L of each finite length message x_i (where $i = 1, 2, \ldots, q$) to be queried.
Let $R : \{0,1\}^b \times \{0,1\}^k \rightarrow \{0,1\}^k$ be a family of all functions and a distribution over them is uniform. Let $G : \{0,1\}^L \rightarrow \{0,1\}^k$ be a family of all functions and

a distribution over them is uniform.

Let A_N be the distinguisher that tries to break $NMAC$ as a PRF. Namely, A_N is given an oracle of the function g chosen at random from one of the two families; $NMAC$ or G. Now A_N's task is to distinguish the function g from a random function. A_N queries the oracle g with x_i and gets the response $g(x_i)$ for every queried message x_i where i ranges from 1 to q. Finally, A_N succeeds if it distinguishes $NMAC$ from G after looking at q input-output examples of g in time t. Let ϵ_N be the probability of success of A_N. After querying the oracle of the function g with q queries, A_N outputs a 0 or 1 bit. The output is 1 if A_N succeeds in correctly distinguishing $NMAC$ from G and the output is 0 if A_N fails in distinguishing $NMAC$ from G.
Using A_N, an attacker A_f is built which aims to tell whether the function f belongs to F' or R on inputs of messages of length b by sending q queries to the oracle of the function f in time t with a maximum probability of ϵ^*. That is, the goal of A_f is to distinguish the family F' from the random function family R.

The algorithm of A_f is given below:

Choose random K_2
For $i = 1, \ldots, q$ perform the following steps:

1. $A_N \rightarrow x_i$
2. A_f computes $F_{K_2}(x_i)$
3. A_f queries f with $\overline{F_{K_2}(x_i)}$ and gets $f(\overline{F_{K_2}(x_i)})$ where $\overline{F_{K_2}(x_i)}$ denotes the padding of $F_{K_2}(x_i)$.
4. $A_N \leftarrow f(\overline{F_{K_2}(x_i)})$

A_N outputs its decision, a 0 or 1 bit.
A_f outputs its decision, a 0 or 1 bit.

Analysis of success probabilities

We now analyze the success probability ϵ^* of the distinguisher A_f in distinguishing f_{K_1} from a truly random function. Let ϵ_F be the maximum probability that there exists at least one collision in F_{K_2} when A_N queries the NMAC function. The distinguisher A_f fails:

1. whenever A_N fails to distinguish $NMAC$ from G and outputs a bit 0.
2. whenever A_N outputs a bit 1 and the inner function F_{K_2} is not weakly collision resistant i.e $F_{K_2}(x_p) = F_{K_2}(x_q)$ for distinct p and q. In this case, the answer of A_N is not useful to tell whether the outer function f_{K_1} belongs to F' or R because $g(x_p) = g(x_q)$ for both f_{K_1} and a truly random function.

If F_{K_2} is weakly collision resistant then the probability that there exists p and q such that $F_{K_2}(x_p) = F_{K_2}(x_q)$ is negligible. Then, the answer of A_N is useful to tell whether the outer function is f_{K_1} or a truly random function because the inputs to the outer function are almost always distinct.

Hence, $\Pr[A_f \text{fails}] \leq \Pr[A_N \text{ fails}] + \Pr[A_N \text{ succeeds } \wedge \exists p, q\ (p \neq q\ \wedge\ F_{K_2}(x_p) = F_{K_2}(x_q))]$

$\Rightarrow 1 - \epsilon^* \leq 1 - \epsilon_N + \epsilon_F$
$\Rightarrow \epsilon_N \leq \epsilon^* + \epsilon_F$
Hence, the probability of breaking NMAC as a PRF is at most the sum of the maximum probabilities of finding collisions for the inner keyed compression function and distinguishing the outer function from that of a random function. □

Remarks:

1. Let $\epsilon_{max} = max(\epsilon^*, \epsilon_F)$

   $\Rightarrow \epsilon_N \leq \epsilon_{max} + \epsilon_{max}$
   $\Rightarrow \epsilon_N \leq 2.\epsilon_{max}$
   $\Rightarrow \epsilon_{max} \geq (1/2).\epsilon_N$.

   Given an adversary that distinguishes the NMAC function from a random function, one can explicitly show an algorithm that using the same resources to break the underlying hash function with at least half of that probability.

2. The proof of security of NMAC as a PRF also applies to HMAC as HMAC is a special case of NMAC as shown in Section 7.1.1.

7.6 On designing a MAC using a Hash Function with a Single Secret Key

The design and analysis aspects of MAC schemes based on hash functions studied in Section 7.1 poses a question on the possibility of designing a provably secure MAC scheme based using a hash function or compression function with only one secret key. It is quite clear that it is impossible to solve this problem using plain Merkle-Damgård mode of operation of hash functions as shown by the attacks on the secret prefix and secret suffix MAC schemes in Section 7.1. In addition, NMAC keyed with one secret key was shown to be either insecure or does not always result in a secure MAC in Section 7.2 depending on the position of the secret key in the MAC scheme.

The problem of designing a provably secure MAC function based on Merkle-Damgård hash functions (or precisely the compression function primitive) with just one secret key can be compared to the problem of designing a provably secure MAC scheme using a block cipher with just one secret key. Note that there is a provably secure MAC scheme based on a block cipher called one-key MAC (OMAC), a variant of CBC-MAC, which uses just one secret key [122][5]. In addition, note that the authors of OMAC pointed out that such a saving in the length of the secret key has made its proof of security substantially harder than the proof of security of TMAC, a two-keyed CBC-MAC [160] and XCBC, a three-keyed CBC-MAC [33, 34].

The problem of designing a MAC scheme using hash functions with just one secret key leads to a question on what modes of operation of popular and widely deployed compression functions such as SHA-1 and SHA-256 would assist in designing and proving the security of such a scheme. One simple approach is to construct a secret prefix MAC scheme using the wide-pipe hash function [174] by keying the IV of the hash function. For example, SHA-384 [211] is the wide-pipe hash function which uses the compression function of SHA-512 [211] producing an output of 384 bits instead of 512 bits. This hash function can be converted into a secret prefix MAC scheme by keying the IV of SHA-512 and using 384 output bits as the authentication tag. Hence, the attacker has to spend an effort of about $2^{512-384} = 2^{128}$ computations of SHA-384 to conduct the forgery on this

[5]NIST has recommended the OMAC equivalent cipher based MAC (CMAC) as a block cipher mode of operation for authentication [74].

scheme using preimage attack or length extension attack. Obviously, the amount of information on the tag available to the attacker becomes less when the output is truncated to 384 bits. The compression function used in designing this type of MAC scheme must be at least near-collision resistant as near collisions, for example, on SHA-512 might produce collisions for SHA-384 and hence for the secret prefix MAC scheme based on SHA-384.

In addition, the output length of such a MAC scheme must be long enough to make it hard for the attacker to guess the truncated bits. For example, if the secret prefix MAC scheme is designed using SHA-224, the attacker can guess the complete output of SHA-256 before it is truncated to 224 bits with a probability of at most 2^{-32}. Hence the output length must at least meet the security level of the birthday attack on the MAC scheme. It has been established that if the compression function is a random oracle, the wide-pipe hash function works as a random oracle [50]. One possible approach for proving the security of this MAC scheme is by assuming the compression function as a family of fixed length input PRFs (FI-PRFs) and then showing that secret prefix MAC scheme based on the wide-pipe hash function is a variable length input PRF (VI-PRF) family. It is well known that any PRF family is a secure MAC [15, 16, 96, 97] and hence such a secret prefix method works as a MAC function. However, such a security proof is out of the scope of this document.

The following section looks at the possibility of converting the **3C** mode of operation of hash function proposed in Chapter 3 as a MAC scheme. This MAC function is called one-keyed NMAC (O-NMAC). Its security is analysed against all the known attacks on the MAC schemes based on hash functions. However, only high-level discussion on the proof of security of this scheme based on the pseudorandomness of the compression function is provided and the task of completing this security proof is left as future work.

7.6.1 The O-NMAC Function

The specification for O-NMAC is given below following the terminology provided for NMAC in Section 7.1.1.

The message x to be authenticated using O-NMAC is processed using the hash function H iterated over the compression function f with the secret key K keyed as IV as shown in Figure 7.5. The secret key K is a key chosen at random from the

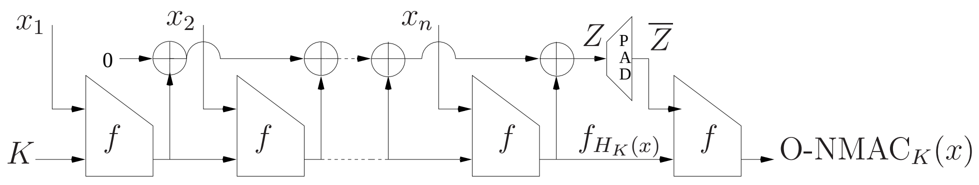


Figure 7.5: The O-NMAC construction

set of all possible keys $\{0,1\}^k$ each of length k bits. The message x is padded using a standard padding technique where a bit 1 followed by sufficient number of 0 bits are appended to x until total length of the message x is an exact multiple of the message block length b of the compression function f. Let $x_1, x_2, \ldots, x_{n-1}$ are all message blocks including padding. Finally, Merkle-Damgård strengthening with an extra block x_n containing the binary encoded representation of $|x|$ is appended to the padded message x. Under this setting, the message x to be authenticated using the MAC function O-NMAC is processed using the expression (7.6).

$$\text{O-NMAC}_K(x) = f_{H_K(x)}(\overline{Z}) \tag{7.6}$$

In the expression (7.6), $Z = \bigoplus_{i=1}^{n} H_i$ where $H_i = f(H_{i-1}, x_i)$, $H_0 = K$, $\overline{Z} = \text{PAD}(Z)$ and PAD is a padding function which appends to Z a bit 1, followed by a sufficient number of 0 bits and the binary encoded representation of $|Z|$. Similar to NMAC and HMAC, O-NMAC employs an extra compression function f which takes $\overline{Z}$ and $H_n = H_K(x)$ as the data block and chaining state inputs respectively. Note that the value $H_K(x)$ is basically a data and secret key dependent chaining input to the function f.

7.6.2 Analysis of O-NMAC

At this stage, one possible approach that could be adapted in giving a formal proof of security for O-NMAC based on the pseudorandomness of the compression function is provided.The complete proof is seen as future work. The outline of this possible approach for the proof of security is given below.

1. If the compression function is assumed as a FI-PRF family F, then the Merkle-Damgård iterated mode of this compression function family works as a VI-PRF family $F^{(*)}$ against prefix-free distinguishers [14]. Such a VI-PRF family is secure against the prefix-free distinguishers and is called basic cascade construction [14].

In the case of **3C**, the function family $F^{(*)}$ produces a two chain output: one is $F_K^{(*)}(x)$ and the other one is Z for every query x. While the Theorem 3.1 and its proof from [14] can be used in showing that if the keyed compression function F is a FI-PRF family then $F_K^{(*)}$ is a VI-PRF family against prefix-free distinguishers, it is unsure whether Z preserves the pseudorandomness. Prefix-freeness can be ensured by prepending the message with its length using some appropriate encoding mechanism and then iterate the function F (i.e, applying $F^{(*)}$).

2. Prepending messages with their length encoding at the beginning requires the machine evaluating the pseudorandom function to first measure the length of the message which requires first buffering the message and then processing it. However, MAC schemes and PRFs are supposed to process messages of arbitrary length "on the fly". Hence, Bellare, Canetti and Krawczyk [14] have proposed an *append cascade* construction (ACSC) which requires another small random key of length δ at the end but requires no prefix-free length encoding as an alternative for basic cascade construction to process messages of arbitrary length.

   By assuming that Z obtained from the VI-PRF family $F^{(*)}$ preserves pseudorandomness, one can follow the lines of Theorem 4.1 and its proof from [14] to show that if $F_K^{(*)}$ and Z are pseudorandom then F_K^{3C} is indeed pseudorandom. However, it is unsure how reasonable is the assumption that Z is pseudorandom.

3. Similar to Corollory 4.2 of [14], one can combine steps 1 and 2 above to show that if the compression function is a FI-PRF family then F_K^{3C} is a VI-PRF.

4. It is well known that any PRF family is a secure MAC [15,16,96,97]. So, **3C**-VI-PRF as a MAC function works as an effective variant to the NMAC and HMAC functions having some advantages at the level of key management over NMAC and HMAC.

The outline of the formal analysis given above shows that the F^{3C} construction works as a PRF and MAC, by assuming that the compression function defines a FI-PRF family. Since, the correctness of the above possible formal approach is unsure at this stage, the MAC and PRF security of F^{3C} scheme is analysed below using some cases.

1. **XORing data blocks:** Consider the case of processing of two distinct one block (or less than one block) messages x_1 and x_2 using O-NMAC with the key K. Then the O-NMAC function used to process these two blocks is defined in expressions (7.7) and (7.8) respectively.

$$\text{O-NMAC}_K(x_1) = f_{F_K(x_1,x_{pad1})}(\overline{Z_1}) \tag{7.7}$$

$$\text{O-NMAC}_K(x_2) = f_{F_K(x_2,x_{pad2})}(\overline{Z_2}) \tag{7.8}$$

   where $Z_1 = F_K(x_1) \oplus F_{F_K(x_1)}(x_{pad1})$ and $Z_2 = F_K(x_2) \oplus F_{F_K(x_2)}(x_{pad2})$; x_{pad1} and x_{pad2} are padded blocks for messages x_1 and x_2 respectively due to the Merkle-Damgård strengthening. Assume $|x_1| = |x_2|$. Then $x_{pad1} = x_{pad2}$.

   Now consider the case of $Z_1 \oplus Z_2$. This is given by $F_K(x_1) \oplus F_{F_K(x_1)}(x_{pad1}) \oplus F_K(x_2) \oplus F_{F_K(x_2)}(x_{pad1})$. Clearly, there are no XOR cancellations in this case as for pseudorandom compression functions, $F_K(x_1) \neq F_K(x_2)$ with significant probability resulting in $F_{F_K(x_1)}(x_{pad1}) \neq F_{F_K(x_2)}(x_{pad1})$. This can be generalised to any number of distinct data blocks.

2. **XORing data blocks with the first block fixed:** This is a special case of the above case. Consider processing of two distinct 2-block messages x_1, x_2 and x_1, x'_2 with their first block x_1 fixed and $x_2 \neq x'_2$. As in the above case, assume $|x_1, x_2| = |x_1, x'_2|$. Then $Z_1 \oplus Z_2$ is given by $F_K(x_1) \oplus F_{F_K(x_1)}(x_2) \oplus F_{F_K(x_1,x_2)}(x_{pad1}) \oplus F_K(x_1) \oplus F_{F_K(x_1)}(x'_2) \oplus F_{F_K(x_1,x'_2)}(x_{pad1})$. This results in $F_{F_K(x_1)}(x_2) \oplus F_{F_K(x_1,x_2)}(x_{pad1}) \oplus F_{F_K(x_1)}(x'_2) \oplus F_{F_K(x_1,x'_2)}(x_{pad1})$. Clearly this case has now become equivalent to the above case and can be generalised to any number of blocks.

Applicability of known attacks on O-NMAC

One can ask whether the assumption on the compression function as a PRF can be weakened still further producing a secure O-NMAC. In this section, security analysis of O-NMAC is provided against the known attacks on the MAC schemes based on hash functions in the literature assuming that the construction uses good compression functions possessing all the secure properties required from compression functions.

- **Extension and key-less collision attacks on hash functions**

  It is well-known that by just assuming that the compression function is a secure MAC one cannot guarantee that the iterated construction $H_K(x)$ is also a secure MAC due to extension attacks [13]. In O-NMAC, this attack is prevented by the application of the outer compression function.

  An off-line collision attack on the plain hash function leads to collisions on the secret suffix MAC scheme $H_{IV}(x||K)$ where the secret key K is appended to the message [234]. An attacker first finds collisions for two distinct messages x and x' using IV of the hash function such that $H_{IV}(x) = H_{IV}(x')$. Then the attacker requests the oracle of $H_{IV}(x||K)$, the MAC value of x, and shows this as the MAC value for the message x'. Note that the attacker cannot find offline collisions for O-NMAC as it cannot interact with the legitimate owner of the key. As we will discuss below, though the birthday attacks of finding collisions in hash functions lead to collisions in the iterated MAC schemes, they are not practical for message digest lengths like 224 or 256 bits.

- **Birthday attack**

  The generic birthday attack to find collisions in cryptographic hash functions can be applied to MAC schemes based on iterated functions [14, 234]. These attacks apply to NMAC, HMAC, CBC-MAC and to O-NMAC as well. Note that they are the best known forgery attacks on O-NMAC. The practicality of the birthday attack on O-NMAC depends on the size of the authentication tag which is also the size of the secret key. These attacks are impractical when one uses the compression functions of hash functions such as SHA-224 and SHA-256 [211] as it is infeasible to find collisions using the birthday attack on these compression functions. This shows that the actual assumptions required in the analysis of O-NMAC might even be weaker than the pseudorandomness of the compression function. In addition, birthday attacks on keyless hash functions are more efficient than those on O-NMAC as attacks on the latter require interaction with the owner of the key to produce MAC values on a large number of chosen messages which can not be parallelized as the collision attack on the MD5 compression function [280].

- **Key-recovery attacks**

  The birthday attack which produces a forgery on MAC schemes was extended in the form of "slice by slice trail key recovery" attack to extract the trail key of some versions of the envelope MAC schemes as explained in Section 7.1. This attack is not applicable on the O-NMAC scheme as it does not have the trail secret key. The divide and conquer secret key recovery attack on the envelope scheme, which applies to NMAC and HMAC as disclosed in Section 7.1 serves to show that the use of a single K-bit secret key in O-NMAC does not weaken the function against exhaustive key search.

7.6.3 Comparison with NMAC, HMAC and ACSC Constructions

Table 7.2 shows the comparison of several aspects of O-NMAC with NMAC, HMAC and ACSC[6] constructions. The performance of MAC schemes is compared in terms of number of calls to the compression function f to process N blocks of data.

Table 7.2: Comparison among MAC schemes

MAC Scheme	# f calls	# keys	Reduction	Best Attack
NMAC	N + 2	2	CF: family of FI-PRFs	$\approx 2^{k_1} + 2^{k_2}$
HMAC	N + 4	2	CF: family of FI-PRFs	$\approx 2^{k_1} + 2^{k_2}$
ACSC	N + 1	2	CF: family of FI-PRFs	$\leq 2^{k} + 2^{\delta}$
O-NMAC	N + 2	1	CF: family of FI-PRFs	2^{k}

The main advantage of O-NMAC over NMAC is that it requires management of a single secret key compared to managing two secret keys as in NMAC which does not even provide security over the combined key length against key-recovery [13] due to the divide and conquer key recovery attack [234–236].

Similarly, when one uses the ACSC construction as a MAC, the sender and receiver of the message have to generate the random secret key along with the

[6]Note that the intention behind introducing ACSC construction in [14] was to show that the iterations of the compression function preserve pseudorandomness of the compression function but not to show ACSC as a secure MAC. Here, it is treated as a secure MAC for comparison.

shared key K every time they wish to communicate. In O-NMAC, that equivalent secret value Z of size k bits is generated "within" the function and there is no need for the communicating parties to generate and share any other key.

It should be noted that from the implementation point of view, HMAC requires two extra computations over NMAC which is significant when it is used to authenticate short data streams. It was noted [13] that an implementation can avoid this extra processing by storing the keys K_1 and K_2 as the keys to the NMAC function. In addition, it was noted in the ISO 9797-2 standard [119] where HMAC requires the same number of calls to the compression function as NMAC when both the keys are pre-computed and used as IVs to the compression function. However, when the keys vary frequently, an implementation of HMAC cannot avoid this extra over-head by caching the keys. In such a scenario, the keys $K_1 = f_{IV}(\overline{K} \oplus \text{opad})$ and $K_2 = f_{IV}(\overline{K} \oplus \text{ipad})$ need to be generated first and this is accomplished using the compression functions of hash functions such as MD5 and SHA-1 as pseudorandom functions. This would involve two computations of the compression function followed by keying them as IVs for the NMAC function. The advantage of O-NMAC is that it just requires a single key unlike NMAC which requires two keys. In Appendix G, the implementation issues of O-NMAC are compared with HMAC and NMAC when they use MD5 and SHA-1 as compression functions.

7.7 Conclusion

This chapter gives a formal notion for the security of a MAC function. It provides a survey on the analysis and design of MACs based on Merkle-Damgård based cryptographic hash functions. The important outcome of this chapter is the analysis of NMAC based on weaker assumptions on the underlying hash functions which shows that a one-way and collision resistant external function and weakly collision resistant inner function do not always imply a secure NMAC function. This chapter provides an efficient NMAC variant and compares it with other efficient MAC proposals based on hash functions. Based on the observations on keying hash functions to produce MACs, this chapter proposes a new variant for NMAC by altering the keying position in NMAC. Finally, the notion on the pseudorandomness of the function families is given in this chapter and the proof of security of NMAC and HMAC functions as pseudorandom functions is provided.

Bellare [12] has proved the pseurandomness of NMAC and HMAC functions using *only* the pseudorandomness of the compression function. That is, as long as the compression function defines a family of FI-PRFs, the NMAC and HMAC functions define a family of VI-PRFs. It is well known any VI-PRF scheme works as a secure MAC and the reduction from PRF to MAC is standard [15,16,96,97]. This new result supersedes the results on the proof of security of NMAC and HMAC as PRFs provided in this chapter. The reason for this is that the collision attacks on the MD5 [285, 291] and SHA-1 [287, 290] hash functions which work for any chosen chaining value at random, demonstrate that these hash functions are not weakly collision resistant. Though these attacks do not reflect any actual weaknesses on the NMAC and HMAC functions, the proofs of security of NMAC and HMAC as PRF and MAC based on the weakly collision resistant assumption on the hash function does not provide any proof-based security guarantee. The pseudorandomness of the compression function is a stronger assumption than the weakly collision resistance property of the hash function and so far there are no powerful distinguishing attacks on the compression functions of standard hash functions violating their pseudorandomness properties.

Recently, Kim *et al.* [141] have performed distinguishing and forgery attacks on the HMAC and NMAC functions based on some standard hash functions. Their results show that HMAC-MD4, for example, can be forged using 2^{58} chosen texts and can be distinguished from HMAC with a random function using $2^{121.5}$ chosen texts. Though no forgery attacks on HMAC-MD5 were given in [141], it was shown that it requires $2^{126.1}$ chosen texts to distinguish HMAC-MD5, (using 33-round MD5), from a random function. While HMAC-SHA-1 with SHA-1 reduced to 34 rounds can be forged using 2^{53} chosen texts, HMAC-SHA-1 with SHA-1 reduced to 43 rounds can be distinguished from a random function using $2^{154.9}$ chosen texts. In parallel, Contini and Yin [47] have also shown that the compression functions of MD4, MD5, SHA-0 and reduced SHA-1 are not pseudorandom functions. They have also carried out the key recovery attacks on the HMAC and NMAC functions based on MD4, SHA-0 and reduced SHA-1 functions and NMAC based on the MD5 hash function. These cryptanalytical results are significant for the interpretation of the impact of the security proofs of NMAC and HMAC for some hash functions.

In practice, the currently formulated forgery or distinguishing attack, for example on HMAC based on reduced SHA-1, is of little concern even if it gets closer to a full-round attack. For example, a typical IPsec session never has more than 2^{32} messages and it is common that a good cryptographic practice requires re-keying much more frequently [292]. These results suggest that while HMAC should not use MD4 anymore due to the practicality of the attacks, they do not trigger any panic on using HMAC either with MD5 or SHA-1. It should be noted that these results do not contradict the proofs of the pseudrandomness of NMAC and HMAC given in [12] and those given in this chapter. They do develop our understanding of the security of NMAC and HMAC based on broken hash functions.

Chapter 8

Side-Channel Cryptanalysis of MACs using Hash Functions

A cryptographic primitive can be perceived in two ways [139, 240]:

1. As an abstract mathematical object, typically, a transformation with a secret key as a parameter mapping an input to an output.
2. As a concrete implementation of the mathematical object.

The classical analysis has been directed towards analysing the mathematical object, the results of which apply to any concrete implementation. Differential cryptanalysis [30] and linear cryptanalysis [177] of block ciphers are two good examples of the analytical tools that come under this category. Side-channel analysis has been directed towards analysing the implementation specific characteristics of the primitives to recover the secret keys, the results of which are specific to a given implementation. Hence these attacks are also called implementation attacks [302]. Side-channel attacks can also make use of known classical cryptanalytical techniques. Often, side-channel attacks are more powerful and are of a huge concern as they can be performed quickly and some times can be implemented using readily available hardware with an investment ranging from a few hundred to thousand dollars [9].

It is a well known observation that side-channel attacks on the devices that implement cryptographic algorithms exploit the correlation between side-channel information and internal state of the processing device to extract secret keys [139, 140, 153, 154, 240, 245, 302]. These attacks on the cryptographic applications implemented in the embedded devices exploit the correlation between side-channel information such as power consumption, electro magnetic radiation, execution time etc. and the secret key related internal state of the processing device, to extract the key. Timing attacks [153], simple power analysis (SPA), differential power analysis (DPA) [154] and reverse differential power analysis (RDPA) [221] are some of side-channel attacks. It is not the scope of this document to provide classification of side-channel attacks, interested readers can refer [240, 302].

