Wouter Teepe

Reconciling Information Exchange and Confidentiality

A Formal Approach

kwahjsdfgkgfadhg

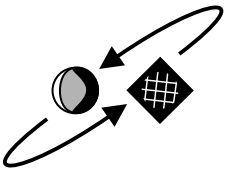
Reconciling Information Exchange and Confidentiality

A Formal Approach

Wouter Teepe

Rijks*universiteit* Groningen

The research reported in this thesis has been funded by the University of Groningen.
http://www.rug.nl



Department of Artificial Intelligence
The research reported in this thesis has been carried out at the Multi-Agent Systems group (MAS) of the research institute for Artificial Intelligence and Cognitive Engineering (ALICE) of the University of Groningen. http://www.ai.rug.nl

SIKS Dissertation Series No. 2007-02
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems. http://www.siks.nl

The research reported in this thesis has been carried out by Wouter Teepe. http://www.teepe.com

Paranymphs: Judith Grob & Leendert van Maanen

The author can be reached at wouter@teepe.com
Supplemental material and errata will be published at http://www.teepe.com/phdthesis

photo back cover: Jeroen van Kooten
cover design: Frans Boon

NUR: 993, 995

	printed edition	electronic edition
ISBN-10	90-367-2810-X	90-367-2811-8
ISBN-13	978-90-367-2810-2	978-90-367-2811-9

RIJKSUNIVERSITEIT GRONINGEN

Reconciling Information Exchange and Confidentiality

A Formal Approach

Proefschrift

ter verkrijging van het doctoraat in de
Gedrags- en Maatschappijwetenschappen
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. F. Zwarts,
in het openbaar te verdedigen op
donderdag 18 januari 2007
om 14.45 uur

door

Wouter Gerard Teepe

geboren op 23 februari 1977
te Darmstadt

Promotor:	prof.dr. L.R.B. Schomaker
Copromotor:	dr. L.C. Verbrugge
Beoordelingscommissie:	prof. dr. W. van der Hoek
	prof. dr. B.P.F. Jacobs
	prof. dr. J.-J. Ch. Meyer
	prof. dr. G.R. Renardel de Lavalette

to Lotte and Lucie

Contents

I Introduction **1**
1 Introduction . . . 3
2 Preliminaries . . . 17

II Tools **27**
3 Cryptographic Hash Functions . . . 29
4 Authentication Logics . . . 47
5 'Unsoundness' of BAN logic . . . 55
6 Extending GNY Logic . . . 67

III Approaches **79**
7 Information Designators . . . 81
8 Knowledge Authentication . . . 101

IV Protocols **121**
9 1-to-many Protocols (T-1) . . . 123
10 Many-to-many Protocols (T-2) . . . 145

V Conclusion **169**
11 Conclusion . . . 171

VI Appendices **177**
A Remarks to Authentication Logics . . . 179
B Summary of GNY Logic . . . 183
C Remarks to Knowledge Authentication . . . 187
D The Secret Prover . . . 191
E Notation . . . 207

Bibliography . . . 211
Author Index . . . 226
About the Author . . . 234
Samenvatting . . . 235

Detailed Contents

I Introduction 1

1 Introduction 3
- 1.1 The Privacy Debate . . . 4
- 1.2 Guarantees of Availability and Confidentiality . . . 6
- 1.3 Thesis Contents . . . 8
- 1.4 Relation to the Author's Other Publications . . . 10
- 1.5 A Case Study: the Dutch Police . . . 10
- 1.6 Considering Central Storage . . . 14

2 Preliminaries 17
- 2.1 Encryption . . . 18
- 2.2 Authorization and Authentication . . . 19
- 2.3 Complexity . . . 20
- 2.4 Probabilistic Algorithms . . . 21
- 2.5 Oblivious Transfer . . . 22
- 2.6 Adversary Models . . . 22
- 2.7 Secure Multiparty Computation . . . 23
- 2.8 Zero-Knowledge Proofs . . . 24

II Tools 27

3 Cryptographic Hash Functions 29
- 3.1 Normal Hash Functions . . . 30
- 3.2 Special Properties . . . 32
- 3.3 The Random Oracle Model . . . 36
- 3.4 Design Paradigms . . . 37
- 3.5 Common Applications . . . 41
- 3.6 (Non-) Incrementality . . . 44
- 3.7 Conclusion . . . 46

4 Authentication Logics 47
- 4.1 The Goals of an Authentication Logic . . . 48
- 4.2 The Taxonomy of Any Authentication Logic . . . 49
- 4.3 Using an Authentication Logic . . . 52
- 4.4 The BAN Logic Debate . . . 54
- 4.5 Conclusion . . . 54

5 'Unsoundness' of BAN logic 55
- 5.1 Cryptographic Hash Functions and Justified Beliefs . . . 55
- 5.2 On the Computational Justification of Beliefs . . . 57
- 5.3 The Two Parrots Protocol . . . 58
- 5.4 Used Inference Rules . . . 61

5.5 Proof of 'Unsoundness' of BAN logic . . . 61
5.6 The Semantic Approach . . . 63
5.7 Conclusion . . . 66

6 Extending GNY Logic **67**
6.1 Why Authentication Logics Are So Tricky . . . 67
6.1.1 Unstated Assumptions: Length-Concealment and Non-Incrementality . . . 67
6.1.2 Omitted Inference Rules: The Key to Incompleteness . . . 69
6.2 Proofs of Knowledge and Ignorance . . . 71
6.2.1 New Inference Rules for Proving Possession . . . 72
6.2.2 Proving That Principals Do Not Learn Too Much . . . 75
6.3 Conclusion . . . 77

III Approaches **79**

7 Information Designators **81**
7.1 Information Integration and its Challenges . . . 83
7.1.1 Overlapping Ontologies . . . 84
7.1.2 Information Propagation . . . 85
7.2 A Joint Approach to Privacy, Anonymity and Information Integration . . . 87
7.2.1 Information Designators . . . 87
7.2.2 Dependency and (Un)linkability . . . 88
7.2.3 Operations on Designators . . . 89
7.3 An Example: the Datamining Bookshop . . . 90
7.3.1 Organizational Setting . . . 91
7.3.2 Designators in Action . . . 92
7.3.3 Observations About the Use of Subqueries . . . 95
7.4 Methods for Restricting Designator Uses . . . 96
7.5 Discussion and Related Work . . . 98
7.6 Conclusion . . . 100

8 Knowledge Authentication **101**
8.1 Application Areas of Gossip . . . 102
8.1.1 Police Investigations . . . 102
8.1.2 The Passenger Name Record . . . 103
8.2 Comparing Information Without Leaking It and Reference . . . 105
8.3 Adversary Models for CIWLI . . . 108
8.4 Possible Set Relations . . . 109
8.5 Secure Protocols for Computing Set Relations . . . 112
8.6 Domain Compression . . . 116
8.7 Conclusion . . . 119

IV Protocols **121**

9 1-to-many Protocols (T-1) **123**
9.1 Prerequisites . . . 124
9.2 Protocol Description (Simple, no Encryption) . . . 126
9.3 Making the Protocol More Efficient (Elaborate, Encryption) . . . 129
9.4 Correctness Proof in GNY Logic . . . 133
9.4.1 Knowledge Preconditions . . . 134
9.4.2 Claims and GNY Idealization . . . 135
9.4.3 The Easy Part of the Proof . . . 136
9.4.4 Different Options to Complete the Proof . . . 137
9.4.5 Proving principals do not learn too much . . . 140
9.4.6 Modeling the beliefs and possessions of an attacker . . . 141
9.5 Conclusion . . . 142

10 Many-to-many Protocols (T-2) **145**
10.1 Using Prefix Trees for Efficiency . . . 145
10.2 Specification of the T-2 Protocol . . . 148
10.2.1 Subprotocol for Determining Intersection . . . 149
10.2.2 Subprotocol for Proving Possession . . . 154
10.3 Making the Protocol Efficient by Restrictions . . . 157
10.4 Determining Communication Complexity . . . 159
10.5 Conclusion . . . 166

V Conclusion **169**

11 Conclusion **171**
11.1 Information Designators . . . 171
11.2 Knowledge Authentication . . . 172
11.3 Hash Functions and Authentication Logics . . . 174
11.4 Relevance to the Privacy Debate . . . 175

VI Appendices **177**

A Remarks to Authentication Logics **179**
A.1 A Taxonomy of Versions of the BAN Paper 179
A.2 A Short Survey of Critisisms on BAN Logic 180

B Summary of GNY Logic **183**
B.1 Formal Language 183
B.2 Inference Rules 184

C Remarks to Knowledge Authentication **187**
C.1 The 'French Approach' 187
C.2 On the Probabilistic Communication Complexity of Set Intersection 188
C.3 Fuzzy Private Matching 189

D The Secret Prover **191**
D.1 Starting Up and Connection Control 192
D.1.1 Opening a Connection Listener 193
D.1.2 Making a Connection 193
D.2 Managing Hash Pools 196
D.3 Running the Protocol 198
D.3.1 Initiating a Protocol 199
D.3.2 Responding to a Protocol 200
D.3.3 A Side Note on Hash Pools 200
D.3.4 Challenging 201
D.3.5 Proving 203
D.3.6 Verifying 204
D.3.7 Faking 206
D.4 Closing 206

E Notation **207**
E.1 Symbols 207
E.2 Letters 208

Bibliography **211**

Author Index **226**

SIKS Dissertation Series **230**

About the Author **234**

Samenvatting **235**

List of Figures

1.1 Dependencies between the chapters that make up the main body of the thesis at hand . . . 9
1.2 Matching of police information within the VROS . . . 13

2.1 A trivial primality testing algorithm . . . 21
2.2 The Miller-Rabin primality testing algorithm . . . 21
2.3 A Rubik's cube . . . 25

3.1 A 'normal' hash function in action . . . 31
3.2 The relation between various properties of cryptographic hash functions . . . 35
3.3 A Merkle-Damgård hash function . . . 38
3.4 A hash function of the randomize-then-combine paradigm . . . 39
3.5 Incremental hash function in action . . . 45

4.1 The signing parrot protocol, plain description . . . 49
4.2 GNY idealization of the signing parrot protocol . . . 51
4.3 GNY annotation of the signing parrot protocol . . . 52
4.4 Heavy GNY annotation of the signing parrot protocol . . . 53

5.1 The two parrots protocol, graphical illustration. . . . 58
5.2 The two parrots protocol, plain description . . . 60
5.3 BAN idealization of the two parrots protocol . . . 60
5.4 GNY idealization of the two parrots protocol . . . 60
5.5 Heavy BAN annotation of the two parrots protocol . . . 63

7.1 The main aims and interests for organizations participating in information integration . . . 86
7.2 An information dependency graph containing the four organizations of the example . . . 92
7.3 A global SQL query which would provide the local bookshop with the information it desires . . . 94

8.1 The relations possible between two sets X and Y . . . 110
8.2 Some interesting set functions for which secure protocols exist . . . 110
8.3 Special cases of the sizes of two sets . . . 111

9.1 T-1 protocol, no encryption, verifier initiates . . . 127
9.2 T-1 protocol, no encryption, prover initiates . . . 127
9.3 T-1 protocol, no encryption, mutual proof . . . 128
9.4 A rough paraphrase of the T-1 protocols . . . 129
9.5 The initialisation and maintenance of the look-up table . . . 130
9.6 T-1 protocol, encryption, verifier initiates . . . 131
9.7 T-1 protocol, encryption, prover initiates . . . 131
9.8 T-1 protocol, encryption, mutual proof . . . 132
9.9 GNY idealization of the T-1 protocol, no encryption, verifier initiates 136
9.10 The output of the protocol parser for the T-1 protocol . . . 137
9.11 GNY proof of the T-1 protocol . . . 139

10.1 Sets KB_A, KB_A represented as binary hash value prefix trees . . 146
10.2 Interleaved subprotocols for establishing the intersection, shown as a colored surface, with $l = 4$ 152
10.3 Interleaved subprotocols for establishing the intersection, shown as a colored surface, with $l = 16$ 152
10.4 The number of communicated bits in the restricted T-2 protocol with cooperative participants, shown as a density plot 163
10.5 The number of communicated bits per compared secret in the restricted T-2 protocol with cooperative participants, shown as a density plot 164

D.1 Main application window 192
D.2 Opening a connection listener 193
D.3 Filling in a name 193
D.4 Making a connection 194
D.5 Filling in connection details 194
D.6 An initiated connection (outgoing) 194
D.7 Receiving a connection (incoming) 195
D.8 An authentication warning 195
D.9 An authentication mismatch 195
D.10 Main application window, with connections 195
D.11 A new hash pool window 196
D.12 Adding files to a hash pool 196
D.13 A hash pool with files added 196
D.14 Computation of hash values 197
D.15 A ready hash pool 197
D.16 Adding files to an existing hash pool 197
D.17 A new protocol window for the initiator 199
D.18 A protocol window, configured by the initiator 199
D.19 A protocol window of the initiator for a protocol that has started 199
D.20 A new protocol window for the responder 200
D.21 The responder has filled in the nonce 200
D.22 The responder has committed the nonce 200
D.23 The verifier chooses whether he will halt the protocol 202
D.24 The verifier has challenged the prover 202
D.25 The prover has received a challenge 202
D.26 The prover sends some fake hash value h_2 203
D.27 The prover sends a genuine hash value h_2 203
D.28 The prover hash halted the protocol 204
D.29 The verifier receives an unexpected value of h_2 205
D.30 The verifier receives the h_2 he expected 205
D.31 The verifier has been informed that the prover has halted the protocol 205

List of Tables

3.1 Some commonly used cryptographic hash functions 40

7.1 Two relational tables which can be combined to relate courses to birth dates 83

7.2 The schemata of the information that is maintained by the civic authority, the local school and the book publisher 93

8.1 All known well-documented secure protocols for computing set relations 114

8.2 Protocols which can be used for knowledge authentication 116

9.1 Basic messages used in the T-1 protocol. 125

10.1 Binary encoding of some hash prefix trees 147

10.2 Basic messages used in the T-2 protocol 148

10.3 A sample run of interleaved subprotocols for establishing the intersection 151

10.4 Interleaved subprotocols for establishing the intersection, shown as a growing binary tree 153

10.5 State variables in a subprotocol for proving possession 155

10.6 A sample run of the subprotocol for proving possession 156

10.7 Encoding for sets R_p where $\forall s\colon |s| = 1$ and p may be omitted . . 158

10.8 A sample protocol run of the restricted T-2 protocol, efficiently encoded 159

10.9 The worst case communication complexity for the restricted T-2 protocol, depending on the strategies 160

10.10 The ten conditions of the experiment to estimate the average communication complexity of the restricted T-2 protocol with cooperative principals 162

10.11 Descriptive statistics of the number of communicated bits in the restricted T-2 protocol with cooperative participants 163

10.12 Descriptive statistics of the number of communicated bits per compared secret in the restricted T-2 protocol with cooperative participants 164

10.13 Upper bounds on average communication complexities of the T-1 and the T-2 protocol 165

Acknowledgements

It is said that the the parts of a thesis that are read most often are the acknowledgements and the bibliography. Though probably true, this is fascinating because in many aspects the acknowledgements and the bibliography are the most boring parts of a book. They are rather dull lists of names, combined with respectively compliments or article details. If anything, the real meat of a thesis is not to be found in any of those two sections. What is it that attracts so many people to these sections?

Let me guess. Lists of names give the reader a peek into the personal life of the author. Who are his friends? What does his professsional network look like? As a researcher on the subject of privacy, this begs the question. Actually, two questions: Why does one voluntarily give up one's own privacy? Why don't all the mentioned people object to being mentioned? The answer to the first question boils down an undisclosed mixture combination of politeness, window-dressing, and sincerety. The answer to the second question is more intriguing.

There is an implicit contract that one shall only say positive things about the people mentioned in the acknowledgements. Privacy infringements are not that bad if only positive information is disclosed. But what if politeness urges one to mention somebody, though the author is not generally positive about the person in question? The trained reader immediately recognizes such cases. Typically, supervisor X has been praised extravagantly, and supervisor Y is acknowledged only for "the interesting discussions". Why stick to the contract?

Though the availability of dirty laundry is not a constraining factor, I will leave the question unanswered. Instead, I will focus on another dilemma, proportionality versus exhaustiveness. When I want to express my gratitude towards someone, a logical place to do so is in the acknowledgements. As not all people have contributed equally, some people deserve more attention than others. Consider the situation where person X did everything possible, and even a bit more, while person Y was so polite to answer an informational email. Would it be disproportional if I took only ten times as many words to express my gratitude towards X?

I daresay it would. Absolutely. As such, and in this context, exhaustiveness implies disproportionality. Still, and in this context, exhaustiveness is a negotiatable concept. A close fried of mine chose to simply thank virtually everybody he knew.[1] Instead of listing my whole address book, I wonder whether there exist valid criteria for omitting people from the list.

[1] Moreover, he thanked everyone he forgot to mention to whom he owes gratitude for not being offended by that. [Koo03, page x]

One possible criterion for omitting people is the question whether their contribution was to be expected based on their job profile. It seems superfluous to thank somebody for just doing his or her job. However, in a context where meeting one's obligations cannot taken for granted, this criterion is far too general. Another possible criterion is to only mention people who actually influenced the contents of my thesis. This does not work either, as soon as one observes that one's general well-being influences one's work. Where does this leave me?

Factually, it leaves me with an acknowledgements section which is composed completely by subjective opinions, and filtered to meet some unwritten norms. As a bonus, the first page is not a dull list of names.

Of the people who have been professionally involved in the process of this thesis, the utmost important person to acknowledge is Rineke Verbrugge, my daily supervisor. She is this good, that if I *would* have had any problem with her, I could be sure the problem would have been on my side. If anyone is a role model for a good supervisor, she is. Any list of appreciated competences would be glaringly incomplete, and therefore I will not even try to enumerate just the most important ones.

Lambert Schomaker, my promotor, has been honest with me: our expertises and interests are not even close. I admire his courage to acknowledge this upfront, and I am grateful for the trust and freedom he has given me to develop my own interests without his interference. Even more, he has protected my research against many 'evil powers' from the outside.

In the team of people involved in the ANITA research project[2], I have found many amicable conspirators, soundboards and keen questioners. Special mention deserve Kees de Vey Mestdagh, Pieter Dijkstra, John-Jules Meyer, Frank Dignum, Huib Aldewereld, Laurens Mommers, and Jaap van den Herik. By means of the ANITA project, I was introduced to a few valuable contacts at the police. Paul Elzinga, information architect of the Dutch police, has been very helpful in explaining how the Dutch police handles many kinds information. Tom van der Velde, privacy officer of the police in Groningen, provided the most inspiring example of my research period (Section 1.5).

The award for the most valued partners-in-crime go to fellow-PhD students Kathy Cartrysse (Delft University of Technology) and Sieuwert van Otterloo (The University of Liverpool). With both, I spent a large number of long afternoons discussing many in-depth issues. Their knowledge, enthusiasm and interest have been an immense factor in the quality of my research.

Many others have contributed by important but little things like answering emails with my questions. Often, these questions turned out to expose my lack of knowledge of specific areas, but the following people have been so kind as to patiently answer them: Ross Anderson, Niels Ferguson, Antonina Mitrovanova, and Leen Torenvliet.

[2] The ANITA project was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 634.000.017.

Another simple, but very important contribution to my research has been the unbounded enthusiasm expressed by Rafael Accorsi, Hans van Ditmarsch, Hans Kamerbeek, Alessio Lomuscio, Gerben Wierda, and many others. Without supporters, it is hard to play a match. More strenuous have been the contributions of those who have patiently proof-read articles and chapters, and have unscrupulously reported on the many weaknesses they found: Anton Ekker, Katrin Franke, Jan-Willem Hiddink, Bart Verheij, Gerben Wierda, and the anonymous and not-so-anonymous reviewers.

Some particularly unanoymous reviewers were Wiebe van der Hoek, Bart Jacobs, John-Jules Meyer, and Gerard Renardel de Lavalette, jointly known as the manuscript committee. The value and quality of their feedback was tremendous, and their comments triggered many significant improvements.

Research can only flourish in a good environment, and it has been a great pleasure to share an office with Marius Bulacu, Judith Grob, Leendert van Maanen, Karin Zondervan, and Egon van Baars. The availability of instant reflection, warm friendship and a relaxed atmosphere is exclusively their contribution. Outside of the office, but still at the same department, I have had the pleasure to enjoy the comforting company of Tjeerd Andringa, Fanny Huitenga, Dirkjan Krijnders, Marcia van Oploo, Hedderik van Rijn, Esther Stiekema, Niels Taatgen, Rineke Verbrugge, Bart Verheij, Geertje Zwarts, and many others.

Not easy to classify, but important nonetheless have been the conversations with Frank Ankersmit, Łukasz Chmielewski, Jaap-Henk Hoepman, Onno Kubbe, Gerard te Meerman, Reind van de Riet, Martijn Warnier, Maurice Wesseling, and Edwin Wolffensperger.

With Marc Hooghe, Stefaan Walgrave and Jochum de Graaf I have had the pleasure of building and discussing *party profile websites* ("stemadviezen"). Alex Allersma, Marieke Berghuis, Rutger van Koert and Bart van Oudenhove have relieved me from various taxing tasks. Jeroen van Kooten allowed me to use the picture on the back cover, which was designed by Frans Boon.

Here is the Hollywood part.

Indirect — but hugely important — contributions to this thesis have been made by everyone who improved my general well-being. With Ron Gerrits, Wemke Tuinier and Iris Vermeeren I have enjoyed many sessions of climbing and much comradeship. Gea Bakker, Wierd Jansma, Barteld Kooi, Stephanie Hoenders, and Judith Grob are the best friends to have around.

The remark about exhaustiveness and dispropotionality made above, applies in particular to the people mentioned below.

It is impossible to exaggerate the importance of Lotte Douze and Lucie Teepe in my life. Lotte has been my loving and caring girlfriend all these years, and I am elated about our recent marriage. Lucie is our daugther since April 2005, and I am a happy man to have these two wonderful wonders around.

Wouter Teepe
Groningen, November 2006

Part I

Introduction

The privacy debate is introduced, and it is explained how this thesis relates to it. The focus of this thesis is given: strong guarantees from cryptographic hash functions and formal analysis of security protocols. The structure and contents of this thesis are revealed. We present a case study of a potential and inspiring application domain: the police. We finish with some deliberations on the merits of central storage of sensitive information.

Chapter 1

Introduction

Securing sensitive information against unauthorized use, while guaranteeing access for its intended uses, is difficult. One of the difficulties lies in the fact that 'intended use' is a rather vague term, which is often explained as 'need to know', a similarly vague term.

Securing sensitive information in such a way that only specific, listed people have access to it, is not that difficult. For example, many police records are protected in such a way that police officers have access to it, but others not. The problem with this type of protection is, that police officers do not need access to *all* police records, but only to those *relevant* to them. Though police officers are generally trustworthy people, they do have access to information they have nothing to do with.

Whether there is a need to know, seldom coincides with a list of people that is easy to construct. If one wants to improve the guarantees against unauthorized use of sensitive information, it seems logical to grant access to information based on whether there is a need to know. The problem is that it is difficult to precisely state and operationalize what constitutes a need to know. Moreover, the question whether one should be granted access, often depends on the information itself, to which there is no access yet: a Catch-22.

It is the aim of this thesis to help solve this problem, by offering technical solutions: we propose new *methods* to organize information in databases, and we propose cryptographic *protocols*.

- Using the *methods*, it is possible to compartmentalize the information into pieces that are so small that the need to know is easier to operationalize.

- Using the *protocols*, it is possible to make access decisions based on secret information without disclosing the information.

1.1 The Privacy Debate

Informally, 'sensitive information' is information about individuals or organizations, for which there is some commonly accepted norm that not all that information should be available to everybody. Typically, this is in the interest of the individuals or organizations that the information is about. The aim of the confidentiality of sensitive information is to prevent misuse of the information.

The trouble is, that for many kinds of sensitive information, there are genuine reasons why in certain circumstances, the confidentiality has to be sacrificed. When one wants to facilitate such 'genuine sacrifices', there needs to be a 'backdoor' in the protection of the information. The appropriate way to implement this proverbial backdoor is the subject of a large public debate, the *privacy debate*.

The following examples help to get an impression of the kinds of sensitive information to which the privacy debate applies:

Criminal records Information about the criminal behavior of people is kept secret in order to guarantee the state monopoly of justice. By keeping criminal records confidential, it is prevented that people take the law into their own hands or interfere with their own prosecution, which would result in disproportional and unequal punishment.

Medical files Information about the particulars of someone's health is kept secret as an extension of the integrity of the body. The medical history of someone is considered as very intimate information. Keeping medical files confidential promotes solidarity. If medical files were public, persons who fall into risk categories of whatever kind, would have great difficulty in applying for jobs and obtaining insurance.

Mail The confidentiality of mail is there to guarantee the freedom of thought and speech. If one has no control over who will receive a mail message when it is sent, one cannot verify whether the recipient is a friend or possibly a foe. Lack of confidentiality of mail will lead to some form of self-censorship.

Financial files Details of the financial status of an individual are kept secret to protect the negotiation position of individuals, and to protect wealthy individuals from targeted crime. Confidentiality of financial files should prevent exploitation.

The underlying norms that make this information 'sensitive' are not controversial, they are not the subject of a serious debate. Nevertheless, it is also commonly understood that the confidentiality of this information cannot be absolute. For certain jobs in which people carry a high responsibility, the criminal records of applicants have to be verified. If someone carries a highly contagious and dangerous disease, health organizations will use this information in order to prevent an epidemic. When it is certain that a person is planning a terrorist attack, his mail will be read and his phone tapped, in order to prevent

the attack. When someone goes bankrupt, the curator will have full access to that person's financial files.

There is a fierce public debate about at what point the interests of the society as a whole should take precedence over the interests of the individual. For example: how strong should evidence be before the privacy of an individual is infringed? If the evidence is required to be 100% decisive and based on *sound science*,[1] the interest of the society will in practice almost never take precedence. If the requirements on the evidence are set too low (say, only 1%), one faces the risk of sacrificing the fundamental values of justice [Sus06]. Here is a Catch-22: if the evidence is in fact 100% decisive, the information is probably not even needed anymore, but to know that it is indeed 100% decisive, one needs access to the information.

This *privacy debate* is widely believed to have zero-sum characteristics[2]: the wishes of those who defend privacy are (supposedly) fundamentally incompatible with the wishes of those who give priority to fighting crime and terrorism. The thought that describes this, can be summarized as 'Either you infringe everyone's privacy, or you do not catch any terrorists.'

In this thesis, the author does not wish to engage in this debate. Its connotation is primarily political and judicial, and though the author is well informed on these subjects and has a stake in this debate, he does not wish to claim scientific expertise on them. The opinion of the author is a *political* one, and not one *based on science*. Personal opinions do not belong in a scientific work.

Whatever comes out of the privacy debate, whether it be by means of consensus or decree, it will be a policy that tries to reconcile confidentiality of sensitive information with the guarantee of its use in urgent circumstances. When such a policy is to be implemented, scientific questions arise, such as:

- How can we organize information in such a way that a policy is enforced?
- What kind of guarantees on confidentiality and availability are precisely required?

It is not uncommon that a chosen policy cannot be implemented as such, but has to be adjusted before it can be implemented: not everything that is desirable is also possible (or affordable). Adjustment of a policy prior to implementation may mean that delicate agreements reflected in the policy are sacrificed for the sake of being able to implement the policy at all. A good policy is one which can be implemented without sacrificing essential parts of the policy. As such, technology has a *constraining* effect on policy-making, while technology should ideally be *facilitating* for policy.

[1] 'Sound science' is a political euphemism for requiring a very high burden of proof [Moo05].

[2] To see the zero-sum characteristics, it helps to perceive the privacy debate as a game between the privacy advocate and the terrorist fighter. The goal of the game is to settle on an information exchange policy. From the established policy, player payoffs are calculated. If infringing privacy is required for catching terrorists, the interests of the players are (at least to some extent) diametrically opposed [Bin92, page 237] [OR94].

The aim of this thesis is to increase the *facilitation* of technology for policies regarding the exchange of sensitive information. In that perspective, the key contributions of this thesis are the following:

Information Designators A new method for structuring information, which demonstrates that exchanging information on the one hand, and privacy and confidentiality on the other hand, can go hand in hand (Chapter 7).

Knowledge Authentication A set of efficient protocols which allow the comparison of information without disclosing the information itself. This has applications in, for example, passenger information exchange between the European Union and the United States of America (Chapters 8–10).

Thus, although this thesis does not defend any position in the privacy debate, it provides input to this debate. The techniques presented in this thesis offer new solutions to settle the privacy debate. The relevance of these techniques is high because they demonstrate that in certain cases it is possible to simultaneously accommodate many wishes of either side of the privacy debate: to respect the privacy of innocent citizens to a high degree, while the information required for protecting the society as a whole (against terrorism, contagious diseases, etc.) is provided.

It is not to be expected that any reasonable civil liberties activist will object if the privacy of proven criminals or terrorists is infringed on purely for the act of bringing them to justice[3]. We present technologies that do just that, without infringing on the privacy of innocent citizens. Thus, we demonstrate that settling the debate is not a zero-sum game at all if the right technology is deployed.

1.2 Guarantees of Availability and Confidentiality

The meaning of a 'guarantee' in a legal context is quite different from the meaning of a guarantee in a mathematical or cryptographic context. This applies also to guarantees of availability and confidentiality.

A legal guarantee of *availability* of information, such as the *Freedom of Information Act* (FOIA)[4] gives someone a right to certain information, but not a means to quickly exercise this right. A FOIA request may take months to complete, and the request may fail for a number of reasons.

A legal guarantee of *confidentiality* of information, such as a *privacy law* or a *non-disclosure agreement* (NDA) obliges someone to keep certain information secret, but it does not physically prevent disclosure of the information. An

[3] Because it implies the information collection procedure is selective [Jac05].

[4] This is the name of two similar laws in the United States of America and the United Kingdom. The Dutch equivalent of this law is the *Wet Openbaarheid Bestuur* (WOB). The FOIA is best summarized as "The Freedom of Information Act gives everyone the right to access information held by the public sector." (http://www.direct.gov.uk/RightsAndResponsibilities/RightsAndResponsibilitiesArticles/fs/en?CONTENT_ID=4003239&chk=xi42h7)

NDA gives the owner of the information the right to hold the discloser of the information liable in court. This leaves the owner of the information at risk: he may be unable to detect the disclosure of the information, while he does suffer from damages as a result of the disclosure.[5] Even if he is able to detect the disclosure of the information, he may be unable to prove this convincingly in a courtroom setting. Not to mention that going to court is generally a tedious, time-consuming and expensive process.

While legal guarantees of availability and confidentiality effectively serve a large number of purposes, such as the protection of press freedom and intellectual property, they are often considered insufficient for parties at either side of the privacy debate. A police officer cannot do his duty if he depends on time-consuming procedures to obtain the information he needs to investigate a crime. Similarly, a fighter for civil liberties will not be satisfied with a mere *promise* that his phone will not be tapped, as is it highly unlikely that he will ever be informed of such a tap.

Cryptography is not like law at all. One of the aims of cryptography is to guarantee information availability and confidentiality to absurdly high degrees.[6] Instead of writing lengthy contracts in legalese that describe what should happen when something goes wrong, cryptography aims to *prevent* things from going wrong. For example, to break the best modern encryption, which is a means to guarantee confidentiality, one needs thousands of years of computation. Colloquially, the likelihood of guessing an encryption key correctly is smaller than the likelihood of getting struck by lightning several days in a row, and surviving.

Cryptography and modern IT infrastructure form a tandem which has the potential of offering both confidentiality and availability guarantees. Admittedly, IT infrastructure does not always live up to its expectations, as it sometimes fails to deliver availability – also in cases where confidentiality is of no concern. If cryptography is improperly used, it does not offer any confidentiality guarantees.

This thesis aims to help develop the potential of the cryptography/IT tandem for guaranteeing both confidentiality and availability of sensitive information.[7] In this perspective, we focus on the following:

Cryptographic Hash Functions A type of algorithm for creating *fingerprints* of information, in such a way that the information itself is kept confidential. Our contribution is a new application area of cryptographic hash functions, and the concept of a *non-incremental hash function* (Chapter 3 and throughout the thesis).

[5] This risk is not hypothetical: for example, it is not unusual that a creditcard or mortgage company refuses someone as a client based on information gathered from an information broker (a 'mass-scale private investigator') that gathered information from obscure sources.

[6] Other typical aims of cryptography are non-repudiation and guarantees of integrity.

[7] Digital Rights Management (DRM) technology is similar but different. DRM technology guarantees confidentiality and availability of copyrighted material, which is typically owned by the entertainment industry. The goal of our research is to promote both confidentiality and availability of sensitive information about individual citizens.

Authentication Logics A method for analyzing security protocols. Our contribution is the extension of a particular authentication logic (GNY logic) to handle cryptographic hash functions, and a proof that another authentication logic (BAN logic) improperly handles cryptographic hash functions. (Chapters 4 and 5)

This thesis focuses on strong guarantees for availability and confidentiality. We use formal methods to reason about the knowledge of persons involved in security protocols, and analyze what these persons can and cannot derive from the information they have got.

1.3 Thesis Contents

In the previous pages, we have motivated and introduced the research that is presented in this thesis. The structure of the thesis at hand is as follows:

In Part I we motivate the relevance of the research conducted in this thesis. What is sensitive information, and why do we need technology to handle sensitive information? (Chapter 1). For reference and the uninitiated, we summarize some important concepts of security and cryptography (Chapter 2).

Part II introduces the main security building blocks used to construct and analyze the protocols presented in this thesis. The *cryptographic hash function* is the most important cryptographic primitive applied in this thesis. We introduce the concept of a *non-incremental* cryptographic hash function (Chapter 3). To analyze the security properties of the presented protocols, we use *authentication logics*, of which the basics are explained (Chapter 4). BAN logic, the 'mother of all authentication logics', contains a flaw with how it handles cryptographic hash functions. We prove that as a result of this, BAN logic is not 'sound' (Chapter 5). GNY logic, another authentication logic (summarized in Appendix B), is extended in such a way that our protocols can be analyzed using GNY logic (Chapter 6).

Part III is the conceptual heart of the thesis, and presents two approaches to handling sensitive information. Instead of propagating information throughout information systems, which is essentially a massive disclosure of information, information can also be linked by using *information designators*. This idea may sound trivial, but it is in fact a radical departure from how information is integrated nowadays. If the idea is fully applied, there are benefits for both information availability and confidentiality (Chapter 7). A typical problem with confidential information is how to establish that two people both know something without actually disclosing it, which we dub *knowledge authentication*. The chicken-and-egg problem 'do you know the secrets that I know?' turns out to be difficult to formalize. We distinguish the case *1-to-many*, where one single secret (that I know) is compared to many secrets (that you know) and the case *many-to-many*, where many secrets (that I know) are compared to many secrets (that you know). We survey what solutions currently exist for this problem, and what their merits are (Chapter 8).

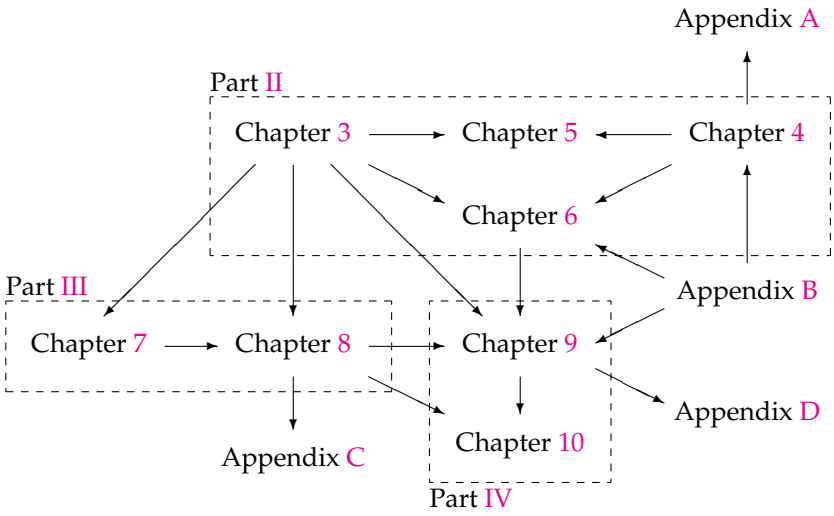


FIGURE 1.1: Dependencies between the chapters that make up the main body of the thesis at hand. The arrow $y \longrightarrow x$ means that to *fully* understand chapter x, chapter y should be read.

In Part IV, new security protocols are presented which implement knowledge authentication. The T-1 protocol does so in the 1-to-many case. In the T-1 protocol a message is sent that can only be recognized if one knows a particular secret. When one claims to know the secret, a (NP-hard) puzzle is created that can only be solved if one knows the secret. The T-1 protocol is very efficient in terms of communication and computational requirements. It uses non-incremental cryptographic hash functions, and is proven correct using our extension of GNY logic (Chapter 9). The T-2 protocol implements knowledge authentication in the many-to-many case. The T-2 protocol is a parallel composition of the T-1 protocol, and as such inherits its security properties from the T-1 protocol. In the T-2 protocol the players work together to efficiently determine the intersection of their secrets. The communication complexity of the T-2 protocol depends on the cooperativeness of the players, and we estimate this complexity experimentally (Chapter 10).

In Part V, we draw some conclusions (Chapter 11).

The appendices in Part VI contain some background information to various chapters. Appendices A and C contain some in-depth explanations to the literature referenced in Chapters 4 and 8, respectively. Appendix B gives a summary of GNY logic which is used in Chapters 4, 6 and 9. Appendix D describes the prototype software of the T-1 protocol presented in Chapter 9. Appendix E gives a summary of the formal notations used in this thesis.

When we focus on the main body of the thesis at hand (leaving out the introduction and conclusion part) we can visualize the dependencies between

the chapters as in Figure 1.1. When chapter x depends on chapter y (notation $y \rightarrowtail x$), this means that to *fully* understand chapter x, chapter y should be read. For Chapters 7 (information designators) and 8 (knowledge authentication), special care has been taken to make sure they are still rather intelligible without reading their background knowledge chapters first (Chapters 2, 3 and 7). In Chapter 9, it is only the correctness proof in GNY logic of the T-1 protocol, given in Section 9.4, that depends on Chapter 6.

Chapters 3 (cryptographic hash functions) and 4 (authentication logics) introduce the background on which the results in this thesis are based. As such, these chapters mainly summarize results of other researchers, though some new concepts are introduced in these chapters: *non-incremental cryptographic hash functions* (Chapter 3) and *heavy GNY annotations* (Chapter 4). From Chapter 5 onwards, all material presented is essentially new, though Chapters 7 and 8 contain various discussions of related research.[8]

1.4 Relation to the Author's Other Publications

Chapter 5 has been presented at FAMAS'06 [Tee06a]. An earlier version of Chapter 7 has appeared in the International Journal of Computer Systems Science & Engineering [Tee05b].

The T-1 protocol which is presented in Chapter 9 was first published at the Knowledge and Games Workshop 2004 [Tee04], and its proof in GNY logic (Section 9.4) first appeared in Synthese [Tee06b]. Chapter 8 (in particular Section 8.2) contains traces of both of these publications. Our opinion letter in Het Financieele Dagblad [Tee05c] can be considered a summary for laymen of the T-1 protocol. All our publications mentioned above are subsumed by this thesis.

The author's publications on workflow analysis and security [TvdRO03, TvdRO02, Tee99] and on expert systems for online voting advice ('party profile websites') [HT07, Tee05a, TH05] are not reflected in this thesis.[9]

1.5 A Case Study: the Dutch Police

A typical example of sensitive information that has to be kept confidential, is the information maintained by crime investigators of the police. Dissemination of this information to criminals would render the research of those police officers almost worthless. At the same time, it has to be prevented that two or more research teams investigate the same people without knowing this. When police teams are unaware of such a situation, their actions can easily harm one another's research. For example: one team is shadowing a suspect in the hope

[8] Moreover, Appendices A, B and C do not present new work, but contain discussions of related research. Appendix D can be considered new work, as it presents a prototype of our new T-1 protocol.

[9] The publications mentioned here do not constitute an exhaustive list.

that his actions will reveal new pieces of evidence; now another team runs in and arrests the suspect for another crime. If police teams cooperate, such situations can be prevented, and moreover collaboration of teams may help them to mutually exchange incriminating evidence.

In this section, we will describe how the Dutch police currently handles this situation and tries to reconcile information exchange and confidentiality.[10] It shows the current practice of how an organization of professionals with a high responsibility deals withs sensitive information.

It should be noted that there are many more information exchange systems operational within the police, for several purposes (criminal records, missing persons tracing, fingerprints, license plate registrations, etc.). The system described is just one of the systems deployed by the Dutch police.

For every investigation project performed by the police, an electronic dossier[11] is created. Access to this dossier is only granted to the officers dedicated to the project, and a number of superiors. Such dossiers are created and maintained locally, at a local police department[12], and can have any subject, varying from 'the great train robbery' to 'pickpockets in Amsterdam'.

An electronic dossier consists of records[13] which are added to the dossier over time. Every record has a time stamp. A record consists of various fields[14], in which the actual information is stored. There are fields for plain text, but also fields to store specific types of information. The most important types are:

BTK for persons[15],

ORG for organizations,

LOK for locations, such as physical addresses,

COM for means of communication, such as phone numbers, and

VTG for vehicles[16].

For all of these five field types, there are instructions on how the information should be encoded. For VTG, the encoding is simply the license plate number, for other types, the encoding is more complicated. For BTK for example, there is a method to derive a 20-character string from a name and birth date, of which it is supposed that it uniquely identifies any person.

[10] Interview with Tom van der Velde, privacy officer of the Police in Groningen, conducted on November 28, 2002, together with Pieter Dijkstra. The information has been verified by Paul Elzinga, IT specialist of NPOL, the organization which manages the information exchange infrastructure of the Dutch police. The interview occured in the context of the ANITA project, which was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 634.000.017.

[11] In Dutch police jargon, an electronic dossier is called a *registratie*.

[12] In Dutch police jargon, this is at the *korps* or *regio* level.

[13] In Dutch police jargon, a record is called a *mutatie*.

[14] In Dutch police jargon, a field is called an *object*.

[15] BTK stems from *betrokennen*.

[16] VTG stems from *voertuigen*.

Hopefully, the police officers will nicely follow the instructions and fill in all relevant places in the prescribed way. As with every database which is manually filled, the electronic dossiers also suffer from inconsistent annotation, and use of text fields where the specific fields could have been used.

When the electronic dossier has as subject a crime for which it is possible to obtain an arrest warrant[17] and it is to be expected that the investigation will take more than one week, there should be a record in the electronic dossier which is tagged MRO[18]. MRO records can only contain fields of the types BTK, ORG, LOK, COM and VTG. Electronic dossiers can be grouped into two classes: those that contain an MRO record[19], and those that do not contain an MRO record[20]. The former class is typical for investigations of serious crimes, the latter class is typical for exploratory investigations. There are some rules about what information should be filed in an MRO record, and what information should not be filed in an MRO record. Roughly, *primary* information about the investigated crime, such as the suspect and the crime scene location, belongs in an MRO record, and *secondary* information, such as the phone numbers found in the agenda of the victim do not.

There is one very special electronic dossier, which is called ZWACRI[21]. It is essentially a black list of known 'professional' criminals. In the ZWACRI dossier, only the fields BTK and ORG are allowed.[22]

To improve the individual investigations, the electronic dossiers are automatically compared. The system that performs the comparison is called the VROS[23], and is housed in a heavily protected bunker which would not look misplaced in a James Bond movie. Every local police office has a dedicated, encrypted data connection to the VROS, and every Sunday morning, all electronic dossiers which contain an MRO record and the ZWACRI dossier are transmitted to the VROS. Only fields of the types BTK, ORG, LOK, COM and VTG are transmitted; text fields are omitted.

In the VROS, the information is divided into the following groups:

MRO which contains the MRO records,

Black Box which contains records that are not tagged MRO,

ZwaCri which contains the ZWACRI dossier, the black list of known 'professional' criminals.

---

[17] i.e., for which arrest and detention is allowed, in Dutch *voorlopige hechtenis*, artikel 67 Wetboek van Strafvordering.

[18] MRO stands for *Melding Recherche Onderzoek*, which translates roughly to 'notice of police investigation'.

[19] In Dutch police jargon, these are called *Lopende Recherche Onderzoeks Registraties* (LROR).

[20] In Dutch police jargon, these are called *Aandachtsvestigingsregistraties*.

[21] ZWACRI stems from *zware criminaliteit*.

[22] Within the ZWACRI dossier, there are two groups of listed people: the 'S' group (for *subject*) contains those who are permanently on the black list, and the 'V' group (for *voorlopig*), who are only temporarily on the black list. The 'V' group is also called *grijze velders*. The ZWACRI dossier is also called the *S/V-index* or *Subjecten-index*.

[23] In Dutch, this stands for *Verwijsindex Recherche Onderzoeken en Subjecten*.

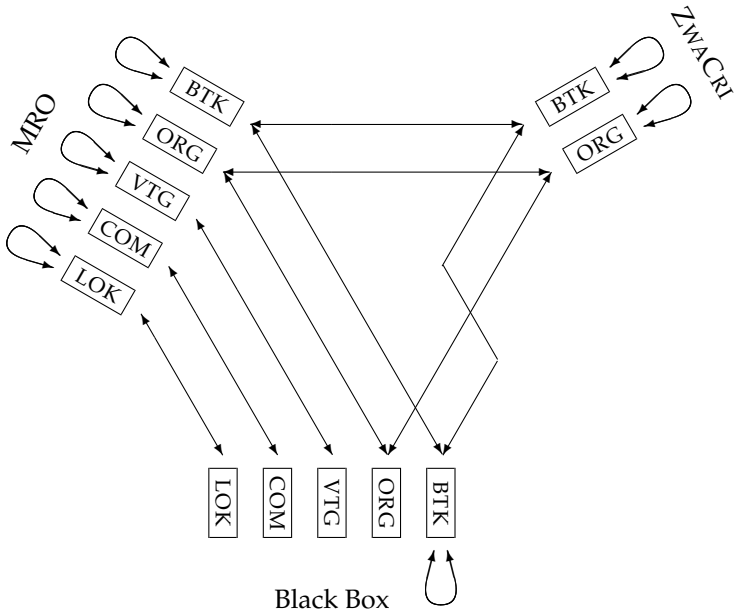


FIGURE 1.2: Matching of police information within the VROS. Every arrow represents a comparison (matching) procedure.

Within every group, all information of every field type (such as COM) is searched for duplicates (except in the Black Box, where this search is only performed for BTK fields). When in two groups the same type of information exists (such as COM), these groups are also searched for duplicates.[24] When a duplicate (a match) is found, the owners of the corresponding electronic dossiers are informed of this, with one another's contact information. Thus, every Monday morning, many police officers call one another with questions like 'I heard you are also investigating Willem Holleeder, could we find out if we can share some information?'. From there on, the process of information exchange depends on the sagacity and discretion of the police officers. The comparison that is performed by the VROS is best depicted by Figure 1.2.

After the comparison process is carried out, the information resident in the Black Box group is deleted. The information in the MRO and ZWACRI group are retained until the next Sunday morning, and is available for queries by police officers. During the week, this information at the VROS is not updated. The next Sunday morning, the process starts all over again. When a duplicate (a match) is found where both copies stem from a record which is more than

[24] Note that this matching process handles the information in non-MRO records differently from information in MRO-records: matches between two non-MRO records are only found when it involves a person. Thus, *secondary* information is only mutually compared if it involves a person. *Primary* information of any type is compared with all other primary and secondary information.

one week old (which can be seen from the time stamp), the match is no longer reported. This prevents matches from being reported more than once.

To summarize, the VROS can be consulted in two ways. The first way is implicit: by just adding records to an electronic dossier, matches are reported once every week. The second way is explicit: by typing in a search query during the week. One can only search for exact matches, queries like 'give me all people suspected of or convicted for blackmail' are not supported.

1.6 Considering Central Storage

The information exchange in the police organization, which we described in the previous section, is a sophisticated system. Sophistication should not be confused with quality of protection. There are a number of weak spots in the system.

First of all, the information that resides at the VROS is unencrypted. A small group of insiders with access to the 'James Bond' bunker in which the VROS resides, can effectively read, use and distribute the information of the VROS at their own will. This is 'the risk of the malicious system administrator'. Because the system is large and the stakes are high, the potential damage is huge. The concentration of information in the VROS makes it an interesting target for crime, whether it be in the form of corruption (bribes) or attack (computer break-in).

Secondly, the VROS is not protected against frivolous, unnecessary queries. Any police officer that performs investigations has access to the VROS. Any police officer can check whether relatives and celebrities are subject of investigation, for example. Because the number of police officers is large, and the corpus of information they have access to is large, dissemination of the information can almost be taken for granted.[25]

The central problem with the quality of the protection of the police information in the VROS boils down to central storage of information, and too generous access to the central information.

The 'VROS solution' does not scale up internationally. Police organizations are often willing to assist their sister-organizations in another country, but they do not unconditionally trust one another[26]. Moreover, the law would probably prohibit such information exchange in various ways. Thus, central storage of police information is not feasible on an international scale.

These problems are not unique for the Dutch police, but apply to all sensi-

[25] Another police information system where this is very clear is the Dutch license plate registration, to which officially only the police and some selected government organizations have full access. In practice, it is rather easy to collect this information for a given license plate number.

[26] Within a single country, the police departments are essentially obliged to trust one another by the central government. In an international context, such a central authority is lacking (by definition).

tive information held by the government. Let us take for example the discussion about the Dutch BSN[27].

The Dutch government maintains a lot of information about individual citizens, such as their employment status, marital status, enrollments in educational institutions, possession of real estate, etc. When one applies for welfare (financial aid) one has to provide this information to the government, while the information is already within the possession of the government. This situation creates an unnecessary administrative burden for the citizen, and also a possibility for fraud [HS06]. Following this line of reasoning, one might conclude that a central storage of the information about individual citizens, not much unlike the VROS, linked by means of the Dutch BSN can prevent fraud and increase service of the government.

The criticisms against the BSN are very similar to the central problems of the VROS, identified above. Mommers[28] writes [Mom06][29]:

> "The problem is, that for proper use of the BSN, we have to assume an outright incorruptible government, which will never use information for other purposes than those which are strictly necessary, and which will never undesirably give the information to others. Moreover, we must assume that it is desirable that the correct information is available always and in every circumstance."

The argument of the incorruptible government ('risk of the malicious system administrator') also holds into the future, as long as the information is retained. One may trust the current government, but maybe not the future government. In the second world war, the Nazis gratefully used the Dutch municipal inhabitants register, which recorded for every citizen the religious affiliation, including the Jews.[30] Alberdingk Thijm[31] can be considered more than just critical, and mentions a somewhat similar scenario [SOL06, page 71]:

> "Privacy violations will be omnipresent the coming years. The passed few years [sic], it was not considered decent to stand on the barricades for the right to privacy. This is largely due to 9/11 and the cry for more anti-terror legislation and less privacy-protection. The department of justice recently set up a system that enables civilians to participate in crime-fighting by recording criminal events they witnessed with mobile phones. It is striking that when such a system is proposed, the Data Protection Board[32] does not object.

[27] BSN stands for *Burger Service Nummer*, which translates into 'citizen service ID'. It is comparable to the American *social security number*.

[28] Associate professor at eLaw@Leiden, who performs research on accessibility of legal information and the interaction between law and technology.

[29] This quote is translated from Dutch by the author, and authorized by Mommers.

[30] The underground resistance eventually identified this problem, and started destroying municipals registers. The Amsterdam municipal register was set on fire on March 27, 1943.

[31] Alberdingk Thijm is also known as the attorney of Kazaa and Skype.

[32] College Bescherming Persoonsgegevens — WT

> It is obvious that there are many problems attached to such a system, such as the authenticity of the material sent by the civilians, access-control and the duration of the storage of it. Only when the supermarket bonus card[33] administration is hacked by Al Qaida [sic], or something similar, civilians will start worrying about their privacy again."

All in all, there is a large number of reasons why central storage of sensitive information is not advisable or desirable [Jac05]. To guarantee the availability of sensitive information where it is needed, *some form* of central storage is required. But is it required to store the sensitive information *itself* in a central location? Can we find a solution in which only *references* to the sensitive information are stored centrally?

The VROS was designed with these considerations in mind: the electronic dossiers are not *completely* sent to the VROS, and the VROS offers only *pointers* to other electronic dossiers and their contact persons.

A similar thing is now happening with medical files in the Netherlands: there is an index of medical files, the LSP[34] [Spa05], which links medical files to persons based on their BSN. As a result, though the LSP by itself does not contain any medical file, it is rather trivial for almost anybody in the health business to estamate the size of a persons medical record, which is of course a proxy for the general health of the person in question[35]. This demonstrates that a centralized index which does not contain any 'sensitive file' by itself, but which does contain links to such files, may still disclose information that should be better protected.

In this thesis, we present technical solutions (the T-1 and T-2 protocols, Chapters 8–10) that allow the VROS to do exactly what is does now, but without storing the sensitive information in a legible form, as it does now. Thus, the risk of central storage ceases to exist.[36] The same techniques can be used for comparing bodies of information when there is no central authority, such as the exchange of passenger information exchange between the European Union and the United States of America.

Similarly, the *information designator* (Chapter 7) is a technique that guarantees availability without creating central storage of sensitive information.

[33] Loyalty program of Albert Heijn, a large supermarket brand in the Netherlands — WT

[34] In Dutch, this stands for *Landelijk SchakelPunt*.

[35] Note that though the BSN of a person is officially secret, it is really easy to obtain this number, as it is printed on many documents, such as the drivers' licence and the passport.

[36] Moreover, this may save the Dutch police a few pennies because the need for the 'James Bond'-bunker ceases to exist.

A number of common subjects in security and cryptography literature are briefly explained: primitives, protocols, encryption, authorization, authentication, probabilistic algorithms, oblivious transfer, adversary models, secure multiparty computation and zero-knowledge proofs.

Chapter 2

Preliminaries

When one talks about security in the context of computers, there are roughly two types of security to distinguish. *Computer security* is concerned with the protection of the computer itself, typically against viruses, malware, Trojan horses and hacker attacks. *Information security* is concerned with the protection of valuable information that may reside on or pass through a computer, typically against unauthorized manipulation and theft. In practice these two types of security are closely related: when a computer is compromised, the information that resides on the computer is no longer secure against theft and unauthorized manipulation. The two types of security can be independent targets of crime: many viruses only compromise computers to send out spam, and not to steal information; information can be stolen without compromising a computer.

In this thesis, we focus on the *information security* type of security.

The discipline of *cryptography* is concerned with hiding the meaning of a message (rather than hiding its existence). There are roughly two subdisciplines of cryptography. One focuses on *cryptographic primitives*, which are algorithms which transform information in cryptographically relevant ways. Examples of cryptographic primitives are encryption and cryptographic hashes. The other subdiscipline focuses on *cryptographic protocols*, also called *security protocols*, which are communication methods that use cryptographic primitives. Typical goals of a cryptographic protocols include *authentication* (proving that you are who you claim to be) and secure communication (the messages sent cannot be interpreted or modified by an adversary).

In this thesis, we focus on *cryptographic protocols*. As cryptographic primitives are the building blocks of cryptographic protocols, they figure frequently. We *design* new protocols, and *use* primitives.

The remainder of the current chapter introduces and explains a vast number of concepts and subjects common to cryptography which are used throughout this thesis. One cryptographic primitive is so essential for this thesis, that a separate chapter is devoted to it: Chapter 3 is about cryptographic hash functions. A particular tool for analysis of cryptographic protocols is so essential for this thesis, that a number of chapters is devoted to it: Chapters 4–6 are about *authentication logics*.

2.1 Encryption

An *encryption algorithm* is an algorithm that transforms an intelligible message (also called plaintext) into an unintelligible message (also called ciphertext). To an encryption algorithm belongs also a *decryption algorithm* which transforms the ciphertext back into the plaintext. To prevent that anybody can decrypt every ciphertext, encryption and decryption algorithms use *keys*. A key is a piece of information that is essential for either successful encryption or successful decryption of a message. Thus, to protect a piece of ciphertext from being decrypted, one only has to keep the *decryption key* secret.

Keeping a key secret only makes sense when it is difficult to guess the key correctly. For example, if a particular algorithm would only support two different keys, an adversary could simply try both keys, and find out which one reveals an intelligible message. Therefore, encryption and decryption algorithms use keys that stem from a very large domain. The domain has to be this large, that it is infeasible to try out all keys, or even any reasonable fraction of all keys. Trying out all possible keys until one works is called a *brute-force attack*. Trying out keys which stem from a precompiled list is called a *dictionary attack*.[1]

Ciphertext messages are unintelligible, but that does not mean that one cannot infer anything from a ciphertext. Most importantly, the length of a ciphertext is often closely related to the length of the corresponding plaintext. This is called *message length revealing* (as opposed to *message length concealing*).[2]

For some encryption and decryption algorithms the *encryption key* (the key used to create ciphertext) and the *decryption key* (the key used to reveal the plaintext) are equal. This is called *symmetric encryption*. For other encryption and decryption algorithms the keys are not the same. This is called *asymmetric encryption*. In asymmetric encryption, the encryption key is also called the *public key*, and the decryption key is also called the *private key*.

Somebody, say, a person named Whitfield, can create a public key/private key pair, and publish his public key. Anybody who wants to send a ciphertext message to Whitfield that only Whitfield can decrypt, can simply encrypt his plaintext with Whitfield's public key. Using some tricks which go beyond the scope of this explanation, Whitfield can also send a message to anyone using

[1] The terms brute-force attack and dictionary attack do apply to the process of guessing any secret information, not just decryption keys.

[2] For a comprehensive discussion of what a ciphertext may reveal, consult [AR02].

his private key, in such a way that anybody who knows the public key can verify that Whitfield sent the message originally. This is called a *cryptographic signature*.

A good introduction to cryptography is [Sin99]. For more technical and in-depth explanations, consult [Sch96] (slightly outdated) or [FS03].

2.2 Authorization and Authentication

Cryptographic protocols are often used where *access control* is of concern, in order to limit who can view or modify particular information. The process of establishing whether someone should be given access to information is called *authorization*.

The difficulty of authorization is often not on the implementation side (does the implemented policy reflect the actual policy?[3]) but on the policy side (does the actual policy reflect the intention of the policy?[4]). For example, to protect medical information, a policy could consist of only granting physicians access to medical files. A system that does just this perfectly can still fail in its objective to protect the medical files against misuse, as the policy is arguably too tolerant: it grants every physician access to every medical file, not just to the files of his clients.

A simple and often-used method for authorization is an *access list*: a list of persons who are granted access (for example, a list of a specific group f physicians). To actually get access, someone has to identify himself (which is called *authentication*) and it has to be verified whether the person is on the list. To some it seems that authentication is necessary for authorization, and that therefore these terms can be used interchangeably, but that is certainly not the case. Authorization and authentication can occur independently of one another. This can best be explained with two examples. To get physical access to the interior of a car, one has to put the correct key into the lock and turn it (authorization). When a police officer obliges you to identify yourself, you have to present your passport or drivers' license, and doing so does not result in any access whatsoever[5] (authentication).

To complicate matters even further, *authentication* has a number of other related meanings, depending on the context. To authenticate a *message* is to verify that a message has not been tampered with. In Chapter 8, we coin the term *knowledge authentication* for proving that someone has particular knowledge without disclosing it. This may seem distant from authentication in the sense of proving that you are who you claim you are, but it is not. In cryptographic protocols, the most common way to prove you are who you claim you are is to prove that you have a particular private key without disclosing it. As such, *knowledge authentication* is a generalization of 'just' *authentication*.

[3] Answering this question is called *verification* of a system.
[4] Answering this question is called *validation* of a system or policy.
[5] Nevertheless, failing to identify (authenticate) oneself may result in access to the interior of the police office.

2.3 Complexity

A central theme in computer science is *complexity*, and this theme figures in virtually every discipline of computer science. The fundamental field of *complexity theory* is so extremely important and intricate, that one single section in an introductory chapter can only explain a few very basic concepts. The purpose of this section is to give the reader who has no background in computer science an idea of what *complexity* is about, and to refresh the memory of the reader whose years in college may have drifted from the mind.[6]

Complexity theory is the study of which amount of resources an algorithm requires, as a function of the size of the input (the 'problem description'). The most important resources are *time* and *memory*. The time assessment of an algorithm is called its *computational complexity* and its memory complexity is called its *space complexity*.

Complexities can be determined for a particular algorithm, but also for the fundamental problem that an algorithm solves. The complexity *of a particular problem* is a property of the set of *all algorithms* that solve the particular problem. More precisely, the complexity of a particular problem is a measure of the amount of resources that the most efficient algorithm for that particular problem requires.

Using complexity theory, one can assess whether it is feasible to use a particular algorithm to solve a particular problem of a particular size. One can also assess whether feasible algorithms for a particular problem exist at all. Whether an algorithm is feasible obviously depends on the amount of available resources.

A notorious class of infeasible problems is the class of NP-complete problems. Informally, an NP-complete problem is a problem in which a solution has to be found, and a solution can be *verified* to be correct in a number of steps which is polynomial in the size of the input. However, it may be that the only way to *find* a correct solution is to try *all possible* solutions (of polynomial length). An example of an NP-complete problem is the subset sum problem: for a given, finite list of integers, find out whether any subset of the integers sums up to zero. For any given subset, it is very easy to verify whether it sums up to zero. But there is no algorithm known that determines whether such a subset exists that is significantly faster than trying every possible subset, which is very slow. NP-complete problems are so difficult to solve, that by increasing the problem size slightly, the resource requirements increase dramatically.

NP-complete problems play a central role in cryptography: an encryption algorithm is only considered good if to construct the plaintext from the ciphertext, without access to the decryption key, is an NP-complete problem.

Complexity does not only apply to algorithms, but also to communication protocols. The *communication complexity of a particular protocol* is a measure of how many bits are exchanged in a protocol run. A protocol can be seen as a way to compute a particular function $f(X, Y)$ (the 'problem'), where X and Y are

[6] For a thorough treatment of complexity, consult [BDG88].

```
trivial-prime-test(n)
   for i = 2 to squareroot(n) do
      if (divisor(i, n)) then return composite;
   return prime;
```

FIGURE 2.1: A trivial primality testing algorithm. This algorithm gives a definite answer to the question whether n is prime. The algorithm could be optimized somewhat, but the running time remains $O(\sqrt{n})$.

```
miller-rabin-prime-test(n, k)
   for i = 1 to k do
      set w to some randomly chosen number 0 < w < n;
      if (not witness(w, n)) then return composite;
   return prime;
```

FIGURE 2.2: The Miller-Rabin primality testing algorithm. The accuracy of this algorithm depends on the security ('certainty') parameter k, a higher value of k will increase the accuracy. When this algorithm answers that a number is composite, then it is indeed composite. On every 4^k occasions that this algorithm answers a number is prime, the expected number of errors is at most one. The function `witness` is a subroutine performing some specific arithmetic test between w and n. The running time of this algorithm is $O(k \ \ln\ln\ln n)$.

known to separate parties. The *communication complexity of a particular problem* is a measure of how many bits need to be exchanged between the parties in the most efficient protocol that computes $f(X, Y)$ [Yao79, Kus97].

2.4 Probabilistic Algorithms

For many computational problems, it is very easy to specify how they should be solved. For example, a program that gives a definite answer to the question whether a specific number is prime, is only a few lines long (see Figure 2.1). For large numbers, this primality testing algorithm would take a prohibitively long time to compute. It is possible to improve dramatically on the computation time of this test, if we allow the test to be wrong in its answer in only a negligible fraction of the occasions it is invoked. Such an algorithm is a *probabilistic* algorithm, and it requires access to some source of randomness (like a virtual coin which it may flip).

Probabilistic algorithms have the very tricky property that they typically detect *the opposite* of the property one is interested in. When it fails to detect the opposite, it guesses the affirmative to be the case. A well known example of a probabilistic algorithm is the Miller-Rabin primality test [Mil75, Rab80] (shown in Figure 2.2). This algorithm tries to find proofs of compositeness[7] of a number n. If it fails to find such a proof for a sufficiently long time, it will

[7] Compositeness is the opposite of primality.

assume that n is a prime. This is not just some 'guess', it can be proven that the chance that the assumption is wrong can be made arbitrarily small, depending on the time the algorithm invests in finding proofs of the opposite.

Thus, a probabilistic algorithm is an algorithm that may give the wrong answer, but only in very few cases. Probabilistic algorithms are employed because they are often much much faster than non-probabilistic algorithms. Almost all algorithms used in practice in cryptography are probabilistic.

2.5 Oblivious Transfer

Oblivious transfer (OT), introduced by Rabin [Rab81] is a type of protocol in which the sender sends a bit with probability $1/2$, and remains oblivious whether it was received. Even, Goldreich and Lempel generalized this type of protocol to *1-out-of-2* oblivious transfer [EGL85]. In *One out of two* oblivious transfer, the sender sends two messages, and the receiver can choose to read one of the messages. The receiver can read that single part of the message, but not the other. *What* message has been chosen, remains secret to the sender.

This rather weird type of protocol has a wide range of applications. For example, it can provide anonymity to buyers of digital goods. The seller (the sender in the protocol) cannot determine what the buyer has bought, but the seller can verify the item is paid for and only one single item is delivered [AIR01]. Oblivious transfer also has applications in auctions [Lip04]. In general, oblivious transfer is a building block for more complex protocols. Therefore, though oblivious transfer is technically a kind of cryptographic protocol, it is often considered a cryptographic primitive.

2.6 Adversary Models

The strength of a security protocol depends on the strength of the party that tries to break the protocol. Thus, when assessing the security of a protocol, one has to make assumptions about what the *adversary* is willing to do and about what he is capable of. Such an assumption is called an *adversary model* (or a *threat model*). There are three common adversary models:

honest The adversary is supposed to completely adhere to the protocol specification, and not to do anything more than the protocol specification.

honest-but-curious (HBC) The adversary is supposed to completely adhere to the protocol specification, but is allowed to perform extra calculations on the information it receives. This model is sometimes called the *semi-honest* adversary model. An attacker in this model is sometimes referred to as a *passive* attacker.

malicious (Dolev-Yao) The adversary is not required to adhere to the protocol specification, and is allowed to perform extra calculations on the information it receives. Moreover, the adversary is capable of intercepting

> any message sent, and is capable of sending any message he is able to compose. An attacker in this model is sometimes referred to as an *active* attacker. [DY83]

In the honest-but-curious and the malicious adversary model, the adversary may perform calculations not specified in the protocol, but the amount of calculations is polynomially bounded. In particular, the adversary cannot perform brute-force attacks to find secret keys.

The honest adversary model is very weak, and therefore only erroneously used. Consider the following protocol for oblivious transfer:

> Alice sends Bob a message, but according to the protocol, Bob is allowed to only look at either the first half of the message, or the second half of the message.

This protocol is only secure if Alice knows that Bob is honest, that is in the *honest adversary model*. Trivially, this protocol is insecure in the honest-but-curious model.

When a protocol is insecure in the honest-but-curious adversary model, it is possible for some principal (i.e., a particupant in the protocol or an external observer) to obtain information that should be kept hidden from the principal. When a protocol is insecure in the malicious adversary model, it is possible for some principal to 'deceive' some other principal into obtaining false beliefs.