From Chapter 7, it is clear that the security of MAC schemes based on hash functions depends on how the secret key and the hash function are used in the architecture of the MAC function. This chapter extends this observation showing that the security of MACs based on hash functions against side-channel attacks depends not only on side-channel information but also on the architecture of the MAC function and the underlying compression function designs.

Cryptographic algorithms are embedded in ubiquitous computing devices such as mobile phones and sensor networks to achieve security goals of message integrity, confidentiality and authentication. Side-channel attacks are becoming a serious threat to these information processing applications where the attacker tries to extract the secret keys by correlating physical measurements such as power consumption and computing time with the internal state of the processing application which is related to the secret key(s) [153, 154].

These devices often use the standard HMAC function [13] to verify the integrity and authenticity of the messages communicated between them. The security of HMAC based on hash functions such as SHA-1 deployed in the computing devices against side-channel attacks has been a concern. Okeya [221] has analysed HMAC using 12 provably secure PGV compression functions based on block ciphers [35, 232] against DPA and RDPA cryptanalytical techniques. The results of [221] shows the vulnerability of 11 of these schemes against side-channel attacks even if the underlying block cipher is secure against these attacks. This result shows that HMAC based on the widely deployed hash functions such as MD5 and SHA-1 that use compression function of Davies-Meyer structure are vulnerable against side-channel attacks as one of the secret keys of NMAC and HMAC can be recovered using these attacks.

This chapter analyses a variant of NMAC called M-NMAC proposed in Chapter 7 with 12 provably secure PGV compression functions against side-channel attacks. This analysis shows that complete secret key can be recovered using DPA for M-NMAC with eight PGV compression functions under the assumption that the underlying block cipher is ideal. *Only* part of the secret key can be extracted using RDPA for M-NMAC with the Matyas-Meyer-Oseas scheme. While M-NMAC with the Davies-Meyer scheme is secure against both DPA and RDPA, the security of M-NMAC with the two other PGV schemes depends on the order of the inner XOR operations used in those schemes. These results also apply to a variant of the envelope MAC scheme proposed by Preneel and van Oorschot [235] due to its similarity in design with M-NMAC.

This chapter is organised as follows: In Section 8.1, the side-channel attacks on the NMAC and HMAC functions are reviewed. In Section 8.2, side-channel analysis on M-NMAC is performed and the analytical results are compared with the results on NMAC and HMAC. The chapter concludes in Section 8.3 with some remarks.

8.1 Side-channel Attacks on NMAC and HMAC

Here side-channel attacks on the HMAC function with 12 PGV schemes using DPA and RDPA attacks are reviewed [221]. For 12 PGV compression functions refer to Appendix H.

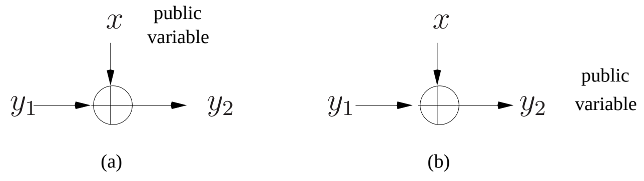


Figure 8.1: The DPA and Reverse DPA models

8.1.1 DPA and Reverse DPA Settings

By employing DPA, the attacker observes the power consumption of the computing device as side-channel information and uses the statistical tools such as the average of the power consumption to eliminate the noise in order to extract the secret key [154]. The application of DPA and RDPA are considered on the target XOR operation in MACs as shown in Figure 8.1 which is also valid for other operations such as modular addition [221, 222]. The XOR operation in Figure 8.1(a) (resp. Figure 8.1(b)) has two inputs: y_1, a constant secret and x, a public variable (resp. a secret), and one output: y_2, whose value is unknown to the attacker (resp. a public variable).

The task of the attacker in this setting for DPA (resp. RDPA) is to extract the secret y_1 by performing the following steps:

1. The attacker guesses some bit of y_1 and classifies the input x (resp. output y_2) into two groups depending on the target bit of y_2 (resp. x) which is 1 or 0 according to the variable input x (resp. y_2).

2. The attacker then observes the power consumed by the XOR operation due to several inputs of x (resp. outputs of y_2).

3. The attacker then computes the average power consumption for each group. Then, the difference between two averages is computed.

4. If positive spikes appear in the difference, the attacker assumes that the original guess for the bit of y_1 was correct. Otherwise, a bit inversion is performed.

Under the Hamming weight model [188], power consumption is correlated with the Hamming weight of the manipulated data. Thus, positive spikes imply that the target bit of y_2 is manipulated as expected, and the averaging eliminates the impact of the other bits on the value of y_2 since they behave randomly.

Note that, by observing sufficient counts of the power consumptions, reclassifying y_2 and computing average power consumption, the attacker does not have to re-observe them in order to find other target bits. See [221] for the experimental results of these attacks.

8.1.2 Side-channel Attacks on NMAC and HMAC with 12 PGV Schemes

Okeya [221] has analysed HMAC with 12 provably secure PGV compression functions against the DPA and RDPA assuming that the block cipher is ideal.This analysis is straight forward on NMAC due to the relationship between NMAC and HMAC as shown in Chapter 7.

The following setting was considered in the analysis of NMAC and HMAC: In the setting of NMAC and HMAC, x_1 is public and variable and H_0 is secret and fixed for the attacker. If the compression function contains the XOR operation as $x_1 \oplus H_0$, the attacker would directly apply the DPA attack on this operation to extract the secret H_0 following the steps given in Section 8.1.1.

Out of the 12 PGV compression functions, eight compression functions f_i for $i \in \{2, 4, 6, 8, 9, 10, 11, 12\}$ contain the operation $x_1 \oplus H_0$ and hence are vulnerable against the DPA attack. It is clear that RDPA works on the XOR operation in the compression function f_5, namely the Davies-Meyer scheme. The application of DPA/RDPA on the compression functions f_3 and f_7 depends on the order of the XOR operation used in these functions. The paper [221] refined f_3 (resp. f_7) as $H_i = (G_{H_{i-1}}(x_i) \oplus H_{i-1}) \oplus x_i$ (resp. $H_i = (G_{x_i}(H_{i-1}) \oplus H_{i-1}) \oplus x_i$) where G is the block algorithm.

8.2 Side-channel Attacks on M-NMAC

The M-NMAC function was proposed in Section 7.4 of Chapter 7. It uses the trail key K_1 as a block instead of as a state (which is done in NMAC) to the external function f. The pictorial representation of M-NMAC is shown in Figure 7.4 in Chapter 7 and is defined below:

$$\text{M-NMAC}_K(x) = f_{H_{K_2}(x)}(\overline{K_1}). \tag{8.1}$$

In the following subsections, M-NMAC with 12 PGV compression functions is analysed with respect to side-channel attacks and is compared with the analysis on NMAC and HMAC.

8.2.1 Target Compression Functions in M-NMAC

The execution of the compression function except the final function in M-NMAC at any state i is written as $H_i = f(H_{i-1}, x_i)$. The execution of the final compression function $f(H_n, \overline{K_1})$ gives the M-NMAC output. In M-NMAC, the keys K_1 and K_2 are secret. The message x split into blocks $x_1, x_2, \ldots, x_n$ and is known to the attacker as it is public. At any state i, in the execution of M-NMAC using the compression function $f(H_{i-1}, x_i)$, the value H_{i-1} is secret and fixed and x_i is public and changeable whenever $x_1, \ldots, x_{i-1}$ are fixed.

The task of the attacker is to extract H_{i-1} using the DPA technique with the power consumption relating to the output H_i. The other target compression function is the final function $f(H_n, \overline{K_1})$ which gives M-NMAC output. Here K_1 and H_n are secrets and the output of M-NMAC is public but is not under the control of the attacker. The task of the attacker is to extract K_1 using the RDPA and the attacker cannot control the output of M-NMAC and does not care about the other input H_n.

8.2.2 Key Recovery Attacks on M-NMAC with 12 PGV Schemes

In this section, the M-NMAC function based on 12 PGV schemes is analysed against the DPA and RDPA attacks to see which compression functions are vulnerable with M-NMAC.

M-NMAC with f_i where $i \in \{2, 4, 6, 8, 9, 10, 11, 12\}$. For M-NMAC with these compression functions, the attacker extracts the secret key K_2 by applying the DPA technique discussed in Section 8.1.1 by choosing a message block x_1 as a public changeable value. After extracting K_2, the attacker can compute $z = H_{K_2}(x)$ for any message x. If the external function f is also one of the eight compression functions f_i for $i \in \{2, 4, 6, 8, 9, 10, 11, 12\}$, it has the XOR operation $\overline{K_1} \oplus z$ with public changeable z and the constant secret $\overline{K_1}$. Applying the DPA model, the secret key k_1 is retrieved.

Now the applicability of RDPA on the final compression function in M-NMAC will be seen. The functions f_i with $i \in \{1, 4, 9, 12\}$ directly XOR the key K_1 with the value obtained after processing the chaining value H_n with the block cipher G. In this setting, the key K_1 is secret which the attacker aims to recover, the value obtained as the output of the block cipher is also a secret which the attacker does not care about and the XOR of these two values gives the M-NMAC output. For this setting, the attacker applies the RDPA attack described in Section 8.1.1 to extract the secret K_1.

The applicability of RDPA on the compression functions f_3 and f_7 depends on the order of the execution of the XOR operation. The following three implementations are considered in these two compression functions:

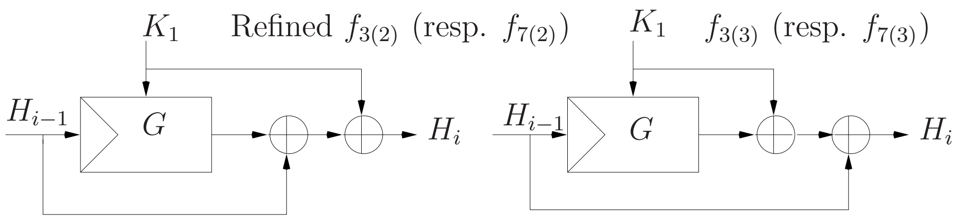

K_1
Refined $f_{3(2)}$ (resp. $f_{7(2)}$)
H_{i-1}
G
H_i
K_1
$f_{3(3)}$ (resp. $f_{7(3)}$)
H_{i-1}
G
H_i

Figure 8.2: Refined $f_{3(2)}$ (resp. $f_{7(2)}$) from [221] and $f_{3(3)}$ (resp. $f_{7(3)}$)

$f_{3(1)}$ (resp. $f_{7(1)}$): $H_i = (K_1 \oplus H_{i-1}) \oplus G(\cdot)$
$f_{3(2)}$ (resp. $f_{7(2)}$): $H_i = (G(\cdot) \oplus H_{i-1}) \oplus K_1$
$f_{3(3)}$ (resp. $f_{7(3)}$): $H_i = (G(\cdot) \oplus K_1) \oplus H_{i-1}$

Here $G(\cdot)$ stands for $G_{H_{i-1}}(K_1)$ (resp. $G_{K_1}(H_{i-1})$). Note that these three implementations are the same compression function in view of their input/output.

The first case $f_{3(1)}$ (resp. $f_{7(1)}$) is vulnerable against the DPA attack due to the operation $K_1 \oplus H_{i-1}$. The second case $f_{3(2)}$ (resp. $f_{7(2)}$) is vulnerable against the RDPA attack due to the operation $H_i = y_1 \oplus K_1$ where the attacker extracts the key K_1, and y_1 denotes $G(\cdot) \oplus H_{i-1}$. Note that $f_{3(2)}$ (resp. $f_{7(2)}$) is recommended for use with NMAC and HMAC in the paper [221]. Finally, neither DPA nor RDPA applies to the third case $f_{3(3)}$ (resp. $f_{7(3)}$) which are described in Figure 8.2. Hence, the proposed compression functions $f_{3(3)}$ and $f_{7(3)}$ are immune to the DPA and RDPA.

From Section 8.1.2, it is clear that secret key K_2 for the Davies-Meyer compression function f_5 in the HMAC and NMAC functions cannot be recovered using the DPA attacks. This is also applicable to the M-NMAC as the positioning of the key K_2 is the same in all these MAC schemes. The RDPA does not apply to recover the key $\overline{K_1}$ for the Davies-Meyer compression function f_5 in the M-NMAC setting due to its usage in a separate block. Hence M-NMAC instantiated with hash functions MD5 and SHA-1 is secure against the known techniques of DPA and RDPA attacks. This shows the significance of the architectural difference of M-NMAC compared to NMAC and HMAC in using the trail secret key as a block rather than as a state. Finally, this analysis of M-NMAC is also applicable to the variant of the envelope MAC scheme [235] which uses the trail secret key in a separate block as M-NMAC. While M-NMAC uses the other key as an IV, the variant of the envelope MAC scheme uses the other key as a data block.

8.2.3 Extending Side-channel Analysis on M-NMAC to Forgery Attacks

Table 8.1 shows the results of side-channel analysis on M-NMAC based on 12 PGV compression functions f_i with $i = 1, \ldots, 12$. Note that N/A in Table 8.1 denotes "Not applicable". The results show that for M-NMAC based on f_i with $i \in \{2, 4, 6, 8, 9, 10, 11, 12\}$, side-channel attacks recover both the keys. Hence the attacker can perform selective forgery on the M-NMAC scheme based on these compression functions for a message of the choice. The attacker can *only* recover the trail key K_1 of M-NMAC based on f_1 and refined $f_{3(2)}$ (resp. $f_{7(2)}$) [221]. In this case, the security of M-NMAC against forgery attacks reduces to the secret prefix MAC scheme where the attacker forges the M-NMAC scheme using the length extension attack exploiting the Merkle-Damgård iterative structure used in M-NMAC.

The following is the attack algorithm:

1. The attacker queries the M-NMAC oracle for an authentication tag of the message x. The tag is $\mathrm{M-NMAC}_k(x) = f_{H_{K_2}(x)}(\overline{K_1})$.

2. Then the attacker uses the key K_1 to perform existential forgery by computing $f_{\mathrm{M-NMAC}(x)}(\overline{K_1})$ which is the tag for the new forged message $x||\overline{K_1}$.

Of course, if the attacker knows *only* key K_2 without the knowledge of the key K_1, the attacker would forge the scheme by performing length extension attack. Finally, the attacker cannot forge M-NMAC based on f_5 and the proposed $f_{3(3)}$ (resp. $f_{7(3)}$) compression functions using DPA or RDPA as no keys can be recovered using these attacks.

8.2.4 Comparison with NMAC and HMAC

The attacker can perform a selective forgery attack on NMAC and HMAC based on eight compression functions f_i with $i \in \{2, 4, 6, 8, 9, 10, 11, 12\}$ by recovering both the keys using side-channel attacks. In the case of M-NMAC, these compression functions differ as given in Section 8.2.2. Following Chapter 7, for the NMAC function based on f_5 where the attacker recovers the secret key K_1, NMAC does not always result in a secure MAC even if the function f_{K_1} is one-way and collision resistant and the function H_{K_2} is a weakly collision resistant hash function.

Table 8.1: Results of side-channel attacks on M-NMAC

Compression function	Secret key K_2	Secret key K_1
f_1 (Matyas-Meyer-Oseas)	N/A	RDPA
f_2	DPA	DPA
$f_{3(1)}$	DPA	DPA
$f_{3(2)}$ [221]	N/A	RDPA
$f_{3(3)}$	**N/A**	**N/A**
f_4	DPA	DPA/RDPA
f_5 (Davies-Meyer)	**N/A**	**N/A**
f_6	DPA	DPA
$f_{7(1)}$	DPA	DPA
$f_{7(2)}$ [221]	N/A	RDPA
$f_{7(3)}$	**N/A**	**N/A**
f_8	DPA	DPA
f_9	DPA	DPA/RDPA
f_{10}	DPA	DPA
f_{11}	DPA	DPA
f_{12}	DPA	DPA/RDPA

For M-NMAC based on the function f_1 where the attacker recovers the trail key K_1, the MAC scheme can be forged using length extension attacks. See Table 8.2 for the secure compression functions in these MAC settings against side-channel attacks.

Table 8.2: Secure Compression Functions for NMAC/HMAC and M-NMAC

NMAC/HMAC	M-NMAC
f_1 (Matyas-Meyer-Oseas)	-
$f_{3(2)}$ [221]	$f_{3(3)}$
-	f_5 (Davies-Meyer)
$f_{7(2)}$ [221]	$f_{7(3)}$

8.3 Conclusion

This chapter shows that the MAC schemes M-NMAC and a variant of the envelope MAC scheme with the Davies-Meyer compression function resist side-channel attacks that work on NMAC and HMAC schemes with the Davies-Meyer scheme.

In the light of improvements in the cryptanalysis of SHA-1, NIST has planned to phase it out by 2010 due to the advances in computing power in favour of secure hash functions SHA-224, SHA-256, SHA-384 and SHA-512 [213, 214]. All these stronger hash functions like scrutinized hash functions MD5 and SHA-1 use the Davies-Meyer compression function. When a MAC scheme is designed using a hash function, the security of the MAC scheme should not be compromised with out any flaws in the underlying hash function. That is, one should not use good grapes to make a bad wine. The counter argument is some times a wine which has been good made with bad grapes might turn out bad as the time advances. This makes sense and in the context of HMAC based on MD5 and SHA-1, refer to some of the latest cryptanalytical results [47, 141]. Now, HMAC based on stronger hash functions such as SHA-256 and SHA-512, must be strong from all types of attacks.

However, HMAC has been attacked using side-channel analysis exploiting the Davies-Meyer compression function independent of the strength of the hash function. Since, it is likely that hash functions from the SHA-256 family might replace SHA-1 in the future protocol implementations, it is of at most importance that such schemes offer security against various attacks including side-channel attacks. In this context, it is strongly recommended to use the M-NMAC function with hash functions such as SHA-256 for protection against side-channel attacks. That is, transition must happen both at the hash function and MAC levels supporting a similar observation made in the context of digital signatures based on hash functions [19] in the light of collision attacks on MD5 and SHA-1 [172, 287, 290, 291] hash functions.

Finally note that, though this chapter has not performed side-channel analysis of the other well known MAC scheme called MDx-MAC [234] mentioned in Section 7.1 of Chapter 7, it seems that it also offers protection against the DPA and RDPA attacks. The MDx-MAC scheme uses three keys, the first key replaces the IV of the hash function, the second key is exclusive-ored with some constants and is appended to the message and a third key influences the internal rounds of the compression function. These three keys are derived from a single master key. Unlike HMAC, MDx-MAC does not use a hash function as a black-box and requires changes to the hash functions from the MDx family. Note that by treating compression functions as a black-box, this scheme looks very similar to M-NMAC and a variant of the envelope MAC schemes except that it XORs some constants to the trail secret key. In this sense, the results of M-NMAC and the variant of the envelope MAC scheme with the Davies-Meyer scheme based hash functions seems straight forward on the MDx-MAC. Further analysis of MDx-MAC against side-channel attacks is considered as part of the future work.

Chapter 9

Collision Attacks on MD5 and SHA-1: Is this the "Sword of Damocles" for Electronic Commerce?

By the early 1990's, the National Science Foundation in the USA permitted the Internet to be open to commercial entities for non-research activities to transact business and thus was born the modern electronic commerce environment [167]. This has resulted in a massive uptake by commercial entities in exploiting this new environment. Traditional boundaries have been dissipated and from a legal perspective new and varied conundrums have confronted legislators and regulatory authorities in managing the various policy/legal issues that have arisen out of this new frontier. The electronic commerce environment is like the hydra which has many heads that need to be addressed. One of these heads involves the so called trusted technologies which can be used to effect non-face-to-face transactions. A fundamental underlying technology for non-face-to-face transactions is the use of cryptographic hash functions.

As noted in chapter 1, hash functions serve an essential role within a wide range of information security applications. These applications provide certain security services such as data integrity, authentication and to a partial extent non-repudiation [179]. These applications also include non-exhaustively, (a) digital signature generation and verification, (b) session key establishment in key agreement protocols, (c) management of password schemes and (d) commitment schemes in cryptographic protocols such as electronic auctions and electronic voting.

In many applications, it is significantly important from a legal perspective that a hash function should be collision resistant. The effectiveness of the collision resistance of the hash function can in some applications support or undermine the legal position of the application that relies upon the hash function technology. As will be discussed later in this chapter, this is especially so where authentication is a fundamental element underpinning the legal validity of the transaction. MD5 and SHA-1 are the widely deployed functions in applications where collision resistance is demanded.

The collision attacks on MD5 [144, 145, 172, 264, 285, 291] show that it should no longer be used in any application as a collision resistant hash function. The collision attacks on the hash function SHA-1 [27, 287, 290] call into question the long-term future use of SHA-1 as a collision resistant function. This chapter focuses on the legal implications of the collisions associated with MD5 and SHA-1 because this issue has not been researched since these attacks and further there are some fundamental implications that could substantially undermine the general layman's trust [93] that currently exists in the electronic commerce environment. For the practical implication of hash function collision attacks on various protocols see [19, 112]. The practical implications of collision attacks on MD5 and SHA-1 will be indicated wherever it is essential in this chapter.

9.1 Practical and Legal Implication of Collision Attacks on MD5 and SHA-1

This section focuses on the practical and legal impact of collision attacks on MD5 and SHA-1 as they are embodied in certain important applications where either of these functions are widely deployed. The applications include digital signatures on messages and digital certificates, software protection and message authentication codes based on hash functions. Note that this legal analysis can be generalized to any widely used hash function which is vulnerable to collision attacks.

9.1.1 Digital Signatures on Messages

Theoretically, digital signatures are used to authenticate the signers of electronic messages. In fact, digital signatures are based upon the substantial underlying assumption that the private key has not been compromised. Digital signatures do not actually establish who affixed the digital signature. They only establish that a particular private key was used to affix the signature. Digital signatures substantially depend upon the document that is being digitally signed, as well as the private key and the algorithms used to affix the digital signature. If the document changes in any way then the digital signature will also substantially change. Consequently, a digital signature is the result of applying the digital signature algorithm to the hash of the document that is being digitally signed, using a private key allegedly held by the person causing the digital signature to be created. Anyone, with the public key that corresponds with the private key used to affix the digital signature, should be able to verify the signature of the signer. The basic premise behind Public Key Technology (PKT) [65] and its ability to provide authentication and restricted non-repudiation services is that the private key remains private and has not been compromised by its disclosure to third parties.

Consider the following scenario between two parties Alice and Bob, wherein Alice wishes to send a digitally signed message to Bob. Assume that the parties use the MD5 hash function as part of the signing technology.

The following steps would be performed by Alice.

1. Alice hashes the message x that she wishes to send to Bob using MD5. MD5(x) is the value of the message digest. Let MD5(x) be h.

2. Alice then transforms h using her private key $kpriv_A$ using some public key algorithm such as RSA to compute the digital signature sig_A. This is expressed as follows.

   $sig_A(x) = E_{RSA}(h, kpriv_A)$

3. Alice sends message x and the signature on x, sig_A, to Bob. Further, Alice informs Bob as to what algorithms were used to generate sig_A[1].

[1]This is actually achieved through an X 509 V. 3 certificate.

The following steps would be performed by Bob after receiving x and sig_A to verify the digital signature of Alice.

1. Bob hashes the message x that he received from Alice using MD5. MD5(x) is the value of the message digest. Let MD5(x) be h'. Note that this is the same initial step performed by Alice above.

2. Bob verifies the signature sig_A using the public key of Alice $kpub_A$ to get the message digest h''.

   This is expressed as follows.

   $h'' = D_{RSA}(sig_A, kpub_A)$

3. Bob compares the digests (h' and h'') obtained in steps 1 and 2 and if they are equal then the integrity of the message is established and provided the private key remains private, the person attributed as signing the message is authenticated.

Considering the feasible nature of finding many collisions on MD5 [144, 145, 291], it is now possible for both the signer and the verifier of a digitally signed message that relies upon the MD5 hash algorithm to cheat each other and thus obviate the non-repudiation property that is widely regarded as being an essential property of digital signature technology. This will be shown in the following two scenarios.

Scenario 1: The collision in hash function can easily be used by Alice to cheat Bob. Let the innocuous message x, that Alice wants to sign, be something like:

> I, Alice, am selling my property of Blackacre comprising 5 acres to
> Bob for a price of AUD $ 123, 456.

Assume that Alice can come up with an alternate message x' that gives the same digest as x. Let x' be:

> I, Alice, am selling my property of Blackacre comprising 2 acres to
> Bob for a price of AUD $ 223, 456.

Note the decrease in the area of Blackacre that is being offered and the increase in the associate price. Let it also be assumed that these two messages have the characteristic that MD5(x) = MD5(x')[2]. Alice then sends x and the signature sig_A computed on MD5(x) - (noting that it would be the same on MD5(x')) to Bob. Bob believes that the message x is the legitimate message but later Alice claims x' as the legitimate message by producing the same signature sig_A on x'. Furthermore, Alice has destroyed all evidence regarding the digital signing of x. Now Alice may have an opportunity to sell Blackacre post signing with Bob to another party Trent and she decides that she needs to get out of the first contract of sale with Bob so that she can enter into the more lucrative contract with Trent. Hence, she goes to the elaborate exercise of creating the later message x' that has the same message digest as x.

From an evidentiary perspective this raises some serious doubts as to the probity of the two messages and in particular the fact that the same digital signature can be verified for both messages. At common law the onus of proof is generally placed upon the plaintiff. That is, it is the plaintiff who has the onus of proving his/her case. In this case, if Alice raises a dispute over the contents of the electronic communications that have transpired between them, she will accordingly inform Bob of the problem. If Bob commences proceedings to prove his case in order to obtain specific performance of the contract, he will have the onus of proving the value of the contract; but in doing so he will encounter the defense that Alice has evidence that the contract was different to that which Bob claims. The difficulty for the arbiter of fact, namely the judge in this case, is that the judge will need to decide which message is the correct reflection of the contract. Other evidence will need to be presented by Bob, which could cause substantial expense being placed upon Bob in proving his case. Remembering that Bob, does not have access to Alice's private key; so Bob cannot generate a new digital signature for x'. Alice would argue that she does not know how this occurred. She can also show that her public key can verify the digital signature attached to x' that is in her possession.

Alice could argue that x' is the original message and that Bob has somehow developed x such that MD5(x) = MD5(x'). Further, she may claim that Bob has stripped the original signature from x' and attached it to x [180]. Bob may have other evidence such as time of receipt of x that has the digital signature of x, but this kind of evidence can be easily spoofed or altered without leaving

[2]It must be noted that the collision attack on MD5 does not actually results in a collision for these two messages.

a trace by the recipient of an electronic communication. Such a case could be decided solely upon the oral evidence of the parties and not upon the technology that underpins the case. Of course, a court in this case could rightly make some substantial disparaging remarks about the technology and its lack of the so called non-repudiation and authentication properties.

Scenario 2: The collision attack on MD5 can also be used by Bob to cheat Alice. This is possible when Alice signs the messages as directed by Bob. Let x and x' be the two messages on which Bob gets a collision before getting into contract with Alice. Let x and x' be respectively as follows:

> I, Alice, am selling my property of Blackacre comprising 5 acres to Bob for a price of AUD $\$223,456$.

> I, Alice, am selling my property of Blackacre comprising 10 acres to Bob for a price of AUD $\$123,456$.

Once Alice signs the message x at the will of Bob, Bob strips the signature of the message x and attaches it to the message x'. Bob, later claims that the message signed by Alice is x' not x.

In this case, if Bob raises a dispute over the contents of the electronic communications that have transpired between them, he will accordingly inform Alice of the problem. If Alice commences proceedings to prove her case, she will have the onus of proving the value of the contract; but in doing so she will encounter the defense that Bob has evidence that the contract was different to that which Alice claims. Further, Bob does not have access to the private key of Alice as it is known only to her and hence the signature on x' could only have been signed by Alice. That is, since Bob is able to verify the digital signature affixed to x' using Alice's public key and Bob has no access to Alice's private key then Bob will argue that Alice is trying to defraud him of the contract value.

Bob may be able to successfully force Alice to admit that her private key has not been compromised, which means that theoretically she is the only person who had access to the relevant private key and was the only person capable of activating its use to digitally sign x'. Since Bob had no access to Alice's private key, and it is Alice's public key that is used to verify the digital signature affixed to the electronic communication that is being relied upon by Bob, it will be very difficult for Alice to discredit this evidence in the court proceedings, even though the original message x has been substituted by Bob for x'.

The collision for both x and x' is a substantial flaw in the digital signature technology, where the hash algorithm used such as MD5 is not collision resistant. Such a flaw will completely undermine the concept of non-repudiation regarding forged digital signatures. The concept of non-repudiation has been a principal attribute promoted by a number of digital signature technology providers.

In Scenario 2, a court, at a minimum could come to the conclusion that there is some uncertainty regarding the validity of the two messages x and x' and at a maximum the electronic communication in the possession of Alice that has Alice's digital signature affixed, has been altered by Alice, which could give rise to an allegation of giving false evidence under oath or to an allegation of tampering with evidence.

In general, it can be seen that attacks on hash functions with meaningful collisions greatly undermine the evidential value of digital signature technology where such hash functions are used for digital signature purposes. The collision attacks on MD5 can be used to construct ASCII message sequences which was demonstrated in [56].

There are no legal cases where digital signatures have been specifically disputed, though it is generally accepted that a digital signature will not be non-repudiatable because there are many legal reasons where a party may be able to successfully repudiate a digital signature attributed to them, such as unconscionable conduct or undue influence or duress [179]. What has been generally taken as the base position is that digital signature technology will greatly reduce the incidence of forgeries, but since the successful attacks by Wang *et al.* even the issue of forgeries has now become an issue, which undermines the concept of non-repudiation even further. When the underlying hash function technology is weak, it could result in the compromise of the non-repudiation security property.

A further effect of the undermining of the non-repudiation property is the long term archiving of digitally signed documents. It is not unusual for a dispute to take a substantial amount of time to elapse before it will be heard by a judge. During this intervening time both parties have to ensure that the evidence they have in their possession does not become tainted and maintains its integrity. In Australia, section 11(3) of the Electronic Transactions Act 1999 (Cth) [ETA] [8] provides that an electronic document can be endorsed by a third party for the purposes of integrity. The Uniform Electronic Transactions Act [UETA] [207] that has been adopted by many US States does not have a similar provision.

If the PKT used to affix a digital signature is undermined due to some technological advancement, it is not correct to get the parties to resign the document as this would alter the document in a substantial manner due to the effluxion of time. It may also be impractical for this to occur as one of the parties may not be available or may have even died in the mean time. The better approach is to get a trusted third party to endorse the document by affixing another digital signature which uses a more robust PKT. This approach is akin, though not completely analogous, to what a notary will undertake when endorsing a document. It is not uncommon for a notary to endorse a document by affixing the notary's seal to an original document. The affixation of the notary's signature and seal does not cause the notary to be bound by the contents of the document as a signatory. Instead, the notary is endorsing the document for some special purpose which must fit within the notary's powers of authenticating documents presented. Therefore, even though the notary may sign a document, this process does not make the notary bound to the document in the same way as the signatory is bound to the contents of the document. This position also arises when a witness affixes his/her signature to a document. The difference between the paper based environment and the digital environment is that in the paper based environment it is very difficult to alter a signature or any other aspects of the document without some visible trace after the signature has been affixed to a document, especially if the signature has been affixed using some indelible ink as opposed to some pencil markings.

An important aspect of the digital endorsement mechanism is that it can be undertaken multiple times. The original document with its original digital signature or signatures is maintained and thus the integrity of the document is preserved. From an evidential perspective, the long term preservation of the integrity of the document, which must include the digital signatures embodied or associated with the digital document is paramount. If there are any questions that adversely call into question the integrity of the digital document then a Court having jurisdiction could decide to not admit the document into evidence or if admitted into evidence, gives the document such a low weighting due to its unreliability that its evidential value is useless for the presenting party. The trusted third party will need to be noted as an endorser so as not to be confused as an original signatory. When the case is finally heard by a Judge, each endorsement of a digital signature will be verified until such time as the original signatures can be verified. For long term archiving purposes this approach is recommended. The UETA that has been adopted by a substantial number of US States does not specifically recognise the endorsement mechanism as provided in the Australian Electronic Transactions Act. However, UETA does make provision for security procedures which are to be used to maintain the integrity of digitally signed documents and therefore the procedure could be similar as noted above for endorsement purposes.

The issue of obviating the non-repudiation property has an even more catastrophic affect when digital certificates are the documents that are being digitally signed. One of the difficulties in using PKT is the deployment of the corresponding public keys that are used to verify digitally signed documents. It should be remembered that it is the public key that corresponds to the private key that is used to verify the digital signature. To distribute public keys, digital certificates were proposed [155] and developed, which are currently based upon the ISO standard X.509 v. 3. [114].

9.1.2 Digital Signatures on Digital Certificates

A digital certificate is a structure used to convey information about a user for identification purposes [92]. It is issued by a certification authority (CA) and attempts to bind an identity to the public key and ip so facto the possessor of the corresponding private key. An entity in need of a digital certificate presents to the CA its identification details that evidences their identity, the public key and other required information. It contains among other things the name, a unique serial number for the certificate, expiration date, a copy of the certificate holder's public key (the holder should solely have the corresponding private key), the public key algorithm and the hash algorithm used and the digital signature of the CA so that a recipient can verify that the certificate is authentic. The most widely used digital certificate standard is X.509 version 3. Hence, the implications of MD5 collisions on X.509 version 3 digital certificate will be discussed in this section.

The X.509 version 3 digital certificate issued to an end user by a CA contains the following fields in order: Version number (v3), Unique serial number, Name of the signature algorithm (for example, MD5 with RSA), the name of the issuer (for example, secure server CA, Verisign CA), Validity period of the certificate, Subject of the certificate which is the end user, Public key of the subject, Optional set of extensions, Signature algorithm and Signature value.

The impact of MD5 collisions on the X.509 version 3 digital certificates depends on the order of the fields.

Case 1: Attack on a new certificate request

The concept of this attack is simple. The attacker prepares in advance a genuine certificate request whose hash value collides with that of a fraudulent certificate (which may have for example, a different name). The CA's signature on the genuine certificate is transferred to the fraudulent one. To generate the colliding request data, the attacker proceeds as follows.

Given any fixed prefix message p and two different desired messages $m1$ and $m2$, create two suffix messages $s1$ and $s2$ that provide a collision. That is concatenating the prefix message p with $m1$ and then concatenating the resultant with $s1$. In like manner, p,$m2$ and $s2$ are also concatenated. This is visualised as $h(p||m1||s1) = h(p||m2||s2)$. In the case of digital certificates, the serial number and validity period are not fixed. Serial numbers are generally harder to predict than the validity period. For example, Verisign the most widely used CA sets the validity period in the certificate based on the current time. This can be predicted to within a few minutes.

A malicious attacker can generate two certificate requests with two different subjects as follows. "Digital certificate request for the website www.centralpark.com and here are my personal details" ($m1$) and "Digital certificate request for the website www.centralbank.com and here are my personal details"($m2$). These two are the desired messages $m1$ and $m2$ of the attacker. Following the above order of field selection in the digital certificate, version number, name of the signature algorithm, name of the issuer, validity period (under the assumption that it can be easily predicted) are the fixed prefix fields. Public key field containing a public key (there would be corresponding private key to each public key) that comes after the subject field forms the suffix.

Once the attacker crafts the field patterns for the different name fields containing two distinct messages in advance, the attacker sends the initial request to get the digital certificate signed by the CA and later inserts the signature on to the fake second certificate thereby making a perfect forgery. Any browser can easily trust the www.centralbank.com's digital certificate as the genuine certificate and masquerades as a bank thus sneaking the personal details of a customer. This attack follows the same principle as the one described in Section 2.1 where the non-repudiation property of security is violated. This type of attack could be used to great effect in the incidence of the email "phishing" scam that has become so prevalent in recent times.

The mathematical equation describing this attack on digital certificate is

$$\mathrm{h(v3||serialnumber1||p||m1||s1) = h(v3||serialnumber2||p||m2||s2)} \tag{9.1}$$

The practical possibility of this attack is still in doubt and further research is required. The reason is that when the CA signs a certificate, CA specifies a unique serial number in the certificate. It depends on how much control the attacker has on the serial number field. X.509 version 3 certificates use a serial number of length 128 bits. The probability of predicting a correct serial number is 2^{-128}. The collision attacks demonstrated on MD5 [144, 145, 172, 264, 285, 291] have 1024-bit freedom for choosing the message patterns. But in this scenario the degree of freedom is really short which is 128 bits.

A more devastating attack would be one in which an attacker can obtain a legitimate server certificate with a collision to the one containing a wild card[3] for the domain name and an expiration date far in the future [247]. It appears that the use of unpredictable serial numbers may prevent such attacks.

Lenstra *et al.* [169,170] have shown a method to construct a pair of valid X.509 certificates in which the "to be signed" parts form a collision when MD5 is used in the X.509 digital certificates. The attack relies on the ability of the attacker to create two different public keys that would cause the body of the certificate to have collisions when hashes with MD5. For the real possibility of this attack, the attacker has to predict the contents and structure of the certificate before it is issued, including the identity, the serial number and the start and finish dates of the validity period of the certificate. This attack demonstrates that a relying party using a public key certificate based on MD5 can not be certain that the alleged owner actually possesses the corresponding private key. Since the identity in the two certificates is the same, there are probably no real-world scenarios where this kind of attack would get the attacker any advantage.

It was suggested in [112] that if a trusted third party who issues the digital certificates wants to avoid the above kind of attack, they can prevent the attack by making other signed parts of the certifcate random enough (for example, by

[3]A wild card is a character which can be substituted for another character.

adding a randomly chosen component to the identity) to eliminate any advantage gained the attack.

Daum and Lucks [56] have constructed a pair of postscript documents that look different but hash to the same digest using the random collisions in MD5 [291] and exploiting the iterative structure of the Merkle/Damgård construction. The Postscript documents permit this attack to work by binding two different documents in the same file revealing only one document for signing purposes and at the same time hiding the other. In a practical sense, this attack would allow an intern at a company to create a file where one document is simply a reference that the employer would digitally sign while the other hidden document could contain unauthorized permissions such as the viewing of the employer's private data [22]. Considering the practical significance of these attacks, it is not recommended that MD5 continue to be used when collision-resistance property of MD5 is required by the application.

The practical impact of the collision attacks on SHA-1 [287] on many existing applications is still limited as any successful attack on SHA-1 based on the new result would need a huge amount of computer processing [243]. Hence, the latest attack on SHA-1 is unlikely to have any significant impact on current digital signatures, though it remains still possible that the results could be improved in future.

It is expected that CAs will move away from using MD5 for signing new digital certificates due to the threat of the described attack.

Case 2: Attack on an existing certificate

At this point of time, the attacks on MD5 and SHA-1 cannot be used to tamper with existing certificates, for example Secure Socket Layer (SSL) webserver certificates, as it needs an attack violating the 2nd pre-image resistance property of these functions. This is important, as all Banking sites utilise the SSL protocol. If there is developed a feasible attack that could tamper with existing SSL server certificates then the trust currently reposed in banking sites would be substantially undermined and this could be a catastrophic effect upon the economy.

So far, the best known attack to violate this property is the brute-force attack which requires 2^{128} hash computations for the MD5 algorithm and 2^{160} hash computations for the SHA-1 algorithm respectively. Obviously, performing these tasks is computationally infeasible. In this case, the attacker cannot find a collision to any arbitrary existing digital certificate. The current attacks on MD5 do not violate the 2$^{\text{nd}}$ pre-image resistance property [105].

Bellovin and Rescorla [19] have observed that the collision attacks on MD5 and SHA-1 cannot be translated into demonstrable attacks on real-world certificate-based protocols such as S/MIME, TLS and IPsec. Their results show that the transition to stronger hash functions is necessary, though not immediate. Moreover, their analysis reveals that major internet protocols such as S/MIME and TLS were not designed properly for such a transition. In particular, if the signature algorithm is linked to a particular hash function, as Digital Signature Algorithm (DSA) is tied to SHA-1, both the signature algorithm and the hash algorithm need to be changed. Hoffman and Schneier [112] also in an independent work have identified that most protocols use hash functions in a way that makes them immune to harm from collision attacks.

As stated, Lenstra *et al.* [169, 170] have shown that a relying party will not have sufficient evidence to satisfy itself as to the named subscriber in the X.509 v.3 certificate as actually possessing the private key corresponding to the public key noted in the certificate. The effect of the collision attacks on MD5 substantially undermine the value of any certificate where MD5 hash algorithm has been used. From a legal perspective, these attacks bring into the question the commercial value of PKI and the technical/legal value of any certificates that utilize the MD5 hash function. So far no practical attacks on the certificates based on SHA-1 have been reported and hence the legal status of certificates based on SHA-1 is maintained.

9.1.3 Message Authentication

Hash functions are used in constructing message authentication codes (MACs) as in the construction of HMAC [13]. A MAC function is said to be forged when an attacker finds the MAC value for a previously unseen message without knowing the secret key, which is one of the inputs to the MAC function. HMAC construction is provably secure under certain assumptions on the security of the underlying hash function. An attacker that tries to forge the MAC function HMAC would also be able to break the underlying hash function in one of the following two ways:

1. The attacker would be able to find collisions to the underlying hash function

keyed through the IV of the hash function.

2. The attacker would be able to find an output of the external hash function which is keyed with a random and secret initial value.

The Transport Layer Security (TLS) protocol which is defined as a proposed Internet standard version for SSL version 3 in [64] uses HMAC algorithm defined in [13, 159] as a MAC function and also as a Pseudo Random Function (PRF). HMACs are protected by a shared secret key, and PRFs do not require collision resistance as their security property [98]. The HMAC function used in the TLS protocol is instantiated either with MD5 or SHA-1 hash functions.

The collision attacks on MD5 and SHA-1 would not enable an attacker either to find collisions starting from random secret initial values or to produce known outputs from random secret initial values. Hence, the ability to find collisions in the hash functions MD5 and SHA-1 do not lead to forgery attacks on the HMAC function, and thus the legal status of the TLS structures is maintained. This is highly relevant as TLS is in many cases used as the underlying security technology for Internet banking. We also note that the collision attacks on MD5 and SHA-1 would not enable an attacker to forge the other variants of HMAC [225]. The recent results on the analysis of HMAC based on MD5 and SHA-1 hash function provided in chapter 7 suggests this too.

9.1.4 Software Protection

Software vendors want to protect the integrity of the software they sell to the users [228]. Hash functions are used to check the integrity of the software that the user receives from the vendor and make sure that user would not receive virus or malicious programs that infect the user's PC. A user can get the software or any application from the vendor through the CDs/DVDs, as downloads from the vendor's website or from vendor's mirror website or through some other means. Sometimes users may also download the software or files for free from the Internet and expect them to be free from viruses or malicious content.

There are so many free tools available on the Internet today [48] that use MD5 algorithm for data integrity control and verification of network file transfer, E-mail messages and files or software downloaded from the Internet. For example, the MD5 algorithm is used to check the integrity of a Cisco Internetworking Operating System(IOS) software image [45]. The MD5 file validation feature of the Cisco IOS uses MD5 to create a 128-bit checksum of the Cisco IOS software image on some of the Cisco released products and compares that with the MD5 checksum of the images on those releases posted on the Cisco website. Apache web-server [1], the most popular web-server on the Internet, develops and maintains an open-source Hyper Text Transfer Protocol (HTTP) server for the Unix and Windows NT operating systems. It uses MD5 hashes as one of the options, the other being the Pretty Good Privacy (PGP) signatures, to ensure the verifiability of the downloads from its home page and other mirror sites [2]. It uses appropriate MD5 embedded programs on the Unix and Windows distribution sites to achieve this. The Solaris Fingerprint Database (sfpDB), a free Sun Microsystems security tool, uses MD5 to verify the integrity of the files distributed with the Solaris Operating Environment [275]. The sfpDB ensures that its official binary distributions contain authentic files but not adapted ones that compromise the system security. The sfpDB uses MD5 to compare the digest of the binary distributions with the trusted hashes stored on its homepage and hence identify any mismatches if present.

In the wake of attacks on MD5, there would not be any threat to the above existing applications where MD5 is used as a CRHF to achieve data integrity [181]. Using the current collision attack techniques on MD5, it is impossible for a malicious attacker to generate the same digest on a new software with a back door as already exists for a certified software and then replacing the later with the former [244]. This attack requires violation of 2^{nd} pre-image resistance property of the hash function. It is still unsure whether the attack strategy on MD5 violates this property when there is a larger degree of freedom for the attacker to manipulate message formats as in this example. This case is quite different from the case of attacking existing digital certificates. In digital certificates, the degree of freedom available to the attacker is quite limited unlike here. The analysis on MD5 collisions given by Hawkes *et al.* [105] shows that MD5 is still 2^{nd} pre-image resistant and the attacks by Wang *et al.* [285, 291] would not compromise this property of MD5. Hence we can say that at present the collision attacks do not allow tampering with arbitrary programs available on the Internet.

If a successful attack is developed then a malicious attacker can masquerade as a genuine vendor and develop collisions for the genuine code and malicious code using the attack technique on the MD5 algorithm. The attacker could place the malicious program (for example code with virus) and the corresponding hash code on the Internet. A similar attack is possible where an attacker uses the collision attack technique on MD5 and can enable a trusted compiler/verifier to accept and sign the innocuous program, which could then be substituted for the malicious one [247]. Hence, when the hash function used for the integrity checking is not robust, the end user cannot identify the infected code from the true code. The infected code could contain a deliberate security vulnerability. It is not out of the question for a malicious attacker to implement this attack on software patch mechanisms that are frequently released.

This then raises the duty of care that software manufacturers have to their end users in distributing software that could possibly be altered with out authority by third parties so as to insert a virus/trojan horse or some back-door code that can later be used by such third parties for nefarious activity [3]. Due to the substantial publication of the attacks on MD5 in the press both generally and computer specific it is unlikely that a software manufacturer could successfully argue that it was not aware of these attacks and their ramifications [116]. The legal status of software manufacturers would it is submitted be no different to other manufacturers where such manufacturers have to take reasonable care to ensure that their products do not cause damage to their constituent end users [3]. This duty of care issue has recently arisen in the USA in the Sony-BMG matter [262]. SONY-BMG was identified as having distributed a substantial number of music CDs that contained an alleged technological protection measure (TPM). The TPM was a root kit mechanism that included the property of hiding itself within the kernel. Further, it was not possible to easily remove the root kit from an infected system and if removed could adversely affect the operations of the infected computer system. Finally it has been identified that the SONY-BMG root kit created a security vulnerability for window users which can be exploited by third parties [262]. Further, in support of this duty of care requirement is the recent determination in Australia by the administrative Appeals Tribunal that non customised software (COTS) is goods for the purposes of the Export Markets Development grants Act [246]. The ratio of this decision is highly persuasive and it is likely that it would be accepted by a court having jurisdiction. The effect of this decision is that if software is legally determined to be goods then software manufacturers will now be within the scope of the product liability provisions of the Trade Practices Act 1974 (CTH). In order to better protect the distribution of such software, software manufacturer's should ensure that their software is digitally signed using an application that utilised an appropriately secure hash function. Thus based upon the effects of the recent attacks mentioned in this paper, MD5 should not be relied upon. This recommendation does not currently apply to SHA-1 as the attacks on SHA-1 are of limited value, though it is possible that future developments of these current attacks could undermine the security of SHA-1 when utilised in the digital signing of software applications. In fact, NIST has already recommended that stronger hash functions be used for digital signing purposes and other purposes (such as SHA-224, SHA-256, SHA-384, SHA-512) and the phasing out of SHA-1 by 2010 [213].

In light of the attacks to MD5, it is recommended that MD5 not be used as a CRHF to achieve data integrity. The continued use of MD5 for software protection measures can no longer be assured of security and therefore should be immediately be stopped and a more secure hash mechanism as identified above should be used instead. From a legal perspective, any software manufacturers who do not so change are potentially exposing themselves to an unnecessary risk.

9.2 Conclusion

The legal implications arising out of the successful identification of the cryptographic hash function attacks need to be properly understood so that the law does not over react to such attacks and thus undermine the economic benefits that commerce has taken advantage of. Notwithstanding this, it is clear that electronic commerce currently relies upon many software applications that in turn are dependent upon secure hash functions. Even if the NIST recommended hash functions [211] are used which necessiates the immediate phasing out of MD5 and the gradual phasing out of SHA-1 by 2010, there is no guarantee that other yet to be discovered attacks are developed on the strong hash functions like SHA-256[4]. This then brings commerce back to the issue of whether the use of such technology will always result in a sword of damocles situation which is a risk that is ever present.

Table 9.1 summaries the technical implications of the collision attacks on MD5. As noted above, the legal implications are affected by how MD5 is used. For example, where a digital signature application utilises MD5 then the au-

[4]It should be noted that the analysis of SHA-256 has already started [95, 226, 298].

Table 9.1: Summary of impact of collisions in MD5 on some applications

Application	Attack possibility	Property violated
Message Signatures	Yes	Non-repudiation Integrity
Existing certificate signatures	No	
New certificate signatures	Yes	Authentication Non-repudiation
HMAC applications	No	
Tampering existing software	No	
Tampering new software	Yes	Non-repudiation Authentication Integrity

thentication and data integrity properties could be greatly undermined from the evidentiary perspective. This position is also applicable where the integrity and authentication of software is important.

Chapter 10

Conclusion and Future Research

Cryptographic hash functions have been in use for about two decades now providing security for computers and electronic communications. However, hash functions are little known elements of cryptography even now. The popular MD5 hash function which has been used to protect most of the financial transactions on the Internet is not collision resistant any longer [130, 172, 191, 264, 291]. The security of the other widely used hash function SHA-1 is questionable now [287, 288, 290]. NIST has conveyed that Federal agencies must stop relying on some applications based on SHA-1, notably digital signatures [213, 214] by the end of 2010. In addition, the security guarantees of the Merkle-Damgård hash function design framework which was followed in the design of MD4 family of hash functions is very limited [137, 138, 174]. The very few practical alternatives [52, 149] to the MD4 family of hash functions have been attacked effectively. Though the security of the HMAC and NMAC authentication schemes with MD5 and SHA-1 hash functions is not a big concern at the moment, it is not a good practice to use vulnerable hash functions for such an important application in the long run. These recent results in the cryptanalysis of hash functions and protocols that use hash functions highlight the importance of conducting in depth investigations into the security of these algorithms.

This thesis has studied the analysis, design and applications of cryptographic hash functions with some significant contributions to the current state of art of hash functions. In particular, focus has been the design and analysis of the variants/alternatives of the popular Merkle-Damgård hash function construction. From the application point of view, analysis and design issues of popular HMAC and NMAC constructions are studied. In Section 10.1, thesis contributions are reviewed. In Section 10.2, avenues for further research in this area are explored.