The malicious (Dolev-Yao) model is the strongest model, in the sense that if a protocol is secure in this model, then it is really very secure. As can be expected, it is rather difficult to prove a protocol to be secure in this model.

2.7 Secure Multiparty Computation

In his seminal paper "Protocols for Secure Computations" [Yao82], Yao has given a clear definition of what constitutes *secure multiparty computation* (SMC). Suppose there are two principals, Alice who possesses X and Bob who possesses Y. Alice and Bob are both interested in the value of some function $f(X, Y)$. A protocol for determining $f(X, Y)$ is an SMC if it satisfies the following conditions:

privacy The inputs X and Y are not mutually disclosed: Alice does not learn anything about Y except $f(X, Y)$, and Bob does not learn anything about X except $f(X, Y)$.

validity The protocol actually establishes $f(X, Y)$ and not something else. If one of the principals cheats, the other principal can detect this.

fairness Either both Alice and Bob learn $f(X, Y)$, or neither of them learns $f(X, Y)$. Essentially, the probability that one principal knows $f(X, Y)$ and withholds it from the other principal, is very small.

autarky Bob and Alice can determine $f(X, Y)$ without the assistance of a third party.[8]

Yao showed that if $f(X, Y)$ can be computed on a binary circuit that has m input wires and n gates, then there exists an SMC protocol with m oblivious transfers and communication complexity of $O(n)$, in a constant number of rounds [Yao86].[9] This may seem a promising result, but the feasibility of this solution is questionable at least. In 2004, eighteen years after publication of [Yao86], it was demonstrated that computing the rather trivial function $f(X, Y) = [X < Y]$ with $0 \leq X < 16$ and $0 \leq Y < 16$ would already take 1.25 seconds [MNPS04][10].

Feige, Kilian and Naor have proven that SMC is possible for any function $f(X, Y)$, if the condition of autarky is relaxed somewhat: their solution involves a third party, which can only learn the outcome of $f(X, Y)$, but nothing else. This third party is then supposed to honestly inform Alice and Bob about the outcome [FKN94]. The communication complexity of their solution, however, is not convincingly attractive, just as [Yao86].

SMC is not to be confused with *secure circuit evaluation* (SCE) [AF90], in which one principal provides the input X, and the other principal provides the function $f(\cdot)$. The aim in SCE is to compute $f(X)$ without mutual disclosure of X and $f(\cdot)$. Importantly, SMC is not a special case of SCE because in SMC it is known to both participants which function is computed. Conversely, SCE can be perceived as a special case of SMC, where the inputs are X and $g(\cdot)$, and where the function $f(X, g(\cdot))$ applies function $g(\cdot)$ to X.

2.8 Zero-Knowledge Proofs

Goldwasser, Micali and Rackoff introduced the concepts *zero-knowledge* and *interactive proof systems* in 1985 [GMR85].

Suppose again that there are two principals, now called Peggy (the prover) and Victor (the verifier). Peggy wants to convince Victor that she has some very special knowledge. For example, she knows the secret ingredients to make Coca-Cola[11], she can solve any Rubik's cube[12] (shown in Figure 2.3), or alternatively: she knows her own private key. Peggy wants to convince Victor, but without disclosing the special knowledge itself.

We will first focus more generally on protocols which convince Victor of Peggy's special knowledge. Protocols that convince Victor without disclosing the secret are a subclass of these protocols.

[8] The condition of autarky has not been explicitly named as such in [Yao82]. It is included here for clarity.

[9] This result has been generally accepted, but a proof has never been published.

[10] This is on two PCs with 2.4 GHz processors, and a communication link with 617.8 MBPS bandwidth and a latency of 0.4 ms. If the communication link is changed to a wide-area setting, e.g. a bandwidth of 1.06 MBPS and a latency of 237.0 ms, the computation time increases to 4.01 seconds.

[11] Coca-Cola is a registered trademark of The Coca-Cola Company.

[12] Rubik's Cube is a registered trademark of Seven Towns Ltd.

FIGURE 2.3: A Rubik's cube. Picture courtesy of Seven Towns Ltd.

Definition 2.1 (Interactive Proof Systems[13])**.** *An* interactive proof system for a set S *is a two-party game between a* verifier *executing a probabilistic polynomial-time (based on polynomial p) strategy (denoted V) and a* prover *executing a computationally unbounded strategy (denoted P), which satisfies the following conditions:*

completeness *For every $x \in S$ the verifier V always accepts after interacting with the prover P on common input x.*

soundness *For some polynomial p, it holds that for every $x \notin S$ and every potential strategy P^*, the verifier V rejects with probability at least $1/p(|x|)$, after interacting with P^* on common input x.*

The terms *soundness* and *completeness* have a meaning different from the meaning in logic (where it refers to the relation between logics and models).

The soundness condition may seem tricky or weak, but observe that when such a proof system is repeated $O(p(|x|)^2)$ times, the probability that V accepts for an $x \notin S$ reduces to $2^{-p(|x|)}$, which is 'close to zero'. The fact that the prover is not computationally bounded should not automatically be considered a structural problem either: the soundness criterion holds the prover to not cheating, and any practical implementation of an interactive proof will be forced to bounded computational resources by mere reality.

If Peggy would like to prove that she knows the solution for any Rubik's cube, she could challenge Victor for a scrambled Rubik's cube. Victor may choose or construct some scrambled Rubik's cube x, and hand it over to Peggy. Peggy could in turn, under the watchful eye of Victor, solve the puzzle. This rather trivial interactive proof is both complete and sound. It is complete because if Peggy knows the solution for x, she can solve it. Though Victor may not be proficient in solving Rubik's cubes, he can easily verify Peggy's solution. This makes the strategy sound. This trivial protocol which proves Peggy's knowledge of how to solve a Rubik's cube, discloses the solution to Victor.

There is also a solution which does not disclose the solution for x to Victor: when Peggy wants to prove knowledge of the solution of Rubik's cube x, Peggy presents another Rubik's cube y. Victor may then either (1) ask Peggy to solve y, or (2) ask Peggy to show how x can be transformed into y. Victor

[13] This definition, which is cited from [Gol02], is a slight variation of the original definition, which can be found in [GMR85].

may not ask both, but he may choose one of both as he wishes. If Peggy is able to solve every Rubik's cube, she will be always be able to perform both the solution of y, and the transformation of y into x.[14] These two together constitute the solution of x. But that solution will never be disclosed to Victor, as he will only see (1) or (2), but never both. The first time Peggy and Victor run this protocol, Victor might believe that Peggy has had the sheer luck that she knows the half of the solution that Victor asked for. But if they repeat the protocol k times, the chance that Peggy does not know the solution reduces to 2^{-k}, which converges to 0 as k increases.

This second protocol is a *zero-knowledge proof*: it convinces Victor of Peggy's knowledge, without disclosing the knowledge itself. Roughly, a zero-knowledge proof is an interactive proof in which the verifier learns nothing more than the assertion proven. The more formal definition of 'nothing more' states that anything that Victor learns by means of the protocol, can be computed just as efficiently from the assertion proven by the protocol alone. For an elaborate discussion of the definition of 'nothing more', consult [Gol02]. For rather simple explanations and examples of zero-knowledge proofs, consult [QQQ+90, NNR99].

It has been shown that every NP-complete set has a zero-knowledge proof [GMW91], provided that one-way functions exist.[15] This is a very valuable result, because it can be used in a cryptographic protocol to convince other parties that one adheres to the protocol specification without disclosing ones private inputs. Instead of disclosing the private inputs, principals must provide a zero-knowledge proof of the correctness of their secret-based actions. This means that any protocol which is secure in the honest-but-curious adversary model can be transformed into a protocol which is secure in the malicious adversary model [Gol02]. Unfortunately, the computational complexity and communication complexity of such a transformed protocol which is secure in the malicious adversary model may well be very unattractive.

There are some correspondences between secure multiparty computation (see Section 2.7) and zero-knowledge proofs. Zero-knowledge proofs satisfy properties roughly equivalent to the the *privacy* and *validity* properties of a secure multiparty computation. *Privacy* because no information on the private inputs is disclosed, and *validity* because an interactive proof is both *sound* and *complete*.[16] Because the roles of the principals in interactive proof systems are not symmetric, *fairness* is not a relevant property. *Autarky* applies to interactive proof systems in the sense that no third party is involved.

[14] Remember that Peggy may choose y. If Peggy is knows the solution to x, she can construct a y in such a way that she knows both how to solve y and how to transform y into x. If Peggy does not know the solution to x, she can guarantee only one of both, giving her a 50% chance of failing to provide a convincing proof on y.

[15] Cryptographic hash functions are a particular type of one-way functions, and are explained in the next chapter.

[16] Again: logicians, take note that sound and complete here are used with the meaning of Definition 2.1 shown on page 25.

Part II

Tools

There are many kinds of cryptographic hash functions and we present many definitions. We give a survey of design paradigms and common applications of cryptographic hash functions. We introduce the concept of non-incrementality, which is necessary when we want to use cryptographic hash functions in proofs of knowledge.

Chapter 3

Cryptographic Hash Functions

Typically, cryptography is about encoding and decoding. One can take a message and encode it in such a way that only specific people can reconstruct (decode) the original message from the encoded message (because they have the right secret key). Thus, though messages are mangled in such a way that the ciphertext looks like random noise, it is possible to reconstruct the original message from the ciphertext. This chapter is not about this type of cryptography. This chapter is about cryptographic functions (primitives) which have no inverse: it is impossible (or at least hard) to reconstruct the original message from the ciphertext. Such functions are called *cryptographic hash functions*[1] On first sight it may seem that this type of functions has no sensible application, but this is not the case. In this chapter, we will show a number of well-known applications. Later on in this thesis, we will show a whole new type of application of this type of functions. In Section 3.6 we will elaborate on these new applications, which will be shown in greater detail later on in Chapters 8–10 of this thesis.

In this chapter, many hash function-related concepts necessary for understanding this thesis will be explained. In cases where multiple names for the same concept exist, a footnote at the first use of the concept will list equivalent names of the concept. A more detailed taxonomy of cryptographic hash functions can be found in [Pre93, Pre98]. A highly valuable reflection on the definition of cryptographic hash functions can be found in [And93].

Cryptographic hash functions can be regarded as a special case of hash functions. Therefore, Section 3.1 will explain what a 'normal' hash function actually is, and Section 3.2 will elaborate on what distinguishes cryptographic

[1] Sometimes cryptographic hash functions are called *one-way functions* [DH76].

hash functions from normal hash functions. In Section 3.3 we will explain the *Random Oracle Model*, which can safely be regarded the theoretical ideal of a cryptographic hash function. In Section 3.4 two approaches to construct cryptographic hash functions will be described: the *Merkle-Damgård paradigm*, and the *randomize-then-combine paradigm*. In Section 3.5 we will give brief survey of applications of cryptographic hash functions.

The concept of cryptographic hash functions as they are generally used is not strong enough for our purposes in this thesis. Therefore, we will explain and define the concept of non-incrementality in Section 3.6. In the final section of this chapter, we will briefly summarize what properties of a cryptographic hash function are required for the new applications described later on in this thesis (in Chapters 8–10),

3.1 Normal Hash Functions

In this section, we will give a definition of hash functions, elaborate on this definition, and explain what hash functions are typically used for.

Definition 3.1. *A function is a hash function if it*

1. *takes its input from a large (possibly infinite) domain,*[2]
2. *has a bounded range,*
3. *is designed to minimize* collisions,
4. *is designed to be fast to compute, and*
5. *is deterministic.*

The *input* to a hash function is often called a message or pre-image. The *output* of a hash function is often called the *hash value*, or just simply *the hash*.

If two different pre-images $M_1 \neq M_2$ have the same hash value ($H(M_1) = H(M_2)$), we have a *collision*. Because of the birthday paradox[3], one can be sure that in practical situations collisions will occur.

A few observations about the list of properties of a hash function should be made here. Properties 1 and 2 imply that a hash function is in practical terms always many-to-one. Property 3 implies in practical terms that the output of a hash function should depend on the complete input of the hash function.

[2] In theory, a hash function is a function $H\colon \{0,1\}^* \to \{0,1\}^k$, but in practice the input domain is bounded by a very high number, for example SHA-512 ($H\colon \{0,1\}^{2^{128}} \to \{0,1\}^{512}$) can handle messages with a length of 2^{128} bits [Nat02]. To give some perspective, there are not even 2^{128} bits stored together on all hard disks worldwide.

[3] The birthday paradox states that if there are 23 or more people in a room, the chance that two of them have the same birthday is more than 50%. This number of 23 is much lower than what one might expect from intuition. It stresses that the chances of a "collision" are often underestimated by humans.

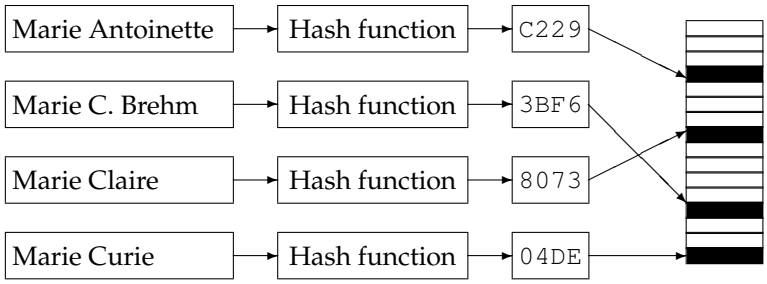


FIGURE 3.1: A 'normal' hash function in action. To the left is a list of (highly non-uniform) names. All of these names are each led through a hash function, resulting in a list of 'more-or-less uniform' hash values. To the right is a data structure with allocated slots (black) and unallocated slots (white). The hash function minimizes the chance that two highly similar names should be stored in the same slot.

Property 4 simply follows from the general goal in computer science to produce efficient computer programs. Later on in this chapter, however, we will see applications where "fast to compute" should apply only to specific uses of hash functions, and not to others, such as, for example, the inverse of a hash function. Property 5 is so obvious for computer scientists that it is often taken for granted, and therefore it is often not considered as a part of the definition. However, since many cryptographic primitives, such as encryption, are often non-deterministic, we deem it important to stress that hash functions are *always and by definition* deterministic.

Definition 3.1 does not mention a typical application, but hash functions are normally used to optimize data structures, and are often not 'visible' from the outside of the data structure. Range sizes of such hash functions are typically only a few orders of magnitude larger than the number of elements stored in the data structure (say a few thousand, or maybe a few million at the most).

In data structures it is often needed to store data values together with some *identification* (an *index*) in such a way, that it is efficient to locate the corresponding data value if given an identification. For example, a teacher would like to find the grades of a student quickly, if he is given a student name. To provide this functionality, one needs a look-up-table (LUT), a data structure that maps indexes to the corresponding data. A LUT has to be designed with a balance between storage requirements and search time. The naive solution is to reserve a *memory slot* for each possible index. This solution wastes an incredible amount of storage space, since usually the number of stored data values is only a tiny fraction of the number of possible indexes. Indeed, the domain of indexes may be infinite. A less naive solution could be to reserve a single slot for a number of large chunks of possible indexes, and hope that it will not happen that the corresponding slot will be needed more than once at the same time.

Therefore, one has to be smart in the way the full domain of possible in-

dexes is divided over the allocated slots. This is where a hash function can be used to optimize data structures. A hash function can be used to determine, given an index, in which allocated slot the corresponding data value should be stored. Figure 3.1 shows an example of how a hash function attributes hash values to indexes and how this determines slot usage.

In most applications, the actual indexes have a highly non-uniform distribution over the domain of possible indexes. If no precautions are taken, many indexes will be projected onto a small set of hash values, which results in a lot of collisions and a lot of unused slots. Thus, a hash function should be designed in such a way that even for highly non-uniform input, its output should be more or less uniform.

Moreover, it should be noted that so far, there is nothing secretive about hash functions. Given a specific hash function, it will generally be easy to construct different pre-images in such a way that they will lead to a collision in a given hash function. Also, it may be easy to infer some properties of the pre-image from the output of the hash function.

3.2 Special Properties

From a cryptography point of view, hash functions are interesting if it would be possible to strengthen some of their properties to the extreme. Hash functions need to be strengthened in such a way that, for example, for any set of messages, (1) the set of hash values is indistinguishable from a uniform distribution, and that (2) collisions are not just unlikely, but actually hard to find. If this could be achieved while the hash function is still cheap to compute, it would open up a whole lot of applications, most of which will be described in section 3.5.

The special properties which distinguish *cryptographic* hash functions from 'normal' hash functions are all related to computational complexity. It definitely goes beyond the scope of this chapter, even this thesis, to give full formal definitions of the hardness properties introduced later on in this section. There are many different ways to define them, and the technicalities involved in these definitions are not relevant for our purposes. Nevertheless, we feel that some clear indication of what we mean by *easy* and *hard* should be provided.

The parameters which play a role in the complexity figures of cryptographic hash functions are l, the length in bits of the pre-image of the hash function, and k, the length in bits of the output of the hash function. For a given hash function, k is always given, and l (obviously) depends on the pre-image with which one feeds the hash function. For a hash function to be *easy to compute*, or *computationally feasible*, means that the number of operations is at most polynomial in l.[4] A computation is considered *hard to compute*, or *computationally infeasible* if the number of required operations is superpolynomial in terms of the input. Specifically, we consider a hash function hard to invert if, given a hash value h,

[4] In practical cases, it is common that the number of operations is at most linear in l.

it requires $O(2^k)$ operations to find a (second) pre-image. Of course, an attack that requires less than $O(2^k)$ may in practical terms still be infeasible.

It depends on the application how far, and in what way, the properties of a hash function must be strengthened. There is no such thing as an all-purpose cryptographic hash function, because requirements of one application may be incompatible with requirements of other applications. To choose a type of cryptographic hash function is therefore to select properties from a menu of possible options. Some options are mutually exclusive, some options can be combined, and some options imply others.

Deciding what properties a cryptographic hash function should have is a very complex task, and in [And93] some stunning examples of overlooked, but required properties in certain applications are shown. Terminology is partially to blame, as the operationalization of a concept is often given the name of the concept itself, though the operationalization is often far from perfect. The most blatant example of this is that a hash function that is called 'collision-free' actually has infinitely many collisions. It is not advisable to literally interpret the linguistic meaning of the words which make up the name of a concept.

For a hash function, one has to choose its *keyedness*, its *freedom*, its *key dependency level* and its *incrementality*. The full menu of items and options to choose from is shown in Figure 3.2.

Keyedness The first choice on the menu is whether or not the cryptographic hash function should be *keyed*. A keyed cryptographic hash function is often called a Message Authentication Code, or simply MAC.[5] A MAC takes two inputs: a key and the pre-image. In the context of MACs, the key is a piece of information which makes computing the function actually feasible: without the key, not only the inverse is hard, but also the 'forward' direction of computation is impossible. Any application of a MAC should include a specification of which principals should know the key, and which principals should not. Otherwise, if the key were to be publicly known, the keyed cryptographic hash function would reduce to a non-keyed cryptographic hash function. (In fact, in many circumstances it reduces to even *less* than a non-keyed cryptographic hash function, which will be explained later on in this section.) If it is unspecified whether a cryptographic hash function is keyed or not, a non-keyed cryptographic hash function is assumed.

Freedom The second choice on the menu is about how obscure the relation between the input and the output should be. In the definitions to come, the parts between square brackets [] give the definitions for MACs. The bitwise exclusive or is written as $\oplus$. In these definitions, *computationally infeasible* means that there exists no polynomial-time function to perform the task mentioned, given the usual assumptions about the complexity hierarchy.

[5] The opposite, a cryptographic hash function that is not keyed, is often called a Manipulation Detection Code, or simply MDC.

One-Way A hash function $H(M)$ [$MAC(K, M)$] is one-way[6] if, for a given hash value h, it is computationally infeasible to construct a message M [and key K] in such a way that $H(M) = h$ [$MAC(K, M) = h$].

Weakly Collision-Free A hash function $H(M)$ [$MAC(K, M)$] is weakly collision-free[7] if, for a given message M_1, it is computationally infeasible to find a message M_2 such that $M_1 \neq M_2$ and $H(M_1) = H(M_2)$ [$MAC(K, M_1) = MAC(K, M_2)$].

Strongly Collision-Free A hash function $H(M)$ [$MAC(K, M)$] is strongly collision-free[8] if it is computationally infeasible to find any two messages M_1 and M_2 such that $M_1 \neq M_2$ and $H(M_1) = H(M_2)$ [$MAC(K, M_1) = MAC(K, M_2)$].

Correlation-Free A hash function $H(M)$ [$MAC(K, M)$] is correlation-free if it is computationally infeasible to find any two messages M_1 and M_2 such that $M_1 \neq M_2$ and the Hamming weight[9] of $H(M_1) \oplus H(M_2)$ [$MAC(K, M_1) \oplus MAC(K, M_2)$] is less than what one would expect if one were to compute $H(M_1) \oplus H(M')$ [$MAC(K, M_1) \oplus MAC(K, M')$] for a lot of randomly chosen M' [Oka93, And93][10].

Correlation freedom means in practical terms that not only collisions are very unlikely and hard to find, but also that *near misses*[11] are unlikely and hard to find. Thus, correlation freedom is a strictly stronger property than strong collision freedom. Moreover, strong collision freedom is a strictly stronger property than weak collision freedom, and weak collision freedom is strictly stronger than one-wayness. This is summarized in Figure 3.2.

Key Dependency For MACs, there are a few more relevant properties, which a MAC is generally assumed to satisfy:

Key-Dependent A MAC $MAC(K, M)$ is key-dependent if, given a pre-image M, it is hard to compute $MAC(K, M)$ (that is, without K)[12].

Chosen Text Attack-Resistant A MAC $MAC(K, M)$ is resistant against a chosen text attack, if given any number of freely chosen pairs $\{M', MAC(K, M')\}$, it is still hard to compute $MAC(K, M)$ for any $M \neq M'$.

---

[6] One-way is also called *first pre-image resistant*.
[7] Weakly collision-free is also called *second pre-image resistant*.
[8] Strongly collision-free is also called *collision-resistant*.
[9] The Hamming weight of a binary string is the number of nonzero bits is the particular string.
[10] Of course, the Hamming weight of the bitwise exclusive or ($\oplus$) of two bitstrings is their Hamming distance.
[11] A near miss is, roughly, that two different messages produce hash values which, though different, are very similar. For example, two hash values are identical except for one or two bits.
[12] Or likewise, one could say that it is hard to guess $MAC(K, M)$ with a chance of success significantly higher than $1/2^k$, where k is the length in bits of the hash value.

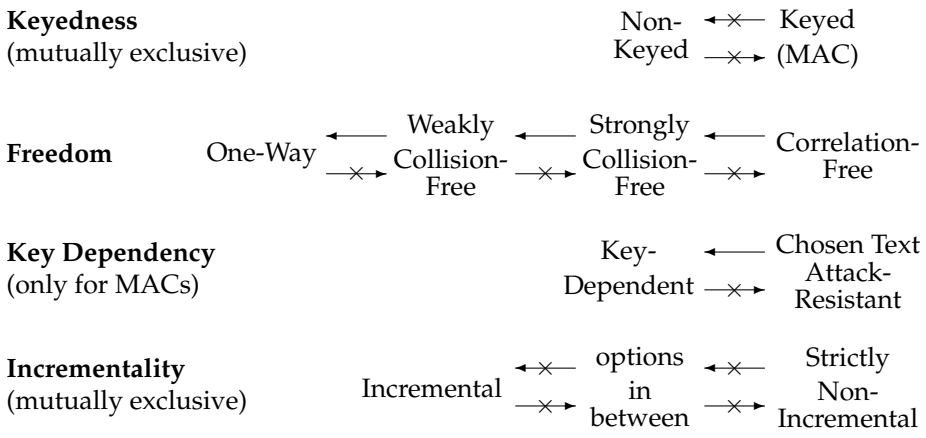


FIGURE 3.2: The relation between various properties of cryptographic hash functions. For the menu items *keyedness* and *incrementality*, the options are mutually exclusive. For the menu items *freedom* and *key-dependency*, the options are increasingly stronger from left to right (e.g. correlation freedom implies strong collision freedom, but not vice versa).

Incrementality For an explanation of the property of incrementality, we refer to Section 3.6. We mention it here for completeness only.

With the complete menu given, we can introduce some commonly used terms for cryptographic hash functions: A *one-way* hash function (OWHF) is one that is not keyed, one-way and weakly collision-free. A *collision resistant* hash function (CRHF) is an OWHF that is also strongly collision-free. There also exists something like a *universal one-way* hash function (UOWHF), which is stronger than an OWHF but weaker than a CRHF [NY89]. We omit its definition for reasons of simplicity.[13]

A Message Authentication Code (MAC) is considered to be key-dependent and resistant against a chosen text attack. It should be noted that key-dependency and resistance against a chosen text attack jointly imply that a keyed hash function is (strongly) collision-free and one-way for someone who does not know the key K. However, a MAC may, by design, actually not be one-way and not collision-free for someone who *does* know the key K. Thus, a MAC $MAC(K, M)$, with a publicly known key K, should not be considered equivalent to a CRHF (or even a OWHF) $H(M)$: the MAC may have none of the interesting properties of the CRHF (!)[14]

[13] For completeness, there are also things like *universal*$_n$, *strongly universal*$_n$ and *strongly universal*$_\omega$ classes of hash functions [CW79, WC81].

[14] For this reason, many experts feel that a MAC should not be considered a cryptographic hash function at all. For completeness and clarity, we have chosen to include MACs in this survey.

3.3 The Random Oracle Model

It has been proven that strong collision freedom is an insufficient property to guarantee *information hiding* and *randomness* [And93]. Information hiding and randomness are, stated informally:

Information Hiding The hash value ($H(x)$) does not leak any information on the pre-image (x).

Randomness The output of the hash function is indistinguishable from random noise.

The history of the definition of cryptographic hash functions is littered with problems popping up every now and then. One cycle of history typically includes: (1) the formal definition of one of the abovementioned properties, (2) the use of a function satisfying the property in some protocol, (3) finding out that the protocol can be broken by some attack on the hash function, and (4) adjusting the definition of a hash function to defy the attack. The properties of one-wayness, weak collision resistance, strong collision resistance and correlation freedom should be regarded as iterations of progressive insight. Not so long ago, strong collision resistance was considered sufficient for many applications, whereas now for the same applications correlation freedom is considered necessary. I would not at all be surprised if correlation freedom will at some point in the near future be proven insufficient for many applications as well.

In fact Preneel, one of the most respected researchers in the field of cryptographic hash functions, recently stated that "we understand very little about the security of hash functions" and "designers have been too optimistic (over and over again...)" [Pre05].

This symptomatic practice leads to the more fundamental question of what notion it is we would like to actually define. What 'real, practical' properties do we believe a 'real' cryptographic hash function actually has? The ideal cryptographic hash function differs from a random function only in that it is deterministic and easy to compute. This defies any formal expression, and the *random oracle model* is the next-best thing one can get. We will introduce this model now.

The purpose of the random oracle model [BR93], introduced by Bellare and Rogaway, is to provide protocol designers with a clear definition of what they can expect from an 'ideal' cryptographic hash function (the random oracle). Whether such ideal cryptographic hash functions actually exist, is a completely different question. The random oracle satisfies 'any property one generally addresses to the notion of a cryptographic hash function'. When designing a protocol, this is of course very useful. A random oracle is defined as follows:

Definition 3.2 (Bellare-Rogaway)**.** *A Random Oracle* $R\colon \{0,1\}^* \rightarrow \{0,1\}^\infty$ *is a map available to all parties*[15]*, good and evil, and every party is allowed to ask the*

[15] There are no such things as 'private oracles'.

oracle only polynomially many questions. Each bit of $R(x)$ is chosen uniformly and independently.

In practical terms, when an oracle is given a question q, it does the following:

1. If the oracle has seen the question q before from whatever party, it gives the answer it gave upon the previous time when it was asked question q.
2. If the oracle has never seen the question q before, it returns a random string of infinite length.

The poser of the question may (and in all practical cases will) instruct the oracle not to physically return the full length random string, but just a prefix of this string of a certain given length.

It is easy to see that a Random Oracle satisfies the properties of information hiding and randomness, as well as all the properties given in Section 3.2.

When using the Random Oracle Model, all calls to a cryptographic hash function are replaced with calls to the oracle (which one might call a black box [Sch98]). Then the protocol is proven correct within this setting. Because such oracles do not seem to exist in real life, the oracle consult has to be replaced again by a call to a 'suitable' cryptographic hash function[16], when such a protocol is deployed in real life.

The Random Oracle methodology is not sound: though it can help detecting protocol flaws, protocols proven secure in the Random Oracle Model cannot be assumed secure when the oracle is replaced by an implementation of a cryptographic hash function [CGH98]. Moreover, it is shown that to replace the oracle with an implementation, one faces some very serious problems [BGI+01]. Nevertheless the Random Oracle methodology is a very valuable one. It provided the best formalization of the properties addressed to cryptographic hash function so far. It has been of great value to protocol design and analysis. Protocols proven correct in the Random Oracle methodology can 'in real life' only be broken by an attack on the internal structure of the hash function, within the setting of protocol interactions [BM97].

3.4 Design Paradigms

The operational design of a hash function is as complex as its definition. But how are cryptographic hash functions actually designed? There are currently two paradigms, the *Merkle-Damgård paradigm*, and the *randomize-then-combine paradigm*. Both paradigms chop up the pre-image of the hash function in a series of fixed-length blocks of bits, and then combine these blocks in some specific way. Depending on the paradigm, the combining specifics differ.[17]

[16] The notion of 'suitable' has not yet been formalized in the literature.

[17] Furthermore, when these paradigms are compared with modes used in cryptography, one can see the Merkle-Damgård paradigm has some similarities to the cipher-block chaining (CBC) mode, and the randomize-then-combine paradigm has some similarities to the electronic codebook (ECB) mode.

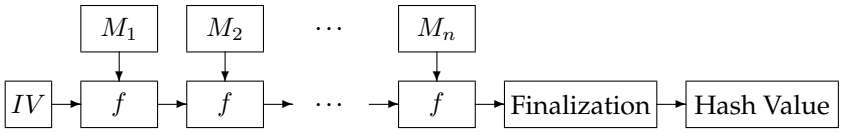


FIGURE 3.3: A Merkle-Damgård hash function. The message M consists of n blocks of the same size, $M_1, M_2, \ldots M_n$. The initialization vector IV has a fixed value. The function f is called the *compression function*.

Because the length of individual blocks is fixed, the original message has to be modified (padded) in such a way that its length is a multiple of the block length. This can be done by adding zeroes at the end of the message until its length is a multiple of the block length. Care has to be taken to make sure that this padding procedure does not weaken the hash function by mapping different pre-images to the same modified pre-image. This problem is solved by encoding the length of the pre-image into the padded message.[18]

Merkle-Damgård In the Merkle-Damgård paradigm [Mer90b, Dam90], the individual blocks are combined by means of a *compression function* f, which takes as input the 'hash so far' and the next message block. At the beginning of the message, the 'hash so far' is a constant *initialization vector* (IV). When all blocks have been processed, the 'hash so far' may undergo some *finalization*, which results in the hash value. In Figure 3.3 the information flow of a hash function using this design is shown graphically.

A hash function in the Merkle-Damgård paradigm is *chaining* (or iterative): blocks earlier in the sequence influence how the blocks later in the sequence are processed. One implication of this is that this type of hash functions cannot be parallellized: adding more hardware cannot speed up the computation of a hash value. Examples of hash functions using this paradigm are MD5, SHA-1 and RIPEMD-160 [DBP96]. Merkle and Damgård have proven that if the compression function is strongly collision-free, then so is the whole hash function [Mer90b, Dam90].

Randomize-then-combine Another approach to cryptographic hash function design is provided by the *randomize-then-combine paradigm* by Bellare et al [BM97, BGG95, BGG94]. Instead of sequentially processing all blocks of the message, the n blocks are processed independently by a *randomizing function* g[19]. The n obtained results are then combined using some well-chosen *combining function* $\odot$. This combining function $\odot$ is chosen in such a way that it is associative[20] and commutative[21] within a group.

[18] This technique is sometimes called MD-strengthening.

[19] In [BM97] the randomizing function is denoted with h, but we will use g instead. The randomizing function could be considered the equivalent of the compression function f in the Merkle-Damgård paradigm.

[20] The parentheses may be moved or removed, e.g. $((x + y) + z) = (x + (y + z)) = x + y + z$.

[21] The terms may be re-ordered, e.g. $x + y = x + y$.

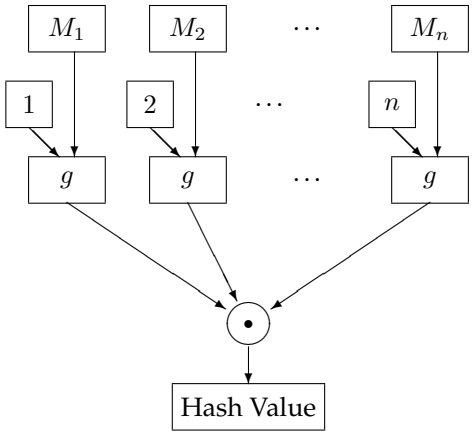


FIGURE 3.4: A hash function of the randomize-then-combine paradigm. The message M consists of n blocks of the same size, $M_1, M_2, \ldots M_n$. The *randomizing function* g combines a message block M_i with its sequence number i. The *combining function* $\odot$ combines the results of all invocations of the randomizing function g into one hash value.

Thus, there is also an identity element[22] 1 and an inverse[23] $^{-1}$ with respect to the combining function $\odot$. The function g takes as input not only a message block, but also the sequence number of the message block. This has to do with the commutative nature of the combining function: without the sequence number it would be trivial to construct collisions for the hash function as a whole. A graphical picture of the information flow in a hash function of this paradigm is shown in Figure 3.4.