10.1 Summary of Thesis Contributions

- Chapter 1 introduces hash functions in cryptology and emphasizes that they deserve the status of one of the main objects of cryptology alongside symmetric key and public key cryptosystems.
- Chapter 2 classifies attacks on hash functions. It provides an overview of the Merkle-Damgård hash function design framework and explains the known generic attacks on the hash functions designed following this structure. Appendix B provides an evolution of the MD4 family of hash functions that follow the design principles of Merkle-Damgård iterative structure. Following the collision attacks on MD5, SHA-0 and SHA-1 hash functions, this chapter defines and classifies multi-block collision attacks on hash functions.
- Chapter 3 presents a new mode of operation for hash functions called **3CG** and provides analysis of a linear checksum variant of **3CG** called **3C** with respect to the known generic and multi-block collision attacks on hash functions. The result is that known techniques of performing generic 2^{nd}-preimage and herding attacks on the Merkle-Damgård hash functions do not work on **3C**, and all variants of **3CG** prevent straight forward length extension attacks. The valuable insight of this chapter is that while iterating the compression function results in an efficient hash function, the manner in which the iterations of the compression function are performed is essential for the security of the hash function.
- Chapter 4 shows that the security of Merkle-Damgård hash functions with linear checksums that include **3C**, GOST-L, F-Hash and **3C-GOST-L** against the generic 2^{nd}-preimage and herding attacks is not much larger than the Merkle-Damgård construction itself. It also extends the known results of performing multi-block collision attacks on **3C** to its variants F-Hash, GOST-L and **3C-GOST-L**. In addition, it improves the known

attack complexities to find multiple 2nd preimages for hash functions and cascade constructions. This chapter shows that maintaining twice the internal state for the hash functions using the linear checksum of chaining values or message blocks or both does not provide better security than hash functions. It provides a valuable insight that while maintaining twice the internal state is essential for an improved security of a hash function, the manner in which it is maintained is essential to achieve it.

- Chapter 5 proposes a new approach to design hash functions called iterated halving. Some analysis for this design is provided. This chapter opens a question on the possibility of designing efficient hash functions that are useful for streaming and non-streaming applications in a different style moving away from the iterative style of hash functions.

- Chapter 6 improves the known cryptanalysis of CAVE to find all the valid inputs for a given 128-bit digest of the 4-round (resp. 8-round) CAVE algorithm. While it takes 2^{91} (resp.2^{93}) evaluations of 4-round (resp. 8-round) CAVE to find one valid preimage using a 128-bit digest of 4-round (resp. 8-round) CAVE, it takes 2^{72} evaluations of 4-round CAVE and 2^{72} evaluations of 8-round CAVE to find all the valid preimages for a given 128-bit digest. It provides a valuable insight that increasing the number of rounds of CAVE from four to eight has only increased the effort of the new attack by two times, showing that doubling the number of rounds is not adequate to enhance its security confirming the previous analysis [192].

- Chapter 7 extensively studies MACs designed using hash functions. It provides analysis of the standard NMAC and HMAC functions using weak assumptions on the hash functions. The result is, under weak assumptions, the NMAC and HMAC functions are either insecure or do not always provide a secure MAC function depending on the assumptions. It provides an efficient variant for the NMAC function called NMAC-1 to authenticate short messages. This chapter proposes a modified variant for the NMAC function called M-NMAC by altering the position of the trail key in NMAC to obtain variable trail key lengths as in HMAC without using any pad constants. It provides security proofs for the NMAC and HMAC functions as pseudorandom functions using the weakly collision resistance property of the hash function and pseudorandomness of the compression function. This chapter proposes O-NMAC, a MAC scheme designed using a hash function with just one secret key and analyses it against the known attacks on MAC schemes based on hash functions. It provides one possible approach of the proof of security of O-NMAC and complete proof will be considered as part of the future work (see Section 10.2). This chapter provides a valuable insight that designing a MAC function using a mode of operation of hash function with a single secret key is a challenging task.

- Chapter 8 identifies that the NMAC and HMAC functions, based on the popular Davies-Meyer compression function, are vulnerable against the side channel attacks. It analyses the M-NMAC function proposed in Chapter 7 based on 12 PGV schemes against the DPA and RDPA attacks. The result is that these attacks do not work on M-NMAC based on the Davies-Meyer compression function. The valuable insight of this chapter is that the architectures of the compression function and the position of the secret key and the compression function in the MAC scheme influence the side channel attacks on MAC schemes.

- In Chapter 9, practical and legal implications due to the collision attacks on hash functions are discussed. In particular, implications that arise due to the collision attacks on the widely used MD5 and SHA-1 hash functions are discussed.

10.2 Future Directions

In the past three years, cryptanalysis of the Merkle-Damgård iterative design and hash functions designed on these principles have been actively pursued. This includes specific analysis oriented towards individual designs such as MD5 and SHA-1 [288, 290, 291] and generic analysis such as the long message second preimage [138] and herding [137] attacks directed towards the Merkle-Damård construction. In addition, during this period, some new hash function designs have been proposed [147, 149, 174] including modifications to the existing designs, [81, 127, 128, 274] to resist several known attacks on hash functions. On the other hand, equally active is the implication of hash function attacks on some important applications such as MAC schemes [12, 47, 109, 141] and digital signatures [19, 56]. This active research in several areas of hash functions provides opportunities to better understand the issues in the properties, design, cryptanalysis and applications of hash functions.

One of the major innovations in the research on the analysis and design of hash functions is the generic analysis of the Merkle-Damgård iterative hash function construction performed by Kelsey and Kohno [137]. They have found that iterated hash functions must possess a new security property called CTFP preimage resistance as noted in Chapter 2. The interesting issue here is it is not reasonable to classify this property as one of the fundamental properties of hash functions as it has evolved due to the weakness of the iterated hash function structures to the herding attack. This property might not be essential for completely different hash function structures and such structures might require some other properties which iterated hash functions do not require. For example, it would be interesting to see whether hash functions designed using modular arithmetic such as provably collision resistant very smooth hash function (VSH) [47] require this new property. In addition, it is an open question on what other properties, apart from the fundamental properties, hash functions based on the iterated halving approach proposed in Chapter 5 require. However, it is quite essential to classify CTFP preimage resistance as the fundamental property for iterated hash functions as widely used hash functions such as MD5 and SHA-1 are iterated hash functions. Following the lines of Kelsey and Kohno [137], it is important to investigate other properties that iterated hash functions might require by analysing the applications where hash functions are used.

In addition to the collision attacks on the popular hash functions MD5 and SHA-1 [288, 290, 291], the other significant contribution in the area of hash functions is the 2nd preimage attack on MD4 by Yu *et al.* [299]. It is essential to study the techniques in the analysis of the 2nd preimage attack on MD4 and see their application on MD5. The reason is, as identified in Appendix 9, security of some applications that use MD5 relies on the 2nd preimage resistance property rather than on the collision resistance. For example, attacks such as tampering of existing certificates and software based on the MD5 hash function require violation of this property.

The results of the Chapters 3 and 4 and the content described in the papers [111, 197, 203, 282] provides an area for many potential research problems.

The security analysis of the combination of iterated, concatenated and expanded (ICE) hash functions dealt with in [111, 203] and linear checksum of message blocks or chaining values against the generic attacks. How to increase the internal state of the hash function efficiently without widening the compression function and still resisting the attacks presented in this chapter? In other words, what variants of **3C** and GOST-L can resist all forms on known generic attacks including the ones presented in this chapter. It seems that there is some connection between Joux multicollision attacks and the results on the generalization of the birthday problem. It is worth investigating any connections between these two important papers. It will be interesting to see whether the GOST hash function structure [220] survives the known generic attacks on hash functions. The Chapter 4 shows that it is difficult to perform to generic attacks using the cryptanalytic collision finding algorithms such as those on MD5, SHA-0 and SHA-1 hash functions.

Chapter 5 of this thesis defines Iterated Halving (IH) approach as a new paradigm to design hash functions as alternative to the plain iterative schemes such as Merkle-Damgård construction. A specific instance of this family of hash functions called CRUSH is proposed. This chapter only provides a brief analysis for the Iterated Halving (IH) style of hash function design and no analysis for CRUSH is provided. Further analysis of IH class of hash functions and analysis of CRUSH is a future research goal. In addition, it would be an interesting research problem to see whether proof of security can be given for this scheme using some reasonable assumptions on the security of the block cipher.

One of the interesting research questions in hash functions is designing a MAC scheme using hash functions with just one secret key which was successfully achieved with block ciphers [74, 122]. O-NMAC proposed in chapter 7 is a valuable attempt along these lines but requires a solid proof of security. Considering the significance of designing MAC schemes with one secret key, O-NMAC requires further analysis. While the plain Merkle-Damgård iterative hash function scheme does not provide a secure MAC scheme even if the compression function is assumed as a PRF due to the length extension attacks, the only possible approach is to use a mode of operation of hash function to design such MAC schemes. One of the interesting research problems on these lines is to design provably secure MAC schemes using the wide-pipe and double-pipe modes of operation of hash functions [174] by keying the initial states of these designs using reasonable assumptions on the compression functions.

Side-channel cryptanalysis of MAC schemes based on hash functions is a very new area of research. The popular HMAC and NMAC schemes are shown to be vulnerable against these attacks when they are deployed with MD4 family of hash functions [221]. Chapter 8 of this thesis shows that M-NMAC, a variant of NMAC, with MD4 family of hash functions is secure against side-channel attacks. It is an interesting research problem to analyse the security of hybrid MACs (MACs designed with two different functions) using hash functions with respect to these attacks. Very recently it was noted that [273], formal models allowing one to understand the side-channel attacks and are also directly meaningful to practice is an open research problem. This question was addressed [273] from the point of view of block ciphers and it is interesting to illustrate this issue for MAC schemes using block ciphers and hash functions.

Many cryptographic schemes in use today especially, digital signatures, assume the existence of secure hash functions, without much justification [62]. This applies to popular hash functions SHA-1, SHA-256 and SHA-512 proposed by NIST where no analysis was given before their standardisation as FIPS-PUB 180-2 [211]. At least, there were some reasons behind the proposal of MD5 as indicated in Chapter 2 to thwart the early cryptanalytical attacks on MD4. Again, this is a heuristic approach. This case applies to the **3C** and **3C+** hash function constructions that are aimed at providing security against the multi-block collision attacks and generic attacks on hash functions but still be efficient, which they are. However, as shown in Chapter 4, new cryptanalytical techniques are developed showing that attacking them is not at all harder (as originally conjectured against multi-block collision attacks in Chapter 3) compared to Merkle-Damgård hash functions.

So there is a necessity for efficient, provably secure hash functions. The cryptologic community seems to be unsure at the moment on how to design generic hash function constructions with proofs of security from which practical designs can be developed. True security of these algorithms will not be found until the underlying Boolean functions are more properly studied [193]. A different idea was suggested in [62] in the context of designing very efficient truly proven secure block ciphers, basing assumptions on Boolean function circuits. These issues seem to be worth looking at in order to design provably secure hash functions.

It seems that SHA-256 and RIPEMD-160 will work as replacements for the suspect SHA-1 in the next few years. As far as our knowledge is concerned, not many secure and efficient hash function alternatives exist at the moment to the Merkle-Damgård construction. Hence, it is likely that significant research results of the analysis and design aspects in this fascinating branch of cryptology, will be seen in the near future.

Appendix A

Finding Expandable Messages for Hash Functions

A.1 Fixed Point Expandable Message Algorithm

The following algorithm is used to find expandable messages for a t-bit hash function using fixed points in the compression function.
ALGORITHM: Make FixedPoint Expandable Message(H_0)
Variables:

1. H_0 = Initial chaining value for the expandable message.
2. FindRandomFixedPoint() = An algorithm returning a pair (H_i, M_i) such that $H_i = f(H_i, M_i)$.
3. A, C = Two lists of hash values.
4. B, D = Two lists of message blocks.
5. i, j = Integers.
6. $M(i)$ = A function that produces a unique message block for each integer i such that $i < 2^t$.

Steps:

1. Construct a list of $2^{t/2}$ fixed points:
   - For $i = 0$ to $2^{t/2} - 1$
     - $h, m =$FindRandomFixedPoint()
     - $A_i = h$
     - $B_i = m$
2. Construct a list of $2^{t/2}$ hash values that can reach from H_0:

   - For $i = 0$ to $2^{t/2} - 1$
     - $h = f(H_0, M(i))$
     - $C_i = h$
     - $D_i = M(i)$
3. Find a match between lists A and C; let i, j satisfy $A_i = C_j$
4. Return expandable message $(D_j, D_j||B_i)$.

Work: About $2^{t/2+1}$ calls to f assuming that much memory.

A.2 Finding Expandable Messages using Generic Multicollision

ALGORITHM: FindCollision(α,H_0)
Find a collision pair with lengths 1 and α blocks, starting from H_0.

Variables:

1. $\alpha =$ Desired length of second message.
2. $A, B =$ Lists of intermediate hash values.
3. $q =$ A fixed "dummy" message used for getting the desired length.
4. $H_0 =$ The input hash value for the collision.

5. H_{tmp} = The intermediate hash value used in the attack.
6. $M(i)$ = The i^{th} distinct message block used in the attack.
7. t = Width of hash function chaining value and output in bits.

Steps:

1. Compute the starting hash for the α-block message by processing $\alpha - 1$ dummy message blocks.

   - $H_{tmp} = H_0$
   - For $i = 0$ to $\alpha - 2$:

     • $H_{tmp} = f(H_{tmp}, q)$

2. Build lists A and B as follows:

   - For $i = 0$ to $2^{t/2} - 1$:

     • $A_i = f(H_0, M(i))$
     • $B_i = f(H_{tmp}, M(i))$

3. Find i, j such that $A_i = B_j$.
4. Return colliding messages $(M(i), q||q||q|| \dots, q||M(j))$ and the resulting intermediate hash $f(H_0, M(i))$.

Work: $\alpha - 1 + 2^{t/2+1}$ calls to the compression function.

A.2.1 Building a Full $(d, d + 2^d - 1)$-expandable Message

This section corrects the pseudocode given in [138, p.480] to find $(d, d + 2^d - 1)$-expandable message as the pseudocode given in that reference does not produce the expandable message needed.

ALGORITHM: MakeExpandableMessage(H_0, d)

Make a $(d, d + 2^d - 1)$-expandable message

Variables:

1. H_{tmp} = the current intermediate hash value.

2. C = a list of pairs of messages of different lengths; $C[i][0]$ is the first message of pair i, while $C[i][1]$ is that pair's second message.

Steps:

```
1. Let $H_{tmp} = H_{in}$.
2. For $i = 0$ to $d - 1$
   - $(M_0, M_1, H_{tmp}) = \text{FindCollision}(2^i + 1, H_{tmp})$
   - $C[i][0] = M_0$
   - $C[i][1] = M_1$
3. Return the list of messages $C$ (Array of $d \times 2$ messages)
```

Work: $d \times 2^{t/2+1} + 2^d \approx d \times 2^{t/2+1}$

A.3 Long Message 2^{nd} preimage Attack using Expandable Messages

The following algorithm is used to carry the long message second preimage attack using expandable messages found by either of the above techniques.
ALGORITHM: LongMessageAttack(M_{target})
Find the second preimage for a message of $2^d + d + 1$ blocks.
Variables:

1. M_{target} = the message for which a second preimage is to be found.

2. M_{link} = a message block used to link the expandable message to some point in the target's message sequence of intermediate hash values.

3. A = a list of intermediate hash values.

4. h_{exp} = intermediate chaining values from processing an expandable message.

Steps:

```
1. $C = \text{MakeExpandableMessage}(d)$
```

2. H_{exp} = the intermediate hash value after processing the expandable message in C.

Steps:

1. Compute the intermediate hash values for M_{target}.

   - M_i is the i^{th} message block and H_0 is the IV. - $H_i = f(H_{i-1}, H_i)$, the i^{th} intermediate hash output block.

2. Find a message block that links the expandable message to one of the intermediate hash values for M_{target} after the d^{th} block.

   - Try linking messages M_{link} until $f(H_{exp}, M_{link}) = H_j$ for some $d + 1 \leq j \leq 2^d + d + 1$.

3. Use the expandable message to produce a message M^* that is $j - 1$ blocks long.

4. Return second preimage $M^*||M_{link}||M[j+1]||M[j+2]\ldots M[2^d+d+1]$ (if $j = 2^d + d + 1$ then no original message blocks are included in the second preimage)

Work: Effort to find the expandable message plus the effort to find the linking message.

Appendix B

Evolution of MD4 Family of Hash Functions

The popular hash functions MD4, MD5, SHA-0, SHA-1, RIPEMD, RIPEMD-128/160, SHA-256, SHA-224, SHA-384 and SHA-512 are designed following the design principles of Merkle-Damgård construction. These hash functions are referred to as hash functions from the MD4 family[1]. The block size, number of compression function steps required to process one block of message and the size of the digest of the hash functions are listed in the Table B.1. Except SHA-384 and SHA-512 hash functions, the other functions are optimized to work on 32-bit processors with word size of 32 bits as shown in Table B.1. Schneier and Kelsey have made an interesting observation in their classical paper [267] that the compression functions of these hash functions are basically unbalanced Feistel networks (UFN). This section provides a high-level explanation of the evolution of these hash functions showing that the compression functions of these hash functions are UFNs operating in a word-based non-linear feedback shift register mode of operation. For detailed explanation on the working details of these hash functions and attacks on hash functions in the literature see the appropriate references or some dissertations on hash functions [55, 125, 228, 276].

[1]In some parts of the literature [201, 208] RIPMED hash functions are referred to as part of the RIPEMD family and hash functions SHA-0/1,SHA-224/256 and SHA-384/512 are considered to be part of SHA family. However, this thesis considers all these hash functions as part of the MD4 family.

An explanation on the UFNs is provided in section B.0.1 followed by the evolution of MD4 family of hash functions in section B.

B.0.1 Balanced and Unbalanced Feistel Networks

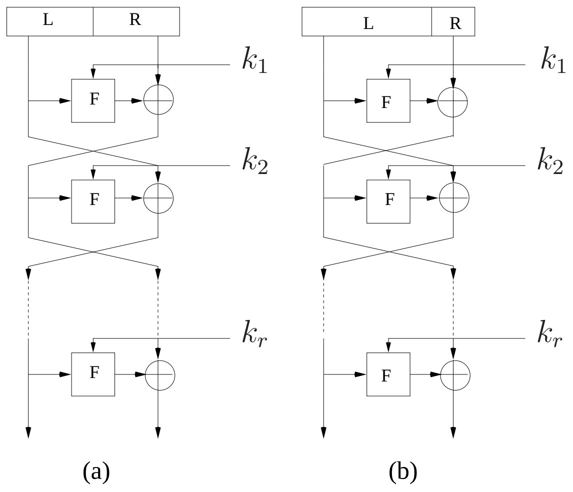


Figure B.1: Balanced (a) and Unbalanced Feistel Networks (b)

The Feistel network [78] used to design block ciphers was invented by the "father of modern block ciphers" Horst Feistel. This structure is used to transform any function, usually called an F-function, into a permutation as shown in the r-round Feistel network in Fig B.1. The F-function is defined as $F : \{0,1\}^{n/2} \times \{0,1\}^k \rightarrow \{0,1\}^{n/2}$ where n is the size of the block split into left and right halves of equal size of $n/2$ bits and k is the size of the round key input k_i for $i = 1$ to r used in each round as shown in Fig B.1. A key-scheduling algorithm is used in deriving the round keys $k_1, k_2, \ldots, k_r$ from the master key K. The Data Encryption Standard (DES) is an example of the cipher which uses balanced Feistel networks [212]. One round of a balanced Feistel network is defined by $X_{i+1} = (F_{k_i}(\text{MSB}_{n/2}(X_i)) \oplus \text{LSB}_{n/2}(X_i))||\text{MSB}_{n/2}(X_i)$ where X_i is an n-bit input to the round, X_{i+1} is an n-bit output of the round, $\text{MSB}_{n/2}(X_i)$ are the most significant $n/2$ bits of X_i and $\text{LSB}_{n/2}(X_i)$ are the least significant $n/2$ bits of X_i.

An unbalanced Feistel network (UFN) is a modified Feistel network where the left and right half of the inputs are not of equal size. The block ciphers MacGuffin [37],MARS [38] and Skipjack [152] are examples with this type of structure. One round of an UFN is defined by $X_{i+1} = (F(\text{MSB}_s(X_i), k_i) \oplus$

$\mathrm{LSB}_t(X_i)||\mathrm{MSB}_s(X_i)$ where $\mathrm{MSB}_s(X_i)$ are the most significant s bits of X_i and $\mathrm{LSB}_t(X_i)$ are the least significant t bits of X_i. Often, if the input to the F function is more than half of the total state size ($s > t$) then the UFNs are source heavy and if $s < t$ they are target heavy. A UFN is said to be homogeneous when the F-function is identical in each round and heterogeneous when the F-function is not identical in each round.

B.0.2 Hash Functions in the MD4 Family

Hash function	Block size	Registers	Steps	Digest size
MD4	512 (32×16)	4	48	128
MD5	512 (32×16)	4	64	128
RIPEMD	512 (32×16)	2×4	64	128
RIPEMD-128	512 (32×16)	2×4	2×64	128
RIPEMD-160	512 (32×16)	2×5	2×80	160
RIPEMD-256	512 (32×16)	2×4	2×64	256
RIPEMD-320	512 (32×16)	2×5	2×80	320
SHA-0	512 (32×16)	5	80	160
SHA-1	512 (32×16)	5	80	160
SHA-224	512 (32×16)	8	64	224
SHA-256	512 (32×16)	8	64	256
SHA-384	1024 (64×16)	8	80	384
SHA-512	1024 (64×16)	8	80	512

Table B.1: Merkle-Damgård Hash Functions

The MD4 message-digest algorithm, designed by Rivest [252, 254], is the first hash function in the MD4 family and was designed to work fast on 32-bit machines. It's design was mainly influenced by Damgård's design [53] and many hash functions designed after MD4, have followed its style of iterating the compression function to obtain the message digest. The working of its compression function can be viewed as the operation of UFN or a word-based non-linear feedback shift register (NLFSR) as shown in Figure B.2.

The MD4 compression function uses 3 rounds with each round consisting of 16 steps to process a message block of 512 bits. A 512-bit message block is represented with sixteen 32-bit words. A step is denoted with i. In Fig B.2, F_i is the non-linear Boolean or auxiliary function, W_i is the order of sixteen 32-bit message words, K_i is an additive constant used in every step of the algorithm.

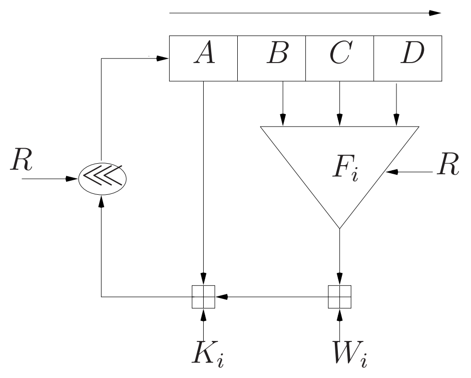


Figure B.2: Core of the MD4 compression function

The amount of circular shift of the result in every step i depends on the round number R.

After every step, the contents of the registers are shifted to the right by one word as shown in Figure B.2. At the end of three rounds, the compression function maps a 4-register chaining value and a 16-word message block to a new chaining value of four registers. After 3 rounds, the registers values are added to the register values at the beginning of the rounds using the feed-forward mode to achieve irreversibility. This value is used as the starting state to process the next message block of 512 bits. In every step of the compression function, the function F_i takes data only from the 3 registers as shown in Fig B.2 and hence this compression function is a 96/32 source heavy UFN [267].

The collision attacks on hash functions are performed using the well established differential cryptanalysis approach used to analyse block ciphers. This is briefly discussed in Appendix C. The permutation used for the message words in the steps 17-48 is similar and also the choice of left shift rotation is similar in some rounds. Boer and Bosselaers [59] have identified these as weaknesses and exploited then for finding collisions on the last thirty-two rounds of MD4. Collisions for the first two rounds of MD4 were found by Vaundenay [281] exploiting the lack of multipermutations introduced by the majority Boolean function (see [252]) in the compression function of MD4. This attack on the complete MD4 hash function results in two messages that yield near collisions.

The first collision attack on the MD4 hash function was performed by Dobbertin [67, 71]. This attack finds collisions for MD4 using two different messages that differ in just one word and having the same chaining value with a complexity of 2^{20} MD4 hash function evaluations. Kassleman [131] has later improved this collision attack by a factor of 64. Wang *et al.* [286] have improved this attack by finding collisions for the MD4 hash function with a complexity of at most 2^8 evaluations of the MD4 hash function. Naito *et al.* [202] have improved this collision attack on MD4 to a complexity of at most three times that of the hashing operations of the MD4 algorithm.

Dobbertin [72] has shown that MD4 with only the first two rounds is not preimage resistant with a complexity of about 2^{32} evaluations of MD4. Kuwakado and Tanaka [162] have proposed an algorithm that can be used to find preimages in about 2^{32} evaluations of the algorithm if the second round of the compression function is omitted.

The MD5 message-digest algorithm is more conservative in design but slightly slower than MD4 [255]. Following the MD4 style, the compression function of MD5 is a 96/32 heterogeneous source heavy UFN operating in the word based NLFSR mode as shown in Fig B.3.