There are various options for choosing a combining function, such as multiplication within a suitable group G (MuHASH) or modular addition (AdHASH). Bellare and Micciancio have proven that if the discrete logarithm in G is hard and g is "ideal", then MuHASH is strongly collision-free; and that AdHASH is a UOWHF if the *weighted knapsack problem* (which they define)[24] is hard and g is "ideal" [BM97]. Similar results have been obtained by Impagliazzo and Naor [IN96]. Computationally these combining operations are cheap. The bitwise exclusive or ($\oplus$) is no good candidate for the combining function in the case of a non-keyed hash: it can easily be proven insecure [BM97]. However, within a MAC the bitwise exclusive or *can* be applied [BGR95].

[22] $x \odot 1 = x$

[23] $x \odot x^{-1} = 1$

[24] The *weighted knapsack problem* is a modification of the *subset sum problem*, which is a special case of the *knapsack problem*. These are all problems in combinatorial optimization. The subset sum problem and the knapsack problem are well known NP-complete problems. The weighted knapsack problem is defined in [BM97] and is assumed to be NP-complete.

hash function	bit size	year of publication	reference	status
MD5	128	1992	[Riv92]	obsoleted by [WY05]
RIPEMD-160	160	1996	[DBP96]	
SHA-1	160	1992	[Nat92]	obsoleted by [WYY05]
SHA-224	224	2004	[Nat04]	
SHA-256	256	2002	[Nat02]	
SHA-384	384	2002	[Nat02]	
SHA-512	512	2002	[Nat02]	

TABLE 3.1: Some commonly used cryptographic hash functions and the size of the hash values they produce.

Because of the independency of the computation of g, and the nature of the combining function $\odot$, the computation of a hash value can be done in parallel. There is another advantage of the randomize-then-combine paradigm: it is *incremental*. This roughly means that once a hash value $h = H(x)$ is computed, and the pre-image x is modified into x', the time required to compute $h' = H(x')$ is "proportional" to the "amount of difference" between x and x' [BGG94]. The implications of incrementality will be addressed in Section 3.6.

In both paradigms, there is a strong dependency on the strength of an internal function: the compression function f in case of the Merkle-Damgård paradigm, and the randomizing function g in case of the randomize-then-combine paradigm. Neither paradigm gives specific instructions on how to construct this function, except that it should be strongly collision-free. In practice, these functions are chosen to be complex myriads of bitwise operations such as shifts, rotations, ors, exclusive ors, ands and negations. That such a function f or g constructed in this way is strongly collision-free, is no more than a bold claim. For the two most-used hash functions MD5 and SHA-1, the conjectures that its compression functions are strongly collision-free, have been shown to be overly optimistic [WY05, WYY05].

The fixed size of the hash value influences the strength of the corresponding cryptographic hash function: it is trivial to find collisions for a hash function that produces hash values of, say, only 8 bits in length, and it will be impossible to find collisions on a good cryptographic hash function which creates hash values of 2^{16} bits long. What hash sizes are practical and whether it is computationally feasible to find collisions depends on the state of the art of computing power: if the hash size is chosen too small, collisions are easily detected, and if it is chosen too large, the computational and storage requirements grow too large. For illustrational purposes, Table 3.1 shows the hash sizes for various hash functions, commonly known and used in 2006.

3.5 Common Applications

In this thesis, we will introduce a new application domain of cryptographic hash functions. To be able to see how our application differs from existing applications of cryptographic hash functions, we will describe the spectrum of uses for cryptographic hash functions:

1. password protection
2. manipulation detection
3. optimization of existing cryptographic signature schemes
4. creation of new cryptographic signature schemes
5. random-number generation
6. creation of (symmetric) encryption schemes
7. commitment schemes
8. computational currency and spam protection

In all applications, the description of the hash function is public. We will explain the applications one by one in more detail:

1. Probably the oldest application of cryptographic hash functions is protection of the *password file*. On a computer system with password-protected user accounts, one needs to store passwords in such a way that (1) the passwords cannot be derived from the password file, but (2) the password file should contain enough information to positively identify someone who claims to know a specific user password. The solution is to store the user name and the hash value of the password together in the password file[25].

2. The best known application of cryptographic hash functions is to protect data from being manipulated by a malicious party: a hash value can be used to establish message integrity. Computing hash values and communicating these over the same communication channel as the message itself will however not help anything, since the hash value is subject to the same manipulation powers of the adversary as the original message.

   When a MAC is used, the integrity of the message depends on the secrecy of the key used: it depends on the quality of the key management[26]. When a non-keyed cryptographic hash function is used, the integrity of the message is transferred to the integrity of the hash value: the hash value needs to be protected against manipulation in some way or another. One way to accomplish this is to use a separate authenticated communication channel, another way is to cryptographically sign or encrypt the

[25] This solution is not without problems, though. Badly chosen passwords can be detected by dictionary attacks.

[26] Note that the keys used in MAC are *symmetric*: the sender and the verifier share the same key. The method for message authentication presented by Tsudik in [Tsu92] is roughly equivalent to the use of a MAC.

hash value, thereby reducing the problem of integrity to a problem of key management.

3. Transferring the integrity of a large amount of data to the integrity of a smaller amount of data (as explained in the previous application) can also help to optimize schemes in which large amounts of data must be cryptographically signed. Methods for cryptographic signatures are generally computationally expensive, and computational costs often grow superlinearly in the size of the data to be signed. This makes cryptographically signing a message of one megabyte in size very — if not prohibitively — expensive. Instead, one can compute the hash value of the same message, and sign the hash value [Dam88]. Using this two-step signature scheme, the cost of signing a message is linear in size of the input instead of superlinear. Moreover, this scheme saves storage space, since a cryptographic signature is generally about the same size as the message signed. When using two-step signing, the signature is about the size of the hash value, instead of the size of the original message.

   In communication, cryptographic signatures facilitate integrity, authenticity (the receiver can verify that the sender is who he claims he is) and non-repudiation (the sender cannot deny having sent the message). In software protection, cryptographic signatures facilitate the detection of *malware* and *Trojan horses*.[27] Theoretically, cryptographic hash functions are not required for these applications, but cryptographic hash function help to optimize these applications up to the point that they are actually feasible.

4. It is also possible to build cryptographic signature schemes from cryptographic hash functions alone, but these schemes are merely a proof of concept and they are not very practical because they require a large public storage. There are four known hash-based digital signature schemes: Diffie-Lamport [DH76, page 35], Winternitz [Mer90a, page 227], Merkle [Mer90a] and Rabin [Rab78].

5. Yet another use of a cryptographic hash function is to use it to build a pseudorandom number generator (PRNG) or even a *cryptographically secure* pseudorandom number generator (CSPRNG). In the latter case it is required to keep the pre-image[28] secret.

6. The output of a CSPRNG can in turn be used as a keystream for a one-time pad (OTP)[29], which is an algorithm for symmetric encryption[30]. An algorithm for symmetric encryption can also be built directly from a cryptographic hash function, by making it the F-function in a Feistel cipher

[27] The best known example of such a malware detection software is Tripwire, which runs on UNIX systems.

[28] In the design of PRNGs, the pre-image is also called the *seed*.

[29] The OTP is also called the Vernam cipher.

[30] Of course, a *true* OTP uses a keystream that is truly random, and does not depend on a seed of a fixed size. Therefore, an OTP using a hash function based keystream is not unbreakable, as opposed to a *true* OTP.

[Fei73, FNS75][31].

7. Commitment schemes can also be built using cryptographic hash functions. In a game or protocol, a player can commit to a particular chosen value M without instantaneously disclosing the value by publishing only the hash value $H(M)$. Later on in the game the player must disclose M. Since the player has published $H(M)$, he cannot choose to publish another message M' since $H(M') \neq H(M)$ [DPP94, Dam97, CMR98].

8. A relatively recent application of cryptographic hash functions is to create some kind of *computational currency unit*. The idea is this: the difficulty of finding a collision in a cryptographic hash function depends on the bit length of the hash values it produces. Thus, by trimming the size of a hash value to an appropriate length, it is possible to create puzzles that are moderately hard to solve, but at the same time still tractable (cf. [DN93]). For example, it is feasible to create two pre-images such that the first 20 bits of their hash values are the same[32]. We call this a *partial hash collision*. The interesting feature here is that though it is costly to *find* partial collisions, it is very cheap to *verify* them. By showing two pre-images which result in a partial hash collision, one can 'prove' to have invested a certain amount of computation time. This can be used to combat spam, by accepting only emails for which sufficient computation time has been invested (e.g. Hashcash [Bac02]). It can also be used to construct a transferable currency unit in peer-to-peer (P2P) systems [GH05].

As can be seen from the examples just described, cryptographic hash functions have a lot of applications. This does not mean that every application of a cryptographic hash function is legitimate in the sense that it offers cryptographic guarantees. For example, the hash values used to identify files in P2P filesharing systems do not provide any guarantees on the files exchanged via such a P2P filesharing system. Fortunately, no such guarantees are suggested by P2P systems. That it is easy to illegitimately assume guarantees from the use of a cryptographic hash function is demonstrated by security expert Schneier. His first explanation on cryptographic hash functions reads [Sch96, page 31]:

> "If you want to verify someone has a particular file (that you also have), but you don't want him to send it to you, then you ask him for the hash value. If he sends you the correct hash value, then it is almost certain that he has that file."

[31] In general, when choosing an encryption algorithm, the option of a hash-function based one may not be the best choice in terms of efficiency. Nevertheless, there may be legal reasons to choose for such a function. The United States of America have strict rules for the export of cryptographic software; cryptographic software is considered a type of military arms. Within these export rules, cryptographic hash functions are sort-of neglected. Thus, to legally circumvent these export limitations, it can help to use encryption algorithms based on cryptographic hash functions. In the years just before '9/11', these export rules have been relaxed somewhat, but there are still limitations.

[32] By feasible we mean here that such a collision can be found within approximately one second on workstation hardware current in 2006.

Equivalent claims are made in [BAN89b, AvdHdV01]. The precise claim in [BAN89b] will be the subject of Chapter 5. Unfortunately, the claim is false. The problem is that in the above situation sketch, there is no mention that the file should be kept secret. If there is a third person who is willing to publish the hash value of the file, anybody can 'prove' possession of the file.

However, it *is* possible to use cryptographic hash functions to prove possession of specific information. It is more complex than Schneier assumed. Chapter 8 will address the specifics of this application, which are far from trivial. Chapters 9 and 10 of this thesis will actually show protocols for this application. These protocols require a specific kind of cryptographic hash function: the *non-incremental* hash function. In the next section, we will explain and define what that is.

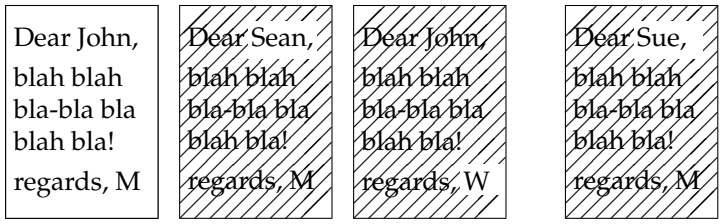
3.6 (Non-) Incrementality

Bellare, Goldreich and Goldwasser have introduced the randomize-then-combine paradigm to show that *incremental* cryptographic hash functions can be built. The concept of incrementality deserves attention, because it shows that certain cryptographic hash functions possess properties we do not desire for our purposes in this thesis. In this section, we will explain what we *do* require. To give some context to our definition, we will first informally explain what incrementality is.

Suppose one wants to send to many different people a signed message that is essentially the same, except for some small parts, such as the addressee field or the date. For every single individual message, the signature has to be recomputed. As the number of similar signed messages grows, the cost of computing the signatures grows as well. Does this mean that signing one million messages is one million times as expensive as signing just one message? In fact, it need not be. Just as buying one million copies of the same book is not one million times more expensive than buying just one copy, it is possible to get a quantity discount on the computational cost of cryptographic signatures.

The terms and conditions of the discount are as follows: the signature should use a two-step signature scheme, and the hash function involved should be *incremental*. The original message must be hashed in the 'old-fashioned' way at least once. To obtain a hash value for every consecutive message, a procedure should be followed, which uses the hash value of the original message, and a description of how the current message differs from the original message. The amount of discount you get on the signature consecutive message is inversely correlated to the size of the *difference description*. Thus, if the body of messages you wish to sign is highly homogeneous, you will get a large discount. If on the other hand the messages are unrelated, your discount will be very small. Similarly, the discount one may get when ordering one million different books will never be as big as when one orders one million copies of the same book. In Figure 3.5 the idea of incremental hash functions is shown.

The practical advantage of incremental hashes and signatures is question-



message number	1	2	3	⋯	∞
individual cost	$\text{size}(M_1)$	$\text{size}(\Delta_2)$	$\text{size}(\Delta_3)$		$\text{size}(\Delta_i)$
incremental average cost	$\text{size}(M_1)$	$\frac{\text{size}(M_1)+\text{size}(\Delta_2)}{2}$	$\frac{\text{size}(M_1)+2\cdot\overline{\text{size}(\Delta)}}{3}$		$\overline{\text{size}(\Delta)}$

FIGURE 3.5: Incremental hash function in action. For the first message, the full message must be consulted. For the next messages, the computation of the hash depends only on the amount of difference of the current message to the first message. As the number of messages grows to infinity, the average cost of hashing a message converges to the average amount of difference. The description of the difference between M_1 and M_i is denoted Δ_i, and $\overline{\text{size}(\Delta)}$ denotes the running average of $\text{size}(\Delta)$. The size of Δ is roughly proportional to the size of the 'not-wiped out' part of messages 2, 3 and onwards.

able. The cost of physically sending a signed message is still proportional to the length of the message. Speeding up the process of creating the signature cannot change that, and will therefore not shorten the whole process of signing and sending a message by more than a constant factor. In fact, the advantages of incremental hashing and signing do only matter if the person generating the hash of the message does not send the message as often as he signs it. In [BGG94, BGG95] some application domains of incremental hashes and signatures are described, including virus protection and signed streaming media.

Incrementality offers some slight advantages, but it also has some clear disadvantages. Consider the following situation:

> Bob wants Alice to prove she possesses M, but Bob doesn't want Alice to send the full message M. Bob knows that the hash value h of M is publicly known, and therefore Bob certainly cannot ask Alice to send just h. Bob decides to create a 'difference description' Δ, which describes how to transform M into M'.
>
> Bob asks Alice to present the hash value h' of M'.

This protocol is faulty: Alice can 'prove' possession of M by exploiting the incrementality of the hash function. That is, Alice can construct $H(M')$ without possessing M or being able to construct M. Thus, for a protocol like this one to be correct, it is necessary that the hash function is not incremental. In that case, the only way to compute $H(M')$ is to construct M' first, and then compute its hash value.

It is not trivial to give a precise formal definition of non-incrementality.[33] Let us give an informal description of the notion of non-incrementality:

Non-Incremental A cryptographic hash function is non-incremental, if it is always necessary to have the full pre-image at hand to compute the hash value of this pre-image.

Hash functions from the randomize-then-combine paradigm are obviously incremental, so that kind of hash functions will not do for our purposes. Hash functions from the Merkle-Damgård paradigm are sort-of incremental, but in a less obvious way. To verify this, see that to compute $H(M_1 \cdot M_2)$, the algorithm computing the hash value passes through a local state where M_1 has been read, but M_2 not yet. This local state can be stored. To compute $H(M_1 \cdot M_2')$, where $M_2 \neq M_2'$, it is sufficient to know the local state and M_2'. This problem can be circumvented by defining $H(M)$ to be equal to $H_{MD}(M \cdot M)$: the pre-image M is 'fed' twice through the Merkle-Damgård hash function $H_{MD}(\cdot)$. As a result of *chaining* (blocks earlier in the sequence influence how the blocks later in the sequence are processed), all blocks of the first iteration of message M, influence the processing of all blocks of the second iteration of M. Therefore, it is not possible to store a local state that allows one to compute $H(M_1 \cdot M_2)$ without knowing M_1.

The solution we propose, to feed the pre-image twice through the hash function is not totally new; Ferguson and Schneier have proposed this solution and similar solutions to fix the length-extension weakness of the Merkle-Damgård paradigm [FS03, pages 90–94]. This is not unexpectedly, as length-extension and incrementality are interrelated properties.[34]

3.7 Conclusion

In this chapter, we have explained what a cryptographic hash function is, and elaborated on some of the difficulties in defining cryptographic hash functions. Moreover, we have shown that incrementality, a property often considered a *feature* of some cryptographic hash functions, can actually be a *drawback*. We have informally but precisely defined non-incrementality, a property that guarantees the specific function does not have this drawback.

In this thesis, we will give a new application of cryptographic hash functions. For this application, we require that it is

1. non-keyed,
2. correlation-free and
3. non-incremental.

We will prove our protocols correct in the random oracle model.

[33] Given the general history of hash function definitions which is sparkled with erroneous definitions, we do not feel safe enough yet to give a formal definition at this point.

[34] It is beyond the scope of this chapter to explain the length extension weakness of the Merkle-Damgård paradigm, consult [FS03, pages 90–94] for a good explanation.

Authentication logics (such as BAN and GNY) offer a way to reason about the knowledge of principals in a security protocol. It is explained how such logics work and what their caveats are.

Chapter 4

Authentication Logics

Formal analysis of security protocols requires one to reason about the knowledge of the participants (principals) in the protocol. Critical for a security protocol is that it should not only guarantee that certain information is communicated, but also that certain other information is *not* communicated. For example, external observers should typically not be able to infer session keys which are exchanged in a security protocol.

BAN logic [BAN89b, BAN89a][1], introduced by Burrows, Abadi and Needham, is an epistemic logic[2] crafted for analyzing security protocols. It models at an abstract level the knowledge of the principals in a protocol. The principals are supposed to have only polynomially many computational resources. It was the first logic of its kind, and has had a tremendous influence on protocol analysis: it has helped revealing weaknesses in known protocols, and many logics are based on it, among others GNY logic. We refer to BAN logic and all the variations of it as *authentication logics*. This is not to say that there has been no criticism of BAN logic. For one thing, a *full* semantics is lacking, and many attempts have been made to fix this problem [AT91, GNY90, KW94, WK96, vO93, SvO94, SvO96, Dek00]. Moreover, the logic fails to detect some very obvious protocol flaws [Nes90].

The general consensus about BAN-descendant logics appears to be that these logics are computationally sound (detected protocol flaws are indeed flaws), but certainly not computationally complete (they may fail to detect certain protocol flaws). Recent work includes attempts to bridge the gap between the formal (i.e., BAN-descendant) approach and the computational approach to security logics [AR02], and attempts to obtain completeness results for BAN-descendant logics in a kind of Kripke-style semantics [CD05a, CD05b]. In the

[1] See Appendix A.1 for a taxonomy of the papers presenting the BAN logic.
[2] For a thorough treatment of epistemic logic, consult [FHMV95, MvdH95].

Multi-Agent Systems world, BAN logic has been widely used (see for example, [AvdHdV01]).

Authentication logics play an important role in this thesis. In Chapter 5, we will prove that BAN logic is not 'sound', due to an inference rule that tries to model cryptographic hash functions. In Chapter 6, we will extend another authentication logic, GNY logic, in such a way that our protocols can be analyzed using GNY logic, which we do in Chapter 9. In the current chapter, we will explain the basics of authentication logics.[3] Where we write formulae, this will be formulae in GNY logic (which is summarized in Appendix B).

4.1 The Goals of an Authentication Logic

The main idea of authentication logics is that we model for each principal, what he knows, and what he can derive. What principals know includes what principals know about the knowledge of other principals: authentication logics are a special kind of *epistemic logic*. The knowledge of principals is modeled by explicitly stating what inference capabilities principals have, what principals know a priori, and what principals observe during a protocol run. The total knowledge of a specific principal is the *logical closure* of all capable inferences over the knowledge he has so far: A principal may know X a priori, infer from this Y, and if Z is inferrable from Y, the agent by definition also knows Z (and so on).

In most authentication logics, the malicious adversary model is used, which means that it is assumed that an adversary can overhear and intercept every message sent, and that an attacker can send any message he can synthesize. Message synthesis of the attacker is however limited to what is cryptographically feasible, i.e., the attacker has only polynomial computing power and cannot perform brute-force analysis to find secret keys [DY83]. (See also Section 2.6.)

Thus, principals are *logically omniscient* in the sense that they may apply some given inference rules unboundedly often. However, principals are not 'mathematically omniscient', i.e., they are not capable of all valid inferences.[4]

Proving that security protocols meet their specification generally involves proving three properties of a protocol:

1. the participating principals learn what they should learn,
2. what the participating principals learn is indeed true, and
3. no principal learns facts he ought not to know.

Thus, a correctness proof requires both a proof of things that are learned, and things that are *not* learned in course of a protocol run. Authentication

[3] For alternative introductions, consult [GSG99] and [SC01].
[4] In particular, principals cannot perform brute-force attacks on cryptographic primitives.

assumptions Alice knows Bob's public key, and Bob knows his own secret key.

the protocol itself Alice chooses a random number and sends it to Bob. Bob signs this number and sends it back to Alice.

claims After Alice receives Bob's message, she knows Bob received and read her message containing the random number.

FIGURE 4.1: The signing parrot protocol, plain description

logics generally focus on the learning part, and less if at all on the not-learning part. If an analysis of a protocol using an authentication logic does not expose a flaw, this means that properties 1 and 2 are not violated, of course assuming that the logic itself is 'correct'. In Chapter 6, we will extend GNY in such a way that property 3 is also addressed.

The specification of a protocol consists of the the following information:

Assumptions A description of the situation before the protocol is executed: who are the players involved (the *principals*), what is the knowledge of the principals, the reasoning powers of the principals, and the knowledge and reasoning powers of those who do not participate in the protocol.

The Protocol Itself A description of the messages that are exchanged in the protocol: how they are constructed, by whom, and when they are sent.

Claims A description of what the protocol supposedly provides: in what way the knowledge of the principals has changed since the beginning of the protocol, and in what way the knowledge of principals, including external observers, has *not* changed since the beginning of the protocol.

For example, consider the protocol shown in Figure 4.1[5]. We will use this protocol in the rest of this chapter in our examples.

4.2 The Taxonomy of Any Authentication Logic

In authentication logics, messages are not modeled as bit strings (as they are in the 'real world'), but as formulae. The principals in a protocol send one another formulae, and privately possess formulae. When a principal possesses a formula this just means he has it stored in his memory. Principals can construct (synthesize) new formulae by applying certain given methods. For example, one such method is symmetric encryption: if a principal possesses a message M and a key K, he can construct the message $\{M\}_K$, the symmetric encryption of M under K.

[5] This trivial protocol does not appear to have a name yet. For easier discussion, we will baptize it the *signing parrot protocol*. This protocol falls in the category often referred to as "Don't try this at home!" Bob's behavior of signing any message he sees, is very unwise.

What happens in the protocol however, *does* have a name. In strand-space terminology, it is called *incomping authentication* [Gut01, Gut02].

Similarly, principals can analyze (deconstruct) messages: if a player possesses $\{M\}_K$ and K, then he can decrypt $\{M\}_K$ and infer M, so he can add M to his 'possessions'. If on the other hand a principal possesses $\{M\}_K$ but not the decryption key K, it is (of course) not possible to infer M. In this way, certain parts of the communication may be hidden from some principals or external observers.

An authentication logic essentially consists of (at least) four parts:

1. A **Formal Language** that describes what *formulae* exist (i.e., what kind of messages can be exchanged). Moreover, the language describes what kinds of assertions can be made. Assertions typically relate formulae to principals. The formal language of GNY logic is summarized in Appendix B.1.

   The expressive power of the language directly influences the class of messages that can be expressed in the logic. For example, if there is no notation accommodated for asymmetric encryption, the logic cannot reason about it. A similar point also holds for statements: if for example the language does not distinguish between possessing a formula and believing the formula, one has to assume a principal believes all formulae he possesses.

2. A **Protocol Idealization Method** that describes how to translate a protocol description into the formal language of the logic. This results in a protocol description in the form of an ordered list of *send statements* $S_1, \cdots, S_n$, each S_i in the form $P \rightarrow Q : X$ (read: 'P sends Q message X') where P and Q are principals ($P \neq Q$) and X is a formula in the formal language of the logic.

   A protocol idealization method typically omits implementation details and tries to focus on the beliefs that are to be conveyed in the message. Protocol idealization is a manual task performed by humans.

3. A **Protocol Parser** which is an algorithm that translates an idealized protocol into an ordered list of *step transitions*. This list is the basis for further analysis of the protocol.

   Most protocol parsers are very trivial, but sometimes the parser does something that could be considered a kind of advanced annotation of the protocol.[6]

4. A **List of Inference Rules** or **Logical Postulates** which defines what a principal can do to construct and analyze formulae. In the following generic presentation of an inference rule with name N, $X_1, X_2, \cdots X_n$, $Y_1, Y_2, \cdots Y_m$ represent statements in the formal language of the logic.

   N $\quad \dfrac{X_1, X_2, \cdots X_n}{Y_1, Y_2, \cdots Y_m}$

[6] Our notion of a protocol parser is a generalization of the protocol parser notion described in [GNY90].

assumptions $A \ni +K, \quad A \models \overset{+K}{\mapsto} B, \quad B \ni -K, \quad A \ni N, \quad A \models \sharp(N)$

the protocol itself
1. $A \rightarrow B\colon N$
2. $B \rightarrow A\colon \{N\}_{-K}$

claims $A \models B \ni N$

FIGURE 4.2: GNY idealization of the signing parrot protocol. Alice and Bob are denoted A and B. Bob's public key is $+K$ and his private key is $-K$. The randomly chosen number is denoted N. For a summary of the formal language of GNY logic which is used here, consult Appendix B

> If $X_1, X_2, \cdots X_n$ all hold, then Y_1 and Y_2 up to Y_m may all be inferred.
>
> The inference rules of GNY logic are summarized in Appendix B.2.
>
> The list of inference rules is typically hand-crafted, small, and often fully given. The set of inference rules should be constructed in such a way that all notions attributed to elements of the language are expressed in the inference rules. For example, if it is possible to formulate a statement essentially saying 'principal P knows X and Y', then there should be inference rules accommodating the inference of 'principal P knows X' and 'principal P knows Y' from *the original sentence*.

To illustrate what the formal language and an idealized protocol look like, the GNY idealization of the signing parrot protocol is shown in Figure 4.2. Appendix B lists the formal language of GNY logic that is used in the figure.

The formal language of authentication logics often has a rather limited expressive power. In particular, explicit negations and disjunctions are oddities (with the notable exception of SVO logic [SvO94]). Implicit negations can however be found easily: most formal languages have a construct denoting that some cryptographic key K is *only* known to two (explicitly named) principals. In BAN and GNY logic, which will be explained later, this is the construct $P \overset{K}{\leftrightarrow} Q$. This of course implies that other principals do *not* know the key K. However, this very same construct also suggests that none of the two named principals would disclose the key K, and it remains the question whether this is realistic. And while disjunctions are not facilitated by the formal language, there are often inference rules which resemble disjunction elimination[7].

Of course, with poor support of disjunctions and negations, these logics can hardly if at all model protocols which have conditional branching: where whether some protocol steps are executed depends on the outcome of previous protocol steps. If one wants to analyze such a protocol using an authentication logic, one has to do this 'outside of the logic', that is: rely on natural language and while doing so, remain precise and concentrated.

[7] For an example of such an inference rule, look at rule **I3** of the GNY logic, shown in Appendix B on page 186. The $*$ sign in the first condition denotes that P did not send the message, and $P \overset{S}{\leftrightarrow} Q$ denotes that only P and Q know S. From this it is inferred that Q sent the message.

$$[A \ni +K, \quad A \models \overset{+K}{\mapsto} B, \quad B \ni -K, \quad A \ni N, \quad A \models \sharp(N)]$$
$$(A \to B : N)$$
$$[B \lhd *N]$$
$$(B \to A : \{N\}_{-K})$$
$$[A \lhd *\{N\}_{-K}, \quad A \lhd \{N\}_{-K}, \quad A \ni \{N\}_{-K},$$
$$A \ni H(N), \quad A \models \phi(N), \quad A \models B \mathrel{|\!\sim} N, \quad A \models B \ni N]$$

FIGURE 4.3: GNY annotation ('correctness proof') of the signing parrot protocol. The assertions in the last postcondition are obtained by application of inference rules **T1**, **P1**, **P4**, **R6**, **I4** and **I6** (in that order). The very last assertion is equal to the claim of the protocol.

4.3 Using an Authentication Logic

When one analyzes a protocol using an authentication logic, one searches for a *legal annotation* of the protocol. A legal annotation is an annotation with some special properties. First, let us explain what an annotation is. An *annotation* of a protocol $S_1, \cdots, S_n$ is roughly something like a transcription in Hoare logic [Hoa69]:

$$[\text{assumptions}]\ S_1\ [\text{assertion } 1]\ \cdots\ [\text{assertion } n-1]\ S_n\ [\text{conclusions}]$$

An assertion is a (comma-separated) list of assertions in the formal language, interpreted as a conjunction. Obviously, the assumptions and the conclusions are a special type of assertion.

The protocol parser provides a list of *step transitions*. A step transition has the form

$$[\text{precondition}]\ (P \to Q : X)\ [\text{postcondition}]$$

For many logics, including BAN and GNY, the precondition is of the form Y, and the postcondition is of the form $Y, Q \lhd X$, which essentially means that whatever was true before the protocol step remains true afterward, and principal Q observes what he is told, namely X. Moreover, the assertion $Q \lhd X$ may be inserted directly after the protocol step. The protocol step itself is the justification for this assumption[8].

Note that the protocol parser does not enforce that the sending party is actually capable of sending the message (i.e., $P \ni X$ is not a formal precondition). In particular, the protocol assumption $B \ni -K$ remains unused in the analysis,

[8] The GNY protocol parser sometimes inserts the not-originated-here sign ($*$) into the inserted assertion. The rules for adding this sign are somewhat complicated, and will for simplicity not be explained in this thesis. These rules can be found in [GNY90, Section 5].

The not-originated-here sign ($*$) rougly means, that the principal who receives a message $*X$, has not previously sent a message X.

1	$A \ni +K$			$(B \rightarrow A\colon \{N\}_{-K})$	
2	$A \models\!\!\stackrel{+K}{\mapsto} B$		7	$A \lhd *\{N\}_{-K}$	[2]
3	$B \ni -K$		8	$A \lhd \{N\}_{-K}$	**T1**(7)
4	$A \ni N$		9	$A \ni \{N\}_{-K}$	**P1**(8)
5	$A \models \sharp(N)$		10	$A \ni H(N)$	**P4**(4)
	$(A \rightarrow B\colon N)$		11	$A \models \phi(N)$	**R6**(10)
6	$B \lhd *N$	[1]	12	$A \models B \mid\!\sim N$	**I4**(8, 1, 2, 11)
			13	$A \models B \ni N$	**I6**(12, 5)

FIGURE 4.4: Heavy GNY annotation of the signing parrot protocol. After each assertion which is not an assumption, a justification is placed. This justification is either the result of a communication step inserted by the protocol parser, denoted with the protocol step between [square brackets], or the name of the inference rule applied, together with the statement numbers of its satisfied preconditions.

while it is neccessary since otherwise B cannot send the message $\{N\}_{-K}$. This issue is addressed further in Section 6.2.2.

To create a *legal annotation* of a protocol is to weld all step transitions of a protocol together such that:

- the precondition of the first step contains only the protocol assumptions;
- the postcondition of the last protocol step contains the protocol claims;
- except for the protocol assumptions and assertions added by the protocol parser, any assertion is derivable (by means of the inference rules) from its prefix[9].

We will use the final requirement, derivability, in a rather strict sense: every statement is obtainable from its prefix by application of at most one inference rule. This does not narrow the class of protocols for which legal annotations can be found, while at the same time it makes it easy to verify whether an annotation is legal. Moreover, we allow repetitions of assertions to be omitted for ease of reading. An example of a legal annotation of a protocol is shown in Figure 4.3.

This type of annotation can be difficult to read and interpret, and therefore in this thesis, we will be a bit more explicit. We will write every statement on an individual line, together with a line number and the applied inference rules for easy reference. An example of such a 'heavy annotation' is shown in Figure 4.4.

It should be noted that this type of logic is monotonic within a single protocol run: all assertions are stable, i.e., once true they remain true (for the time of the protocol run). To prove that a protocol is correct with repsect to a particular claim to give a (constructive) proof of the protocol claim by means of a (heavy) annotation.

[9] i.e., the protocol annotation before ('left of') the statement in question.

4.4 The BAN Logic Debate

Authentication logics provide a very *intuitive* means of analyzing security protocols. Whether the approach is also *accurate* has been subject of a very extended debate. The main criticisms of BAN logic regard the following properties of the logic:

1. the semantics,
2. the notion of belief,
3. the protocol idealization method,
4. the honesty assumption, and
5. the incompleteness.

Appendix A.2 discusses these criticisms in more detail. All existing criticisms of authentication logics might suggest that the approach is rather worthless, but on proper inspection the host of critiques deserves another interpretation: The way of reasoning in authentication logics is highly valuable, and that is why so much effort has been taken to *improve* the original BAN logic. In this thesis, the constructive contribution to authentication logics can be found in Chapter 6.

Our opinion on authentication logics is that the general approach is simply wonderful, while the operationalization of the approach is rather troublesome. In particular, authentication logics *in general* should not be blamed for problems with *early, individual instances* of authentication logics, such as BAN logic. The methodology is simple, intuitive and powerful. Therefore, we feel authentication logics deserve to be one of the basic tools for protocol analysis (next to other methodologies such as strand spaces and the computational approach).

We are not blind to the shortcomings of BAN logic and its descendants. In Chapter 5, we even prove that BAN logic is not 'sound', and Section 6.1 is not particularly praising, either.

4.5 Conclusion

In this chapter, we have briefly explained how an authentication logic works. The role of the formal language, the inference rules, the protocol parser, assertions, annotations and legal annotations have been explained. We have introduced the concept of a *heavy annotation*, which is a legal annotation that is easy to verify.