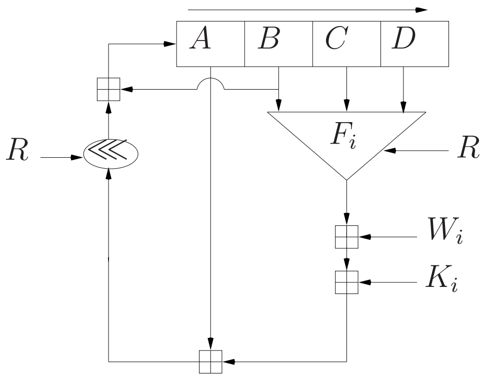


Figure B.3: Core of the MD5 compression function

It has sixteen more rounds than MD4, a less symmetric multiplexer function (see [255]) instead of the majority function to provide the multipermutation property, a different order of the message words in steps 17-48, variable left shift amounts, a unique additive constant K_i for every step and the addition of the result of the previous step to the current step to allow a faster avalanche effect. However, these improvements have been shown to be not enough to meet the security requirements of MD5. Special pseudo-collisions [60] were found for the compression function of MD5 with the data registers initialized to two different chaining values. These two chaining values contain the most significant bits that are complements of each other. It is interesting that this attack works on MD5 but not on MD4 because of the addition of the result of the previous step to the current step in MD5 (see Fig B.3 where this is shown with the content of register B added to the output of left shift). In 1996, Dobbertin [69] showed that it is possible to find collisions for the compression function of MD5 by selecting two messages that differ by a small hamming weight using an arbitrary initial value rather than the one in the specification of MD5.

The first practical collisions on MD5 were found by Wang *et al.* [285, 291] in 2004. Their attack on MD5 is a 2-block differential collision attack [30] with a complexity of 2^{39} hashing operations of MD5. Their technique involves finding nearly collided chaining values for the first message block and then converting these values to a collision after processing second message blocks thus exploiting the Merkle-Damgård iterative structure of MD5. Multi-block collision attacks are explored in great depth in section 2.5.

Since Wang *et al.* published their results on MD5, several improvements have been made for finding collisions for this hash function. See [144, 145, 172, 264, 297] for the improved cryptanalysis of the MD5 hash function. The latest cryptanalysis of MD5 [172, 264] shows that collisions can be found on MD5 with a complexity about 2^{33}-2^{34} hashing operations of the algorithm.

RIPEMD [70], is a two parallel track model of MD4, with XOR combining the outputs at the end. The permutation of the message words, Boolean functions and the cyclic shifts used in the two parallel lines are the same; they just have different additive constants. It was shown in [70] that because of the application of the same order of message words in the two parallel lines, collisions can be found in the compression function with a complexity of 2^{31} hash computations if the first or last round of the compression function is removed. This attack is similar to the one used against MD4 [71]. The first collision attack on the full RIPEMD was performed by Wang *et al.* [285, 286] in 2^{18} hash function computations.

The main alternative to MD5 was the original Secure Hash Algorithm (SHA) (now known as SHA-0) proposed by the National Institute of Standards and Technology (NIST) in 1993 [206]. It uses the design principles of MD4 with greatly improved message pre-processing (a task which one may consider as similar to a key-schedule in a block cipher). The methods used to attack MD4, MD5 and RIPEMD cannot be applied directly on SHA-0 because of the linear message expansion of sixteen 32-bit message words (512-bit block) to eighty 32-bit words (2560 bits) to achieve large hamming distance between any two randomly selected input data values. It is interesting that SHA-1 [210], the revised version of SHA-

0, differs from SHA-0 only in this linear message expansion where the expanded message is shifted to the right by 1 bit. The compression function of SHA-0/1 is shown in Figure B.4. Note that in every step, the output of the F_i function after mixing with a 32-bit word W_i and a constant K_i is added to the target register E. The circular left shift of A and an additional cyclic shift of the part B of the source register in every step is performed making the UFN of SHA-0/1 more complicated than MD4 and MD5. The compression function of SHA-0/1 is a 128/32 source heavy heterogeneous UFN as shown in Figure B.4.

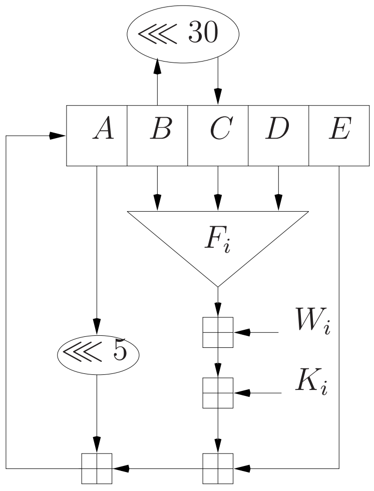


Figure B.4: The core of the SHA-0/1 compression function

The reasons behind this tiny design alteration from SHA-0 to SHA-1 by the NIST was never made public. However, one could imagine that this is due to a collision attack for the compression function of SHA-0 by Chabaud and Joux [42] with a complexity of 2^{61} which is not effective on SHA-1 because of the right shift of the bits after the message expansion in SHA-1. Their attack finds collisions for the 35-step SHA-0 compression function in 2^{14} hashing operations for messages with specific differences. A similar attack on SHA-0 was independently discovered by Wang in 1997 (For example see the reference [289]) but published in Chinese. Biham and Chen [24] have improved this attack on SHA-0 by finding near-collisions on the full SHA-0 with 18-bit difference with a complexity of 2^{40} and full collisions for the 65-round SHA-0 with a complexity of 2^{29}. Biham *et al.* have found the first collision on the full SHA-0 [27] using a 4-block collision finding technique based on near-collisions with a complexity of 2^{51} hashing operations of the SHA-0 hash function. Their attack was further improved by Wang *et al* [289] using a 2-block collision finding algorithm with a complexity of 2^{39} SHA-0 operations.

The reduced versions of SHA-1 were analyzed in [27, 249]. Biham *et al.* [27] have found a 2-block collision on the 40-round SHA-1 with a complexity of 2^{57} hashing operations of SHA-1. Rijmen and Oswald [249] have estimated theoretically that a collision can be found for the 53-step SHA-1 with a complexity of 2^{71} hashing operations of the algorithm. The first full collision attack on SHA-1 was performed by Wang *et al.* [290]. Wang *et al.* have shown techniques that can be used to find full collisions for a 2-block SHA-1 with a complexity of about 2^{69} hashing operations which is less than the 2^{80} theoretical bound of finding collisions in SHA-1 using birthday attack. Recently, Wang *et al.* have improved their attack [287] to a complexity of 2^{63} hashing operations of SHA-1.

Although it is clear that the techniques to find collisions in SHA-1 [287, 290] are viable, Wang *et al.* only estimated the difficulty of an attack, rather than showing any real collision as they had shown on MD4, MD5, RIPEMD, HAVAL and SHA-0 hash functions [285, 286, 289, 291]. Notwithstanding this, 2^{63} hashing operations of SHA-1 is within the reach of a distributed computing effort [274]. It is likely that further improvements to this attack will occur in the foreseeable future. A hardware architecture to break the SHA-1 hash function was proposed by Satoh [265] using Wang's cryptanalytical methods [290]. The estimated \$10 million system built with current hardware technology would consists of 303 personal computers with 16 SHA-1 attacking boards each with an USB interface and each board would have 32 chips. The system consists a total of 9,928,704 SHA-1 macros, and could find a real collision for the full-round SHA-1 in 127 days.

With the advent of early cryptanalytical developments on the hash functions MD4, MD5 and RIPEMD, designers of hash functions have started paying attention to the practical bit security level that a hash function should ideally provide. RIPEMD-160 [68] was considered as an alternative to MD5 and as an extended version of RIPEMD. It is a part of ISO/IEC 10118-3 [120] standard of dedicated hash functions. It outputs 160 bits of hash value and its compression function is executed for 80 steps in two parallel lines (left and right) differing from one another in the permutation of the message words (which is not the case in RIPEMD), the order of five Boolean functions used (reversed order) and the application of additive constants.

The compression function of RIPEMD-160 can be seen as 128/32 heteroge-

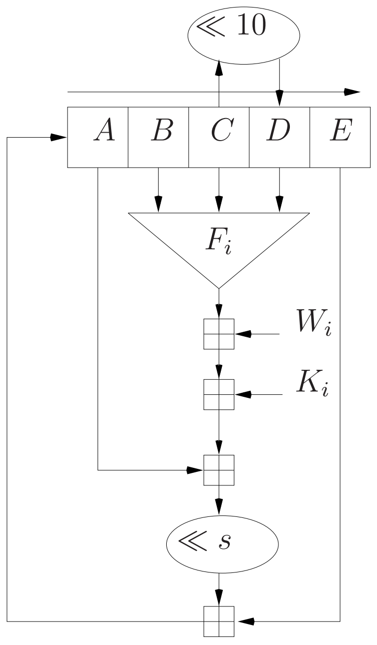


Figure B.5: Core of 1-track of RIPEMD-160 compression function

neous source heavy UFN as shown in Fig B.5. The function F_i is selected by the round and it would be F_j for the j^{th}-round on the left side and F_{6-j} for the j^{th}-round on the right side for which $j = 1, \ldots, 5$. The step constants K_i and rotation values s are different for each step in a round and differ in both the sides. The previous state of the hash function is combined with the state values obtained from both the sides after 80 steps of the compression function are executed to obtain a new starting state to process the next message block. The recent analytical results [185] on RIPEMD-160 shows that it is a collision resistant hash function and the techniques used to attack MD4, MD5 and SHA-1 [286, 288, 290, 291] do not work on RIPEMD-160. RIPEMD-128 is a reduced variant of RIPEMD-160 and uses a chaining variable of 128 bits with only four rounds (or 64 steps) of RIPEMD-160. Recently, a collision attack was demonstrated on the 3-round version of RIPEMD-128 [185].

With the emergence of the Advanced Encryption Standard (AES) [218] as a new secret-key encryption standard with a variable key size of 128, 192 or 256 bits, the 80-bit security level offered by hash functions seems to be inadequate. This initiated the National Security Agency (NSA) to develop three new hash functions to match the security level offered by the AES encryption scheme. These new secure hash algorithms share a similar structure and are called SHA-

224, SHA-256, SHA-384 and SHA-512. These four algorithms along with SHA-1 are included in the secure hash standard [211].

The compression function of SHA-256 can be visualised as a coupling of two finite state machines similar to SHA-1, as shown in Figure B.6. One finite state machine is driven by the input words and round constants while the second machine is driven by the first one. This design approach of coupling two smaller networks makes SHA-256 more distinct from its precursor designs. In addition to the coupling, there are increased security modifications to the basic networks. For the linear mixing operations $\sum_1$ and $\sum_0$ of SHA-256 (see [211]), each output bit depends on three input bits and each input bit affects three output bits, which provides some diffusion where the rotation operation does not. One execution of the compression function of SHA-256 has 64 steps and all these steps use the same Boolean functions except with different constants and message words. Refer to [211] for the complete description of these algorithms. From Fig B.6, it is clear that the compression function of SHA-256 is a source heavy heterogeneous 192/64 UFN.

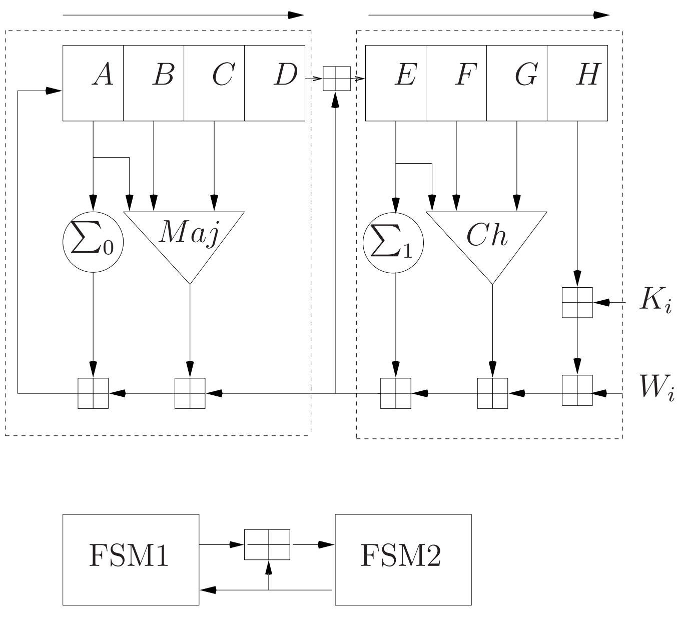


Figure B.6: SHA-256 Compression Function

The SHA-224 hash function uses the compression function of SHA-256 except that it uses a different IV and outputs only left most 224 bits instead of 256 bits as SHA-256. Though NIST has not specified any reasons for proposing SHA-224 as a truncation mode of SHA-256, one could imagine that to meet the same security level as 112-bit key TripleDES block cipher [216]. This means that it would take about 2^{112} iterations of SHA-224 to find a collision using birthday attack and the same effort is required to find the 112-bit secret key of TripleDES using brute-force attack. So far, there are no collision attacks on the SHA-224 and SHA-256 hash functions. Nevertheless, there has been some active research in the analysis of SHA-256 [95, 184, 226, 298]. It should be noted that these analytical results on SHA-256 are straight-forward on SHA-224 with no additional effort. In some cases, the attackers can even be "lucky". For example, a near-collision on the reduced version (up to 19 steps of the compression function execution) of SHA-256 resulted in a full collision on the equivalent version of SHA-224 [184] as the truncation of 256 bits to 224 bits has cut away a few 1 bits in the difference. This attack exemplifies the importance of near-collision resistance property for a hash function.

The operation of hash functions SHA-384 and SHA-512 is similar to SHA-224 and SHA-256 except that they operate on 64-bit words, are designed to work efficiently on 64-bit processors, process blocks of 1024 bits and have 80 steps of compression function iterations. They also use same the Boolean functions as SHA-224 and SHA-256. SHA-384 is the truncated version of SHA-512 with a different IV. While SHA-512 has the same security level as 256-bit key AES, SHA-384 has the same security level as 192-bit key AES. The compression functions of SHA-384 and SHA-512 are source heavy heterogeneous 384/128 UFNs.

B.0.3 Some Hash Functions outside the MD4 family

HAVAL [301], a variable length output hash function, is another hash function whose construction is based on the design principles of MD4. Note that though HAVAL follows the design principle of MD4 family of hash functions, it is not considered to be part of this family due to reasons such as the use of Boolean functions, padding techniques, variable length hash values and a variable number of internal rounds (3, 4 or 5). However, it is mentioned here for completeness. Unlike MD4 and MD5, HAVAL processes messages in blocks of 1024 bits, uses an initial state of 256 bits and padding is specified with the length encoding of the original length of the information, number of rounds each block is processed with and the version number of the algorithm. The output is 256 bits but can be transformed to 128, 160, 192 or 224 bits using an optional output transformation. HAVAL is a 224/32 heterogeneous source heavy UFN. Kasselman and Penzhorn [132] have presented the first collision attack on 256-bit 3-round HAVAL by using only the last two rounds of the three rounds. Park *et al.* [223] have extended this result by finding collisions for the first two rounds and last two rounds of 3-round HAVAL [223]. Rompay *et al* [258] have improved these collision attacks on the reduced version of 3-round HAVAL by finding a collision for the full 3-round 256-bit HAVAL in 2^{29} iterations of its compression function. Her *et al.* [107] have presented the first collision attack on the reduced versions (1,2; 2,3 and 3,4) of 4-round HAVAL. The first collision attack on the 5-round HAVAL with a 128-bit digest was found by Wang *et al.* [285,291] in 2^7 operations of the algorithm.

Hermetic hash functions [51] such as PANAMA [52] are designed based on the principle of MD4 family of hash functions. In hermetic hash functions, the chaining state is split into two parts: the *hashing state a* and the *buffer b* and both parts are affected by a round function. The round transformation is a simple nonlinear invertible updating transformation which has parts of the buffer contents as parameters (these are selected by a selection function) and this transformation updates the hashing state. In an analogy with block ciphers, hash functions of MD4 family use a variable length message which acts as a key and the chaining state which acts as a message block and is updated regularly. In hermetic schemes, the buffer and its updating function act as the key and hashing state which works as the message is updated by the round transformation.

PANAMA is a stream/hash cryptographic module designed to operate on 32-bit software architectures. It works both as a stream cipher and hash function. As a hash function, PANAMA, maps messages of arbitrary length to a hash value of 256 bits. It uses a 544-bit (17 words) state and a 8192-bit buffer that are updated by performing an iteration. It uses two modes of iteration: push and pull. In the push or absorbing mode, input message blocks are injected and there is no generation of output. Pull mode or emitting mode takes no message input and generates output and in the final blank pull iteration output is discarded. Push mode can be viewed as the input message processing stage in the PANAMA and pull mode can be viewed as hashing stage in PANAMA and after thirty three pull iterations, the output is collected from the state. Rijmen *et al.* [250] have demonstrated a theoretical collision attack that can find collisions for the PANAMA hash function with a complexity of 2^{82}.

Appendix C

Differential Cryptanalysis of Hash Functions

Differential cryptanalysis is a technique introduced by Eli Biham and Adi Shamir to analyse the block cipher DES [31]. It is a chosen plaintext attack where the attacker is able to select inputs and examine outputs in an attempt to derive the secret key. If X and X' are two input plaintexts or messages, the difference between them is defined as $\Delta X = X' \; op \; X$. If Y and Y' are the two corresponding ciphertexts (hash values in the case of hash functions), the difference between them is defined as $\Delta Y = Y' \; op \; Y$. The difference measure op can be exclusive-or operation [30, 31] or integer subtraction [46, 68, 70, 71, 286, 291] or may be any other operation. An attacker performing differential cryptanalysis or a differential attack, selects inputs to satisfy a particular ΔX under the assumption that for that particular input difference ΔX, a particular output difference ΔY occurs with a very high probability. The pair $(\Delta X, \Delta Y)$ is referred to as a differential.

Let M and M' are any two b-bit multiple messages (for hash functions MD5 and SHA-1, $b = 512$) where $M = (M_0, M_1, \ldots, M_{l-1})$ and $M' = (M'_0, M'_1, \ldots, M'_{l-1})$ and l is the total number of message blocks. Let ΔH_0 and ΔH be the initial and final value differences of the hash function. For a collision in hash function, ΔH_0 and ΔH should be equal to 0. The differential that produces a collision in a hash function is called collision differential. A full collision differential for a hash function is defined as follows for $i = 0$ to l:

$$\Delta H_i \overset{(M_i,M_i')}{\longrightarrow} \Delta H_{l-1} \overset{(M_{l-1},M_{l-1}')}{\longrightarrow} \Delta H \tag{C.1}$$

Consider an n round hash function with each round containing t number of step operations (for example, for MD5, $n = 4$ and $t = 16$). For this hash function, i^{th} iteration differential $\Delta H_i \overset{(M_i,M_i')}{\longrightarrow} \Delta H_{i+1}$ is represented as follows:

$$\Delta H_i \xrightarrow{P_1} \Delta R_{i+1,1} \ldots \Delta R_{i+1,n-1} \xrightarrow{P_n} \Delta R_{i+1,n} = \Delta H_{i+1}$$

A round or a step differential characteristic is a sequence of input and output differences to the rounds or steps so that the output differences from one round or step correspond to the input differences of the following round or step. The round differential $\Delta R_{j-1} \longrightarrow \Delta R_j (j = 1 \ldots n)$ with the probability P_j is expanded to the following step differential characteristics.

$$\Delta R_{j-1} \xrightarrow{P_{j1}} \Delta X_1 \xrightarrow{P_{j2}} \ldots \xrightarrow{P_{jt}} \Delta X_t = \Delta R_j$$

The probability P of the collision differential $\Delta H_i \overset{(M_i,M_i')}{\longrightarrow} \Delta H_{i+1}$ satisfies the following equation:

$$P \geq \prod_{i=1}^{4} P_j \text{ and } P_j \geq \prod_{t=1}^{16} P_{jt}.$$

C.1 2-block Collision Attack on MD5

The collision attack on MD5 is a differential attack and uses message modification techniques to improve the efficiency of the attack. In a message modification technique some of the message words are fixed before fulfilling the necessary conditions that are required to generate a collision. The attack uses modular integer subtraction as a measure of difference. This differential is called modular differential. This attack on MD5 is an improvement over the previous attack on MD5 [68] by Hans Dobbertin. While the previous attack on MD5 by Dobbertin finds collisions on a pair of one-block messages (512-bit messages) for an IV chosen by the attack, this improved attack finds collisions on a pair of two-block messages (1024-bit messages) using the IV of the MD5 hash function and it indeed works for any chosen IV.

If (a_0, b_0, c_0, d_0) are the initial values for MD5, the attack finds a pair of 2-block messages (M_0, M_1) and (M_0', M_1') such that the following equations are satisfied:

$$\begin{aligned}(a, b, c, d) &= \mathrm{MD5}(a_0, b_0, c_0, d_0, M_0)\\ (a', b', c', d') &= \mathrm{MD5}(a_0, b_0, c_0, d_0, M_0')\\ \mathrm{MD5}(a, b, c, d, M_1) &= \mathrm{MD5}(a', b', c', d', M_1')\end{aligned}$$

This attack finds first blocks (M_0, M_0') by performing 2^{39} MD5 operations and finds second blocks (M_1, M_1') by performing 2^{32} MD5 operations. The attack algorithm when implemented on IBM P690 machine takes about an hour to find M_0 and M_0' and takes only between 15 seconds and 5 minutes to find the second blocks M_1 and M_1'. The attack technique can be used to find collisions in other hash functions MD4, HAVAL and RIPEMD.

The collision differential for a 2-block collision on MD5 is represented as follows:

$$H_0 \overset{(M_0, M_0'), 2^{-37}}{\longrightarrow} \Delta H_1 \overset{(M_1, M_1'), 2^{-30}}{\longrightarrow} \Delta H = 0$$

The algorithm which finds collisions is as follows:

1. A random message M_0 is selected and is subjected to message modification techniques in such a way that M_0 and $M_0' = M_0 + \Delta M_0$ produce the following first iteration differential with a probability 2^{-37}

   $$\Delta M_0 \longrightarrow (\Delta H_1, \Delta M_1)$$

   Then a test is conducted to check whether all the characteristics hold by applying the compression function of MD5 on M_0 and M_0'.

2. A random message M_1 is selected and is subjected to message modification techniques in such a way that M_1 and $M_1' = M_1 + \Delta M_1$ produce the following second iteration differential with a probability 2^{-30}

   $$(\Delta H_1, \Delta M_1) \longrightarrow \Delta H = 0$$

   Then a test is conducted to check whether this pair of messages lead to a collision.

For a deeper explanation on message modification techniques and set of conditions required to find the first and second iteration differentials, refer to [291].

Note that the multi-block collision finding algorithm of Wang *et al.* has an interesting property that it works for any initialization value. However, this was not explicitly demonstrated in [286].

Klima [145] has independently developed multi-message modification techniques for finding collisions in MD5 that are quicker than the methods preseneted in [291]. These improved techniques are about 3 - 6 times faster than the one by Wang *et al.* [291]. In short, the collision finding algorithm chooses an IV arbitrarily and runs the procedure described in [145] and generates two messages of 1024 bits each that give a collision. Note that collisions can be found for any arbitrarily chosen IV only by using the procedure described in [145].

Appendix D

Generic Attacks on GOST-L, F-Hash and 3C-GOST-L

D.1 Long Message 2^{nd}-preimage Attack on GOST-L

The long message second preimage attack on a b-bit block and t-bit digest GOST-L hash function is similar to the one on **3C** except that the multicollision carried over is a 2^b multicollision on 1-block messages.

ALGORITHM: LongMessageAttack(M_{target}) on GOST-L

Find the second preimage for a message of $2^d + d + b + 1$ *blocks.*

Variables:

1. M_{target} = the target long message for which a second preimage is to be found.
2. M_{link} = linking message block used to connect the chaining value at the end of the expandable message to some point in the sequence of chaining values of the target message.
3. H_{exp} = the intermediate chaining value at the end of the expandable message.

4. H_t = the result of the 2^b 1-block multicollision on H starting from the initial state.
5. M_{final} = the second preimage of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.
6. M_{pref} = the checksum control prefix obtained from the CCS to force the linear checksum to the desired checksum.

Steps:

1. Compute the intermediate hash values for M_{target} using H:
   - H_0 is the initial state of H.
   - M_i is the i^{th} message block of M_{target}.
   - $H_i = f(H_{i-1}, M_i)$ is the intermediate hash value of H.
   - The cascade and accumulation chaining states are organised in some searchable structure for the attack, such as hash table. The elements $H_1, \ldots, H_d$ and the elements obtained in the processing of b 1-block messages are excluded from the table as the expandable messages cannot be made short enough to accommodate them in the attack.
2. Build a CSS by constructing a 2^b 1-block multicollision on GOST-L following as described in section 4.3.4. Let H_t be the multicollision value on the cascade chain and M_t be the corresponding checksum block which is random. The value of M_t depends on the choice of b data blocks from the CCS that give the collision H_t.
3. Construct a $(d, d + 2^d - 1)$ expandable message M_{exp} from the end of H_t. Append the expandable message M_{exp} to the CCS. Let H_{exp} be the chaining value at the end of the expandable message M_{exp}.
4. Test the message blocks from the end of H_{exp} to find a linking message block M_{link} such that $f(H_{exp}, M_{link})$ matches one of the chaining values stored in the hash table while processing M_{target}. Let this matching value of the target message be H_u and the linear checksum of data blocks of M_{target} until that point be M_u where $d + b + 1 \leq u \leq 2^d + d + b + 1$.