In the next chapter (Chapter 5), we will prove that BAN logic is not 'sound', and in Chapter 6 we will extend GNY logic in such a way that our protocols can be analyzed in Chapter 9.

For further reading on authentication logics, consult [SC01].

BAN logic, the 'mother of all authentication logics', does not model hash functions in an appropriate way. It is possible to derive false statements from true assumptions, therefore, BAN logic is not 'sound' – even without a semantics. In the (limited and heavily debated) semantics that BAN logic has, this problem also shows up.

Chapter 5

'Unsoundness' of BAN logic

In this chapter we show a problem of BAN logic [BAN89b, BAN89a][1] that has, to our knowledge, not yet been identified, despite all research into formal protocol analysis. The problem is this: BAN logic is not 'sound'. False statements can be obtained by 'valid' inference rules from true assumptions. This behavior is caused by a questionable inference rule. In Section 5.1 we will explain the reasoning mistake behind this questionable inference rule. As a result of the reasoning mistake, the inference rule does not have a *computational justification*, which is discussed in Section 5.2. Section 5.3 shows the protocol we use in our unsoundness proof and Sect. 5.4 shows all inference rules used in our proof. Section 5.5 shows the actual proof. In Sect. 5.6 we will give an alternative proof, but in the questionable semantics of BAN logic; therefore, we regard our proof of Sect. 5.5 more important. We close with some remarks on the relevance of our results.

5.1 Cryptographic Hash Functions and Justified Beliefs

A cryptographic hash function is a function $H\colon \{0,1\}^* \to \{0,1\}^k$ which is computationally feasible to compute, but for which the inverse is computationally infeasible. In particular, computing the inverse of a hash function takes $O(2^k)$ operations. Thus, a cryptographic hash function is *one-way*: it is computationally infeasible to construct a message x such that $H(x)$ yields a given value h [DH76]. For an extensive treatment of cryptographic hash functions, consult Chapter 3.

[1] If you are unfamiliar with BAN logic, you may wish to consult the previous chapter first.

We repeat the most relevant properties of cryptographic hash functions from Chapter 3 here. Cryptographic hash functions have a lot of applications, including password protection, manipulation detection and the optimization of digital signature schemes. Unfortunately however, the class of applications is sometimes overestimated. Consider for example the following quote from security expert Bruce Schneier [Sch96, page 31] (also quoted on page 43):

> "If you want to verify someone has a particular file (that you also have), but you don't want him to send it to you, then you ask him for the hash value. If he sends you the correct hash value, then it is almost certain that he has that file."

Unfortunately, this claim is false. The problem is that in the above situation sketch, there is no mention that the hash value should be kept totally secret. If there is somebody who is willing to publish the hash value of the file, anybody can 'prove' possession of the file.

The authors of BAN logic [BAN89b, BAN89a] made the same reasoning mistake as Bruce Schneier, and incorporated into their logic an inference rule reflecting the abovementioned questionable reasoning[2]. The name of the questionable rule is **H-BAN** and the rule will be shown in Sect. 5.2 on page 58. As a result of this, BAN logic is not 'sound'. Essential in our proof is the fact that belief in BAN logic is considered to be *justified belief*.

But first, let us recapitulate what soundness is. A proof procedure is sound if it proves only valid formulae. In particular, from justified ('true') formulae it should be impossible to infer an unjustifiable ('false') formula. A proof of soundness generally involves a formal system and a class of models (a *semantics*): a proof of soundness essentially shows that every formula that is *derivable* ($\vdash$) in the formal system is *observable* ($\models$) in all relevant models (i.e., $S \vdash X$ implies $s \models X$).

A related concept, 'soundness'[3] ($S \vdash P \mid\!\equiv X$ implies $S \vdash X$) relies on the definition of the modal operator *belief* ($\mid\!\equiv$) in BAN logic which denotes *true justified belief*. As opposed to beliefs in general, which may be ungrounded and false, a true justified belief should be true. To see what the authors of BAN logic consider belief, let us look at the following excerpt from [BAN94, page 7]:

> "More precisely, define knowledge as truth in all states (as in [HM84][4]); our notion of belief is a rudimentary approximation to knowledge, and it is simple to see that if all initial beliefs are knowledge then all final beliefs are knowledge and, in particular, they are true."

In this chapter, we will prove 'unsoundness' in Section 5.5 and unsoundness in Section 5.6. In our 'unsoundness' proof, all initial beliefs are clearly

[2] See Appendix A.1 for a detailed discussion of the papers presenting BAN logic, and which papers exactly contain the reasoning mistake.
[3] Note the quotes, which distinguish 'soundness' from soundness.
[4] This is a reference to a preliminary paper. The final paper is [HM90] — WT.

knowledge, though one of the obtained final beliefs is not knowledge, in particular, it is false. Thus, by inferring an unjustified belief in BAN logic from true assumptions, we prove that BAN logic is not sound. In particular, this means that it is impossible to create a semantics in which BAN logic is sound.

5.2 On the Computational Justification of Beliefs

In the analysis of security protocols, if a principal obtains a new belief, there has to be a computational justification for the newly obtained belief. For example, if a principal sees a message cryptographically signed with private key K^{-1}, it is justified to believe that the message originates from the principal owning private key K^{-1}. The computational justification is in this case that it is computationally infeasible for principals other than the one owning private key K^{-1} to construct a message signed with this key. This type of justification is *essential* if security is of concern.[5]

With this consideration in mind, it is worth noting the following excerpt from page 266 of the BAN paper [BAN89b], (resp. pages 41–42 of [BAN89a]):

> "Obviously, trust in protocols that use hash functions is not always warranted. If H is an arbitrary function, nothing convinces one that when A has uttered $H(m)$ he must have also uttered m. In fact, A may never have seen m. This may happen, for instance, if the author of m gave $H(m)$ to A, who signed it and sent it. This is similar to the way in which a manager signs a document presented by a subordinate without reading the details of the document. However, the manager expects anyone receiving this signed document to behave as though the manager had full knowledge of the contents. Thus, provided the manager is not careless and the hash function is suitable, signing a hash value should be considered the same as signing the entire message."

This quote contains an assumption which is, in our opinion, unreasonable: The manager expects anyone receiving the signed document as though something would be the case *which may not be the case*. Of course, any principal including the manager may be free to desire any behavior from other principals. But is it reasonable to expect beliefs to be obtained which are not computationally justified?

It is reasonable to assume that any principal, upon seeing $\{H(N)\}_{K^{-1}}$ will believe the manager signed $H(N)$, since it is computationally too difficult for any principal *other than the manager* to construct the signature. However, it is not reasonable to assume that any principal, upon believing a manager signed $H(N)$, believes that the manager has seen N, as there is *no computational problem*

[5] Consider the alternative: we do not want principals to believe a message is sent by Santa Claus just because the name 'Santa Claus' is written beneath it; writing the name 'Santa Claus' is an exercise just as easy for Santa Claus himself as it is for anybody else.

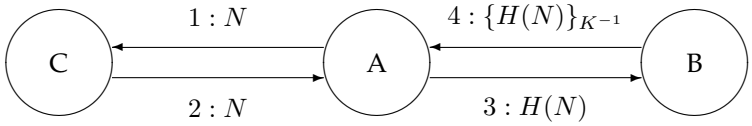


FIGURE 5.1: The two parrots protocol, graphical illustration.

that would justify such a belief. Anybody may have computed $H(N)$ from N, in particular someone may have told the manager $H(N)$ but not N. Therefore, the expectation of a manager that other principals should act as if the manager knows N, is not warranted.

In fact, the text quoted above is the justification of the inference rule **H-BAN** in BAN logic. We believe the identified problematic assumption explains the problems that arise from the inference rule **H-BAN**.

The **H-BAN** *hashing* inference rule reads, as given on page 266 of [BAN89b] (resp. page 42 of [BAN89a])[6]:

H-BAN $$\frac{P \models Q \mid\sim H(X), \quad P \lhd X}{P \models Q \mid\sim X}$$

This rule is problematic, as it essentially infers belief (by P) of "possession" (by Q) of the message X from P believing that Q once conveyed $H(X)$. This rule leads to the 'unsoundness' of BAN logic. Fortunately, none of the authentication logics that descend from BAN logic, adopts the **H-BAN** inference rule.

Because the most commonly used signature schemes use cryptographic hash functions, the **H-BAN** inference rule was added to BAN logic to facilitate the analysis of such signature schemes.

With this inference rule at hand, we can see how the two parrots protocol demonstrates the 'unsoundness' of BAN logic.

5.3 The Two Parrots Protocol

To prove the 'unsoundness' of BAN logic, we rely on a protocol. The rather simple *two parrots protocol*[7], shown in shown in Figures 5.1 and 5.2, will demonstrate the 'unsoundness'. Alice (denoted A) chooses a random number N, sends it to Cecil (denoted C), who returns the number. Then Alice sends the cryptographic hash of the number to Bob (denoted B), and Bob signs this hash value and returns it to Alice. As Bob only sees the cryptographic hash value of N, and a cryptographic hash function is one-way, Bob does not learn N itself. Of course, Cecil might privately disclose N to Bob, but this does not happen

[6] If one would like to add inference rule **H-BAN** to GNY, albeit just for demonstration purposes, one could use exactly the same notation, as the formal languages of BAN logic and GNY logic coincide for the constructs used in this particular inference rule.

[7] The *two parrots protocol* is a variation on the *signing parrot protocol*, which was presented in Chapter 4, Table 4.1.

in the two parrots protocol. Thus, though by private channels Bob might learn N, the protocol certainly does not guarantee this.

Alice cannot, as a result of the protocol, conclude that Bob knows N. Neither can Alice conclude that Bob *does not* know N. However, according to the analysis of the two parrots protocol in BAN logic, Alice will believe that Bob knows N.

In the two parrots protocol, the message N is transmitted without protection. Thus, one can argue that Bob could learn N by mere eavesdropping. For the sake of simplicity, we use a *very simple protocol* that suffices to demonstrate our observation on BAN logic. Of course, protection of N can be achieved by encryption of the messages between Alice and Cecil. Our proof can be easily extended to obtain the same result for such an altered protocol. Moreover, our proof does not rely on Bob eavesdropping.

Thus, though Bob could learn N through either an assistant (Cecil disclosing N to Bob) or through eavesdropping, the communication in the two parrots protocol simply does not warrant Bob knowing N, and therefore also does not warrant Alice believing that Bob knows N.

When we want to formally analyze the protocol in BAN logic, we need to *transcribe* it into BAN logic. A summary of the protocol transcription is given in Figure 5.3. For illustrative purposes, we will also give the transcription into GNY logic in Figure 5.4.[8] First, we have the protocol assumptions which state that A knows the public key K of B, A knows N, and A believes N to be *fresh*. A newly generated random number is particularly fresh.

The protocol description itself is rather straightforward. To quickly see how the two parrots protocol interacts with the **H-BAN** inference rule, observe that message 2 ($C \rightarrow A\colon N$) can be used to obtain the second precondition of **H-BAN**, and that message 4 ($B \rightarrow A\colon \{H(N)\}_{K^{-1}}$) can be used to obtain the first precondition of **H-BAN**. Thus, messages 2 and 4 are the essential messages of the protocol. The other messages can be considered mere 'glue'.

What is achieved by a protocol can be stated in *claims*. For the two parrots protocol, the following claim is true:

$$\text{It will not be the case that } B \models N$$

which essentially states that B will not know N. Note that this is true because

1. B only sees $H(N)$,
2. the inverse of $H(\cdot)$ is hard to compute ($H(\cdot)$ is a one-way function), and
3. B has only polynomially many computational resources.

[8] Note how idealization in BAN logic (Figure 5.3) differs from the idealization in GNY logic as given in Figure 5.4. The assumptions $A \ni +K$ ("A knows the public key $+K$") and $B \ni -K$ ("B knows his own private key $-K$") are omitted, as within BAN logic this is implied by $A \models \overset{+K}{\mapsto} B$. In fact BAN logic does not even explicitly name public and private keys individually. Therefore, the signed message in protocol step 2, has the form $\{H(N)\}_{K^{-1}}$ (BAN logic) instead of $\{H(N)\}_{-K}$ (GNY logic). Moreover, as BAN does not distinguish between possession ($\ni$) and belief ($\models$), the assumptions and claims are rewritten accordingly. ($A \ni N$ in GNY logic is $A \models N$ in BAN logic; $A \models B \ni N$ in GNY logic is $A \models B \models N$ in BAN logic.)

assumptions Alice knows Bob's public key, and Bob knows his own secret key.

the protocol itself Alice chooses a random number and sends it to Cecil. Cecil sends this very same number back to Alice. Alice computes the (cryptographic) hash value of this number and send the hash value to Bob. Bob signs the hash value and sends it back to Alice.

claims Alice knows that knows Bob received her message containing the hash value of the random number. Bob does not know the random number itself by means of the protocol, though Bob might learn it by other means (e.g. Cecil tells Bob the number in private). Alice has no stance on whether Bob knows the random number.

FIGURE 5.2: The two parrots protocol, plain description

---

assumptions $A \models \overset{K}{\mapsto} B, \quad A \models N, \quad A \models \sharp(N)$

the protocol itself
1. $A \to C\colon N$
2. $C \to A\colon N$
3. $A \to B\colon H(N)$
4. $B \to A\colon \{H(N)\}_{K^{-1}}$

claims It will not be the case that $B \models N$.

problem $A \models B \models N$ can be inferred.

FIGURE 5.3: BAN idealization of the two parrots protocol. In general, the message X cryptographically signed with the private key corresponding to public key K is denoted as $\{X\}_{K^{-1}}$. Thus, any agent that knows K can verify the signature and read X. The assumptions are the *true premises* that lead to the *false belief* which is shown under 'problem'.

---

assumptions $A \ni +K, \quad A \models \overset{+K}{\mapsto} B, \quad B \ni -K, \quad A \ni N, \quad A \models \sharp(N)$

the protocol itself
1. $S_1\ A \to C\colon N$
2. $S_2\ C \to A\colon N$
3. $S_3\ A \to B\colon H(N)$
4. $S_4\ B \to A\colon \{H(N)\}_{-K}$

claims $A \models B \ni H(N)$ and it will not be the case that $B \ni N$.

remark If rule **H-BAN** would be added to GNY logic, $A \models B \ni N$ would be inferrable, which is undesirable.

FIGURE 5.4: GNY idealization of the two parrots protocol. To verify that, if **H-BAN** is adopted, a legal annotation of this protocol exists where $A \models B \ni N$ is inferred, see that the protocol is highly similar to the signing parrot protocol shown in Figure 4.2, and a legal annotation can be derived by adaptation of Figure 4.4.

The problem that we identify in BAN logic (see Sect. 5.5) has the effect that due to inference rule **H-BAN** the following statement can also be inferred in BAN logic:

$$A \models B \models N$$

which states that A will believe that B will know N. This belief of A is not computationally justified (see Sect. 5.2).

5.4 Used Inference Rules

The proof of 'unsoundness' in Sect. 5.5 involves three inference rules of BAN logic[9]. Inference rule **H-BAN** has already been given on page 58, the other two rules are:

1. the *message meaning* inference rule number *ii* as given on page 238 of [BAN89b] (resp. page 6 of [BAN89a])[10]:

   MM $$\frac{P \models \overset{K}{\mapsto} Q, \quad P \lhd \{X\}_{K^{-1}}}{P \models Q \mid\sim X}$$

   This rule formalizes that if P knows Q's public key, and P receives a message X signed with Q's private key, P may infer that Q once sent X.[11]

2. the *nonce-verification* inference rule as given on page 238 of [BAN89b] (resp. page 6 of [BAN89a])[12]:

   NV $$\frac{P \models \sharp(X), \quad P \models Q \mid\sim X}{P \models Q \models X}$$

   This rule formalizes that if P believes X to be *fresh* (it originates in the current session), and P believes Q once conveyed X, then P may infer that Q believes X (in the current session).[13]

5.5 Proof of 'Unsoundness' of BAN logic

In this section, we will present our formal proof. In our proof, we use the term 'false belief'. This might be perceived as unnecessarily harsh or misleading, but

[9] These names of these inference rules have been given by the writer of this text.
[10] The GNY equivalent of this inference rule is **I4**.
[11] Inference rule **MM** has been questioned by Wedel and Kessler, as it is invalid if interpreted according to their semantics [WK96]. However, they point out that it is unclear whether BAN logic itself or their semantics of BAN logic is to blame for that.
[12] The GNY equivalent of this inference rule is **I6**.
[13] This rule relies on the assumption that only beliefs are communicated.

we will argue that this is the right formulation, even in lack of a clear semantics of BAN logic as a whole. The central construct of BAN logic, $\models$, is defined as follows on page 236 of [BAN89b] (resp. page 4 of [BAN89a]):

> "$P \models X$: *P believes X*, or P would be entitled to believe X. In particular, the principal P may act as though X is true. This construct is central to the logic."

In our proof, we obtain a result of the form $P \models X$, where X is *not warranted*. It *might* be the case that X were true, if some more communication were to occur than considered in our proof. Therefore, and in this way, we deem "false belief" the appropriate term for such an X. With this explanation given, let us formulate our main theorem:

Theorem 5.1 ('Unsoundness' of BAN logic)**.** *Within BAN logic (as defined in [BAN89b, BAN89a]) it is possible to derive unjustifiable beliefs. More precisely, a statement of the form $A \models X$ can be derived while the statement X itself cannot be derived.*

Proof (derivability). Consider the two parrots protocol, whose BAN idealization is given in Sect. 5.3. It is trivial to verify that A, C and B are capable of sending the messages they ought to send in the two parrots protocol.

As a result of protocol step 2 (S_2), the following statement is inserted:

$$A \lhd N \tag{5.1}$$

As a result of protocol step 4 (S_4), the following statement is inserted:

$$A \lhd \{H(N)\}_{K^{-1}} \tag{5.2}$$

Using inference rule **MM**, assumption $A \models \overset{K}{\mapsto} B$ and (5.2), we can infer:

$$A \models B \mathrel{|\!\sim} H(N) \tag{5.3}$$

Using inference rule **H-BAN**, (5.3) and (5.1), we can infer:

$$A \models B \mathrel{|\!\sim} N \tag{5.4}$$

Using inference rule **NV**, assumption $A \models \sharp(N)$ and (5.4), we can infer:

$$A \models B \models N \tag{5.5}$$

This inference is also depicted in Figure 5.5 as a *heavy annotation* of the protocol. Statement (5.5) should definitely not be derivable from the two parrots protocol. With all protocol assumptions satisfied and only valid inferences applied, an unjustifiable belief is established. More precisely, A believes $B \models N$, while it also consistent to assume that B does not know N, and nobody tells B about N. Therefore, $A \models B \models N$ is unjustified. □

1	$A \models\!\overset{K}{\mapsto} B$			$(A \to B\colon H(N))$	
2	$A \models N$		6	$B \lhd H(N)$	[3]
3	$A \models \sharp(N)$			$(B \to A\colon \{H(N)\}_{K^{-1}})$	
	$(A \to C\colon N)$		7	$A \lhd \{H(N)\}_{K^{-1}}$	[4]
4	$C \lhd N$	[1]	8	$A \models B \mid\!\sim H(N)$	**MM**(1, 7)
	$(C \to A\colon N)$		9	$A \models B \mid\!\sim N$	**H-BAN**(8, 5)
5	$A \lhd N$	[2]	10	$A \models B \models N$	**NV**(3, 9)

FIGURE 5.5: Heavy BAN annotation of the two parrots protocol. This annotation actually shows that BAN logic is not 'sound', because statement 10 should not be derivable, as it is false.

The culprit is the inference rule **H-BAN**. This problem cannot be fixed by adding inference rules in such a way that $B \models N$ can be inferred, as this would thwart the definition of a cryptographic hash function: then N would be derivable from $H(N)$. Such a 'fix' would increase the number of computationally unjustified inference rules from (at least) one to two.

Note that one more inference step is needed after application of the **H-BAN** rule before a false belief is established. This is because we need to obtain *belief of belief*, which cannot be directly inferred from **H-BAN**.[14]

5.6 The Semantic Approach

In the original BAN papers [BAN89b, BAN89a], a rather limited semantics is given for a part of the formal language of BAN logic. This semantics has been subject to an enormous amount of criticism. For one thing, the semantics is *very* closely tied to the formal language of BAN logic: what is derivable in the logic is by definition observable in the semantics. Arguably, the semantics is *so* closely tied to the formal language that it is of no additional value. Except for it being the subject of criticism, the semantics has hardly ever been used.

In Sect. 5.5 we have explained why we used the formulation 'false belief' in a proof that does not rely on any formal semantics. Therefore, we have consistently used quotes around the term *unsoundness*. In this section we will provide a proof based on a semantics: therefore, we may omit the quotes around unsoundness. However, for this proof we need to disregard all criticisms of the semantics of BAN logic. Therefore, we regard our proof in the previous section as more important. But it is of course up to the reader to choose what he likes best:

1. to agree with our use of 'unjustified belief' in the previous section, and with it agree with the semantics-free proof of 'unsoundness' (shown in the previous section), or

[14] Note that in BAN logic, the semantics of *belief* ($\models$) is defined, while the semantics of *once said* ($\mid\!\sim$) is still "largely a mystery" (literal quote from [BAN89b, BAN89a, BAN88]).

2. to accept the semantics of BAN logic, regardless of all its shortcomings, and with it agree to our proof of unsoundness (shown in this section).

Before we show a run of the two parrots protocol in the semantics of BAN logic, it is appropriate to summarize this semantics:

- A *local state* of a principal P is a tuple $(\mathscr{M}_P, \mathscr{B}_P)$, where $\mathscr{M}_P$ is the set of messages seen ($\lhd$) by P, and $\mathscr{B}_P$ is the set of beliefs ($\models$) of P. These sets enjoy closure properties which correspond to the inference rules of the logic. For compactness and ease of reading, we have only included elements in these sets which are relevant for our purposes.
- A *global state* s is a tuple containing the local states of all principals. If s is a global state, then s_P is the local state of P in s and $\mathscr{M}_P(s)$ and $\mathscr{B}_P(s)$ are the corresponding sets of seen messages and beliefs. In our case the principals are A, B and C, and a global state s is the triple (s_A, s_B, s_C).
- A *run* is a finite sequence of of global states $s_0, \ldots, s_n$.
- A *protocol run* of a protocol of n steps of the form $(P_i \rightarrow Q_i : X_i)$ is a run of length $n+1$, where s_0 corresponds to the protocol assumptions and where $X_i \ni \mathscr{M}_{Q_i}(s_i)$ for all i such that $0 < i \leq n$.

To be able to show a run of the two parrots protocol which is convenient to read, we will first name and give all local states. Then, we will give the full protocol run in which the names of these local states are used. For naming the local states, we adhere to the following convention: $s_P^{n,\cdots,n'}$ is the local state of principal P in the global states $n, \cdots, n'$.

The local states of principals A, B and C are as follows:

$$
\begin{array}{lll}
 & \mathscr{M}_A & \mathscr{B}_A \\
s_A^{0,1} = (& \emptyset, & \{\overset{K}{\mapsto} B, N, \sharp(N)\} \quad) \\
s_A^{2,3} = (& \{N\}, & \{\overset{K}{\mapsto} B, N, \sharp(N)\} \quad) \\
s_A^{4} = (& \{N, \{H(N)\}_{K^{-1}}\}, & \{\overset{K}{\mapsto} B, N, \sharp(N), \\
 & & B \mid\sim H(N), B \mid\sim N, B \models N\} \quad) \\
 & \mathscr{M}_B & \mathscr{B}_B \\
s_B^{0,1,2} = (& \emptyset, & \emptyset \quad) \\
s_B^{3,4} = (& \{H(N)\} & \{H(N), \{H(N)\}_{K^{-1}}\} \quad) \\
 & \mathscr{M}_C & \mathscr{B}_C \\
s_C^{0} = (& \emptyset, & \emptyset \quad) \\
s_C^{1,2,3,4} = (& \{N\} & \{N\} \quad)
\end{array}
\tag{5.6}
$$

The following is a *run* of the two parrots protocol:

$$s_0, s_1, s_2, s_3, s_4 \tag{5.7}$$

where s_i are the global states after the consecutive steps of the protocol:

$$\begin{array}{lccc} & s_A & s_B & s_C \\ s_0 = (& s_A^{0,1} & s_B^{0,1,2} & s_C^{0} \quad) \\ s_1 = (& s_A^{0,1} & s_B^{0,1,2} & s_C^{1,2,3,4} \quad) \\ s_2 = (& s_A^{2,3} & s_B^{0,1,2} & s_C^{1,2,3,4} \quad) \\ s_3 = (& s_A^{2,3} & s_B^{3,4} & s_C^{1,2,3,4} \quad) \\ s_4 = (& s_A^{4} & s_B^{3,4} & s_C^{1,2,3,4} \quad) \end{array} \tag{5.8}$$

Now that we have specified a protocol run of the two parrots protocol, we can give our alternative proof of unsoundness:

Proof (observability). As shown in statement (5.5) of the *derivability* proof in Section 5.5, we can derive in BAN logic the sentence $A \models B \models N$ in *every* run S_1, S_2, S_3, S_4 of the two parrots protocol. Thus, we have:

$$S_1, S_2, S_3, S_4 \vdash A \models B \models N \tag{5.9}$$

Global state s_4 corresponds to the semantics after *a particular* protocol run S_1, S_2, S_3, S_4 of the two parrots protocol. When we take the model as given in equations (5.6)–(5.8), we can observe that 'A believes B knows N': $B \models N \in \mathscr{B}_A(s_4)$, which gives us:

$$s_4 \models A \models B \models N \tag{5.10}$$

On the other hand, we can also observe in our model that 'B does not know N': $N \notin \mathscr{B}_B(s_4)$, which gives us:

$$s_4 \not\models B \models N \tag{5.11}$$

Thus, the belief of A as given in (5.10) is not true in *a particular* protocol run as shown in (5.11). The false belief of A as given in (5.10), is nevertheless derivable (5.9) in *every* protocol run. Thus, it is possible to derive a false belief within BAN logic. □

Let us quote one last excerpt from Section 13, on page 269 of [BAN89b] (resp. pages 47–48 of [BAN89a]):

> "Clearly, some beliefs are false. This seems essential to a satisfactory semantics. [...] Most beliefs happen to be true in practice, but the semantics does not account for this coincidence. To guarantee that all beliefs are true we would need to guarantee that all initial beliefs are true."

The existence of false beliefs in the semantics as such is not a problem, the problem is that some false beliefs are derivable from true ones.

5.7 Conclusion

The formal approach to protocol analysis essentially started with BAN logic. Many critiques of BAN logic have appeared, mentioning its incompleteness (i.e., inability to detect some obvious problems, cf. [Nes90]) and its poor semantics (among many others, see [AT91]). Nevertheless, these critiques have not been a reason to abandon the *way of thinking* introduced by BAN logic [HPvdM03]. The many augmentations to BAN logic (most notably, AT [AT91], GNY [GNY90], AUTLOG [KW94, WK96], VO [vO93], SVO [SvO94, SvO96] and SVD [Dek00]) show the trust in the formal approach which originates from BAN logic. In our opinion, this consensual trust in the *way of thinking* introduced by BAN logic is justified. While obtaining completeness has long been regarded as impossible, the soundness of BAN logic itself has never been seriously doubted. Wedel and Kessler identified rules in BAN, AT and GNY which are invalid in their semantics, but they point out that it is unclear whether the inference rules or their semantics are to blame for that [WK96]. Various more recent results [AR02, CD05a, CD05b, Syv00] provide directions on how completeness could be obtained for formal protocol analysis.

Our unsoundness result does not at all invalidate the formal approach to protocol analysis. It should merely count as a warning to those who wish to *complete* their logic. All augmentations of BAN logic are incomplete in the sense that they do not accommodate all cryptographic primitives known to date. These logics are essentially 'just big enough' to capture the problems the authors intend to capture. And to be fair, this has been difficult enough already. Just a few BAN-descendant logics accommodate cryptographic hash functions, none of them accommodate fancy primitives like (to name just an example) oblivious transfer.

The fact that none of the hash-accommodating BAN-descendant logics adopts the **H-BAN** inference rule, can probably be explained by the observation that constructing a good logic is already so difficult that none of the authors will have felt the urge to include an inference rule into their logic that was not needed to capture the problem the author intended to capture. Nevertheless, it is remarkable that we are apparently the first to find this result on a paper which has been so extensively studied and is 17 years old.

So far, we know of only one publication which relies on the faulty **H-BAN** inference rule [AvdHdV01]. In this publication, the SET protocol[15] is analyzed in BAN logic. It remains open whether the authors' assessment of SET holds in a BAN logic with the inference rule **H-BAN** omitted.

[15] SET stands for *Secure Electronic Transactions* [MV97a, MV97b, MV97c]. The protocol was introduced by VISA and Mastercard for online payments, but it has never been widely adopted or deployed.

GNY logic is not good enough yet. We analyze some general problems of authentication logics. We introduce completeness assumptions, add an inference rule, enhance the protocol parser and introduce maximum belief sets.

Chapter 6

Extending GNY Logic

GNY logic is not powerful enough to analyze the protocols we want to analyze in Chapter 9 of this thesis. Therefore, we will extend GNY logic. Before extending GNY logic, we need to address a few caveats of authentication logics in general.

6.1 Why Authentication Logics Are So Tricky

Arguably the most difficult part of constructing an authentication logic is crafting the list of inference rules. The inference rules should precisely express 'all relevant' properties of cryptographic primitives such as cryptographic hash functions, symmetric encryption and asymmetric encryption. The list of inference rules often has omissions, and individual inference rules can have unstated assumptions, and sometimes even downright fatal flaws. The previous chapter (Chapter 5) elaborated on such a flaw, found in BAN logic. Unstated assumptions and omitted inference rules are sometimes two sides of the same coin, but not necessarily so. An unstated assumption may still be a legitimate assumption. An erroneously omitted inference rule is a Bad Thing: it means that the logic does not detect a flaw which could have been detected if the rule were not omitted.

6.1.1 Unstated Assumptions: Length-Concealment and Non-Incrementality

We will give two examples of unstated assumptions with regard to inference rules. We sketch the problem and offer directions of how to solve the problems that result from the unstated assumptions.

The first example has to do with the constructs used to formalize encrypted messages. Consider the following situation in a fictive university department:

> In the department, all communication is done by placing sealed envelopes on the table in the coffee corner. On these envelopes the names of the sender and the intended recipient are written, and everybody obeys the rule to only open an envelope if he are the intended recipient. Now everybody in the department knows that Alice is writing her long-awaited PhD thesis, which presumably contains shocking results. Everybody is dying to know whether Alice has submitted her manuscript to her supervisor, Bob. As long as the only envelopes from Alice to Bob on the table in the coffee corner are *a few* thin, flimsy ones, everybody will be sure that Alice has not yet submitted her manuscript. This will however change as soon as a one-inch thick envelope from Alice to Bob appears on the table.

Now, reread the description, but read 'encrypted message' where it says 'sealed envelope'. Obviously, if one does not have the right decryption key to an encrypted message, one cannot infer the contents of the message (i.e., look inside the envelope). However, in general it *is* possible to infer the *length* of an encrypted message from the encrypted message without knowing the decryption key (i.e., one can look at the size and measure the weight of a sealed envelope). In fact, length-concealing encryption schemes are a rarity indeed. Within almost every authentication logic however, a length-concealing encryption scheme is assumed. We have not found any paper presenting an authentication logic in which this assumption is explicitly stated. There is only one way to infer that this assumption is actually made: there is no inference rule of roughly the following form[1]:

P-LEN $\dfrac{P \lhd \{X\}_K,\ \mathrm{length}(X, l)}{P \lhd \mathrm{length}(X, l)}$ If a principal is told an encryption $\{X\}_K$ of formula X, and the formula X has length l, then he is considered to have also been told the length l of the formula X.

Length-concealing encryption schemes are, from an information theory perspective, strictly stronger than length-revealing encryption schemes. Therefore, if one analyzes a protocol in an authentication logic, and models a length-revealing encryption scheme as length-concealing, one makes an unjustified assumption. Care should be taken that such an unjustified assumption does not invalidate the correctness proof that is obtained from the authentication logic. An example of a protocol that can be wrongly proven secure using such an assumption is derivable from the university situation described above.

[1] This inference rule follows the notation of GNY logic, see Appendix B. Likewise, the recognizability ($\phi(\cdot)$) concept in GNY logic could be extended to facilitate recognition of the length of formulae.

For our second example on unstated assumptions, we return again to cryptographic hash functions (cf. Chapter 3). Inference rule **P4** of GNY logic[2] states that whoever possesses X, can obtain in a computationally feasible way $H(X)$. There is no inference rule of the following form:

H-INC $\dfrac{P \ni H(X,Y),\ P \ni (Y,Z)}{P \ni H(X,Z)}$

If a principal possesses the hash value of the concatenation of X and Y, and possesses Y and Z, then he is capable of possessing the hash value of the concatenation of X and Z.

Note that in **H-INC**, P may be ignorant of the actual contents of X.

If the cryptographic hash function denoted by $H(\cdot)$ is *incremental* as described in Section 3.6, an inference rule like **H-INC** should be added to the logic. Otherwise, the logic would systematically underestimate the inference capabilities of the principals, which is undesirable. In fact, if this inference rule would be added, the protocols described in Chapters 9 and 10 of this thesis would be rendered worthless. In Chapters 9 and 10, we assume that the used cryptographic hash function is indeed non-incremental.

Thus, one has to be careful and be explicit about whether a hash function, modeled in an authentication logic, is considered to be incremental. It should be noted that the authors of most authentication logics are not to blame for this omitted assumption, as the concept of incremental cryptographic hash functions has emerged years after most authentication logics were conceived.