5. Use the CCS built in step 2 to find the checksum control prefix M_{pref} to adjust the accumulation of data blocks at that point to match the desired checksum value M_u in M_{target}. This is equivalent to adjusting the checksum M_t in the 2^b multicollision of step 2. The prefix M_{pref} is obtained by solving a system of $b \times b$ linear equations as described in section 4.3.5.

6. Expand the expandable message to produce a message M^* which is $u-1$ blocks long.

7. Return the second preimage $M_{final} = M_{pref}||M^*||M_{link}||M_{u+1} \ldots$ $M_{2^d+d+b+1}$ of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

Work: The total computational effort required in performing the 2nd preimage attack on GOST-L consists of the effort in finding a 2^b 1-block multicollision plus the effort in solving a $b \times b$ system of linear equations plus the work in finding the expandable message M_{exp} plus the work to find the linking message M_{link}. So the only additional effort in performing the second preimage attack on GOST-L over Merkle-Damgård hash function is the effort in solving a $b \times b$ system of equations and producing a 2^b 1-block multicollision.

1. Using the generic expandable-message finding algorithm, this effort equals $b \times 2^{t/2} + d \times 2^{t/2+1} + 2^{t-d+1}$ computations of the compression function and b^3 bit-XOR operations.

2. Using the fixed-point expandable-message finding algorithm, this effort equals $b \times 2^{t/2} + 3 \times 2^{t/2+1} + 2^{t-d+1}$ computations of the compression function and b^3 bit-XOR operations.

D.1.1 Long Message 2nd-preimage Attack on F-Hash

ALGORITHM: LongMessageAttack(M_{target}) on F-Hash
Find the second preimage for a message of $2^d + d + 2t + 1$ blocks.
Variables:

1. M_{target} = The target long message for which a second preimage is to be found.

2. M_{link} = Linking message block used to connect the cascade chaining value at the end of the expandable message to some point in the sequence of the cascade chaining values of the target message.

3. H_{exp} = The intermediate cascade chaining value at the end of the expandable message.

4. H_t = The result of the 2^t 2-block multicollision on the cascade chain of H starting from the initial state H_0.

5. M_{final} = The second preimage of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

6. M_{pref} = The checksum control prefix obtained from the CCS to force the linear checksum to the desired checksum.

Steps:

1. Compute the intermediate hash values for M_{target} using H:

   - H_0 and h_0 are the initial states on the cascade and accumulation chains of H respectively.
   - M_i is the i^{th} message block of M_{target}.
   - $(H_i, h_i) = f(H_{i-1}, M_i)$ and $h'_i = \bigoplus_{j=1}^{i-1} h_j$ are the i^{th} intermediate chaining values on the cascade and accumulation chains respectively.
   - The cascade and accumulation chaining states are organised in some searchable structure for the attack, such as hash table. The elements $H_1, \ldots, H_d$ and the elements obtained in the processing of t 2-block messages are excluded from the hash table as the expandable messages cannot be made short enough to accommodate them in the attack.

2. Build a CCS by constructing a 2^t 2-block multicollision on H as described in section 4.3.6 starting from the initial state H_0. Let H_t be the multicollision chaining value. The corresponding checksum value h'_t due to the 2^t 2-block multicollision on H is random and its value depends on the choice of the t 2-block messages from the CCS that give the collision H_t.

3. Construct a $(d, d+2^d-1)$ expandable message M_{exp} with H_t as the starting chaining state using the generic technique to find the expandable messages. Append the expandable message M_{exp} to the CCS. Let H_{exp} be the cascade chaining value at the end of the expandable message M_{exp}.

4. Try different message blocks from the end of H_{exp} to find a linking message block M_{link} such that $f(H_{exp}, M_{link})$ matches some cascade chaining value H_u stored in the hash table while processing M_{target}. Let this matching value of the target message be H_u and the corresponding accumulation chaining value be h'_u where $d+2t+1 \leq u \leq 2^d+d+2t+1$.

5. Use the CCS built in step 2 to find the checksum control prefix M_{pref} to adjust the accumulation chaining value at that point to match the desired accumulation value h'_u in the target message M_{target}. Using h'_u, the desired checksum value at the end of the 2^t 2-block multicollision is calculated and this value is adjusted in such a way that the desired checksum h'_u is obtained. The prefix M_{pref} is obtained by solving a system of $t \times t$ linear equations following section 4.3.3.

6. Expand the expandable message to produce a message M^* which is $u-1$ blocks long.

7. Return the second preimage $M_{final} = M_{pref}||M^*||M_{link}||M_{u+1}\ldots M_{2^d+d+1+2t}$ of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

Work: The computational effort required to perform the second preimage attack on F-Hash is the same as the work to find second preimages for **3C**. Using the generic expandable-message finding algorithm, this effort equals $t \times 2^{t/2} + d \times 2^{t/2+1} + 2^{t-d+1}$ compression function computations and t^3 bit-XOR operations.

D.1.2 Long-message 2^{nd}-preimage Attack on 3C-GOST-L

ALGORITHM: LongMessageAttack(M_{target}) on 3C-GOST-L
Find the second preimage for a message of $2^d+d+2t+1$ *blocks.*
Variables:

1. M_{target} = The target long message for which a second preimage is to be found.

2. M_{link} = Linking message block used to connect the cascade chaining value at the end of the expandable message to some point in the sequence of the cascade chaining values of the target message.

3. H_{exp} = The intermediate cascade chaining value at the end of the expandable message.

4. H_t = The result of the 2^t 2-block multicollision on H starting from the initial state H_0.

5. M_{final} = The second preimage of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

6. D_1 = The linear checksum obtained due to the chaining values in the cascade chain.

7. D_2 = The linear checksum obtained due to the message blocks.

8. M_{pref} = the checksum control prefix obtained from the two CCSs to force both the linear checksums to the desired checksums D_1 and D_2.

Steps:

1. Compute the intermediate hash values for M_{target} using H:

   - H_0 (resp. h'_0, h''_0) is the initial state on the cascade chain (resp. other two chains).
   - M_i is the i^{th} message block of M_{target}.
   - $H_i = f(H_{i-1}, M_i)$, $h'_i = H_i \oplus h_{i-1}$ and $h''_i = M_i \oplus (\bigoplus_{j=1}^{i-1} M_j)$ are the i^{th} intermediate chaining values on the cascade, second and third chains respectively.
   - The cascade and accumulation chaining states are organised in some searchable structure for the attack, such as hash table. The elements $H_1, \ldots, H_d$ and the elements obtained in the processing of t 2-block messages are excluded from the hash table as the expandable messages cannot be made short enough to accommodate them in the attack.

2. Build the CCS by constructing a 2^t 2-block multicollision on H as described in section 4.3.7 starting from the initial state H_0. Let H_t be the multicollision value on the cascade chain. The corresponding checksum values h'_t and h''_t due to the 2^t 2-block multicollision on H are random and these values depend on the choice of the t chaining values and message blocks from the two CCSs.

3. Construct a $(d, d+2^d-1)$ expandable message M_{exp} with H_t as the starting chaining state. Let H_{exp} be the cascade chaining value at the end of the expandable message M_{exp}.

4. Test message blocks from the end of H_{exp} to find a linking message block M_{link} such that $f(H_{exp}, M_{link})$ matches one of the cascade chaining values stored in the hash table while processing M_{target}. Let this matching value of the target message be H_u and the corresponding accumulation chaining values be h'_u and h''_u where $d + 2t + 1 \leq u \leq 2^d + d + 2t + 1$.

5. Use the CCSs built in step 2 to find the checksum control prefix M_{pref} to adjust the accumulation chaining values at that point to match the desired accumulation values h'_u and h''_u in the target message M_{target}. This is equivalent to adjusting the checksum value at the end of 2^t 2-block multicollision. M_{pref} is obtained by solving two sets of system of $t \times t$ linear equations as outlined in section 4.3.7.

6. Expand the expandable message to produce a message M^* which is $u - 1$ blocks long.

7. Return the second preimage $M_{final} = M_{pref}||M^*||M_{link}||M_{u+1} \ldots M_{2^d+d+1+2t}$ of the same length as M_{target} such that $H(M_{final}) = H(M_{target})$.

Work: The computational effort required to perform the second preimage attack on **3C-GOST-L** consists of the effort in finding a 2^t 2-block multicollision plus the effort in solving two $t \times t$ systems of linear equations plus the work in finding the expandable message plus the work to find the linking message.

1. Using the generic expandable-message finding algorithm, this effort equals $t \times 2^{t/2} + d \times 2^{t/2+1} + 2^{t-d+1}$ compression function computations and $2 \times t^3$ bit-XOR operations.

2. Using the fixed-point expandable-message finding algorithm, this effort equals $t \times 2^{t/2} + 3 \times 2^{t/2+1} + 2^{t-d+1}$ compression function computations and $2 \times t^3$ bit-XOR operations.

D.1.3 Herding Attack on GOST-L

The following steps outline the herding attack on a t-bit GOST-L hash function H:

1. Construct a 2^d hash value wide diamond structure as briefed in chapter 2 and output the final hash value H_f. The final hash value H_f could be due to any of the possible 2^{d-1} checksum values or some value chosen arbitrarily. Let M_c be that checksum value.

2. Build the CCS using a 2^b multicollision over 1-block messages and let H_t be the cascade chaining value after the multicollision.

3. When challenged with the prefix message P, process P using H_t as the starting chaining value on the cascade chain. Let $H(H_t, P) = H_p$.

4. Find the linking message M_{link} such that the state $H(H_p, M_{link})$ matches one of the 2^d outermost intermediate chaining values on the cascade chain in the diamond structure. If the match is compared to all the $2^{d+1} - 2$ intermediate chaining values then a $(1, d+1)$-expandable message must be produced at the end of the diamond structure to make sure that the final herded message is always a fixed length.

5. Use the CCS computed in the step 2 to force the checksum of the herded message P to M_c using the techniques to bypass the checksum in the 2^b multicollision described in section 4.3.5. Let M_{pref} be the checksum control prefix obtained after solving the system of equations due to the CCS.

6. Finally, output the message $M_{pref}||P||M_{link}||M_d$ where M_d are the message blocks which contribute to the construction of the diamond structure. The value $H(M)$ will be same as the chosen target H_f.

Work: The work to perform the herding attack on GOST-L is the work required to build the CCS plus the work required to perform the herding attack from [137].

This equals $b2^{t/2} + 2^{t/2+d/2+2} + 2^{t-d}$ computations of the compression function and b^3 bit-XOR operations assuming that only the outermost 2^d chaining values are used for searching in the diamond structure. If all the $2^{d+1} - 2$ intermediate chaining values are used for searching in the diamond structure then the work required equals $b \times 2^{t/2} + 2^{t/2+d/2+2} + d \times 2^{t/2+1} + 2^{t-d-1}$ computations of the compression function and b^3 bit-XOR operations.

D.1.4 Herding Attack on F-Hash

The following steps outline the herding attack on a t-bit F-Hash hash function H:

1. A 2^d hash value wide diamond structure is constructed for H with 2^d different arbitrary states $H_1, H_2, \ldots, H_{2^d}$ as the starting cascade chain hash values. It is constructed by finding 1-block collisions similar to the construction of the diamond structure for the Merkle-Damgård hash functions. The final hash value H_f, which is the output of the compression function g, is computed using any of the possible 2^{d-1} checksum values or some value chosen arbitrarily. Let h'_c be that checksum value.

2. Build the CCS for H using a 2^t multicollision over 2-block messages as described in section 4.3.6. Let H_t be the 2^t 2-block multicollision value on the cascade chain of H.

3. When challenged with the prefix message P, process P using H_t as the starting chaining value on the cascade chain. Let $H(H_t, P) = H_p$.

4. Find the linking message M_{link} such that the state $H(H_p, M_{link})$ matches one of the 2^d outermost intermediate chaining values on the cascade chain in the diamond structure. If the match is compared to all the $2^{d+1} - 2$ intermediate chaining values in the diamond structure then a $(1, d + 1)$-expandable message must be produced at the end of the diamond structure ensuring that the final herded message is always a fixed length.

5. Use the CCS computed in step 2 to force the checksum of the herded message P to h'_c using the techniques as described in section 4.3.6 to bypass the checksum in the 2^t multicollision. Let M_{pref} be the checksum control prefix obtained after solving the system of equations due to the CCS.

6. Finally, output the message $M = M_{pref}||P||M_{link}||M_d$ where M_d are the message blocks which contributes in the construction of the diamond structure. The value $H(M)$ will be the same as the chosen target H_f.

Work: The work to perform the herding attack on F-Hash is the work required to build the CCS plus the effort to solve the system of equations due to the CCS plus the work required to perform the herding attack on the Merkle-Damgård hash functions from [137]. This equals $t \times 2^{t/2} + 2^{t/2+d/2+2} + 2^{t-d}$ compression function computations and t^3 bit-XOR operations assuming that only the outermost 2^d chaining values are used for searching in the diamond structure. If all the $2^{d+1}-2$ intermediate chaining values are used for searching in the diamond structure then the work required equals $t \times 2^{t/2} + 2^{t/2+d/2+2} + d \times 2^{t/2+1} + 2^{t-d-1}$ compression function computations and t^3 bit-XOR operations.

D.1.5 Herding Attack on 3C-GOST-L

The following steps outline the herding attack on a t-bit **3C-GOST-L** hash function H:

1. Construct a 2^d hash value wide diamond structure for H as described in chapter 2 and output the final hash value H_f. The final hash value H_f, which is the output of the compression function g, is computed using any of the possible 2^{d-1} checksum values of the two chains or some value chosen arbitrarily. Let h_c be that checksum value.

2. Build the two CCSs using a 2^t multicollision over 2-block messages. Let H_t be the chaining value on the cascade chain after the 2^t 2-block multicollision on H.

3. When challenged with the prefix message P, process P using H_t as the starting chaining value on the cascade chain. Let $H(H_t, P) = H_p$.

4. Find the linking message M_{link} such that the state $H(H_p, M_{link})$ matches one of the 2^d outermost intermediate chaining values on the cascade chain in the diamond structure. If the match is compared to all of the $2^{d+1} - 2$ intermediate chaining values then a $(1, d+1)$-expandable message must be produced at the end of the diamond structure ensuring that the final herded message is always a fixed length.

5. Use the two CCSs computed in step 2 to force the checksum of the herded message P to h_c using the techniques to bypass the two checksums in the 2^t multicollision described in section 4.3.7. Let M_{pref} be the prefix obtained after solving the two systems of equations due to the CCSs from both the chains.

6. Finally, output the message $M = M_{pref}||P||M_{link}||M_d$ where M_d are the message blocks which contributes to the construction of the diamond structure. The value $H(M)$ will be the same as the chosen target H_f.

Work: The work to perform the herding attack on **3C-GOST-L** is $t2^{t/2} + 2^{t/2+d/2+2} + 2^{t-d}$ compression function computations and $2 \times t^3$ bit-XOR operations assuming that only the outermost 2^d chaining values are used for searching in the diamond structure. If all the $2^{d+1} - 2$ intermediate chaining values are used for searching in the diamond structure then the work required equals $t \times 2^{t/2} + 2^{t/2+d/2+2} + d \times 2^{t/2+1} + 2^{t-d-1}$ compression function computations and $2 \times t^3$ bit-XOR operations.

Appendix E

CAVE Tables

Table E.1: The High and Low CAVE Tables, In Hex Notation

hi/lo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D9	23	5F	E6	CA	68	97	B0	7B	F2	0C	34	11	A5	8D	4E
1	0A	46	77	8D	10	9F	5E	62	F1	34	EC	A5	C9	B3	D8	2B
2	59	47	E3	D2	FF	AE	64	CA	15	8B	7D	38	21	BC	96	00
3	49	56	23	15	97	E4	CB	6F	F2	70	3C	88	BA	D1	0D	AE
4	E2	38	BA	44	9F	83	5D	1C	DE	AB	C7	65	F1	76	09	20
5	86	BD	0A	F1	3C	A7	29	93	CB	45	5F	E8	10	74	62	DE
6	B8	77	80	D1	12	26	AC	6D	E9	CF	F3	54	3A	0B	95	4E
7	B1	30	A4	96	F8	57	49	8E	05	1F	62	7C	C3	2B	DA	ED
8	BB	86	0D	7A	97	13	6C	4E	51	30	E5	F2	2F	D8	C4	A9
9	91	76	F0	17	43	38	29	84	A2	DB	EF	65	5E	CA	0D	BC
A	E7	FA	D8	81	6F	00	14	42	25	7C	5D	C9	9E	B6	33	AB
B	5A	6F	9B	D9	FE	71	44	C5	37	A2	88	2D	00	B6	13	EC
C	4E	96	A8	5A	B5	D7	C3	8D	3F	F2	EC	04	60	71	1B	29
D	04	79	E3	C7	1B	66	81	4A	25	9D	DC	5F	3E	B0	F8	A2
E	91	34	F6	5C	67	89	73	05	22	AA	CB	EE	BF	18	D0	4D
F	F5	36	AE	01	2F	94	C3	49	8B	BD	58	12	E0	77	6C	DA

The CAVE tables are shown in Table E.1. Each CAVE table has 256 4-bit values, with each value appearing exactly 16 times each. The CAVE tables are used separately as substitution boxes. Each entry in the table shows the hexadecimal output for both the high (on the left) and the low cave table (on the right).

Appendix F

Application based on HMAC-SHA-1

Consider the vehicle to remote database application of ISO 15764 (Road Vehicles: Extended Data Link Security) [121]. This application uses HMAC-SHA-1 for entity authentication and data integrity security services to prevent replay attacks. The link between the vehicle and the external test equipment is local to the terminal. It is therefore under the control of the user who will be identified to the server before secure information is processed. The communication chain requires the following security services: entity authentication and data integrity to prevent replay attack, confidentiality to prevent eavesdropping and non-repudiation to prevent the user later denying establishing the link, thereby linking the user to the audit trail. If the HMAC-SHA-1 for authentication and integrity services is not one-way and collision resistant for the user knowing the key, then signing the final message with the RSA private key will not provide non-repudiation for the entire message stream [198]. Considering the recent attacks on SHA-1 [287, 290], one would require the collision resistance property of the underlying hash function in the MAC schemes like HMAC and NMAC to provide additional protection to the application against insiders. Note that this is not the motivation behind the proposals NMAC and HMAC. Nevertheless, as pointed out by Preneel [228, p.20], if some additional protection against insider attacks is obtained from the protection of the MAC, the MAC function must be naturally collision resistant.

Appendix G

Implementation Issues

Tables G.1 and G.2 compare the throughputs of MD5 and SHA-1 respectively, against their HMAC, NMAC and O-NMAC functions. The metrics are obtained on a Pentium-M machine with a CPU speed of 2 GHz, by hashing messages of the given payload repeatedly until one gigabit of digest material has been acquired.

For each hash function, NMAC outperforms HMAC and O-NMAC for short messages. HMAC is slower than NMAC because practical implementations require one additional execution of the compression function, the effect of which diminishes with longer messages. O-NMAC suffers from the requirement that a minimum of three blocks (ie. 192 bytes) be hashed, irrespective of the payload. For payloads in excess of this amount, none of the schemes shown in the tables are especially distinguishable via their throughputs.

An increase in the state size of O-NMAC and its associated hash functions is required to hold the additional chaining variable. For O-NMAC based on MD5, this increase in size amounts to 17%. For O-NMAC based on SHA-1, the chain represent an overhead of 4% . This is a minor disadvantage compared to HMAC and NMAC, neither of which increase the memory requirements of the schemes (except to store the additional key).

Table G.1: Time in seconds taken to hash one gigabit of data using MD5 and MACs based on it

Payload (bytes)	MD5	MD5-HMAC	MD5-NMAC	MD5-O-NMAC
16	4.53	13.79	8.91	11.72
32	2.11	6.36	4.32	5.33
64	1.95	4.06	3.05	2.99
128	1.61	2.66	2.17	1.85
256	1.43	1.97	1.71	1.55
1K	1.30	1.44	1.37	1.33
1M	1.25	1.26	1.26	1.25
1G	1.25	1.25	1.25	1.25

Table G.2: Time in seconds taken to hash one gigabit of data using SHA-1 and MACs based on it

Payload (bytes)	SHA1	SHA1-HMAC	SHA1-NMAC	SHA1-O-NMAC
16	7.73	26.0	18.13	24.34
32	3.61	12.35	8.51	11.99
64	3.37	7.79	5.81	6.51
128	2.67	4.89	3.92	3.7
256	2.33	3.43	2.95	2.58
1K	2.08	2.34	2.22	2.14
1M	1.99	2.00	1.99	1.99
1G	1.99	2.00	2.00	2.02

Appendix H

12 Provably Secure PGV Compression Functions

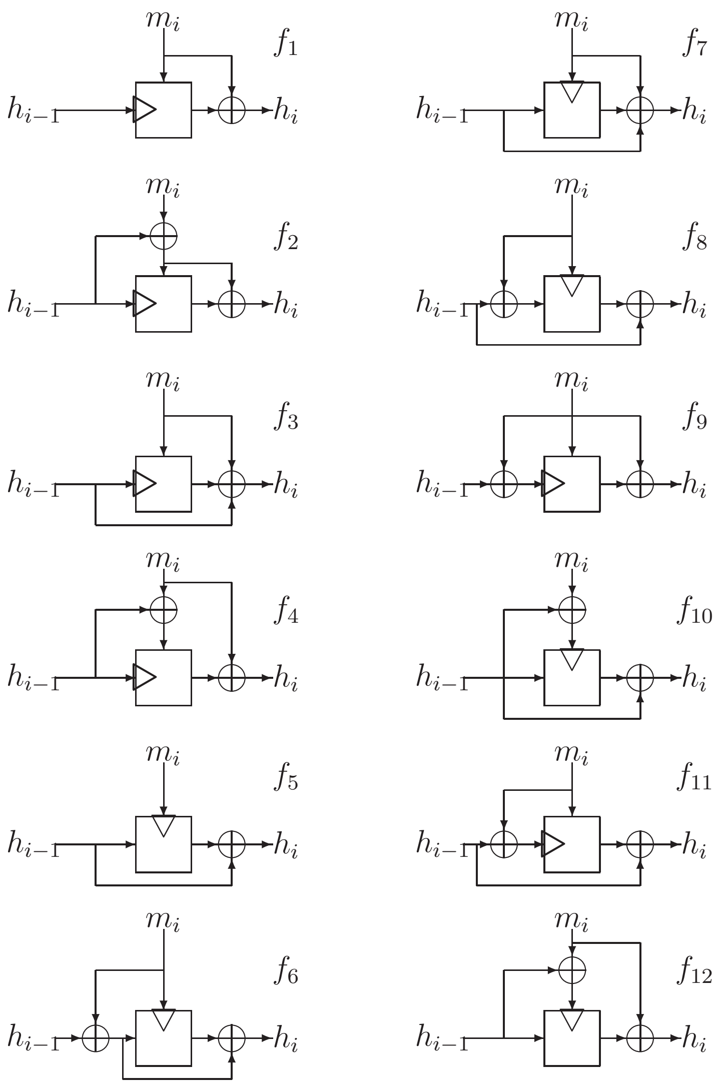


Figure H.1: Compression functions based on PGV construction

Bibliography

[1] Apache HTTP Server Project. This is the homepage of Apache HTTP server. This homepage is available at `http://httpd.apache.org/`. Last access date: 7th of January 2007.

[2] Apache HTTP Server Source Code Distributions. This download page of source code is on the Apache website. This link is available at `http://www.apache.org/dist/httpd/`. Last access date: 7th of January 2007.

[3] Donoghue v. Stevenson (nee Macalister), AC 532. All ER In the locus classicus case for the modern day action for negligence in the UK and in the US the locus classicus case is Palsgraf v. Long Island Railroad Co. 248 N.Y. 339, 162 N.E. 99 (1928), 1932.

[4] Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: BEAR and LION. In Dieter Grollman, editor, *Fast Software Encryption: Third International Workshop*, volume 1039 of *Lecture Notes in Computer Science*, pages 113–120, 21–23 February 1996.

[5] American National Standards Institute (ANSI). *ANSI X9.9: Financial Institution Message Authentication (Wholesale)*, 1986.

[6] American National Standards Institution (ANSI). ANSI X9.71, Keyed Hash Message Authentication Code, 2000.

[7] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis. *Lecture Notes in Computer Science*, 2012:39–56, 2001.

[8] Australian Government Attorney-General's Department (AGD). *Electronic Transactions Act (1999)*, 1999. The pdf version of the document is available at `http://scaleplus.law.gov.au/html/comact/10/6074/pdf/162of99.pdf` Last access date: 13th of January, 2006.

[9] Hagai Bar-El. Introduction to Side Channel Attacks. Technical report, Discretix-Embedding Security Solutions. This white paper is available at `http://www.discretix.com/wp.shtml`. Last access date: 1st of January 2007.

[10] Paulo Barreto. Personal Communication, March 2006.

[11] Mihir Bellare. Personal Communication, December 2004.

[12] Mihir Bellare. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In Cynthia Dwork, editor, *Advances in Cryptology—CRYPTO '06*, volume 4117 of *Lecture Notes in Computer Science*. Springer-Verlag, 20–24 August 2006. Full version of the paper is available at `http://www-cse.ucsd.edu/users/mihir/papers/hmac-new.html`. Last access date: 1st of December 2006.