6.1.2 Omitted Inference Rules: The Key to Incompleteness

It goes almost without saying that it is extremely difficult to create an authentication logic from scratch that captures all possible cryptographic primitives that might be used in a security protocol. To take an example: oblivious transfer is not facilitated by any known authentication logic to date. While it is wise to start small and keep authentication logics as simple as possible, incomplete coverage of cryptographic primitives can have implications for the results obtained by application of an authentication logic. The coverage of cryptographic primitives by an authentication logic is determined by two aspects:

1. *The formal language*
   It must be possible to denote in the formal language of the logic that a certain cryptographic primitive has been applied to some message.

2. *The inference rules*
   The 'essence' of a cryptographic primitive lies in what knowledge or possessions are required to perform specific operations. To correctly reflect a primitive, the inference rules should precisely reflect what can and what can not be done with the use of a certain primitive.

[2] See Appendix B, page 185.

Thus, incomplete coverage of primitives may be due to two types of omissions. Either the the formal language is too restricted, or the list of inference rules does not correctly reflect the workings of the primitives facilitated in the language. The former type omission is not a large problem: if a primitive is omitted in the language, a protocol using the primitive cannot be modeled using the authentication logic, and therefore cannot be falsely 'proven correct'. The latter type of omission poses a serious problem. In fact, it is this type of omission that is partially to blame for the incompleteness of authentication logics.

Omitted inference rules are no rarities in the field of authentication logics. In fact, it is common to add inference rules when needed to prove a protocol correct, and to ignore the inference rules not needed in the particular protocol proof.

To illustrate how an omitted inference rule makes an authentication logic incomplete, consider the following type of protocol. Let us assume we have some protocol which uses asymmetric encryption, but requires the public keys to be kept secret within a certain group of principals. (Actually, we do not know whether such a protocol exists in practice, but there is no reason to render such a protocol unviable.) Within this protocol, at a certain moment, a message $\{X\}_{-K}$ is sent. This is the message X, cryptographically signed with private key $-K$. The message X should remain secret within the group of principals knowing the public key $+K$.

Does this protocol have a major problem? In fact, *it does*! If the signature scheme is like almost any signature scheme used in common practice, it is possible to derive X from $\{X\}_{-K}$ without knowing or possessing the public key $+K$.[3] Thus, for the essentials of such a signature scheme to be reflected properly, an inference rule of the following form is required:[4]

T6′ $\dfrac{P \lhd \{X\}_{-K}}{P \lhd X}$ If a principal is told a formula encrypted with a private key (i.e., a signed formula) then he is considered to have also been told the contents of that formula.

Without an inference rule like **T6′**, an authentication logic fails to find the huge gap in the above-mentioned protocol. If the protocol description were to be adjusted such that it explicitly states that a signature scheme is used that does not leak the message, this should be reflected in an assumption about the inference rules. The assumption should be something like this:

> There is no set of inference rules which allows a principal to derive X from $\{X\}_{-K}$.

This assumption is essentially a *completeness assumption*: it states that the list of inference rules is complete with respect to some essential property of a specific cryptographic primitive. In Chapter 9 we will use completeness assumptions to prove certain principals cannot infer specific information.

[3] This means that is is possible to read a message even if one does not recognize the signature.
[4] This rule **T6′** is of course a strengthened version of rule **T6**.

6.2 Proofs of Knowledge and Ignorance

Proving that security protocols meet their specification generally involves proving three properties of a protocol:

1. the participating principals learn what they should learn,
2. what the participating principals learn is indeed true, and
3. no principal learns facts he ought not to know.

Observe that properties 1 and 2 address mainly *liveness*, and that property 3 addresses *safety*.

Thus, a correctness proof requires both a proof of things that are learned, and things that are *not* learned in course of a protocol run. Authentication logics generally focus on the learning part, and less if at all on the not-learning part. If an analysis of a protocol using an authentication logic does not expose a flaw, this means that properties 1 and 2 are not violated, of course assuming that the logic itself is 'correct'.

If one wants to prove property 3, that principals can *not* infer specific facts, one has to model the limitations of the inference capabilities of the agents, and show that the limitations effectively obstruct principals from inferring certain relevant facts (Cf. [ABV01]). To model the inference limitations of principals, we need to model what inference rules are available to an agent. This is where a nasty property of the authentication logics comes in: none of the authors of these logics claim that the list of inference rules provided in the logic is indeed complete in the sense that no more inference rules can be added. We have to make completeness assumptions (as in Section 6.1.2) to be able to prove property 3 of a protocol. We do not believe nor claim that completeness assumptions are sufficient for proving property 3 of a protocol. This issue will be discussed in Section 6.2.2.[5]

Typically, the (in)ability to draw specific conclusions plays a crucial role in a security protocol: for example showing a message which can only be constructed with knowledge of the specific conclusion constitutes a proof of identity. This is best illustrated with cryptographic signatures. If some principal V sees $\{X\}_{-K}$, and knows $-K$ is the private key of P, then V may believe P once conveyed X. Why is this the case? Essentially, because no principal but P possesses $-K$. This begs the question: is there anything in the authentication logic which prevents a malicious principal to create $\{X\}_{-K}$ out of thin air? The answer is simple: no, there is nothing which prevents a principal to do this *within the authentication logic*. Of course, in an actual 'real world' protocol run, it is impossible to do this. So, the obstruction that one cannot construct messages which are computationally hard to construct, should be incorporated into the logic.[6]

[5] Thus, completeness assumptions are *necessary*, but not necessarily *sufficient*.

[6] Though this type of reasoning is not new in the domain of authentication logics, it has never been incorporated into an authentication logic.

While lacking a means to prove property 3, the meaning of a correctness proof in an authentication logic is only limited. If it exposes a flaw in a protocol, the protocol will indeed be flawed. However, not finding any errors does not guarantee that the protocol is correct. Therefore, proving a protocol correct using an authentication logic, only proves that the protocol has passed a first test of some not-so-obvious flaws. However, we deem such a proof an important step in defending correctness of protocols.

In the light of these considerations, we need to extend the authentication logic we use (GNY) in such a way that

- what *should* be learned can in fact be learned[7], and
- what *should not* be learned, can be proven not to be learned.

What can and cannot be learned should have causal effects in protocol runs. We extend GNY logic in this way in the next two sections.

6.2.1 New Inference Rules for Proving Possession

In Chapters 8–10, we will present new methods for proving possession of information based on cryptographic hash functions. In order to prove these methods correct, we have to model some properties of cryptographic hash functions which have not yet been modeled in any authentication logic. This modeling is done by adding the appropriate inference rule **H2** to GNY logic.

The reasoning behind the added inference rule takes as its starting point another inference rule (**I4**) which reflects a way in which possession can be proven. Slowly we will manipulate this rule until we arrive at **H2**. Note that we do not depart from an inference rule like **H-BAN**, because that rule is faulty (see Chapter 5).

Since we are discussing inference rules which can be applied in protocols in which one principal (the *prover*) proves something to another principal (the *verifier*), we will use in the presented inference rules the names P and V to denote the prover and the verifier (as opposed to using the names P and Q for two arbitrary principals). Moreover, any malicious principal in our discussion will be denoted with the name C (for Charlie). Messages are denoted X.

A well known method for proving possession of a certain message is to sign the message one wants to prove possession of, and then to show this signed message.[8] Obviously this method cannot be used in a setting where the message itself should be kept secret, because the message will be disclosed when showing the signed message. Nevertheless it is interesting to look at the inference rule that captures the interpretation of signed messages, **I4**, repeated below (from [GNY90]; note that the names of the principals have been changed, therefore we name the rule **I4′**).

[7] Of course, without making unjustifiable assumptions.

[8] The signature is required because the communication channel does not reliably 'say' who has sent a particular message. This is a result of using the malicious adversary model.

I4′
$$\frac{V \lhd \{X\}_{-K}, \quad V \ni +K, \quad V \models \overset{+K}{\mapsto} P, \quad V \models \phi(X)}{V \models P \mid\sim X, \quad V \models P \mid\sim \{X\}_{-K}}$$

What the rule says is this: If V sees a signed message $\{X\}_{-K}$, knows the public key $+K$, knows the corresponding private key $-K$ belongs to P, and recognizes X to be a message, then V is entitled to believe that P once conveyed the signed message $\{X\}_{-K}$, and thus also once conveyed the message X itself.

Important to note are two silent assumptions of this inference rule:

1. For any X, no principal C will convey $\{X\}_{-K}$ where $C \neq P$. Thus, P's private key $-K$ is only known to P and P will never convey $-K$.[9]
2. P will, in a sense, be conservative in what he signs: P will only sign intentionally and with consent.[10] P will never sign unseen messages.

The reason for the second assumption is that a digital signature is non-repudiatable.[11]. Making similar assumptions, we could introduce a new identity-related inference rule:

H1
$$\frac{V \lhd *H(X,P), \quad V \ni (X,P)}{V \models P \mid\sim (X,P), \quad V \models P \mid\sim H(X,P)}$$

If V sees a message $H(X,P)$, which V did not send himself previously, and also possesses (X,P), then V is entitled to believe that P once conveyed (X,P) and $H(X,P)$.

The assumptions under which this rule is justified are these:

1. For any X, no principal C will convey $H(X,P)$ where $C \neq P$.
2. P will, in a sense, be conservative in the set of X's for which he (P) conveys $H(X,P)$. More specifically, P will only convey $H(X,P)$ for X's of which he (P) wants to show other principals he possesses X.

These two assumptions tie together just like the two assumptions of rule **I4′**: the first assumption states that only one principal is capable of sending certain messages, and the second states that this principal will only do so with informed consent.[12]

However, the first assumption of inference rule **H1** is not justifiable. Relying on rule **H1**, a malicious principal C, knowing P and any secret X, can

[9] More precisely, we mean that no C will send $\{X\}_{-K}$ before receiving $\{X\}_{-K}$: Thus, C could perform replays of messages, but cannot generate messages signed with the key $-K$.

[10] For example, P will not sign his own death penalty.

[11] It is however important to distinguish the different *intentions* a signature may convey. A signature may convey, for example, a confirmation of a contract, or a receipt, or something completely different. The signer should always assure that he consents the intention which he conveys with his signature. In particular, a principal may sign an unseen message in a challenge-response protocol as long as the context guarantees that the signature only conveys a receipt. This can be assured by using a particular keyset for such signatures, or by including the intention in the signed message itself.

[12] Since **I4′** is just a syntactic variation of **I4**, it of course also applies to **I4**.

'commit' P to conveying the secret X by broadcasting $H(X, P)$. Assumption 1 of inference rule **I4′** does not suffer from such a problem: to construct $\{X\}_{-K}$, one has to possess $-K$.

To prevent malicious principals from creating havoc by sending $H(X, P)$, we should require the message sent to be authenticated, i.e., that it is known that P sent the message $H(X, P)$. Using sender identification, a verifier can distinguish proofs of possession by malicious principals from proofs by intended principals. When we incorporate sender identification, we can introduce a more moderate rule like this one:

H2 $$\frac{V \models P \mid\sim *H(X, P), \quad V \ni (X, P)}{V \models P \mid\sim (X, P)}$$

If V believes P once conveyed the the message $H(X, P)$, which V did not send himself previously, and if V also possesses (X, P), then V is entitled to believe that P once conveyed (X, P). This effectively eliminates the first assumption of rule **H1**.

Rule **H2** is justified under the following assumptions:

1. For any X, no principal C will convey $H(X, P)$ where $C \neq P$.
2. P will, in a sense, be conservative in the set of X's for which he conveys $H(X, P)$ in an authenticated manner (that is, such that P can be identified as the sender). More specifically, P will only convey $H(X, P)$ for X's of which he wants to show other principals that he possesses X.

Though these assumptions are not very different from the assumptions of rule **H1**, the working of rule **H2** is quite different. Firstly, a malicious principal C, knowing both X and P, cannot 'commit' P to conveying the secret X by broadcasting $H(X, P)$. Moreover, if a malicious principal would broadcast $H(X, P)$, and if P would receive it, sign it, and broadcast the signed message, this would still not result in anyone being convinced that P actually possesses X. Some may be convinced that P *conveyed* X, but conveying a message does not imply *possessing* a message!

For any principal V to believe that P possesses X, based on rule **H2**, X should be *fresh*. More precisely, X should contain a term that V believes to be fresh, and then V could apply rule **I6**. Typically, V should construct a fresh term F, and this term should be combined with X, yielding $X' = (X, F)$ and then $H(X', P)$ should be computed. If P possesses X and receives F, then P can construct a convincing proof of possession.

However, if P has an assistant C who possesses X, P might forward F to C, and C might compute $H(X', P)$ and send this term to P. In turn, P could sign this term and send it on to V, who will be convinced. Is this a problem? Well, both yes and no. Yes, because strictly spoken it does not guarantee that P actually possesses X. No, because it does guarantee that either P possesses X or P has a rather cooperative assistant who does possess X and is willing to perform computations on X on behalf of P. Assumption 1 above essentially rules out that such an assistant exists.

There is a slight technical issue with rule **H2**, which also applies to rule **H1**: How can a principal P make sure that he is not sending some message M in an authenticated manner (that is, such that P can be identified as the sender) without actually knowing that he is sending $H(X, P)$? For example, P might be required to sign a challenge M. P cannot verify whether this challenge M in fact is equal to an $H(X, P)$ if he does not possess X. This problem can be solved by adding something like a publicly known and recognizable *speech act token* to the hash value that has to be signed: P would have to sign ("I know", $H(X, P)$) instead of just $H(X, P)$. The speech act token can always be recognized by P, and therefore P can prevent erroneously signing hash values. The inference rule needs to be adjusted to reflect this, giving rule **H3** shown below. In such a way, P can make sure that he never accidentally signs a value that may be interpreted using inference rule **H3**.

$$\textbf{H3} \qquad \frac{V \models P \mid\sim (\text{"I know"}, *H(X, P)), \quad V \ni (X, P)}{V \models P \mid\sim (X, P)}$$

This rule has the same assumption as **H2**, except that for **H3**, P can really make sure the assumption is true, because P always knows it when he sends a signed message may be used using inference rule **H3**. This rule requires that the language of the GNY logic be extended with *tokens*. We decide not to do this (yet), and use rule **H2**, knowing that we can trivially modify protocols and proofs to reflect rule **H3** instead of rule **H2**.

6.2.2 Proving That Principals Do Not Learn Too Much

Authentication logics focus on establishing whether the principals interacting in a security protocol draw correct conclusions. However, for security protocols, it is also crucial to prove that certain principals *cannot* draw some specific conclusions. In this section, we will enhance GNY logic to extend its reasoning capabilities about *not* learning. First, we make sure that not learning has *causal effects* on protocol analysis, and secondly we enhance the logic to allow us to precisely state *in what circumstances* it can be guaranteed that certain facts are not learned.

Our proposal for incorporation is simple and effective. The protocol parser which translates an idealized protocol into a number of step transitions (see Section 4.3), should require that the sending party actually possesses ($\ni$) the message it is supposed to send, before sending the message.[13] Thus, any step transition is of the following form:[14]

$$[Y, \quad P \ni X]\,(P \to Q\colon X)\,[Y, \quad P \ni X, \quad Q \lhd X]$$

whereas in the original GNY logic, the step transition has only this form:

$$[Y]\,(P \to Q\colon X)\,[Y, \quad Q \lhd X]$$

[13] This way of reasoning has also been used in our earlier work [TvdRO03].

[14] For simplicity, we omit the $*$ (not-originated-here) sign which the GNY protocol parser in some cases adds to the postcondition [GNY90, Section 5]. The not-originated-here sign is implicated.

The effectiveness of this modified protocol parser lies in the fact that the protocol parser introduces a precondition that should be derivable from earlier statements. If it is impossible to derive the precondition $P \ni X$, then it is impossible to perform the protocol step $P \rightarrow Q: X$, and it is impossible to create a *legal annotation* of a protocol.

To incorporate this precondition into our notion of a *heavy annotation* (see Section 4.3), we require in a heavy annotation that every assertion of the form '*is told* ($\lhd$)' which is added after a protocol step is not only annotated with the step number, but also with the assertion number which shows that the sender actually possessed the message before sending it.[15]

This modified protocol parser and annotation requirements make sure that, just as in the 'real world', there is a causal connection from *not knowing* X to *not being able to send* X.[16]

Now that we have made sure that the inability to draw certain conclusions has causal effects on the protocol analysis, let us focus on the inability to draw certain conclusions. This should be proven for both active and passive attacks. An *active attack* is an attack in which a malicious principal manipulates the messages exchanged in a protocol in such a way that the honest participating principals learn other things than intended by the protocol. A *passive attack* is an attack in which the attacker learns something he should not learn, while the only capabilities available to the attacker are eavesdropping and inference, and notably *not* message interception and modification, as in the malicious adversary model (Dolev-Yao threat model). The literature about authentication logics generally addresses the case of active attacks (like in the Needham-Schroeder Public-Key protocol (NSPK) [Low96]), but not the case of passive attacks.

We demonstrate an approach to proving that principals cannot learn specific facts in the course of a protocol run. We believe that this is a significant contribution to establishing for concrete protocols a proof of property 3 as mentioned in Section 6.2, page 71.

Normally, when proving a protocol using an authentication logic, assumptions about the participating principals are stated. We introduce an extra principal and show that this principal cannot infer what should be kept secret. This new principal E is Eve the *e*vil *e*avesdropper. We make no assumptions on what role Eve takes in the protocol: Eve may either be one of the participants or an external observer. Just as with any other principal, we list assumptions about what Eve possesses and believes at the beginning of the protocol. The meaning of the assumptions is somewhat different, however. When we state

[15] If we return to the heavy annotation of the example of the signing parrot protocol, shown in Figure 4.4 on page 53, line 6 should carry as justification '[1](4)' instead of just '[1]'. Line 7 cannot be justified right away, but from line 6, using inference rules **P1** and **P8**, $B \ni \{N\}_{-K}$ can be inferred. The line on which $A \lhd * \{N\}_{-K}$ is inserted, should carry as justification '[2](x)', where x is the line number on which $B \ni \{N\}_{-K}$ is inferred.

[16] Note that this modified protocol parser does not rule out attacks in which forwarding of messages plays a role: to forward a message, the intruder still has to observe the message.

an assumption for a 'normal' principal participating in the protocol, this is in some sense a weakness of the protocol: it has to be met in order for the protocol to be correct. When we state an assumption about Eve, this is a strength of the protocol: even if Eve knows or possesses this a priori, the protocol is still correct in the sense that Eve cannot infer the secret. Thus, we establish the maximum amount of a priori beliefs and possessions Eve may have under which it is still impossible for Eve to infer the secret facts, a *maximum belief set*.[17] Just as with 'normal' authentication logic proofs, the list of assumptions allows to reason about subtleties concerning the quality and applicability of a protocol.

To sum up, we model two properties of a passive attacker, namely

1. its beliefs and possessions (by means of a *maximum belief set*), and
2. its inference capabilities (by means of *completeness assumptions*, see Section 6.1.2).

Using these beliefs, possessions and inference capabilities, we can compute what a passive attacker can learn from observing a protocol run. The things that should be kept secret should not be learnable for the passive attacker.

6.3 Conclusion

Authentication logics are powerful instruments that should be created and handled with care. Two types of mistakes that are easily made are (1) making implicit (unstated) assumptions, and (2) omitting inference rules. When all inference rules modeling a particular cryptographic primitive are added to an authentication logic, one can guarantee a limited kind of completeness of an authentication logic.

The important elements of our extension to GNY logic are the following:

Heavy annotations which make sure verification of a protocol analysis is structured and simple. The general structure of heavy protocol annotations that has been explained in Section 4.3 (page 53) is extended in Section 6.2.2 (page 76).

Completeness assumptions which allows one to state that an authentication logic models all essential properties of a cryptographic primitive. Completeness assumptions are introduced in Section 6.1.2 (page 70).

Inference rule H2 which captures an important property of cryptographic hash functions that had not yet been incorporated into any authentication logic. This inference rule is explained and introduced in Section 6.2.1.

A modified protocol parser which requires principals to possess a message before they can send it. This is needed for proving that not learning specific facts has causal effects on protocol evolution. This modified protocol parser is introduced in Section 6.2.2.

[17] This maximum belief set is not necessarily unique.

Maximum belief sets which allow one to reason about passive attackers and what they can and cannot learn in the course of a protocol run, depending on their a priori knowledge and possessions. Maximum belief sets are explained in Section 6.2.2.

Later on in this thesis, in Chapter 9, we will use our extended version of GNY logic to analyze our protocols.

Part III

Approaches

Linking information which stems from various sources, also called information integration, is difficult. Also, enforcing that linked information is kept secret seems impossible. We present the information designator, which is an information pseudonym, a concept that helps to solve both problems simultaneously.

Chapter 7

Information Designators

The discussion about the state of the art in computer security in general, and privacy protection in particular, divides the participants into optimists and pessimists. Consider for example the following question:

> Is security of exchanged information a solved problem?

If one looks at this question from the cryptography perspective, the answer tends to the positive. It is possible to store or communicate information in such a way that only the intended recipients can interpret the information. The *cryptographers* (those who design cryptographic schemes) are currently way ahead of the *cryptoanalysts* (those who try to break cryptographic schemes).

On the other hand, if one looks at the question from a civil liberties perspective, the answer would definitely tend to the negative. In practice, cryptographic techniques are only reluctantly applied to protect the privacy of individual citizens. Information about individuals from various sources is combined for commercial and 'homeland security' purposes, which are not always in the interest of the individual.[1]

The extent to which the privacy of individual citizens *should* be protected is a normative, if not political question, but to what extent it *can* be protected is a scientific question. This latter question will be the focus of this chapter. The trivial answer is that privacy can be protected by making sure that no information about individuals is communicated at all. Arguably, in the current society we have become so dependent on the automated processing of information that we cannot afford such a solution. Thus, the better question would be:

[1] The American Civil Liberties Union has a clear image of one of its worst nightmares, which can be found on http://www.aclu.org/pizza/. In a somewhat exaggerated movie, they tell a story of someone ordering a pizza on the phone, with the person handling the call looking through the client's medical and library files.

> Can privacy of citizens be protected without prohibiting the automated information processing we depend on?

In this chapter, we will demonstrate that it is possible to facilitate intricate, distributed information processing while at the same time protecting the privacy of the individuals involved. Thus, the commonly held belief that *privacy* and *information availability* are not on good terms, is not as rigid as it seems.

We cannot achieve secure information exchange by mere application of some cryptography. Cryptographic techniques often cannot be easily applied, therefore the privacy protection is mediocre in many information systems.

In 'traditional' cryptography, there is a very clear distinction between the *good guys* and the *bad guys*. The former can be fully trusted, the latter not at all. If discussing the exchange of privacy-sensitive information, it is fair to say that not every organization processing such privacy-sensitive information is intrinsically good. In fact, if Alice is afraid Bob might misuse the information, Bob does not belong to the good guys nor to the bad guys. Probably Bob belongs to the *so-so guys*: those not intrinsically bad, but not to be trusted more than strictly necessary. Cryptography assumes a clear distinction between the trusted and the untrusted, and therefore more than just cryptography is needed if privacy needs to be protected in a context where *so-so guys* exist. With this knowledge, we can answer the opening question of this chapter as follows:

> If we accept there are parties who are not unconditionally trusted, but at the same time need to process sensitive information, the security of this information is not a solved problem (yet).

It would be ludicrous to assume that if privacy were of no concern, all information from various sources could easily be combined. Solving problems surrounding information integration properly is already so difficult [RB01, DH05, GK05], that it is no real surprise that issues such as privacy and anonymity are often no substantial part of the initial integration design, if they are included at all.

We believe however, that both information dissemination control and proper information integration can actually be achieved by one and the same instrument. In this chapter, we will present our solution, the *information designator*. This solution is by no means a 'one size fits all'-solution, nor is it easy to implement given the legacy of information systems. On the other hand, our solution is in the end rather elegant and effective, and we would like to present it as a proof of concept.

In Section 7.1, we will introduce the research field of information integration, and how its problems relate to ontologies and dissemination of information. Section 7.2 will present our new approach to these challenges, and the central concept of this approach: the information designator. Phenomena mentioned in Section 7.2 will be illustrated in Section 7.3, where we show an example of how both information integration and dissemination control are solved jointly. Section 7.4 will show how cryptography can be used to establish desirable properties of information designators. In Section 7.5 we discuss

student	course	grade
John Doe	expert systems	A
Joe Average	statistics	C
Jim Doolittle	statistics	E
...	...	...

name	birth date
J. Average	7/6/1946
J. Doe	3/31/1948
N. Chimpsky	11/21/1973
...	...

TABLE 7.1: Two relational tables which can be combined to relate courses to birth dates.

the relevance of our approach and relate it to other research. And of course, we end with some conclusions.

7.1 Information Integration and its Challenges

In this section, we present our analysis of the fundamental challenges that must be faced when integrating information. These problems stem from the fact that some information may be modeled multiple times, but differently (Section 7.1.1), and from the fact that information, once disseminated from its original source, is hard to control (Section 7.1.2).

Information integration is done when a group of organizations decides to pool their information. Typically this is a tedious task in which unrelated, individual (relational) databases have to be combined in such a way that the databases jointly act as if one. A query on the aggregate database must be seamlessly divided into subqueries which operate on the individual databases, and the results of these queries have to be merged into one query result.

To actually integrate the databases, the schemata of the databases are compared, and fields in different databases but with similar semantics are identified. For example, one database may relate students to courses, and another database may relate names to their birth dates: the student and name fields can then be used to relate courses to birth dates. (See Figure 7.1.)

When tying databases together in this way, two problems frequently occur. First, it is difficult to make sure that all matches that should be found between different individual databases are actually established. This is typically due to different ways of encoding the same information in different databases. Second, where the individual databases may be internally consistent, the joint databases may very well be inconsistent.

The common denominator in addressing these problems is to expose more information. Making more information available allows for more matches to be found, and allows for inconsistencies to be detected. Thus it seems necessary to expose a lot of information in order to achieve proper information integration. From the privacy and anonymity perspective, a priori exposing a lot of information is out of the question. This suggests that information integration on the one hand, and confidentiality on the other hand, are not on comfortable terms.

At this point, it is good to make some remarks on what *we* mean by *information integration*. In the abstract sense, information integration is the act or process of making sure that information stored and maintained at separate locations and organizations, can be combined with ease and without introducing inconsistencies.

Roughly, there are two ways to accomplish this goal. The first way is to take a number of information sources, and perform the difficult and tedious task of matching the information at the different locations. This includes among others record matching, data re-identification, record linkage, and this is what is traditionally understood when one refers to information integration [GK05, DH05]. However, there is another, second way of achieving the goal of assuring the easy combination of dislocated information, which will be our approach. The main idea is to anticipate the combining of information at the moment the individual information sources are set up. In Section 7.2, we will show how this can be done without assuming a trusted central authority and without disclosing information which may need to remain confidential. We consider such an approach an important step towards solving the problems of information integration, though it is somewhat nonstandard, if compared to the traditional meaning of information integration.

7.1.1 Overlapping Ontologies

An ontology defines, for a single information source, what the information stored in the source represents, and how it is structured [AvH04]. Within the relational database paradigm, a database schema can be seen as the implementation of such an ontology. When information sources are combined, this is done by comparing the ontologies of the different sources. If the ontologies overlap sufficiently, or if it is possible to map parts of one ontology onto some parts of the other ontology, the information from the two sources can be linked.

The individual information sources are almost always stand-alone information systems by origin. Because of this origin, these systems store many kinds of information, since they have (had) to maximally support the owning organization. For example, a university database typically stores a lot of details about students, like students' previous educations, birth dates, private addresses. This information is stored because at some moment in time the university will need it for some task.

As a result the information sources subject to information integration tend to have a rather large ontology. It can even be argued that information integration happens because the ontologies grow so large that it is no longer viable for one single organization to maintain all information within one stand-alone information system. Keeping track of how all information should be modeled, as well as actually obtaining all the information for a single, large stand-alone information system becomes very complicated when information from sources outside of the organization have to be included.

It can be expected that in the example of the university database, inaccuracies will exist in the information that comes from outside of the organization.

Minor inaccuracies may arise from data-entry, bigger inaccuracies may arise from updating the information infrequently or not at all. Intricate inaccuracies may occur when the ontology does not have enough expressive power to facilitate the information that should be stored. When inaccurate information from various sources is combined, this will almost inevitably lead to inconsistencies.

It should be expected that the information in the university database concerning the core university activities, such as course enrollments, grades and diplomas given, is essentially, if not by definition, correct.

An organization which creates new information is probably the best suited organization to model this information, and to maintain an ontology of this information. However, it is not unusual for such an organization to maintain an ontology covering more than the *core business* of the organization itself, but also to maintain a part of its ontology which is error-prone, and essentially a duplicate of many parts of many other ontologies of other organizations.

If the overlapping parts of the information sources' ontologies contain personal information, this means that this personal information is stored at several sites. If for whatever reason this information should be kept under some restricted disclosure regime, *all* sites storing this information should adhere to the restricted disclosure regime. Obviously, it may be impossible to enforce this, which means that the information is kept private just insofar the weakest link does not disclose it. Information stored at only one site is easier to control, since there is only one party which has to adhere to a specific disclosure regime.

7.1.2 Information Propagation

The reason for linking information sources, i.e., to perform information integration, is twofold from the perspective of a participating organization. First, the organization wants to *retrieve* authoritative information from external sources. When retrieving data, the desiderata are *availability* and *integrity* of the information. Second, the organization wants to *publish* information, but possibly only to a restricted set of *consumers* for some restricted set of *application uses*. When publishing data, enforcing *dissemination policies* is the main challenge.[2] These aims and interests of the participating organizations are depicted in Figure 7.1.

To maximize integrity of information, it would be good to verify the information at the authoritative source, as shortly as possible before actually using the information. Better could even be to just *fetch* the authoritative information at use-time. To prevent unwanted dissemination of information, best would be to verify that for each time the information is used, there is a legitimate reason to use this information. This can be achieved by requiring authorization for each individual 'shipment' of information, and to make sure the information can only be used for the purpose stated in the authorization procedure.

[2] It is rarely if ever the case that an organization would want to *directly* alter information that is within the realm of another organization.

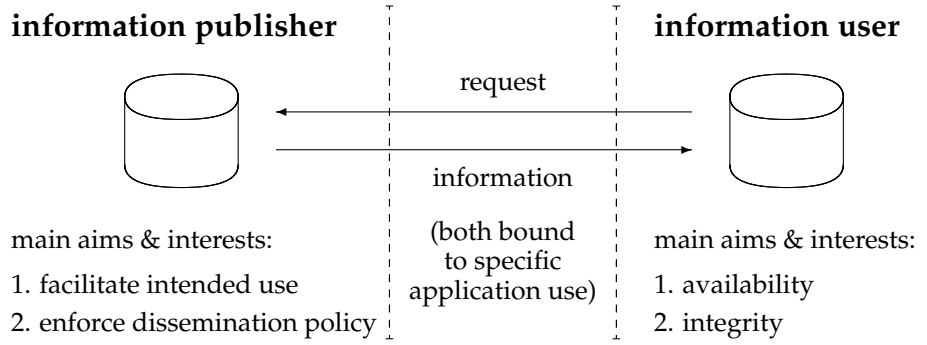


FIGURE 7.1: The main aims and interests for organizations participating in information integration. Virtually every organization in an information integration setting is both *user* of some externally published information, and *publisher* of some other information. This figure depicts the interests for one organization in the role of *information publisher* and another in the role of *information user* with respect to one 'piece' of information.

This leads to a central adage in our approach:

Don't propagate, but link!

Information should only be disclosed when it is really about to be used, and not at any time before that. At the very best, the disclosed information should be destroyed immediately after use.

This adage may seem very unrealistic in two ways. First, it has to be properly defined what 'using information' actually means. If it is too widely defined, it does not really restrict dissemination. For example, if counting the existence of a piece of information (such as when counting students in a room) is regarded as 'using' information, counting a student would lead to disclosure of his personal information. If 'using information' is too strictly defined, it prevents any sensible use of information.

Information designators, introduced and explained in the next section, will solve this apparent paradox. Second, one may question whether not propagating information would lead to unacceptable performance bottlenecks in the resulting information system. Assuring proper information granularity will minimize, if not circumvent this problem. Information designators are the instrument that will offer us the flexibility to reason about information that is not *physically present*. This may lower the capacity of information sources to disseminate information, but it will give the information holder much more control over who has access to what information.

7.2 A Joint Approach to Privacy, Anonymity and Information Integration

In this section, we will present our approach to solving information integration and dissemination control. First, we introduce the *information designator* in Section 7.2.1. Section 7.2.2 explains how, using information designators, information from various sources can be tied together, while these sources remain in control over their information. Moreover, in Section 7.2.3 we explain how an organization that provides information designators to others, can accurately manipulate which others can actually use the provided information designators, and to what extent.

7.2.1 Information Designators

The central instrument in our approach is the *information designator*, which is a piece of information whose sole purpose it is to refer to other information without containing the other information and without any reference to a context. Every designator contains an address at which a software agent, an *exchange agent*, can be contacted to translate the designator into the information it refers to. An exchange agent may place restrictions or conditions on the information requester before it translates a designator into the information it refers to.

An example of a designator could be `12345.67890`. If Bob were to ask Alice her home address, she could give Bob this designator. Bob then knows that if he wants to send postal mail to Alice's home, he must contact the exchange agent at `12345`[3], and hand over to the exchange agent the full designator `12345.67890`. In turn, if Bob meets the conditions set by the exchange agent, Bob will receive Alice's home address. The fact that the designator refers to Alice's home address, cannot be inferred from the designator itself. Bob only knows the designator has this semantics because Alice told Bob so. Alice should make sure that the exchange agent will answer Bob's call for information in the right way.

The process of Bob obtaining Alice's home address is now a two-step process, as follows:

- The *principal* step is the one in which Bob asks Alice her home address, and possibly after some combination of authorization and agreeing on some terms, Alice hands over the information designator to Bob. From that moment on, until Bob contacts the exchange agent, the designator is something like an 'I owe you' (IOU) of Alice to Bob, where the debt of Alice is the information that stands for her home address. Though Alice has granted Bob access to the information of her home address, she has still control over it. Alice can change her home address without any administrative burden to Bob. Also, Alice can *retract* her designator

[3] This could be a phone number, IP address, or something else that allows setting up a communication channel in an automated way.

by instructing the exchange agent not to give Bob the information the designator refers (or: referred) to.

- The second step is the *materialization* step, in which Bob contacts the exchange agent. If Alice hasn't retracted the designator, and Bob meets the conditions set by the exchange agent, Bob will obtain the information that is Alice's home address.

  The use of this kind of mapping allows for changing of the information referred to without the need to update references. This would allow telecom operators to redistribute phone numbers, or the city council of Tel Aviv to rename the "Malchei Yisrael Square" into the "Yitzhak Rabin Square" without introducing inconsistencies into databases where these numbers or names are referred to.[4]

This flexible use of designators has benefits for both the users of information and the providers of information. The users of information have access to the information they need, but they do not need to worry about the housekeeping of this information. Barring unforeseen exceptions, the users are guaranteed access to the information. At the same time, the providers of information are given greater control over the dissemination of the information, and can individually audit the use of the information.

The architecture presented here could be considered to be a peer-to-peer (P2P) data management system (PDMS), like the Piazza PDMS [HIM+04]. However, the PDMSs we know of lack the concept of an information designator, and do not distinguish between *raw* information, and a reference to such information. In fact, techniques used in the web services and the semantic web [ACKM04] and PDMSs are generally a vehicle to ease the problem of schema integration, whereas the information designator is a means to *bypass* the problem of schema integration.

7.2.2 Dependency and (Un)linkability

It may seem that by using information designators, the users of information are subject to possible arbitrary behavior of the providers of information. For example, the providers might choose to instruct their exchange agents to further deny any information to the users. We do not believe that this scenario is any more likely to happen than in a context where another mechanism for information integration is used. Even stronger, we believe the *possibility* to retract designators on an individual basis may well happen to be an essential requirement for many organizations to participate in an information integration project. More organizations will be willing to provide information, because

[4] Thus, because the 'raw data' such as a street name is separated from the concept of what it represents in the data structure, it is possible to perform database transactions on the 'raw data' without even touching the database records that link to the 'raw data'. From the perspective of complex database transactions and the frame problem, this is an interesting feature [Rei95].

they have the option to retract the information in the case of an unlikely or unforeseen event.

Using information designators makes existing informational dependencies of organizations explicit. If an organization depends for some task on information from another organization, this will inevitably lead to an infrastructure in which designators are used whose corresponding exchange agents operate under the auspice of the organization depended on.

The information designator approach has the very interesting property that if it is fully applied, there need not be overlapping ontologies. Different organizations *provide* information under their own, simultaneously provided ontology. If this information is used, the provided ontology will be used. If this information is related to information from some other ontology, it will be related by means of a designator in the one ontology, pointing to information in the other ontology. Technically this means that instead of multiple information sources storing identical information, there is one information source that stores the original information, while other information sources store references (information designators) to this original information. In this sense, designators are the *glue* between ontologies, that allows ontologies to be disjoint, but integrated at the same time.

Disjointness of ontologies is an extremely useful feature from both the information integration and from the privacy and anonymity perspective. It effectively makes it impossible for conflicting information on one subject to be established, which seriously limits the class of possible inconsistencies that can arise from linking information.[5] At the same time, information can be linked without automatically disclosing a part of the linked information: information 'normally' (otherwise) made public can be kept private.

7.2.3 Operations on Designators

One could wonder whether introducing designators actually improves privacy and anonymity, by reasoning that the designators themselves will fulfill the role of identifying information; that a person is not identified by his or her name, but by the designator that refers to his or her name. This would indeed be the case, if for each piece of information, there would only be one designator referring to it. If multiple parties would have this same designator, they could recognize that the information they individually have is about the same person or artifact.

However, it is *nowhere necessary* that each piece of information has only one designator pointing to it. In fact, the introduction of designators would have little to offer on the privacy and anonymity front if each piece of information would have its unique corresponding designator. An organization handing out designators could in fact every time it hands out a designator, create an

[5] The claim is somewhat weak, and this is on purpose: there might be other classes of inconsistencies we have not thought of. As we cannot prove to prevent *all* types of inconsistency, we will not claim so.

extra 'fully anonymous' designator for the information it needs to point to.[6] In this scenario, the organization handing out designators knows for sure that the designators it handed out cannot be combined in any way to find matches between designators.

There are excitingly many policies between strictly unique designators on the one hand and fully anonymous designators. Here, we will mention just a few. Designators to the same piece of information could be the same, if given to the same requesting organization, or if given to an organization in some given group, thereby allowing the organization or group of organizations to compare their designators. It is totally at the discretion of an organization handing out designators to decide whether its designators will have these properties. Also, it could provide these properties to some users of information, and not to others. The closer the policy is to strictly unique designators, the more recombination possibilities there are that need no consent of the organization that handed out the designators.

An organization handing out designators does not have to fully decide on its policy when it starts handing out designators. For example, it could by default hand out only fully anonymous designators, and upon special request exchange some of the designators for designators that can be recombined in some specific way. A user or group of users could for example ask the specific question if within a specific set of their designators, some refer to the same information. The organization handing out designators could in turn translate the given specific set into other designators in such a way that only within this set duplicates can be detected.

Depending on policy decisions, the extent to which designators are valuable to users can be varied in a very precise way. Organizations handing out designators can choose to make their designators on a per-user and per-transaction basis, homomorphic to the information the designators refer to.

7.3 An Example: the Datamining Bookshop

The information designator is more than a theoretical concept. In fact, we have built a prototype system which demonstrates several of the above-mentioned properties. The prototype illustrates an example of information integration and information exchange which would, without information designators, either be impossible or it would seriously infringe privacy. We present the prototype here for three purposes:

1. to stress that information designator systems *can actually be built* [Hid04],

[6] Creating an extra designator every time a designator is handed out will not have any serious impact on the required storage capacity of the exchange agent. This can be achieved for example by designators that actually are encrypted versions of a *master designator*, of which the exchange agent is the only agent knowing the decryption key. For more examples of designator obfuscation, see Section 7.4.

2. to show how an information designator system works internally, thereby illustrating the subject matters explained in the previous section, and
3. to give an application example which demonstrates how information designators help in protecting privacy and maintaining unlinkability.

7.3.1 Organizational Setting

Our example is about information flow between the following four organizations.

Civic Authority This organization has the task to maintain the municipal inhabitants register, which contains inhabitants' names, birth dates, and residence addresses.

Local School The students of the local school live in the domain of the civic authority. The local school keeps record of its students, their results, their course enrollments and required literature for courses.

Local Bookshop This organization is located conveniently next to the local school. The local bookshop wants to provide for the literature demands from the local school students, but does not want to overstock.

Book Publisher This organization publishes the books that are used in the courses of the local school. The book publisher maintains information about books and their details, such as titles, authors and ordering information.

There are many relations between the information maintained by these organizations. The students of the local school are all registered at the civic authority. Contrary to the book publisher and the local bookshop, the local school has the right to access some of the information stored and maintained by the civic authority. The books the local school recommends for their various courses, are all published by the book publisher. The book publisher is fairly liberal in allowing access to the information about its books, however, it has some extra information for its known resellers, one of which is the local bookshop.

The local bookshop has a strong desire not to overstock books, and at the same time the local school wishes all their students to have their obligatory books when the term starts. As a result, the local school depends on the behavior of the local bookshop, and the local bookshop depends on information from the local school. A very naive way to solve this dependency would be that the local school gives the local bookshop full access to the local school administration. This would obviously lead to unacceptable privacy infringements, even if the local school would limit the access to things like course enrollments (and hide exam results). A slightly less naive solution would be that the local school gives the local bookshop an update of the expected number of required books once in a while. However, these updates are just snapshots. It would be ideal

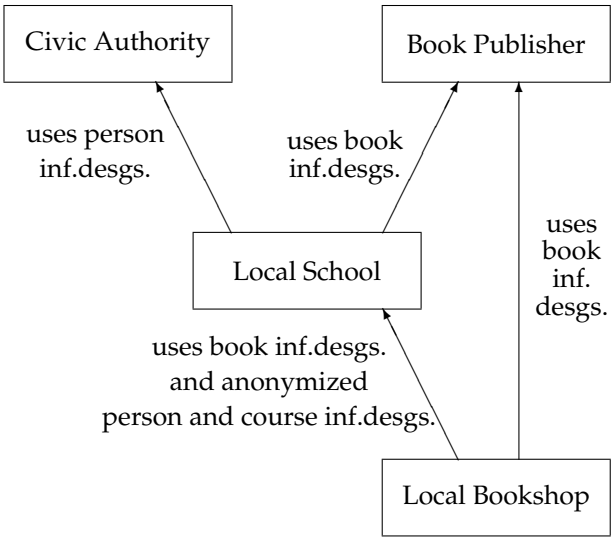


FIGURE 7.2: An information dependency graph containing the four organizations of the example. The organizations and their information demands are described in Section 7.3.1. An arrow from organization A leading to B means that A is interested in information maintained by B. For example, the local bookshop depends on (desires) information from both the local school and the book publisher. 'Information designators' is abbreviated to 'inf.desgs.'.

for the local bookshop to directly look in the administration of the local school at the moments relevant for the local bookshop. If this would not infringe on the privacy of the students, the local school would probably find such a solution fairly unproblematic.

Figure 7.2 shows how the four organizations relate to one another with respect to their information needs.

The example may seem a perfect case for setting up a Web Service framework [ACKM04]. However, a Web Service framework would offer only a means for exchanging information, while the use of information designators offers a means for assuring mutual information integrity and consistency while keeping almost all information confidential. The confidentiality and integrity is not manually crafted into the architecture, it is a mere consequence of using information designator technology.

7.3.2 Designators in Action

The information that is maintained by the organizations is summarized in table 7.2. The table shows the schemata of the local databases. These might be plain vanilla relational databases, in which the 'person' field contains a string which denominates the person's name. This is however not the case. All fields

providing organization	table name	field 1	field 2
civic authority	names	person	**name**
civic authority	birthdates	person	**date**
local school	students	-	*person*
local school	courses	course	**name**
local school	enrollments	course	*person*
local school	literature	course	*book*
book publisher	book_details	book	**details**

TABLE 7.2: The schemata of the information that is maintained by the civic authority, the local school and the book publisher. The fields written in *italics* contain designators from an external organization. The fields in **bold** contain raw data, that is, information which is not a designator, and therefore readily interpretable. The fields written in normal font, are designators which are locally defined.

contain *information designators*. Some designators are created by the local organization, like the designators stored in the 'course' fields. The content of these fields is fully defined by the local school; the local school creates the designators that refer to the various courses offered by the local school. Some other designators are *foreign*, they originate from outside the organization. The 'person' designators are created by the civic authority, and the local school's 'person' fields are an example of fields which will be filled by such foreign designators. In this way, the local school database is linked to the database of the civic authority. A similar link exists to the database of the book publisher. The 'names', 'birthdates', 'courses' and 'details' are the only tables also containing raw data that is not encoded via a designator.

The local bookshop desires a summary which states how many copies of each book can be expected to be sold. Executing the global SQL query shown in Figure 7.3 would provide this information. The local bookshop should make sure that this query is executed, and that parties providing necessary information cooperate sufficiently.

To execute this query, access is needed to the 'enrollments' and 'literature' tables from the local school, and to the 'book_details' table from the book publisher. There are essentially two ways to execute the query. First, the query could be divided into two subqueries. The first subquery is executed by the local school, its results are sent to the book publisher, which performs the second subquery, and the merged result is forwarded to the local bookshop. This solution works, but for more complex queries, it will become quite difficult to divide the query into subqueries. Also, the intermediate query results could leak information. The second solution could be to grant the local bookshop read access to the required tables. If these tables were 'plain vanilla' relational databases, access to these tables would have disclosed detailed information about the interests and advances of named students. This would be a very ob-

```
SELECT COUNT(DISTINCT person),details
   FROM enrollments
   JOIN literature USING (course)
   JOIN book_details USING (book)
GROUP BY book;
```

FIGURE 7.3: A global SQL query, which would provide the local bookshop with the information it desires: details of each of the books needed by the students of the local school, and the number of required copies of each of these books. To obtain this desired table, the 'enrollments' and 'literature' table of the local school are consulted, as well as the 'book_details' table of the book publisher.

vious example of privacy violation. However, if the following three conditions are met, the privacy conditions are much improved.

1. The information in the tables does not contain sensitive information.
2. The information in the tables cannot be used to retrieve sensitive information.
3. The information in the tables cannot be combined with external tables to infer sensitive information.

We will show how designators can be used to make sure the tables of the local school satisfy these properties. First, by using designators, it is ensured that no raw identifiable data is stored in the tables, hereby meeting condition 1. Satisfying condition 2 is somewhat more complicated, but well doable. It should be made sure that though the designators can be used by *the local school* to retrieve information, this cannot be done by others. In fact, it can be expected that the civic authority would only grant the local school access to its information in case it can make sure the local school will not leak the information. The solution to condition 2 lies in the civic authority, which can create designators especially for use by the local school in such a way that others, such as the local bookshop, cannot materialize the designators. How this is done technically and in an efficient way is shown in Section 7.4.

Condition 3 can be met by making sure the designators given to the local bookshop do not match designators referring to sensitive information the local bookshop may have found elsewhere. Thus, the designators given to the local bookshop should be unlinkable. However, the internal correspondences between the tables should remain intact. In our example, if a student occurs multiple times in the enrollments table, all these occurrences should be replaced by the same designator. Yet what the *actual content* of the designator is, is irrelevant and may therefore be altered. A way to create such designators on the spot is shown in Section 7.4.

If all three conditions are met, there is no problem in granting the local bookshop full access to the 'enrollments' and 'literature' tables as maintained

by the local school. The book publisher grants the local bookshop access to its 'book_details' table and everything is solved. That is, everything is solved from the privacy and unlinkability perspective, while still giving the local bookshop a royal amount of freedom in accessing the information it desires to have. The local bookshop can get up-to-date information at any moment it wishes.

Still, there is a lot to optimize. Of course, the local bookshop might retrieve the full contents of the 'enrollments' and 'literature' tables, and perform the *joins* by itself, but it is easy to see that this would require a high amount of communication. It may well be the case that using subqueries and executing subqueries at various different locations is resource-wise a more optimal solution. Therefore, the ideal approach should be liberal in allowing queries to be divided into subqueries. Our approach is such an approach, and this will be the focus of the next section.

7.3.3 Observations About the Use of Subqueries

The approach to the question whether or not to use subqueries when assessing a global query may seem unusual. First, we found subqueries difficult, information-leaking instruments. So instead, we granted access to all information sources, but we ensured that nothing sensitive was left in these information sources. Then, we observed that though operating correctly, our solution would be very inefficient so we re-allowed the use of subqueries.

However, in making a detour away from and back to the use of subqueries, we have ensured a very important property. Namely, we have obtained that any result from any subquery cannot be linked to sensitive information, because the information it stems from cannot be linked to sensitive information. Thus, we have a guarantee about the unlinkability of the subquery results. Not only have the tables from the civic authority not been accessed during query execution, also the subquery results and query result offer nothing that might help in getting access to the civic authority's tables.

The alternative to this detour would be that for each query it would need to be assessed whether the answer would somehow leak too much information. In this assessment, answers received from previous queries should be taken into account. This easily would become complex, not to say unmanageable. The designator approach is liberal in the sense that any query which can be resolved using the 'obfuscated tables' is allowed, and restrictive in the sense that any query which cannot be resolved in this way is not allowed. In effect, linking information across organizations and hiding information from other 'third' organizations can go hand in hand in an elegant and easy way.

The detour has in fact something more to offer. Since subquery results cannot contain sensitive information, global queries may be divided into subqueries in any way that happens to be resource-wise the most optimal. The subqueries could be executed by the organizations offering the information (e.g. the local school), but also be executed by mobile agents on behalf of the information users (e.g. the local bookshop).

7.4 Methods for Restricting Designator Uses

In Section 7.3.2, we have assumed that it is viable to ensure certain properties of designators, such as that it is impossible to recombine designators in specific ways. In this section, we sketch tentative solutions for creating designators which satisfy these properties.

All examples are about three organizations, A, B and C. Organization A (the *information publisher*) is always the organization handing out a designator to B, sometimes also to C (the *information users*). Most of the solutions we present assume (deterministic) asymmetric encryption with signatures (e.g. RSA [RSA78]). When a cryptographic hash function $H(\cdot)$ is used, it is assumed that it is a correlation-free non-incremental cryptographic hash function (see Chapter 3 and Section 3.6).

Organization A internally uses designators, which we will refer to as *master designators*. The designator it hands out to organization B will be called a *user-bound designator*. Organization A has a private secret, S. The public and private keys of A are $+K_A$ and $-K_A$, and similarly the public and private keys of B and C are $+K_B$, $-K_B$, $+K_C$ and $-K_C$, respectively.

The methods described in thesis section can easily be combined within one step, if necessary. The purpose of showing these methods is to show that it can be done, and roughly how, omitting the deepest technical details. We do not claim that these ways of solving the problems are necessarily the best or most efficient ones.

1. *Designators that can only be materialized by a specific user*

   Consider an organization A that would like to hand out designators to its own information to organization B, granting B access to the information maintained by A. At the same time, A wants to make sure only B can materialize the designators. However, A lacks the capacity to maintain a record of each individual designator it hands out, since this would require storage space for each designator handed out, and it would require computation time to look up each designator in this storage at the time of materialization.

   Now, if A wants to grant B access to the information referred to by the master designator D, it hands out the user-bound designator D^B:

   $$D^B = \{D, +K_B, \text{access-specification}, S\}_{+K_A}$$

   where access-specification may be some extra information restricting the access of B to D. The user-bound designator D^B is given to B. Nobody but A can decrypt D^B. If at some moment later in time B wishes to materialize the designator, it has to send $\{D^B\}_{-K_B}$ (a signed copy of the designator D^B)[7] to A. In turn, A will decrypt D^B (using his private key

[7] It has to be made sure that A can decrypt D^B before verification of the signature, since the public key $+K_B$ required for verification is stored within D^B. A 'two-step' signature scheme can facilitate this.

$-K_A$), and verify whether the signature matches the public key $+K_B$ found in the decrypted D^B. If either decryption fails, or the signature cannot be verified, or the secret is not present, or the access-specification is not met, then A will refuse to present the materialization of D.

If D^B_M falls into the hands of a third organization, say C, this third organization cannot materialize the designator since C is unable to forge B's signature.

2. *Designators that cannot be recombined by multiple users*

   Consider an organization A that wants to hand out designators to both B and C, but wants to prevent that B and C can combine their information. Designators should be unique with respect to the information they refer to, but only within the realm of one single user. Thus, if B receives two designators D^B_1, D^B_2, it can infer whether they refer to the same information by verifying whether D^B_1 itself is equal to D^B_2. However, if C receives designator D^C_3, B and C should not be able to find out whether D^C_3 is equal to either D^B_1 or D^B_2 (without cooperation of the organization that handed out the designators, namely A).

   If A wants to create such a user-bound designator to B, it hands out the following designator to B:

   $$D^B = \{D, B, S\}_{+K_A}$$

   If the designator never needs to be looked up by organization A, the following simpler solution would also suffice:

   $$D^B = H(D, B, S)$$

   Because all steps in generating the user-bound designator are deterministic, uniqueness of designators is preserved as long as the requesting user (i.e., B) remains the same. However, if both B and C get a designator which refers to the information D refers to, these designators will not be mutually comparable.

3. *Designators that cannot be recombined over time*

   Consider an organization A that would like to allow users to analyze the structure of the information at a specific moment in time, but does not want to allow the users to analyze how the structure evolves over time. For example, in the local bookshop scenario, the local school would like to prevent the local bookshop from finding out how long students are studying at the local school. Thus, designators should only be uniquely referring to information if these designators are all obtained at the same moment in time.

   To enforce this property, A can create time-dependent designators D^t in the following way:

   $$D^t = \{D, t, S\}_{+K_A}$$

where t is the moment in time when the designator is created. Essentially, t is a time interval, and some care must be taken in choosing the size of this time interval. To be useful, t should not be too small, because otherwise too little designators from the same time frame would exist to make *any* snapshot inferences. Depending on the application domain, the interval could be as long as a minute, day, week or possibly even a longer period of time.

If the designator never needs to be looked up by organization A, the following simpler solution would also suffice:

$$D^t = H(D, t, S)$$

Note that this solution does not require a global clock, but only a local clock for A.

The space requirements (i.e., size) of designators are only limited. A designator which is constructed using a cryptographic hash function is trivially bounded in length, with current cryptographic hash functions only a few hundred bits. A designator which is constructed using an encryption step is bigger than the designator it encapsulates by a constant factor. A designator never reveals the length of the information it designates.

7.5 Discussion and Related Work

The use of information designators that we introduce in this chapter allows information systems to fulfill many different roles at the same time. They can simultaneously be a transaction system, a public information system, subject to datamining, and still hide the information contained. Moreover, integrity can be guaranteed to an extent higher than normal for information integration systems. Two important properties of the information designator system enable the seamless combination of these roles:

1. The information system can supply to different users different 'views' of the information it has, but these views are only mutually comparable if the providing information system explicitly allows and enables this.

2. The information contained in these views (i.e., in the returned records) is not interpretable without the explicit cooperation of the providing information system.

As a result, an information system can choose to allow extensive analysis of its information, without disclosing sensitive records within this information [LP00]. This is useful in applications where it is undesirable for individual records to be disclosed (this would for example harm someone's privacy) but at the same time it is not a problem to produce and use accurate aggregate

statistics of the information [ESAG02]. Simultaneously, administrative information exchange about such details between organizations remains possible.

An information designator can be seen as a pseudonym for information. While pseudonyms are typically associated with persons (as in [Cha81, Cha85, Cha92]), there is no conceptual problem in using codewords to denominate a piece of information which does not refer to a person. In this perspective, a pseudonym is just a special case of an information designator. Moreover, we have generalized the idea of using multiple pseudonyms for one person to using multiple designators for one piece of information. The decision when information designators should and can be materialized is of course essentially a policy issue which has to reflect the opinions of the participants involved. Identity escrow schemes [KP98] and threshold-based privacy solutions [JLS02] can be seen as special cases of solutions possible with our approach.

Information designators offer a mechanism to reason about information that is not physically present. If properly authorized, it is possible to retrieve the information that an information designator refers to. However, it is also possible to retrieve only *some* properties of the information designator at hand. In an insurance company for example, the claim experts normally see the names of the clients, because these are part of the portfolio, and are needed for subsequent steps in the claim handling process. For establishing a good judgment, the claim expert does not need the name of the client; it may even be argued that he will judge more fairly if he *does not know* the name of the client at hand. Similar considerations apply to tasks like the judging of job applications. Using designators, it would be relatively easy to create workflow systems that hide all information but the information relevant in the specific step of the workflow system [TvdRO03].

Reasoning about information without disclosing raw data is also subject of Chapters 8–10 of this thesis, in which we present protocols for comparing secrets for equality without disclosing the contents of the secrets [FNW96]. In Chapters 8–10, we consider two agents, both possessing 'raw data', and these agents are interested in comparing their raw data mutually without disclosing it in case the data is not equal. In that chapter we demonstrate that it is also possible to compare information that is *not even present* at any of the two agents involved. However, the organization that owns the information compared has to deliberately allow this comparison.

Thus, for the sake of the protocols of the next chapters, the information designators could be considered as 'raw data'. This allows for example two organizations, who have pools of 'anonymous data items', to compute the intersection of these pools *without identification of the data items themselves*. Such a rather counter-intuitive computation may have a number of applications, such as privacy-respecting informed policy-making.

In [FGR92, FLW91], cryptography is used to protect the contents of databases on a record level and field level, which has some similarities to our approach. However, in [FGR92, FLW91], no cooperation from the information provider is required to materialize raw data. Our approach allows the infor-

mation provider to refuse materialization of data, which is a means of control *after* information has been disclosed in the form of information designators.

Other approaches choose to protect the privacy of the *users* against analysis of their queries by the information provider (private information retrieval) [CGKS98], or to distrust the information provider to inspect the information it stores [SWP00]. Although these are not primary goals of our approach, we believe that similar concepts could be implemented in information designator systems. Indeed, when an organization stores designators which it cannot materialize, this organization is seriously limited in analyzing and linking its data and the queries it receives from users.

The database representations suggested in our work form a radical departure from some of the basics of relational databases [Cod70]. First, the tables of the database are no longer filled with actual raw data, but with some kind of 'global pointers', i.e., information designators. These designators point to information which is *vertically fragmented* over distributed information providers [CK85, BKK95, Bon02]. The ontologies of these providers do *not* overlap, which is dramatically different from most uses of ontologies [Gua98, UG96], and also noticeably different from the ontology use in the semantic web community [DMDH02].

7.6 Conclusion

In this chapter, we have described a way of structuring and linking information that is totally different from the way that information is structured and linked nowadays. Nowadays, it is common that information systems store raw data, and replicate data almost abundantly. The information designator approach is technically not yet sufficiently fleshed out to be applied to large-scale production-quality information systems. Also, lack of integration with existing legacy systems and lack of a critical mass of information systems using information designators, are currently prohibitive for a widespread adoption.

It is not our goal to present an instantly applicable technique. We want to demonstrate that information integration on the one hand, and privacy, unlinkability, confidentiality and related considerations on the other hand, can go hand in hand. In the presented information designator approach, goals like fluent information integration, information exchange and tight dissemination policy enforcement can be satisfied simultaneously.

In line with this, we believe that the apparent trade-off between privacy and availability of information may not be as rigid as commonly believed. The strong common belief in this apparent trade-off is a result of using information systems in which raw data is exchanged. Therefore, we believe abandoning information systems which mainly manipulate raw data may be part of the way to overcome the misunderstanding that information exchange and privacy can not be simultaneously established.

The question 'do you know the secrets that I know?' is a tricky one. We explore what a protocol must do in order to provide an answer to such questions. We distinguish the 1-to-many case and the many-to-many case, and survey protocols which solve these cases. There are no protocols in the literature yet which solve these cases where the domain of possible secrets is huge, except for the protocols (T-1 and T-2) we will present in the next chapters.

Chapter 8

Knowledge Authentication

We will introduce the objective of the material presented in this chapter by a rather innocent real-world situation which has actually occurred:

> Geertje and Wouter are two friends and colleagues. Geertje has told Wouter in private that she is expecting a baby.[1] Just a few days later, Wouter meets the secretary at the coffee corner. The secretary looks expectantly to Wouter. Wouter would like to gossip with the secretary about Geertje's pregnancy. But Wouter has also promised Geertje not to disclose the secret of her pregnancy. If Wouter wants to keep his promise, he can only start gossiping about Geertje's pregnancy if he can be certain that the secretary already knew the secret. Wouter cannot simply ask the secretary whether she knew, because such a question would disclose the secret.
>
> Similarly, the secretary might know the secret, and could also have promised not to disclose the secret. In the case that both the secretary and Wouter know the secret, they are allowed to gossip.[2]

Is there a strategy for the secretary and Wouter that enables them to mutually establish whether they know of Geertje's pregnancy without disclosing this secret? The answer is yes. We will call protocols that solve this type of problem protocols for *knowledge authentication*: authentication based on the *knowledge* of an actor — instead of based on the identity or role of the actor.

Protocols for *knowledge authentication* can be used to grant somebody access to confidential information based on his or her knowledge, instead of (only)

[1] The baby was born on May 24, 2004. Her name is Marloes and she is really cute.

[2] Of course, they should make sure not to be overheard, and the coffee corner is probably not the best location for not being overheard.

based on his or her identity and role.[3] In the above example, successful authentication grants access to quotes and sentences which should be described as 'gossip'.

In this chapter, we will precisely define the the fundamental problem that is exemplified in the above story. We will survey and categorize what solutions for this problem exist in the literature. In Chapters 9 and 10, we will present our solutions to this problem, which are more general and more efficient than all existing solutions.

8.1 Application Areas of Gossip

The desire to gossip may be an interesting occasion to devise protocols and touch upon fundamental research issues, but the application areas of the protocols defined in this chapter reach further than the social talk at the coffee corner. We will present two application areas where 'cautious gossip' has valuable applications.

8.1.1 Police Investigations

The situation that initiated the design of protocols for *knowledge authentication* has been arguably a more important than coffee corner gossip:

> When a research team of the police performs an investigation on some crime, they register the people who are victims, the people who are witnesses, and the people who are suspects. If two independent crimes are investigated, it may be the case that somebody is suspect of two independent criminal acts. If the research teams of the two crimes do not communicate, they can easily harm one another's research. For example, one research team may be shadowing the suspect, and the other team may want to arrest the suspect. If the research teams do not communicate, it can be expected that something will go wrong in at least one of the investigations.
>
> How can a research team know that some other research team is investigating the same person?

One solution for this police problem could be that the police has an organization-wide notice board on which all victims, witnesses and suspects are listed. The current solution in the Dutch police[4] is not very much unlike this one: a police officer can enter a name in the computer, and will get a list of research teams investigating the person.

[3] This should not be confused with protocols for *private authentication* [AF04], which are protocols where the identity of a principal is only proven to a restricted set of possible communication partners.

[4] See Section 1.5 for a detailed description of the current solution in the Dutch police.

The Dutch police has thousands of criminal investigators. All of these investigators have made an oath which morally binds them to righteous behavior. The sheer number of police investigators entails that one can be sure that at least some of them will be corrupt and malicious. Thus, an organization-wide notice board that can only be consulted by officers who have made an oath is not a good solution; just a few corrupt officers who leak the information on the notice board can be enough to help some criminals escape the fate they deserve by law.[5]

It is justified to say that the organization-wide notice board is not the source of the problem, but that the few corrupt police officers are the source of the problem. If one is interested in addressing the problem of corrupt police officers, one should of course always tackle the problem at its root: hunt down and eliminate corrupt officers. It would not be realistic to believe that hunting down corrupt officers will be a 100% effective. Therefore, some measures have to be taken to limit the negative impact that corrupt police officers can have.

A first step in limiting the negative impact that corrupt police officers can have, is to prevent 'frivolous queries', that is, to prevent queries for which there does not exist a *need to know*. Of course, the *need to know* is something which is often hard to operationalize precisely. But even if it is only operationalized in a rough, simplistic way, such that only obvious violations are detected, it already limits the impact of corrupt police officers.

A simple operationalization of *need to know* which dramatically limits the frivolous queries of the thousands of researchers: only queries are allowed on names which occur in the electronic dossier the researcher is working on.[6]

There are also police officers which operate the organization-wide notice board, and if the information is stored unencrypted on the notice board[7] these operators will still be able to perform frivolous queries at will.

As long as the software that is running the notice board needs access to unencrypted data, the operators will have a means to access the unencrypted data. However, when the software that operates the notice board does not need access to unencrypted data to match electronic dossiers, it is possible to prevent frivolous access by malicious operators. When protocols for *knowledge authentication* are applied, notice boards do not need unencrypted data.

8.1.2 The Passenger Name Record

The best known application area where protocols for *knowledge authentication* can help is the airline passenger data (the so-called *passenger name record* or PNR). In 2003, in response to the events on 9/11, the United States of America

[5] There have been major leaks in the Dutch police organization, as Dutch top-criminals like Mink Kok have possession of various classified police documents. The police has a hard time identifying the corrupt police officers (various newspaper media, 2006).

[6] Such an operationalization has to be enforced by the software which is used to manage the electronic dossier.

[7] Or equivalently: it is stored encrypted, but the operators have access to the decryption key.

mandated that all airline carriers release the PNR to the Department of Homeland Security (DHS) for all flights from, to, and over US territory:

> The DHS has a 'terrorist list' of people they do not wish close to US territory. When an airplane wants to enter US airspace and it carries one or more people who are on the 'terrorist list', it is refused access.[8] Understandably, the DHS does not want to disclose its terrorist list; such a disclosure would give an unnecessary advantage to terrorists. Al Qaeda would precisely know which of its combattants would be granted access to the US, and which not.
>
> The airline carriers, on the other hand, are not automatically inclined to release the PNR to the DHS. The information has been collected for commercial purposes, and not for security purposes. Release of the PNR would result in infringement on the privacy of innocent citizens. The European Data Protection Directive forbids the release of this information. This resulted in a circus of lawsuits and negotiations.[9]
>
> Can the airline carriers and the DHS compare their lists of passengers and suspected terrorists without mutually disclosing their lists?

This problem is more intricate than the problem of the police investigation information, because the airline carriers and the DHS are not subsidiaries of one larger organization. As such, it will be hard — if not impossible — to find a *trusted third party* (TTP) to compare their lists. Thus, a 'notice board solution' as with the police investigation information is impossible.

Using protocols for *knowledge authentication*, it is possible to determine the intersection of two lists, without disclosing the lists themselves[10], without the need for a TTP.

When we look at the *need to know* of the DHS, we can observe that it is in fact very limited. What the DHS needs to know is *whether* there is a suspected terrorist on board of an airplane. Strictly taken, the DHS does not even need to know *which* terrorist is on board. Importantly, the DHS does *not* need to know the identities of the passengers that are *not* suspected terrorists.

Thus, protocols for *knowledge authentication* can protect the privacy of innocent citizens who fly from, to, or over the US, while the DHS can still perform its task.[11]

---

[8] Remarkably, when a airplane is actually refused access, there is no procedure for a concerted effort to arrest the suspected terrorist. After the airplane lands outside of the US, the suspected terrorist is free to travel elsewhere.

[9] For an extensive treatment of how the European Union and the US settled their dispute on the PNR, consult [Hei05].