[13] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 18–22 August 1996. Full version of the paper is available at `http://www-cse.ucsd.edu/users/mihir/papers/hmac.html`. Last access date: 9th of July 2005.

[14] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, FOCS'96 (Burlington, VT, October 14-16, 1996)*, pages 514–523. IEEE Computer Society, IEEE Computer Society Press, 1996.

[15] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of Cipher Block Chaining. In Yvo G. Desmedt, editor, *Advances in Cryptology - Crypto 94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer Verlag, 1994.

[16] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[17] Mihir Bellare and Tadayoshi Kohno. Hash Function Balance and Its Impact on Birthday Attacks. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 401–418. Springer, 2004. This paper is available at `http://www-cse.ucsd.edu/users/mihir/papers/balance.html`. Last access date: 4th of December 2006.

[18] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *Advances in Cryptology: Proceedings of EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192, 1997. The full version of the paper is available at `http://www-cse.ucsd.edu/~mihir/papers/incremental.html`. Last access date: 19th of September 2006.

[19] Steven Bellovin and Eric Rescorla. Deploying a New Hash Algorithm. Technical report, National Institute of Standards and Technology NIST, October 2005. This paper is available at `http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm`. Last access date: 23rd of January 2007.

[20] Thomas A. Berson. Differential cryptanalysis mod 2^{32} with applications to MD5. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 71–80. Springer-Verlag, 24–28 May 1992.

[21] Thomas Beth. Zur Sicherheit der Informationstechnik. *Informatik Spektrum*, 13:204–215, 1990.

[22] Celeste Biever. Hashing exploit threatens digital security. An article in the New Scientist Magazine, 2005. The note is available at `http://www.newscientist.com/article.ns?id=dn7519`. Last access date: 23rd of January 2007.

[23] Eli Biham. A note on comparing the AES candidates. In National Institute of Standards and Technology, editor, *Second AES Candidate Conference*

Proceedings, March 22–23, 1999, Rome, Italy, Gaithersburg, MD, USA, March 1999. National Institute for Standards and Technology.

[24] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matt Franklin, editor, *Advances in Cryptology-CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, August 15–19 2004.

[25] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matt Franklin, editor, *Advances in Cryptology-CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, August 15–19 2004.

[26] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

[27] Eli Biham, Rafi Chen, Antonie Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

[28] Eli Biham, Orr Dunkelman, and Nathan Keller. Rectangle Attacks on 49-Round SHACAL-1. In Thomas Johansson, editor, *10th International Workshop on Fast Software Encryption (FSE)*, volume 2887 of *Lecture Notes in Computer Science*, pages 22–35. Springer, 2003.

[29] Eli Biham and Vladimir Furman. Impossible differential on 8-round mars' core. In NIST, editor, *The Third Advanced Encryption Standard Candidate Conference*, pages 186–194. National Institute for Standards and Technology, 2000. This paper is available at `http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/papers/07-ebiham.pdf`. Last access date: 23$^{\text{rd}}$ of December 2006.

[30] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems (Extended Abstract). In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes*

in Computer Science, pages 2–21. Springer-Verlag, 1991, 11–15 August 1990.

[31] Eli Biham and Adi Shamir. Differential cryptanalysis of Feal and N-Hash. In D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 8–11 April 1991.

[32] Alex Biryukov. Block ciphers and stream ciphers: The state of the art. Cryptology ePrint Archive, Report 2004/094, 2004. This paper is available at `http://eprint.iacr.org/2004/094.pdf`. Last access date: 23$^{\text{rd}}$ of December 2006.

[33] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In Ronald Cramer, editor, *Advances in Cryptology—EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 526–541. Springer-Verlag, 22–26 May 2005.

[34] John Black and Phillip Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. In *CRYPTO: Proceedings of Crypto*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215, 2000.

[35] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[36] George Blakely. Safeguarding Cryptographic Keys. In *Proceedings of AFIPS*, volume 48, pages 313–317, June 1978.

[37] Matt Blaze and Bruce Schneier. The MacGuffin Block Cipher Algorithm. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, pages 97–110. Springer-Verlag, 14–16 December 1994.

[38] C. Burwick, D. Coppersmith, E. D'Avignon, R.Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N.Zunic. MARS — A candidate cipher for AES. NIST AES Proposal, June 1998.

[39] Christian Cachin. An Information-Theoretic Model for Steganography. In David Aucsmith, editor, *Information Hiding: Second International Workshop*, volume 1525 of *Lecture Notes in Computer Science*, pages 306–318. Springer-Verlag, Berlin, Germany, 1998.

[40] Ran Canetti. Personal Communication, June 2005.

[41] Christophe De Canniere. Guess and Determine Attack on SOBER. This paper is available at `citeseer.ist.psu.edu/549458.html`. Last access date: 22nd of August 2006.

[42] Florent Chabaud and Antoine Joux. Differential collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71, 1998.

[43] David L. Chaum. Verification by Anonymous Monitors. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81*, pages 138–139. U.C. Santa Barbara Dept. of Elec. and Computer Eng., 1982. Tech Report 82-04.

[44] Benny Chor, Amos Fiat, and Moni Naor. Tracing Traitors. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer-Verlag, 21–25 August 1994.

[45] Cisco. MD5 File Validation. A document on MD5 File Validaion available on Cisco Website, 2002. This article is available at `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t4/ft_md5v.htm`. Last access date: 9th of January 2007.

[46] Scott Contini, Ronald L. Rivest, M.J.B. Robshaw, and Yiqun Lisa Yin. The security of the RC6 block cipher, 1998. The paper is available at `citeseer.ist.psu.edu/contini98security.html/`.

[47] Scott Contini and Yiqun Lisa Yin. Forgery and partial key-recovery attacks on hmac and nmac using hash collisions. To appear in AsiaCrypt 2006, 2006. A full version of the paper is available at `http://eprint.iacr.org/2006/319`. Last access date: 29th of September 2006.

[48] FastSum Integrity Control. Data Invariability and Integrity Control. Internet article on a tool which MD5 for integrity purposes, 2004. This article is available at `http://www.fastsum.com/` Last access date: 9th of January 2007.

[49] D. Coppersmith. Two Broken Hash Functions. Technical Report IBM Research Report RC-18397, IBM Research Center, October 1992.

[50] Jean-Sebastien Coron, Yevgeniy Dodis, Cecile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005, 14–18 August 2005.

[51] Joan Daemen. *Hash Function and Cipher Design: Strategies Based on Linear and Differential Cryptanalysis*. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, March 1995.

[52] Joan Daemen and Craig Clapp. Fast hashing and stream encryption with PANAMA. In Serge Vaudenay, editor, *Fast Software Encryption: 5th International Workshop*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74, 1998.

[53] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.

[54] Ivan Damgård. Personal Communication, September 2005.

[55] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr-Universität Bochum, 2005. This thesis is available at `http://www.cits.ruhr-uni-bochum.de/personen/daum.html`. Last access date: 4th of December 2006.

[56] Magnus Daum and Stefan Lucks. Hash Collisions (The Poisoned Message Attack) "The Story of Alice and her Boss". Presented at the Rump Session of EUROCRYPT 2005, May 2005. The result is available at `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/`. Last access date: 23rd of January 2007.

[57] Donald Davies and Wyn Price. Digital Signatures, An Update. In *Proceedings of 5th International Conference on Computer Communications*, pages 845–849, October 1984.

[58] Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999.

[59] Bert den Boer and Antoon Bosselaers. An attack on the last two rounds of MD4. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 194–203. Springer-Verlag, 1992, 11–15 August 1991.

[60] Bert den Boer and Antoon Bosselaers. Collisions for the compression function of MD5. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 1994, 23–27 May 1993.

[61] Yvo Desmedt. Security Goals in Cryptography. Presented at Queensland University of Technology (QUT) on his visit, July 2005.

[62] Yvo Desmedt. The Future of Cryptography for the Next 25 Years. Presented at Queensland University of Technology (QUT) on his visit, August 2005.

[63] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. Internet Request for Comment RFC 2246, Internet Engineering Task Force (IETF), January 1999. This RFC Proposed Standard is available at `http://www.ietf.org/rfc/rfc2246.txt`. Last access date: 9th of June 2006.

[64] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. Internet Request for Comment RFC 2246, Internet Engineering Task Force (IETF), January 1999. Proposed Standard.

[65] Whitfield Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76:560–576, 1988.

[66] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, November 1976.

[67] Hans Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption (FSE)*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996.

[68] Hans Dobbertin. Cryptanalysis of MD5 compress. Presented at the Rump session of Euro Crypto'96 Rump Session, 1996.

[69] Hans Dobbertin. The status of MD5 after a recent attack. *CryptoBytes*, 2(2):1, 3–6, Summer 1996.

[70] Hans Dobbertin. RIPEMD with two-round compress function is not collision-free. *Journal of Cryptology*, 10(1):51–70, Winter 1997.

[71] Hans Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.

[72] Hans Dobbertin. The first two rounds of MD4 are not one-way (extended abstract). In Serge Vaudenay, editor, *Fast Software Encryption: 5th International Workshop*, volume 1372 of *Lecture Notes in Computer Science*, pages 284–292. Springer-Verlag, 23–25 March 1998.

[73] Morris Dworkin. Recommendation for Block Cipher Modes of Operation; Methods and Techniques. NIST Special Publication 800-38A 2001 Edition, 2001. This publication available at `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf` Last access date:26 March, 2006.

[74] Morris Dworkin. Recommendation for block cipher modes of operation: The CMAC mode for authentication. NIST Special Publication 800-38B, May 2005. This publication is available at the link `http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf`. Last access date: 29th of November 2006.

[75] ECRYPT. Call for stream cipher primitives, April 2005. The technical background on the eSTREAM process is avaialable at`http://www.ecrypt.eu.org/stream/call/`. Last access date: 27th of September 2006.

[76] A. Evans, W. Kantrowitz Jr., and E. Weiss. A High Security Log-in Procedure. *Communications of the ACM*, 17:442–445, August 1974.

[77] A. Evans, W. Kantrowitz Jr., and E. Weiss. A User Authentication System Not Requiring Secrecy in the Computer. *Communications of the ACM*, 17:437–442, August 1974.

[78] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5), May 1973.

[79] Niels Ferguson and Bruce Schneier. *Practical Cryptography*, chapter 8, pages 83–96. John Wiley & Sons, 2003.

[80] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[81] Decio Gazzoni Filho, Paulo Barreto, and Vincent Rijmen. The Maelstrom-0 Hash Function. Published at 6th Brazilian Symposium on Information and Computer System Security, 2006.

[82] Federal Information Processing Standard (FIPS). *The Keyed-Hash Message Authentication Code (HMAC)*. National Institute for Standards and Technology (NIST), March 2002. This standard is available at `http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf`. Last access date: 4th of December 2006.

[83] Joanne Fuller, William Millan, and Ed Dawson. Multi-objective Optimisation of Bijective S-boxes. In *CEC 2004: International Conference on Evolutionary Computation*. IEEE, 2004.

[84] Praveen Gauravaram, Shoichi Hirose, and Suganya Annadurai. An Update on the Analysis and Design of NMAC and HMAC functions. To be published in the International Journal of Network Security (IJNS), 2007.

[85] Praveen Gauravaram, Adrian McCullagh, and Ed Dawson. Collision Attacks on MD5 and SHA-1: Is this the "Sword of Damocles" for Electronic Commerce? In Andrew Clark and Mark McPherson and George Mohay, editor, *AusCERT Asia Pacific Information Technology Security Conference Refereed R & D Stream.*

[86] Praveen Gauravaram, Adrian McCullagh, and Ed Dawson. The legal and practical implications of recent attacks on 128-bit cryptographic hash functions. *First Monday*, 11(1), 2006.

[87] Praveen Gauravaram and William Millan. Improved Attack on the Cellular Authentication and Voice Encryption Algorithm. In *International Workshop on Cryptographic Algorithms and their Uses*, 5-6 2004.

[88] Praveen Gauravaram, William Millan, Ed Dawson, Matt Henricksen, Juanma Gonzalez Nieto, and Kapali Viswanathan. Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction (full version). Technical Report QUT-ISI-TR-2006-013, Information Security Institute (ISI), Queensland University of Technology (QUT), July 2006. This technical report is available at `http://www.isi.qut.edu.au/research/publications/technical/qut-isi-tr-2006-013.pdf`. Last access date: 18th of August 2006.

[89] Praveen Gauravaram, William Millan, Ed Dawson, and Kapali Viswanathan. Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction. In *Australasian Conference on Information Security and Privacy (ACISP)*, volume 4058 of *Lecture Notes in Computer Science*, pages 407–420, 2006.

[90] Praveen Gauravaram, William Millan, and Lauren May. CRUSH: A New Cryptographic Hash Function using Iterated Halving Technique. In *Proceedings of the workshop on Cryptographic Algorithms and their uses*, pages 28–39, Goldcoast, Australia, July 4–5 2004.

[91] Praveen Gauravaram and Katsuyuki Okeya. Security Analysis on Keyed Hash Functions from the Viewpoint of Side Channel Attacks. To be presented at the Symposium on Cryptography and Information Security (SCIS), Sasebo, Japan, January, 23-26 2007.

[92] Ed Gerck. Overview of certification systems: X.509, PKIX, CA, PGP and SKIP. *The Bell*, 1(3):8, July 2000. This article is available at `http://www.thebell.net/papers/certover.pdf`. Last access date: 23rd of January 2007.

[93] Anthony Giddens. The consequences of modernity. Stanford University Press, 1990.

[94] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane. Codes which detect deception. *Bell System Tech. J.*, 53:405–424, 1974.

[95] Henri Gilbert and Helena Handschuh. Security analysis of SHA-256 and sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 175–193. Springer, 2004.

[96] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions (extended abstract). In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 276–288. Springer-Verlag, 1985, 19–22 August 1984.

[97] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[98] Peter Gutmann, David Naccache, and Charles Palmer. When Hashes Collide. *Security & Privacy Magazine, IEEE*, 3(3):68–71, May-June 2005.

[99] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 437–455. Springer-Verlag, 1991, 11–15 August 1990.

[100] Helena Handschuh, Lars R. Knudsen, and Matthew J. B. Robshaw. Analysis of SHA-1 in Encryption Mode. In David Naccache, editor, *CTRSA: CT-RSA, The Cryptographers' Track at RSA Conference*, volume 2020 of *Lecture Notes in Computer Science*, pages 70–83. Springer-Verlag, 2001.

[101] Helena Handschuh and David Naccache. SHACAL: A Family of Block Ciphers. Submission to the NESSIE Project, 2002. This article is available at `https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions.html`. Last access date: 23$^{\text{rd}}$ of January 2007.

[102] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), November 1998. This standards track RFC document is available at `http://www.ietf.org/rfc/rfc2409.txt`. Last access date: 9th of June 2006.

[103] Johan Hastad and Mats Naslund. Key Feedback Mode: A Keystream Generator with Provable Security. First Modes of Operation Workshop for Symmetric Key Block Ciphers, Baltimore, Maryland, USA, October 2000. This paper was found at `http://csrc.nist.gov/CryptoToolkit/modes/workshop1/`. Last access date: 9th of January 2006.

[104] Philip Hawkes, Michael Paddon, and Gregory Rose. The Mundja Streaming MAC. This work was presented at the ECRYPT Network of Excellence in Cryptology workshop on the State of the Art of Stream Ciphers, October 2004, Brugge, Belgium, 2004. The paper is available at `http://eprint.iacr.org/2004/271`. Last access date: 29th of September 2006.

[105] Philip Hawkes, Michael Paddon, and Gregory G. Rose. Musings on the Wang et al. MD5 Collision. Cryptology ePrint Archive, Report 2004/264, 2004. This article is available at `http://eprint.iacr.org/2004/264`. Last access date: 7th of January 2007.

[106] Philip Hawkes and Gregory G. Rose. Guess-and-Determine Attacks on SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 37–46. Springer, 2002.

[107] Yong-Sork Her, Kouichi Sakurai, and Shin-Hwan Kim. Attacks for Finding Collision in Reduced Versions of 3-PASS and 4-PASS HAVAL. In *Proceedings of International Conference on Computers, Communications and Systems ICCCS*, pages 75–78, February 2003.

[108] Shoichi Hirose. Secure Block Ciphers Are Not Sufficient for One-Way Hash Functions in the Preneel-Govaerts-Vandewalle Model. In *SAC: Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 339–352. Springer-Verlag, 2002.

[109] Shoichi Hirose. A note on the strength of weak collision resistance. *The Institute of Electronics, Information and Communication Engineers (IEICE) Transaction Fundamentals*, E87-A(5):1092–1097, 2004.

[110] Shoichi Hirose. Personal Communication, January 2006.

[111] Jonathan Hoch and Adi Shamir. Breaking the ICE: Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. To appear in the Proceedings of 13th Annual Fast Software Encryption (FSE) International Conference, 2006.

[112] Paul Hoffman and Bruce Schneier. RFC 4270: Attacks on cryptographic hashes in internet protocols, November 2005. This Informational RFC draft is available at `http://www.rfc-archive.org/getrfc.php?rfc=4270`. Last access date: 11th of December 2006.

[113] Seokhie Hong, Jongsung Kim, Guil Kim, Jaechul Sung, Changhoon Lee, and Sangjin Lee. Impossible Differential Attack on 30-Round SHACAL-2. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT, 4th International Conference on Cryptology in India*, volume 2904 of *Lecture Notes in Computer Science*, pages 97–106. Springer, 2003.

[114] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile, January 1999. Status: PROPOSED STANDARD.

[115] Hideki Imai and Atsuhiro Yamagishi. CRYPTREC Project - Cryptographic Evaluation Project for the Japanese Electronic Government. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 399–400. Springer, 2000.

[116] In Re Verizone, Maine Public Utilities Commission. Docket No. 2000-849.

[117] Japan Information-Technology Promotion Agency (IPA). CRYPTREC PROJECT. This website can be accessed at `http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html`. Last access date: 8th of January 2007.

[118] ISO. *Information Processing Systems — Open Systems Interconnection (OSI) — Basic Reference Model- Part 2: Security Architecture*, 1987.

[119] ISO. *Information technology-Security techniques- Message Authentication Codes (MACs)- Part 2: Mechanisms using a Dedicated Hash-Function*, August 2002.

[120] ISO. *Information Technology- Security Techniques- Hash functions: Part 3-Dedicated Hash Functions*, 2003. The document is available at `http://www.ncits.org/ref-docs/FDIS_10118-3.pdf`. Last access date: 9th of December 2006.

[121] ISO. *Road Vehicles Extended Data Link security*, 2004. Published Standard.

[122] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In *Fast Software Encryption (FSE),2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003. For OMAC page, see the link `http://www.nuee.nagoya-u.ac.jp/labs/tiwata/omac/omac.html`.Last access date:26th of March, 2006.

[123] Don Johnson. Hash Function Lifesycles and Future Resiliency. Technical report, National Institute of Standards and Technology NIST, October 2005. This paper is available at `http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm`. Last access date: 20th of January 2007.

[124] Don Johnson. Improving Hash Function Padding. Technical report, National Institute of Standards and Technology NIST, October 2005. This paper is available at `http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm`. Last access date: 20th of January 2007.

[125] Daniel Joscak. Finding Collisions in Cryptographic Hash Functions. Master's thesis, Department of Algebra, Univerzita Karlova, 2006. This master's thesis is available at `http://cryptography.hyperlink.cz/2006/diplomka.pdf`. Last access date: 28th of December 2006.

[126] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matt Franklin, editor, *Advances in Cryptology-CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, August 15–19 2004.

[127] Charanjit S. Jutla and Anindya C. Patthak. Is SHA-1 conceptually sound? Cryptology ePrint Archive, Report 2005/350, 2005. This paper is available at http://eprint.iacr.org/2005/350. Last access date: 20th of January 2007.

[128] Charanjit S. Jutla and Anindya C. Patthak. A simple and provably good code for SHA message expansion. Cryptology ePrint Archive, Report 2005/247, 2005. This paper is available at http://eprint.iacr.org/2005/247. Last access date: 20th of January 2007.

[129] Burt Kaliski and Matt Robshaw. Message authentication with MD5. *CryptoBytes*, 1(1):5–8, Spring 1995.

[130] Dan Kaminsky. MD5 To Be Considered Harmful Someday. Cryptology ePrint Archive, Report 2004/357, 2004. This paper is available at http://eprint.iacr.org/2004/357. Last access date: 9th of January 2007.

[131] P.R. Kasselman. A Fast Attack on the MD4 Hash Function. In *Proceedings of the 1997 South African Symposium on Communications and Signal Processing (COMSIG'97)*, pages 147–150. IEEE, September 1997.

[132] P.R. Kasselman and W.T Penzhorn. Cryptanalysis of Reduced Version of HAVAL. *Electronic Letters*, 36(1):30–31, January 2000.

[133] John Kelsey. Comments in the Crypto Forum Research Group (CFRG), October 2005. These comments can be found at http://www1.ietf.org/mail-archive/web/cfrg/current/msg01126.html. Last access date: 4th of December 2006.

[134] John Kelsey. Possibly new result on truncating hashes. CFRG Email-list archive, August 2005. This attack note is available at http://www.mail-archive.com/cryptography@metzdowd.com/msg04468.html. Last access date: 8th of December 2006.

[135] John Kelsey. Personal Communication, 15th of June 2006.

[136] John Kelsey and Praveen Gauravaram. Linear checksums Don't Block Generic Damgård-Merkle Attacks. Presented at Crypto'06 Rump Session, 2006.

[137] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Serge Vaudenay, editor, *Advanes in Cryptology-EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.

[138] John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2n̂ Work. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.

[139] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, and Dieter Gollmann, editors, *Computer Security - ESORICS 98, 5th European Symposium on Research in Computer Security*, volume 1485 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 1998.

[140] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2/3), 2000.

[141] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the Security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1. Cryptology ePrint Archive, Report 2006/187, 2006. The full version is available at `http://eprint.iacr.org/2006/187`. Last access date: 14th of December 2006.

[142] Jongsung Kim, Alex Biryukov, Bart Preneel, and Sangjin Lee. On the Security of Encryption Modes of MD4, MD5 and HAVAL. Cryptology ePrint Archive, Report 2005/327, 2005. This article is available at `http://eprint.iacr.org/2005/327`. Last access date: 23rd of January 2007.

[143] Jongsung Kim, Dukjae Moon, Wonil Lee, Seokhie Hong, Sangjin Lee, and Seokwon Jung. Amplified boomerang attack against reduced-round SHACAL. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 243–253. Springer, 2002.

[144] Vlastimil Klima. Finding MD5 collisions A toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 2005. `http://eprint.iacr.org/2005/075`. Last access date: 9$^{\text{th}}$ of January, 2007.

[145] Vlastimil Klima. Finding MD5 Collisions on a Notebook PC using Multi-message Modifications. Cryptology ePrint Archive, Report 2005/102, April 2005. This paper is available at `http://eprint.iacr.org/2005/102.pdf`. Last access date: 8$^{\text{th}}$ of January 2007.

[146] Lars Knudsen. *Block Ciphers: Analysis, Design and Applications.* PhD thesis, Arhus University, 1994.

[147] Lars Knudsen. Personal Communication, March 2006.

[148] Lars Knudsen and Frederic Muller. Some Attacks Against a Double Length Hash Proposal. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 462–473. Springer-Verlag, 2005.

[149] Lars R. Knudsen. SMASH - A Cryptographic Hash Function. In Henri Gilbert and Helena Handschuh, editors, *12th International Workshop On Fast Software Encryption (FSE)*, volume 3557 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2005.

[150] Lars R. Knudsen, Xuejia Lai, and Bart Preneel. Attacks on Fast Double Block Length Hash Functions. *J. Cryptology*, 11(1):59–72, 1998.

[151] Lars R. Knudsen and Bart Preneel. Hash functions based on block ciphers and quaternary codes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT*, volume 1163 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 1996.

[152] Lars R. Knudsen and David Wagner. On the structure of Skipjack. *Discrete Applied Mathematics*, 111(1-2):103–116, 2001.

[153] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 18–22 August 1996.

[154] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[155] Loren M. Kohnfelder. Towards a practical public-key cryptosystem. B.S. Thesis, supervised by L. Adleman, May 1978.

[156] Korea Information Security Agency. A Design and Analysis of 128-bit Symmetric Block Cipher(SEED), 1999.

[157] Hugo Krawczyk. HMAC with MD5 and SHA-1. CFRG Email-list archive, August 2004. This note is available at `http://www1.ietf.org/mail-archive/web/cfrg/current/msg00527.html`. Last access date: 8th of December 2006.

[158] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. RFC 2104: HMAC: Keyed-hashing for message authentication, February 1997. This RFC document is available at `http://www.ietf.org/rfc/rfc2104.txt`. Last access date: 8th of June 2006.

[159] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. RFC 2104: HMAC: Keyed-hashing for message authentication, 1997.

[160] Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-key CBC MAC. In *CTRSA: CT-RSA, The Cryptographers' Track at RSA Conference*, volume 2612 of *Lecture Notes in Computer Science*, pages 33–39. Springer, 2003.

[161] Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-Key CBC MAC. *IEICE Transactions*, 87-A(1):46–52, 2004.

[162] Hidenori Kuwakado and Hatsukazu Tanaka. New algorithm for finding preimages in reduced version of the MD4 compression function. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, E83-A(1):97–100, 2000.

[163] Xuejia Lai and Lars R. Knudsen. Attacks on Double Block Length Hash Functions. In *Fast Software Encryption*, volume 809 of *Lecture Notes in Computer Science*, pages 157–165. Springer, 1994.

[164] Xuejia Lai and James L. Massey. Hash Functions Based on Block Ciphers. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer-Verlag, 24–28 May 1992.

[165] Duo Lei. F-HASH: Securing Hash Functions Using Feistel Chaining. Cryptology ePrint Archive, Report 2005/430, 2005. The paper is available at `http://eprint.iacr.org/2005/430.pdf`. Last access date: 11th of November 2006.

[166] Duo Lei. New Integrated proof Method on Iterated Hash Structure and New Structures. Cryptology ePrint Archive, Report 2006/147, 2006. The paper is available at `http://eprint.iacr.org/2006/147.pdf`. Last access date: 5th of November 2006.

[167] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A Brief History of the Internet. This article is available at `http://www.isoc.org/internet/history/brief.shtml`. Last access date: 23rd of January 2007.

[168] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[169] Arjen Lenstra and Benne de Weger. On the possibility of constructing meaningful hash collisions for public keys. In Colin Boyd and Juan M. Gonzalez Nieto, editors, *Information Security and Privacy*, volume 3574 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 4–6 2005. Full version of this paper is available at `"http://www.win.tue.nl/~bdeweger/CollidingCertificates/"`. Last access date: 20th of December 2006.

[170] Arjen Lenstra, Xiaoyun Wang, and Benne de Weger. Colliding X.509 Certificates. Cryptology ePrint Archive, Report 2005/067, 2005. This paper is available at `http://eprint.iacr.org/2005/067`. Last access date: 20th of January 2007.

[171] Hendrik W. Lenstra. Integer programming with a fixed number of variables. Technical Report 81-03, University of Amsterdam, Amsterdam, 1981.

[172] Jie Liang and Xuejia Lai. Improved collision attack on hash function MD5. Cryptology ePrint Archive, Report 2005/425, 2005. This paper is available at `http://eprint.iacr.org/2005/425.pdf`. Last access date: 20th of December, 2006.

[173] Stefan Lucks. Design Principles for Iterated Hash Functions. Cryptology ePrint Archive, Report 2004/253, 2004. This paper is available at `http://eprint.iacr.org/2004/253`. Last access date: 27th of November 2006.

[174] Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer-Verlag, 2005.

[175] Stefan Lucks. ICE-EM RNSA 2006 Workshop on Recent Advances in Stream Ciphers and Hash Functions, 28th of June 2006. Queensland University of Technology, Brisbane, Australia.

[176] James Massey. SAFER K-64: A byte-oriented block-ciphering algorithm. In R. Anderson, editor, *Fast Software Encryption (FSE)*, volume 809 of *Lecture Notes in Computer Science*, pages 1–17. Springer Verlag, 1994.

[177] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994, 23–27 May 1993.

[178] S. Matyas, C. Meyer, and J. Oseas. Generating Strong One-way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[179] Adrian McCullagh and William Caelli. Non-Repudiation in the Digital Environment. *First Monday–Peer-Reviewed Journal On the Internet*, 5, August 2000. This article is available at `http://www.firstmonday.dk/issues/issue5_8/mccullagh/` Last access date: 23rd January, 2007.

[180] Adrian McCullagh, William Caelli, and Peter Little. Signature stripping: A digital dilemma. *Journal of Information, Law and Technol-*

ogy, 1. http://www2.warwick.ac.uk/fac/soc/law/elj/jilt/2001_1/mccullagh/ Last access date: 23rd of January, 2007.

[181] Declan McCullagh. Crypto Researchers Abuzz over Flaws. This report was published at the CNET News, 2004. This report is available at http://news.com.com/Crypto%20researchers%20abuzz%20over%20flaws/2100-1002_3-5313655.html/. Last access date: 8th of January 2007.

[182] David McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). NIST special publication, National Institute for Standards and Technology. This standard document is available at http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-revised-spec.pdf. Last access date: 27th of November 2006.

[183] David McNett. RC5-64 HAS BEEN SOLVED, September 2002. The information is obtained through personal communication with Greg Rose in 2004 and see http://www.distributed.net/pressroom/news-20020926.txt for the report. Last access date: 21st of January 2007.

[184] Florian Mendel, Norbert Pramstallar, Christian Rechberger, and Vincent Rijmen. Analysis of a step-reduced SHA-256, 2006. This paper is to be published in the Proceedings of FSE'2006.

[185] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. On the Collision Resistance of RIPEMD-160. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2006.

[186] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, chapter Hash Functions and Data Integrity, pages 321–383. The CRC Press series on discrete mathematics and its applications. CRC Press, 1997.

[187] Ralph Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1989.

[188] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

[189] Philip Metzger and William Simpson. RFC 1828- IP authentication using keyed MD5, August 1995. Status: PROPOSED STANDARD.

[190] C. H. Meyer and S. M. Matyas. *Cryptography: A Guide for the Design and Implementation of Secure Systems*. John Willey and Sons Publications, 1982.

[191] Ondrej Mikle. Practical attacks on digital signatures using MD5 message digest. Cryptology ePrint Archive, Report 2004/356, 2004. This paper is available at `http://eprint.iacr.org/2004/356`. Last access date: 9th of January 2007.

[192] William Millan. Cryptanalysis of the alleged CAVE algorithm. In *The 1st International Conference on Information Security and Cryptology*, volume 1, pages 107–119. Korea Institute of Information Security and Cryptology (KIISC), 18-19 December 1998.

[193] William Millan. Personal Communication, June 2005.

[194] William Millan, Ed Dawson, and L.J.O'Connor. Fast Attacks on Tree-Structured Ciphers. In *Selected Areas in Cryptography*, pages 148–158, 1994.

[195] William Millan and Praveen Gauravaram. Cryptanalysis of the Cellular Authentication and Voice Encryption Algorithm. *IEICE Electronic Express*, 1(15):453–459, 2004.

[196] Ilya Mironov. Hash functions: Theory, Attacks, and Applications. Technical Report MSR-TR-2005-187, Microsoft Research, November 2005. This paper is available at `http://research.microsoft.com/users/mironov/papers/hash_survey.pdf`. Last access date: 20th of January 2007.

[197] Ilya Mironov and Arvind Narayanan. Personal Communication during the rump session of Crypto'06, August 2006.

[198] Chris Mitchell. Personal Communication, August 2005.

[199] Shoji Miyaguchi, Kazuo Ohta, and Masahiko Iwata. Confirmation that Some Hash Functions Are Not Collision Free. In I. B. Damgård, editor, *Advances in Cryptology—EUROCRYPT 90*, volume 473 of *Lecture Notes in Computer Science*, pages 326–343. Springer-Verlag, 1991, 21–24 May 1990.

[200] P. Morin. A critique of BEAR and LION. This paper is available at `http://citeseer.ist.psu.edu/124166.html`. Last access date: 21st of January 2007.

[201] Frédéric Muller. The MD2 Hash Function is not One-way. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2004.

[202] Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta. Improved collision attack on MD4. Cryptology ePrint Archive, Report 2005/151, 2005. The paper is available at `http://eprint.iacr.org/2005/151`. Last access date: 8th of January 2007.

[203] Mridul Nandi and Douglas Stinson. Multicollision attacks on some generalized sequential hash functions. Cryptology ePrint Archive, Report 2006/055, 2006. The paper is available at `http://eprint.iacr.org/2006/055`. Last access date: 19th of September 2006.

[204] National Bureau of Standards (NBS). Specification for the Data Encryption Standard. Federal Information Processing Standards Publication (FIPS PUB) 46, January 1977.

[205] National Institute of Standards and Technology (NIST). *FIPS Publication 81: DES Modes of Operation*, December 2, 1980. Originally issued by National Bureau of Standards.

[206] National Institute of Standards and Technology (NIST). Secure hash standard. Federal Information Processing Standards Publication (FIPS PUB) 180, May 1993.

[207] National Conference of Commissioners On Uniform State Laws. Uniform Electronic Transactions Act (1999), 1999. The pdf version of the document is available at `http://www.law.upenn.edu/bll/ulc/fnact99/1990s/ueta99.pdf` Last access date: 23rd of January 2007.

[208] Network of Excellence in Cryptography (ECRYPT). Recent Collision Attacks on Hash Functions: ECRYPT Position Paper. Technical Report Version 1.1, Katholieke Universiteit Leuven (KUL), February 2005. This draft is available at `http://www.ecrypt.eu.org/documents/STVL-ERICS-2-HASH_STMT-1.1.pdf`. Last access date: 7th of January 2007.

[209] Network of Excellence in Cryptology (IST-2002-507932). eSTREAM - the ECRYPT Stream Cipher Project. The home page for the eSTREAM project can be found at `http://www.ecrypt.eu.org/stream/`. Last access date: 8th of January 2007.

[210] National Institute of Standards and Technology (NIST) Computer Systems Laboratory. Secure hash standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1, April 1995.

[211] National Institute of Standards and Technology (NIST) Computer Systems Laboratory. Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-2, August 2002. This standard document is available at `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf`. Last access date: 8th of January 2007.

[212] National Institute of Standards and Technology (NIST). *FIPS Publication 46: Announcing the Data Encryption Standard*, January 1977. Originally issued by National Bureau of Standards.

[213] National Institute of Standards and Technology (NIST). NIST Brief Comments on Recent Cryptanalytic Attacks on SHA-1, February 2005. This short notice by NIST is available at `http://csrc.nist.gov/news-highlights/NIST-Brief-Comments-on-SHA1-attack.pdf`. Last access date: 20th of December 2006.

[214] National Institute of Standards and Technology (NIST). NIST Comments on Cryptanalytic Attacks on SHA-1, 2005. This short notice

by NIST is available at `http://csrc.nist.gov/CryptoToolkit/shs/NISTHashComments-final.pdf`. Last access date: 20th of January 2007.

[215] National Institute of Standards and Technology (NIST). Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. NIST Special Publication 800-56, National Institute for Standards and Technology, 2005. This draft is available at `http://all.net/books/standards/NIST-CSRC/csrc.nist.gov/publications/drafts.html#draft-SP800-56`. Last access date: 9th of June 2006.

[216] National Institute of Standards and Technology. Data Encryption Standard. Federal Information Processing Standards Publication (FIPS PUB) 46-3, October 1999.

[217] National Institute of Standards and Technology. *FIPS PUB 186-2: Digital Signature Standard (DSS).* National Institute for Standards and Technology, January 2000.

[218] National Institute of Standards and Technology(NIST). Advanced Encryption Standard. The details of AES process can be found at `http://csrc.nist.gov/CryptoToolkit/aes/`. Last access date: 21st of January 2007.

[219] Government Committee of the Russia for Standards. GOST R 34.10-94, Gosudarstvennyi Standard of Russian Federation, Cryptographic Protection for Data Processing Systems Government Committee of USSR for Standards, 1989 (in Russian)., 1989.

[220] Government Committee of the Russia for Standards. GOST R 34.11-94, Gosudarstvennyi Standard of Russian Federation, Information Technology, Cryptographic Data Security, Hashing function, 1994.

[221] Katsuyuki Okeya. Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editor, *Australasian Conference on Information Security and Privacy*, volume 4058 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2006.

[222] Katsuyuki Okeya and Tetsu Iwata. Side Channel Attacks on Message Authentication Codes. In Refik Molva and Gene Tsudik and Dirk Westhoff, editor, *Security and Privacy in Ad-hoc and Sensor Networks, Second European Workshop, ESAS 2005*, volume 3813 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2005.

[223] Sangwoo Park, Soo Hak Sung, Seongtaek Chee, and Jongin Lim. On the Security of Reduced Versions of 3-Pass HAVAL. In Lynn Margaret Batten and Jennifer Seberry, editors, 7^{th} *Australasian Conference on Information Security and Privacy ACISP 2002*, volume 2384 of *Lecture Notes in Computer Science*, pages 406–419. Springer, 2002.

[224] Sarvar Patel. This was an unpublished report, 1995.

[225] Sarvar Patel. An efficient MAC for short messages. In *SAC: Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 353–368. Springer-Verlag, 2002.

[226] Norbert Pramstallar, Christian Rechberger, and Vincent Rijmen. Preliminary analysis of the SHA-256 Message Expansion. Technical report, National Institute of Standards and Technology NIST, October 2005. This paper is available at `http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm`. Last access date: 23^{rd} of January 2007.

[227] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Breaking a new hash function design strategy called SMASH. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2005.

[228] Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[229] Bart Preneel. NESSIE: A European Approach to Evaluate Cryptographic Algorithms. In *IWFSE: International Workshop on Fast Software Encryption*, volume 2355 of *Lecture Notes in Computer Science*, pages 267–276. Springer-Verlag, 2002.

[230] Bart Preneel. NESSIE Portfolio of Recommended Cryptographic Primitives, February 2003. This document can be found at `https://www.cosic.esat.kuleuven.be/nessie/`. Last access date: 8th of January 2007.

[231] Bart Preneel. NESSIE Project Announces Final Selection of Crypto Algorithms, February 2003. This document can be found at `https://www.cosic.esat.kuleuven.be/nessie/`. Last access date: 8th of January 2007.

[232] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer-Verlag, 22–26 August 1993.

[233] Bart Preneel, Bart Van Rompay, Sidi Ben rs, Alex Biryukov, Louis Granboulan, Emmanuelle Dottax, Markus Dichtl, Markus Schafheutle, Pascale Serf, Stefan Pyka, Eli Biham, Elad Barkan, Orr Dunkelman, J. Stolin, Mathieu Ciet, Jean-Jacques Quisquater, Francesco Sica, Hvard Raddum, and Matthew Parker. Final report of European project number IST-1999-12324, named New European Schemes for Signatures, Integrity, and Encryption, April 2004. This document can be found at `https://www.cosic.esat.kuleuven.be/nessie/`. Last access date: 8th of January 2007.

[234] Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 27–31 August 1995.

[235] Bart Preneel and Paul C. van Oorschot. On the security of two MAC algorithms. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 19–32. Springer-Verlag, 12–16 May 1996.

[236] Bart Preneel and Paul C. van Oorschot. On the Security of Iterated Message Authentication Codes. *IEEE Transactions on Information Theory*, 45(1):188–199, 1999.

[237] Rene Purnadi and Giridhar Mandyam. DS-41 and UMTS Intersystem Roaming. In *Wireless Communications and Networking Conference, WCNC*, volume 3, pages 1575–1577. IEEE, 2000.

[238] Frank Quick. Common Cryptographic Algorithms- 3rd Generation Partnership Project 2 (3GPP2), January 2002. The document is available at `http://www.3gpp2.org/Public_html/specs/S.S0053-0_v1.0.pdf`. Last access date: 8th of January 2007.

[239] Frank Quick. Personal Communication, June 2004.

[240] Jean-Jacques Quisquater. Side Channel Attacks: State-of-the-art, 2004. This paper is available at `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1047_Side_Channel_report.pdf`. Last access date: 31st of December 2006.

[241] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[242] Doug Rahikka. NIST- Key Management Standard. User Perspectives/Requirements:Wirless Applications, February 2000. These slides are available at `http://csrc.nist.gov/CryptoToolkit/kms/wireless.pdf`. Last access date: 31st of December 2006.

[243] James Randall. Hash Functions Update Due to Potential Weakness Found in SHA-1. A technical notes in the RSA website, March 2005. The note is available at `http://www.rsasecurity.com/rsalabs/node.asp?id=2834`. Last access date: 23rd of January 2007.

[244] James Randall and Michael Szydlo. Collisions for SHA0, MD5, HAVAL, MD4, and RIPEMD, but SHA1 Still Secure. A technical note on the RSA website, August 2004. The note is available at `http://www.rsasecurity.com/rsalabs/node.asp?id=2738`. Last access date: 23rd of January 2007.

[245] Srivaths Ravi, Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security as a New Dimension in Embedded System Design. In *Proceedings of the 41st Annual conference on Design Automation (DAC-04)*, pages 753–760. ACM Press, June 7–11 2004.

[246] Re Amlink Technologies Pty Ltd and Australian Trade Commission [2005] AATA 359.

[247] Cryptography Research. Hash Collision Question and Answer. Crypto News on Attacks on Hash Functions, 2004. This link is available at `http://www.cryptography.com/cnews/hash.html`. Last access date: 20th December, 2006.

[248] Vincent Rijmen and Paulo S. L. M. Barreto. The WHIRLPOOL hash function. This hash function was adopted as a standard by ISO/IEC 10118-3:2004, 2004. The specification is available at `http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html`. Last access date: 27th of November 2006.

[249] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA, The Cryptographers' Track at the RSA Conference*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2005.

[250] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing collisions for PANAMA. In *IWFSE: 8th International Workshop on Fast Software Encryption*, volume 2355 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2002.

[251] Ronald Rivest. Cryptography. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A. MIT Press/Elsevier, Amsterdam, 1990.

[252] Ronald Rivest. The MD4 message digest algorithm. In *Advances in Cryptology – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311, 1991.

[253] Ronald Rivest. The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force, April 1992. This RFC is available at `http://www.faqs.org/rfcs/rfc1321.html`. Last access date: 20th of November 2006.

[254] Ronald Rivest. RFC 1320: The MD4 Message-Digest Algorithm, April 1992. This standard is available at `http://www.faqs.org/rfcs/rfc1320.html`. Last access date: 5th of December 2006.

[255] Ronald Rivest. The MD5 Message-Digest Algorithm. RFC 1321, MIT, RSA Data Security, April 1992.

[256] Ronald L. Rivest. The RC5 encryption algorithm. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96, Leuven, Belgium, 14–16 December 1994. Springer-Verlag.

[257] Ronald L. Rivest, Leonard Adleman, and Michael L. Dertouzos. On Data Banks and Privacy Homomorphisms. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 169–180. Academic Press, 1978.

[258] Bart Van Rompay, Alex Biryukov, Bart Preneel, and Joos Vandewalle. Cryptanalysis of 3-Pass HAVAL. In Chi-Sung Laih, editor, *ASIACRYPT: Advances in Cryptology*, volume 2894, pages 228–245. Springer-Verlag, 2003.

[259] Greg Rose. Authentication and Security in Mobile Phones. Technical report, QUALCOMM, Australia, 1999. This technical report is available at `http://www.qualcomm.com.au/PublicationsDocs/AUUG99AuthSec.pdf`. Last access date: 31st of December 2006.

[260] Greg Rose. Personal Communication, June 2004.

[261] Greg Rose. Personal Communication, June 2005.

[262] M Russinovich. Marks sysinternals blog. This link is available at `http://www.sysinternals.com/blog`. Last access date: 23rd of January, 2005.

[263] Markku-Juhani O. Saarinen. Cryptanalysis of block ciphers based on SHA-1 and MD5. In *IWFSE: International Workshop on Fast Software Encryption*, volume 2887 of *Lecture Notes in Computer Science*, pages 36–44. Springer-Verlag, 2003.

[264] Yu Sasaki, Yusuke Naito, Noboru Kunihiro, and Kazuo Ohta. Improved Collision Attack on MD5. Cryptology ePrint Archive, Report 2005/400, 2005. This paper is available at `http://eprint.iacr.org/2005/400`. Last access date: 8th of January 2007.

[265] Akashi Satoh. Hardware Architecture and Cost Estimates for Breaking SHA-1. In Chi-Ming Hu and Wen-Guey Tzeng, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 259–273. Springer, 2005.

[266] Bruce Schneier. The Twofish encryption algorithm. *Dr. Dobb's Journal of Software Tools*, 23(12):30, 32, 34, 36, 38, December 1998.

[267] Bruce Schneier and John Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In Dieter Grollman, editor, *Fast Software Encryption: Third International Workshop*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer-Verlag, 21–23 February 1996.

[268] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental Poker. In D. Klarner, editor, *The Mathematical Gardner*, pages 37–43. Wadsworth, Belmont, California, 1981.

[269] Adi Shamir. On the cryptocomplexity of knapsack systems. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, STOC'79 (Atlanta, GA, April 30 - May 2, 1979)*, pages 118–129. ACM, ACM Press, 1979.

[270] Adi Shamir. The Cryptographic Security of Compact Knapsacks (Preliminary Report). In *Proceedings of the 1980 Symposium on Security and Privacy (SSP '80)*, pages 94–99. IEEE Computer Society Press, April 1980.

[271] Adi Shamir. A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem (Extended Abstract). In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of Crypto 82*, pages 279–288. Plenum Press, New York and London, 1983, 23–25 August 1982.

[272] Minho Shin, Justin Ma, Arunesh Mishra, and William Arbaugh. Wireless Network Security and Interworking. *Proceedings of the IEEE*, 94(2):455–466, 2006.

[273] Francois-Xavier Standaert, Tal G. Malkin, and Moti Yung. A formal practice-oriented model for the analysis of side-channel attacks. Cryptology ePrint Archive, Report 2006/139, 2006. This paper is available at `http://eprint.iacr.org/2006/139`. Last access date: 21st of January 2007.

[274] Michael Szydlo and Yiqun Lisa Yin. Collision-resistant usage of MD5 and SHA-1 via message preprocessing. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2006.

[275] The Solaris Fingerprint Database- A Security Tool for Solaris Operating Environment Files. Technical Report 816-1148-11, Sun Microsystems, March 2006. This paper is available at `http://www.sun.com/blueprints/0306/816-1148.pdf`. Last access date: 21st of January 2007.

[276] Søren S. Thomsen. Cryptographic Hash Functions. Master's thesis, Technical University of Denmark, November 2005. This master's thesis is obtained through a personal communication from the author.

[277] TIA. Appendix A to IS-54 Rev. B Dual-Mode Cellular System: Authentication, Message Encryption, Voice Privacy Mask Generation, Shared Secret Data Generation, A-Key Verification and Test Data, February 1992. The document is available at the link `http://cryptome.org/0377-cave.htm`. Last access date: 14th of April 2006.

[278] Gene Tsudik. Message Authentication with One-Way Hash Functions. In *IEEE Infocom 1992*, pages 2055–2059, 1992.

[279] Jiri Tuma and Daniel Joscak. Multi-block Collisions in Hash Functions based on 3C and 3C+ Enhancements of the Merkle-Damgård Construction. In Min Surp Rhee and Byoungcheon Lee, editor, *Information Security and Cryptology ICISC*, volume 4296 of *Lecture Notes in Computer Science*, pages 257–266, 2006.

[280] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[281] Serge Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer-Verlag, 14–16 December 1994.

[282] David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[283] David Wagner, Bruce Schneier, and John Kelsey. Cryptanalysis of the cellular message encryption algorithm. In Burton S. Kaliski Jr., editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 526–537. Springer-Verlag, 17–21 August 1997.

[284] David Wagner, Leone Simpson, Ed Dawson, John Kelsey, William Millan, and Bruce Schneier. Cryptanalysis of ORYX. In H. Meijer S. Tavares, editor, *SAC: Annual International Workshop on Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 296–305. Springer, 1998.

[285] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004. `http://eprint.iacr.org/2004/199`. Last access date: 20th of January 2007.

[286] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[287] Xiaoyun Wang, Andrew Yao, and Frances Yao. Cryptanalysis of SHA-1 hash function. Technical report, National Institute of Standards and Technology NIST, October 2005. The slides of this work are available at `http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm` Last access date: 2nd of October 2006.

[288] Xiaoyun Wang, Andrew Yao, and Frances Yao. New collision search for SHA-1. Presented at Crypto'05 Rump Session by Adi Shamir, 2005. There is no official paper yet that explains this new result.

[289] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Efficient collision search attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005, 14–18 August 2005.

[290] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005, 14–18 August 2005.

[291] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[292] Doug Whiting. CFRG Email list, July 2006.

[293] Maurice Vincent Wilkes. *Time-Sharing Computer Systems.* New York:Elsevier, 1972.

[294] Christopher Wingert and Mullaguru Naidu. CDMA 1XRTT SECURITY OVERVIEW. Technical report, QUALCOMM, August 2002. This report is available at `http://www.cdg.org/technology/cdma_technology/white_papers/cdma_1x_security_overview.pdf`. Last access date: 8st of January 2007.

[295] Robert Winternitz. A Secure One-Way Hash Function Built from DES. In *Proceedings of the 1984 Symposium on Security and Privacy (SSP '84)*, pages 88–90. IEEE Computer Society Press, April 1984.

[296] Robert Winternitz. Producing a One-Way Hash Function from DES. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 203–207. Plenum Press, 1984.

[297] Jun Yajima and Takeshi Shimoyama. Wang's sufficient conditions of MD5 are not sufficient. Cryptology ePrint Archive, Report 2005/263, 2005. This paper is available at `http://eprint.iacr.org/2005/263.pdf`. Last access date: 20th of December, 2006.

[298] Hirotaka Yoshida and Alex Biryukov. Analysis of a SHA-256 variant. In *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2005.

[299] Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The second-preimage attack on MD4. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *4th International Conference on Cryptology and Network Security CANS*, volume 3810 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[300] Gideon Yuval. How to swindle Rabin. *Cryptologia*, 3(3):187–189, July 1979.

[301] Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry. HAVAL–A one-way hashing algorithm with Variable Length of Output. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology—AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104. Springer-Verlag, 13–16 December 1992.

[302] YongBin Zhou and DengGuo Feng. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing, 2005. This paper is available at `http://csrc.nist.gov/cryptval/physec/papers/physecpaper19.pdf`. Last access date: 31$^{\text{st}}$ of December 2006.