[10] Thus, the items which are on both lists are mututally disclosed, but the items which are only on one of the lists are kept secret.

[11] Using protocols for *knowledge authentication* the identity of suspected terrorists which are on board of an airplane *is* disclosed to the airline carrier, and in this sense the terrorist list is not kept secret. In the current situation in which the full passenger list is disclosed to the DHS, the

8.2 Comparing Information Without Leaking It and Reference

Protocols for *knowledge authentication* are for comparing secrets, without disclosing the secrets. We need to be more precise on what we consider to be 'secret', and what we mean by 'comparing without disclosing'. We will make this more precise in this section.

What constitutes a 'secret' is relatively simple. A secret of player Q is a bit string, generated by player Q, which player Q is not willing to disclose to others. Whether the bit string can be generated by virtually every other agent does not alter it being a secret of agent Q. Thus, Q may consider 'Stalin sent millions of people to Siberia' to be a secret, while in fact many people know this. Moreover, whether the bit string corresponds to something which is true in the 'outside world' is irrelevant: for example, someone may 'know' (e.g. *believe*) the secret 'there are weapons of mass destruction in Iraq'.

The careful reader has noted that we use the verb 'to know' in a loose way. In epistemic logic, knowledge is at least as strong as *true justified belief*.[12] When we use the verb 'to know', we technically mean 'possessing information x, which may be false'. We use 'to know' in this way because *knowledge* is an intuitive notion for the examples.

What constitutes 'comparing without disclosing' is more complicated. We will focus on *comparing information without leaking it* (CIWLI) *without reference*. What that is, and how it differs from CIWLI *with reference*, will be explaind in the remainder of this section. In zero-knowledge protocols, two players play a game in which the prover (player one) proves to the verifier (player two) that the prover has some *special knowledge*. This special knowledge could be for example knowing a Hamiltonian tour for a graph, or a password to Ali Baba's cave. The verifier (player two) does not possess the special knowledge, nor does he learn it by means of the protocol. Thus, zero-knowledge protocols are convincing but yield nothing beyond the validity of the assertion proven (in the example 'the prover knows a Hamiltonian tour') [GMR85, Gol02, BG93, BFM88].[13]

The type of knowledge that can be proven in zero-knowledge protocols is limited to knowledge within a mathematical context: the two players in a protocol know some x *a priori*, and the prover proves his knowledge of some special object y. The object x may be a public key and y the corresponding private key, or x may be a graph and y the Hamiltonian tour of it, as in the example. The required mathematical relation between x and y is, speaking loosely, that it is NP-hard to compute y from x. It might seem that the requirement of a

terrorist list is not kept secret either, as the airline carrier learns that *at least one* of the passengers on board is on the list upon being refused access to US airspace. In practice, the DHS currently discloses the identity of the suspected terrorists voluntarily to the airline carrier.

[12] Beliefs in general may be ungrounded and false. Even the definition 'true justified beliefs' has some problems [Get63].

[13] For an introduction to Zero-Knowledge protocols, consult Section 2.8.

specific mathematical relation between x and y somehow restricts the possible applications of zero-knowledge protocols.

However, it is also possible to create an NP-hard 'puzzle' on the fly to prove knowledge of any y, *provided that the verifier also knows y a priori*. If the verifier does not know y a priori, he does not gain any information which helps him to compute y. In this thesis we present the first efficient zero-knowledge protocols in which possession of *any* kind of knowledge can be proven. The knowledge need not be special in any mathematical or contextual way.[14] The assertion 'the prover knows y' can only be verified if the verifier also knows (all of) y. The verifier never learns anything more than the prover's knowledge of y, and not y itself.

This type of protocols has applications where securely comparing secrets allows transactions which could not be allowed otherwise. Examples are the comparison of police information (Section 8.1.1) and the exchange of the PNR (Section 8.1.2)

For example, secret agents might like to test each other's knowledge without exposing their own. Many examples can be found where privacy requirements or non-disclosure requirements are an obstruction for performing righteous tasks.

The type of problem that our protocols solve is similar to, but different from, the problem described in [FNW96]. We will first give a description which is broad enough to cover both problems, after which we will describe the difference.

By a secret, we mean information possessed by an agent, which the agent is not willing to share with another agent. Whether other agents indeed possess this information as well is not relevant for it being considered a secret. Here follows the problem "Comparing Information Without Leaking It" (CIWLI)[15]:

> Two players want to test whether their respective secrets are the same, but they do not want the other player to learn the secret in case the secrets do not match.

Not specified yet is *which* particular secrets are to be compared, and how it is *decided* which particular secrets are to be compared. Do the two players each take a specific secret into their mind which they compare? For example, is 'the person I voted for' equal to 'the person you voted for'? Or does one player take a secret 'The General will attack tomorrow at noon' and does the other player see whether he knows this specific secret as well? In the former case, the two players first have to agree upon what they want to compare. I call this CIWLI *with reference*. In the latter case, no a priori agreement is needed and I call it CIWLI *without reference*, because of its lack of an agreement which refers to a secret.

[14] The only requirement is that it can be uniquely encoded in a binary string, which can hardly be considered a limitation.

[15] This is a slight variation from [FNW96, page 78], where it reads "Ron and Moshe would like to determine whether the same person has complained to each of them, but, if there are two complainers, Ron and Moshe want to give no information to each other about their identities."

The difference between CIWLI with reference and CIWLI without reference can be illustrated with the following two secrets:

with reference *'I voted for Pim Fortuyn'*

> This could be a secret because it expresses a stance of the player, which he may want to keep secret for whatever reason. The reason could be fundamental (like 'votes should be secret') or practical (for example to prevent embarrassment, like admitting one still likes AƋBA music).

without reference *'arkjjhhg bwr ufkng'*

> This could be a secret because it might be the access code to a Swiss bank account where someone keeps his fortune.

CIWLI with reference is symmetric in the sense that both players have a specific secret in mind while performing the protocol, whereas in CIWLI without reference, only one of the players has one specific secret in mind.[16]

An example of CIWLI with reference is the Socialist Millionaires' problem, in which two players want to test their riches for equality, but do not want to disclose their riches to the other player [JY96, BST01]. Another example is that two managers each have received a complaint about a sensitive matter, know this of one another, and would like to compare whether the complainer is the same person (without contacting the complainer) [FNW96]. Solutions exist for CIWLI with reference [FNW96, BST01, JY96]. In [FNW96] a series of interesting applications is listed where protocols solving this problem could be used.

It could also be the case that it is not agreed upon between the agents what the secret is about, i.e., that the agents have no particular stance towards the secret as in CIWLI with reference. In that case, we have CIWLI without reference. For example, Alice could have a file on her hard disk, and would like to know whether Bob possesses the same file as well. Alice can not naively show the file to Bob and ask him to search for a matching file, because this will obviously result in Bob obtaining the file (though Bob could be honorable and delete it voluntarily). In cases of CIWLI with reference, it is common that *two* specific secrets are tested for equality, whereas in cases without reference, one specific secret is tested against *numerous* secrets for equality. The file-comparison problem would be a case with reference if the two players would like to know whether two *specific* files are equal. ('Are the instructions you got from Carol the same as the instructions I got from Carol?')

Though secrets $f(X, Y)$ can be computed using some very complicated protocol, what will be the input X (resp. Y) remains under the control of Alice (resp. Bob). This has been acknowledged already in [Yao82, page 162]:

[16] In the field of dymanic epistemic logic, there are riddles about card deals, such as Van Ditmarsch's Russian cards problem [vD03]. It may need notice that CIWLI problems are very different from such card deal problems. Firstly, in CIWLI the number of 'cards' is unlimited (or at least extremely high), and it is not publicly known which 'cards' exist. Secondly, in CIWLI there is no such thing as *exclusive* possession of a 'card'.

> "Since a protocol can never prohibit Alice (or Bob) from behaving as if she had a different variable value X' (or Y')[17], the most that a protocol can achieve is to make sure that this is the only cheating that Alice (or Bob) can do."

In particular, a principal can always refuse to prove possession of some item, while he or she actually possesses the item. The best a protocol can achieve, is to prevent the opposite: it ensures that a principal cannot 'prove' possession of an item he does not have.

For CIWLI with reference, this is a larger problem than for CIWLI without reference. In CIWLI with reference, there is no guarantee that the input of a player is truthful (e.g., that the player *did* vote for Pim Fortuyn, or *does* like ABBA music). In CIWLI with reference, a commitment is required of both parties that their inputs to the protocol satisfy the reference, i.e., they are truthful. (For example, in the socialist millionaires' problem this means that the inputs correspond to the wealth of the players.) In fact, these protocols can only be used to test whether the two inputs are equal, and only assuming truthfulness one can say something about, for example, the riches of the players.

In CIWLI without reference, a successfully proven secret *is* truthful, because the 'truth' that is proven is the fact that the player can *construct* the secret (e.g., the access code to the Swiss bank account). However, a player can always fake *not* possessing a certain file, while he actually *does* possess the file. A player can however never fake possessing something which he does not possess (or only with negligible probability).

In this thesis, we focus on protocols for CIWLI without reference.

8.3 Adversary Models for CIWLI

In CIWLI with reference, it is required that player A cannot infer anything on the input of player B, in case their inputs do not match. This includes that it should not be possible for player A to test the input of player B for likely values, that is to guess and verify whether the guess is correct. This is called semantic security [Yao82, Yao86][18]. Semantic security is important in CIWLI with reference, because what is tested is not whether the other player can imagine or guess some input [WSI03], but whether he actually *states* the input. Thus, cases with reference should withstand guessing attacks (also called *dictionary attacks*, see Section 2.1).

In case of CIWLI without reference, there is no need to withstand guessing attacks of the players. Basically this is because cases without reference test whether the other player possesses a specific file, which is roughly equivalent to being able to imagine or guess it within the limits of its storage capacity and computational resources. In fact, the protocols we present in the next chapters

[17] In [Yao82], it says "i'" instead of "X' (or Y')" — WT

[18] Informally, an encryption scheme is semantically secure, if ciphertexts leak no information about the plaintext.

are based on the fact that a player can verify the other player's knowledge of a file by correctly 'guessing' it. Semantic security is still required in the sense that if a player cannot guess the *complete* input of the other player, he should not be able to infer *anything* of the input of the other player. And, of course, there must be full semantic security with respect to eavesdroppers, third persons other than the two players.

Given these considerations, what adversary models can best be applied?[19] In the *honest-but-curious adversary model*, the principals are supposed to adhere to protocol specification. Thus, it is assumed that in this model, the principals do not try to convince other players of possession of items which they in fact do not possess. They are only secure under the assumption of no cheating. To let go of this assumption, one has to adopt the *malicious adversary model*.

In the rest of this chapter and thesis, we will adopt the malicious adversary model.

8.4 Possible Set Relations

Algorithms for the distributed computation of set relations are tricky. For one thing, different algorithms may seem to compute the same thing, but in fact compute something different. Thus, before listing known algorithms, which we will do in the next section, it should be explained which interesting properties can be computed given two finite sets. In this section, we will explain what set relations we distinguish. All sets we consider are finite.

It is easy to characterize the possible relations between two subsets of a given domain Ω. As a reminder, two sets (each a subset of Ω) can either be:

disjoint, there is no single item that is in both sets,

partially intersecting, at least one item is found in both sets and at least one item is found in only one of the sets, or

equal, any item found in one set is also found in the other set.

The possible relations between to sets are depicted in Figure 8.1. Note that, if $X = Y = \emptyset$, the sets are both disjoint and equal. Otherwise, the relations are mutually exclusive. For example, if one determines that A and B are not equal, one can be sure that A and B are either disjoint or partially intersecting.[20]

When describing the relation between two sets, *partially intersecting* deserves some extra attention: depending on the application domain, it may or may not be required to spell out what elements constitute the partial intersection. To determine what elements constitute the intersection requires more computation than to determine whether the intersection is empty or not. To

[19] For an introduction to adversary models, see Section 2.6.

[20] For a discussion of possible set relations when one also takes the domain of possible items (e.g. the *universe*) into account, consult [KM06].

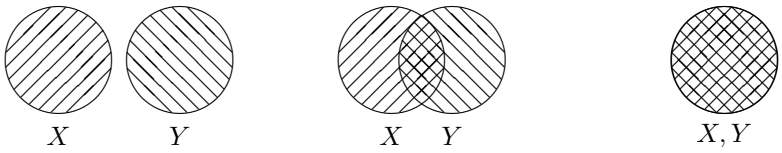


FIGURE 8.1: The relations possible between two sets X and Y.

---

disjointness Are X and Y disjoint? The answer is either a yes (1) or a no (0).

$$f_{\text{disj}}\colon \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \times \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \to \{0,1\} \quad \text{with} \quad f_{\text{disj}}(X,Y) = [X \cap Y = \emptyset]$$

intersection Which items are in both X and Y? The answer is a set.

$$f_{\text{int}}\colon \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \times \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \to \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \quad \text{with} \quad f_{\text{int}}(X,Y) = X \cap Y$$

intersection cardinality How many items are in both X and Y? The answer is a number.

$$f_{\text{ic}}\colon \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \times \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \to \mathbb{N}_0 \quad \text{with} \quad f_{\text{ic}}(X,Y) = |X \cap Y|$$

equality Are X and Y identical? The answer is either a yes (1) or a no (0).

$$f_{\text{eq}}\colon \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \times \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \to \{0,1\} \quad \text{with} \quad f_{\text{eq}}(X,Y) = [X = Y]$$

subset inclusion Is X a subset of Y? The answer is either a yes (1) or a no (0).

$$f_{\text{si}}\colon \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \times \stackrel{<\infty}{\mathcal{P}}(\{0,1\}^*) \to \{0,1\} \quad \text{with} \quad f_{\text{si}}(X,Y) = [X \subseteq Y]$$

This is the only relation that does not commute: $f_{\text{si}}(X,Y) \not\Leftrightarrow f_{\text{si}}(Y,X)$.

FIGURE 8.2: Some interesting set functions for which secure protocols exist.

- Items of sets (the 'secrets') are represented as bit strings.
- The set of all possible finite bit strings is represented as $\{0,1\}^*$.
- The *finite power set* (the set of all finite possible subsets) is denoted as $\stackrel{<\infty}{\mathcal{P}}$.
- The set of non-negative integers is denoted as $\mathbb{N}_0$.
- The cardinality of a set X is denoted as $|X|$.
- The extension of a set X is denoted as $[X]$.
- If X is a condition, then $[X] = 1$ if X holds, and $[X] = 0$ otherwise.

0-to-any At least one set is empty, say $X = \emptyset$. This case is not really interesting. In this case, for any Y, we have

$$f_{\text{disj}}(X,Y) = 1, \quad f_{\text{int}}(X,Y) = \emptyset, \quad f_{\text{ic}}(X,Y) = 0, \quad f_{\text{si}}(X,Y) = 1$$
$$\text{and} \quad f_{\text{eq}}(X,Y) = [Y = \emptyset]$$

1-to-1 Both sets contain only one element. In this case the sets are either disjoint, or equal. In this case, we have

$$f_{\text{disj}}(X,Y) = 0 \quad \Leftrightarrow \quad f_{\text{int}}(X,Y) = X \quad \Leftrightarrow \quad f_{\text{int}}(X,Y) = Y$$
$$\Leftrightarrow \quad f_{\text{ic}}(X,Y) = 1 \quad \Leftrightarrow \quad f_{\text{eq}}(X,Y) = 1 \quad \Leftrightarrow \quad f_{\text{si}}(X,Y) = 1$$

1-to-many One of the sets contains only one element, say $X = \{M\}$, and the other set Y contains more than one element. In this case Y can be disjoint with X, or partially intersecting. In this case, we have

$$f_{\text{disj}}(X,Y) = 0 \quad \Leftrightarrow \quad f_{\text{int}}(X,Y) = X$$
$$\Leftrightarrow \quad f_{\text{ic}}(X,Y) = 1 \quad \Leftrightarrow \quad f_{\text{si}}(X,Y) = 1$$

many-to-many Both sets contain more than one element. This case is not really special in the sense that there are not too many correspondences between the set relations. In this case, we only have

$$f_{\text{disj}}(X,Y) = 0 \quad \Leftrightarrow \quad f_{\text{int}}(X,Y) \neq \emptyset \quad \Leftrightarrow \quad f_{\text{ic}}(X,Y) > 0$$

FIGURE 8.3: Special cases of the sizes of two sets.

encode what elements constitute the intersection also requires more bits than to encode whether the intersection is empty or not.

Now, if one is given two finite sets X and Y, what kind of questions could one be interested to ask? In Figure 8.2, we list the most useful questions for which algorithms have been designed, with their formal definition. Without loss of generality, we assume that an item is modeled as a bit string in $\{0,1\}^*$, which contains all finite strings[21], including the empty string.

In this thesis, we will focus mainly on the *intersection* question. For more details on *intersection cardinality*, consult [FNP04, KS04]. For more details on *subset inclusion*, consult [LLM05, KM06]. In [KS04], protocols are described which determine set relation properties which are only relevant if more than two sets are involved. For the ease of comparison and discussion, we will only address the case where the number of sets to be compared is two.[22]

[21] Thus, the interpretation of $\{0,1\}^*$ is that the pattern $\{0,1\}$ may be repeated any finite number of times, but not infinitely many times.

[22] For example, in [KS04] *threshold intersection cardinality* is defined as the function that determines the set of items that each occur in more than n of the m sets, with $n > 2$ and $m > 2$.

Depending on the sizes of the sets X and Y, four cases can be distinguished. For some of these cases, the definitions of some of the questions above can be simplified, and can become the dual of one of the other definitions. Also, algorithms could be tailor-made for one of these special cases. The first case, in which one set is empty, is only included for completeness. All four cases are shown in Figure 8.3.

Clearly, the many-to-many case is the most general one. The most specific but still interesting case is 1-to-1. For the 1-to-1 case, the questions that can be asked are interchangeable.

An algorithm for the many-to-many case can be constructed by multiple runs of a 1-to-many algorithm, and an algorithm for the 1-to-many case can be constructed by multiple runs of a 1-to-1 algorithm. Usually, these are not the most efficient solutions. In Chapter 10 we show how we efficiently transform a protocol for the 1-to-many case into a protocol for the many-to-many case.

The many-to-many case is also called the *list intersection problem* [NP99]. A solution for the many-to-many case will make it possible to create indexes on distributed, secured databases, which can be searched without leaking information on the contents of the databases.[23]

8.5 Secure Protocols for Computing Set Relations

It is not very difficult to create an algorithm that computes the answer to any single question of Figure 8.2. Also, if X is only known to one principal (Alice), and Y is only known to the other principal (Bob), protocols answering these questions are easy to construct. But it is diffucult to construct protocols which compute the answers to these questions in a secure manner. In this section, we will give an overview of existing protocols for secure computation of the set relations. First, we will explain and discuss the properties that these protocols satisfy. Then, Table 8.1 lists all known, published protocols.

The protocols listed in Table 8.1 satisfy the following properties:[24]

privacy the inputs X and Y are not mutually disclosed;

validity it is impossible for the principals to cheat without the other player detecting this;

autarky the principals do not need assistance of a third party; and

efficiency for some set sizes $|X|$ and $|Y|$ the solution is efficient.[25]

[23] This is similar to, but with wider application areas than the approaches in [FLW91, FGR92].

[24] The properties privacy, validity and autarky listed here are described in more detail in Section 2.7.

[25] It would be better if all protocols are efficient for all set sizes $|X|$ and $|Y|$, but this is not the case. For some set sizes, some protocols are not even feasible. For an introduction to complexity, consult Section 2.3.

The properties *privacy* and *validity* are not easily combined with the properties *autarky* and *efficiency*. Observe that if autarky and efficiency would be dropped as requirements, there would be a trivial protocol involving a *trusted third party* (a TTP): both Alice and Bob privately disclose their inputs X and Y to the TTP, the TTP computes $f(X, Y)$, and informs Alice and Bob.

The *privacy* property is tricky in particular with respect to the *set sizes* of the inputs (i.e., $|X|$ and $|Y|$). It is easy to see that if both X and Y contain many elements, the amount of computation and comparisons required is much larger than if X and Y contain only a few elements. If it is required that $|X|$ and $|Y|$ are not disclosed, it cannot be allowed to design the protocol in such a way that it will take advantage of $|X|$ and $|Y|$ to optimize the communication and computational complexity: a protocol run would leak $|X|$ and $|Y|$ or at least some stochastic information on $|X|$ and $|Y|$. A protocol that does not disclose $|X|$ and $|Y|$ is essentially a protocol that adds dummy elements to X and Y (obtaining X' and Y') such that $|X'| = c$ and $|Y'| = c$ for some commonly agreed upper bound c. Such a protocol only works for sets with at most c elements, and 'discloses' that $|X| \leq c$ and that $|Y| \leq c$. Using such a protocol, with $c = 4{\cdot}10^6$, comparing two sets each of size 4 is just as expensive as two sets each of size 4 million. Thus, nondisclosure of $|X|$ and $|Y|$ implies a tremendous efficiency penalty.[26]

Protocols satisfying the properties listed in the beginning of this section are closely related to *secure multiparty computation* (SMC). In addition, a protocol for SMC also has to satisfy the *fairness* property. That is, both principals learn the outcome of $f(X, Y)$, and they cannot deny the other the outcome of $f(X, Y)$ without denying it oneself. The BST protocol (listed in Table 8.1) can be transformed into a fair protocol [BST01].

There is also a close relation to *zero-knowledge proofs*. If either the set sizes are a priori known to both participants, or the protocols do not leak the set sizes, the protocols can be considered zero-knowledge proofs.

Normally, in zero-knowledge proofs, it is tacitly assumed that the prover is convinced of the truth of the assertion he tries to prove. There is conceptually nothing wrong with trying to prove an assertion of which one does not know the truth value. Proving that 'I know what you know' is a very good example of this: the prover may not know the knowledge of the opponent, but can nevertheless engage in a protocol. Whether the run of this protocol yields a convincing proof depends on the knowledge of the verifier. Moreover, it is not necessary that the knowledge of the verifier is known to the prover, of course. Also, after running the protocol, the prover does not know whether the assertion proven was in fact true, or whether the proof was convincing. In [JY96], Jakobsson and Yung have introduced the concept of an *oblivious prover* which applies well to this situation. An *oblivious prover* is a prover who does not know the truth value of the assertion he tries to prove.

Now that the most important properties have been described, we are ready

[26] This has also been observed in [KM05]. Moreover, [BCC88, page 157] contains a hint to this matter: "However, Vic is not given a clue [...] (except perhaps for an upper limit on its length)."

name	reference	case	computes	adversary model	communication complexity	domain compression
FNW[27]	[FNW96]	1-to-1	equality[28]	various	various	no
JY	[JY96]	1-to-1	equality[28]	malicious	$\ln \lvert\Omega\rvert$	no
BST	[BST01]	1-to-1	equality[28]	malicious	$\ln \lvert\Omega\rvert$	no
NP-1	[NP99]	1-to-many	intersection[29]	malicious	$\lvert Y\rvert \cdot \ln \lvert\Omega\rvert$	no
T-1	Chapter 9	1-to-many	intersection[29]	malicious	1	yes
KM-1[30]	[KM05]	any	disjointness	honest-but-curious[31]	$\lvert\Omega\rvert \cdot \ln \lvert\Omega\rvert$	no
KM-2[30]	[KM05][32]	any	disjointness	honest-but-curious[31]	$\lvert X\rvert \cdot \lvert Y\rvert \cdot \ln \lvert\Omega\rvert$	no
KM-3[30]	[KM05][32]	any	disjointness	honest-but-curious[31]	$\binom{\lvert X\rvert+\lvert Y\rvert}{\lvert Y\rvert} \cdot \ln \lvert\Omega\rvert$	no
KM-4	[KM06]	any	disjointness	honest-but-curious[31]	$(\lvert\Omega\rvert - \lvert X\rvert) \cdot \ln \lvert\Omega\rvert$	no
NP-2	[NP99]	any	intersection	malicious	$(\lvert X\rvert + \lvert Y\rvert) \cdot \ln \lvert\Omega\rvert$	no
AgES-1[33]	[AES03]	any	intersection	honest-but-curious	$\lvert X\rvert + \lvert Y\rvert$	yes
FNP-1	[FNP04]	any	intersection	honest-but-curious	$\lvert X\rvert \cdot \ln \lvert\Omega\rvert$	no
FNP-2	[FNP04]	any	intersection	malicious	unclear	no
KS-1	[KS04]	any	intersection	honest-but-curious	$\lvert X\rvert \cdot \ln \lvert\Omega\rvert$	no
KS-2	[KS04]	any	intersection	malicious	$\lvert X\rvert^2 \cdot \ln \lvert\Omega\rvert$	no
T-2	Chapter 10	any	intersection	malicious	$\lvert X\rvert + \lvert Y\rvert$	yes
AgES-2[33]	[AES03]	any	intersection cardinality	honest-but-curious	$\lvert X\rvert + \lvert Y\rvert$	yes
FNP-3	[FNP04]	any	intersection cardinality	honest-but-curious	$\lvert X\rvert \cdot \ln \lvert\Omega\rvert$	no
KS-3	[KS04]	any	intersection cardinality	malicious	$\lvert X\rvert \cdot \ln \lvert\Omega\rvert$	no
KM-5	[KM06]	any	subset inclusion	honest-but-curious[31]	$\lvert X\rvert \cdot \lvert Y\rvert \cdot \ln \lvert\Omega\rvert$	no
KM-6	[KM06]	any	subset inclusion	honest-but-curious[31]	$(\lvert\Omega\rvert - \lvert X\rvert) \cdot \ln \lvert\Omega\rvert$	no
LLM	[LLM05]	any	subset inclusion	malicious	$\lvert\Omega\rvert \cdot \ln \lvert\Omega\rvert$	no

TABLE 8.1: All known well-documented secure protocols for computing the set relations given in Figure 8.2 (page 110).

to present the full list of protocols which securely compute set relations. It is given in Table 8.1. The following remarks may help to read the table:

- The protocols have been named after the authors of the papers in which they have been presented. Where necessary, protocols have been numbered to distinguish different protocols from the same set of authors.
- The column *case* gives the set sizes for which the protocol has been designed (see Figure 8.3). 'Any' implies any combination of set sizes, and therefore includes the many-to-many case.
- Because the properties *privacy* and *validity* can only be assessed with respect to some chosen adversary model, the adversary model is explicitly stated for each protocol.
- In the columns describing the complexity, X and Y denote the sets of Alice and Bob. The domain of all possible set items (the 'universe') is denoted Ω. The cardinality of a set Z is denoted as $|Z|$. The logarithm base 2 is denoted $\ln$. For many protocols, the communication complexity also depends on a constant factor k, a security parameter k. It has been omitted for ease of reading. Other constant factors have also been omitted.
- The column *domain compression* states whether in the protocol the items of the sets are abbreviated or not. This will be addressed further in Section 8.6.

As can be seen in Table 8.1, a lot of research has recently gone into the subject matter of secure protocols for computing set relations. Omitted from the table are protocols for which it is unclear what their security properties are [DQB95, QBA+98a, QBA+98b, QAD00, Ber04, CC04][34]. Also omitted from the list are 'protocols' which for their security actually rely on another protocol for computing set relations, such as [RCF04]. Another interesting algorithm is the set comparison algorithm of Wegman and Carter [WC81], which can easily

[27] FNW is only listed for completeness. In [FNW96], a total of thirteen protocols is described, some of which are only secure in the honest adversary model. Some of the protocols require physical presence or special purpose devices. None of the protocols simultaneously satisfies the privacy, validity and autarky properties. It is a valuable and enjoyable read nevertheless.

[28] In the 1-to-1 case, all interesting set relations are equivalent (see Figure 8.3). In our opinion, *equality* is the term which fits best here.

[29] In the 1-to-many case, all interesting set relations but equality are equivalent (see Figure 8.3). In our opinion, *intersection* is the term which fits best here.

[30] The protocols KM-1 through KM-3 are called PIPE #1 through PIPE #3 in [KM05].

[32] The protocols KM-2 and KM-3 are also documented in [KM06].

[31] In [KM05, KM06], it is claimed that protocols KM-1 through KM-6 can be transformed into protocols which are secure in the malicious adversary model. The proofs of these claims have not been published, nor were they available upon request.

[33] To avoid confusion with the encryption standard AES, the protocols from [AES03] have been named AgES-n. The AgES-n protocols have been somewhat modified and extended in [OYGB04], where essentially a trusted third party (TTP) is introduced.

[34] For a discussion of these articles, see Appendix C.1

name	reference	case	communication complexity	domain compression
JY	[JY96]	1-to-1	$\ln \lvert\Omega\rvert$	no
BST	[BST01]	1-to-1	$\ln \lvert\Omega\rvert$	no
NP-1	[NP99]	1-to-many	$\lvert Y\rvert \cdot \ln \lvert\Omega\rvert$	no
KS-3	[KS04]	1-to-many	$\ln \lvert\Omega\rvert$	no
T-1	Chapter 9	1-to-many	1	yes
NP-2	[NP99]	any	$(\lvert X\rvert + \lvert Y\rvert) \cdot \ln \lvert\Omega\rvert$	no
FNP-2	[FNP04]	any	unclear	no
KS-2	[KS04]	any	$\lvert X\rvert^2 \cdot \ln \lvert\Omega\rvert$	no
T-2	Chapter 10	any	$\lvert X\rvert + \lvert Y\rvert$	yes
LLM	[LLM05]	any	$\lvert\Omega\rvert \cdot \ln \lvert\Omega\rvert$	no

TABLE 8.2: Protocols which can be used for knowledge authentication This Table is a strict subset of Table 8.1, except for the KS-3 protocol, which has been restricted to the 1-to-many case (in which case *intersection cardinality* is equivalent to *subset inclusion*, see Figure 8.3).

be transformed into a protocol in the many-to-many case for computing set equality with communication complexity 1 (!). This algorithm does not belong in the table because it assumes honest principals.

All protocols which are suitable for the 'any' case leak some information on the sizes of the sets. For the 1-to-many case, the NP-1 protocol leaks the size of the bigger set, while our T-1 protocol does not. For the 1-to-1 case, the mere running of the protocol leaks that the set sizes are both equal to one, but this is rather dull information and can hardly be considered 'leaking information'.

A large number of the protocols in Table 8.1 is proven secure in the *honest-but-curious* adversary model (see Section 2.6). It is important to appreciate what this precisely means. In the honest-but-curious adversary model, the principals are supposed to adhere to protocol specification. Thus, it is assumed that in this model, the principals do not try to convince other players of possession of items which they in fact do not possess. Therefore, protocols which compute the intersection (subset inclusion, equality) problem should definitely not be considered as protocols which prove possession of set items. They are only secure under the assumption of no cheating (as discussed also in Section 8.3).

Protocols for computing intersection (subset inclusion, equality) which are secure in the *malicious* adversary model *can* be considered a proof of possession. These protocols and their main properties are summarized in Table 8.2.

8.6 Domain Compression

As can be seen in Tables 8.1 and 8.2, the communication complexities of the various protocols differ significantly. It is striking that almost all protocols have

a factor $\ln|\Omega|$ in the complexity. The reason for this factor is rather simple: these protocols communicate the set items in encrypted form. If the set of items is restricted to things like secret keys, which are typically a few thousand bits long (say, 4096 bits), the domain size is $|\Omega| = 2^{4096}$. In that case $\ln|\Omega| = 4096$ bits, which is half a kilobyte[35]. With the current cost of computing power and communication bandwidth, this is by no means a prohibitive figure.

If on the other hand the set of items is not restricted to secret keys, but is extended to include binary files (say, up to sixteen megabyte), the domain size is $|\Omega| = 2^{2^{32}}$, and $\ln|\Omega| = 2^{32}$, which is sixteen megabyte. If 2^{32} is *only a factor* in the communication complexity, the feasibility of such a protocol for a domain of such size is questionable at least.

To communicate sixteen megabyte of information only to identify a single file may seem absurd, but it is necessary if communicating less information would lead to unacceptably many identification errors. When we have a set $\Phi \subsetneq \Omega$, a constant c, $|\Phi| < 2^c|\Omega|$, and Φ has a uniform distribution (for any subset of Ω), one can uniquely identify an element $x \in \Phi$ with an error probability ε of $\varepsilon = 2^{-c}$ by communicating only $c + \ln|\Phi|$ bits.

Thus, the domain size $|\Omega|$ does not impose a lower bound on the communication complexity, but $|\Phi|$, the size of the of the domain that is actually used. To obtain a lower communication complexity, we have to *compress* the *domain* Ω onto the smaller domain Φ, hence *domain compression*[36]. We need a function which provides the mapping $H\colon \Omega \to \Phi$, where the output has a uniform distribution.

A protocol that uses domain compression does not operate directly on two sets X and Y, but on two sets X' and Y' which are constructed by application of the mapping H to the sets. (Thus $X' = \{H(x)|x \in X\}$ and $Y' = \{H(y)|y \in Y\}$.)

The logical choice for a function that provides the compression, is a cryptographic hash function.[37] The output of a cryptographic hash function is indistinguishable from a uniform distribution. The output of a cryptographic hash function has a fixed length l, which is typically a few hundred for current hash functions[38], yielding a domain Φ with $|\Phi| = 2^l$. Such domain sizes for Φ are sufficiently large.

All in all, the domain Ω is compressed to a smaller domain Φ. When this is done using a cryptographic hash function, the hash values ($\in \Phi$) can be considered 'abbreviations' of the original set items ($\in \Omega$).

When it comes to computing set relations, where the items of the sets stem from a large or huge domain Ω, one can observe that for fixed sets of secrets X and Y, the larger a domain Ω, the sparser the sets will be. More specifically,

[35] Where we write ln, this is a shorthand for for $\log_2$, the logarithm with base 2.

[36] Though tempting, we refrain from using the term 'compression function', as this term also has a specific meaning in the Merkle-Damgård paradigm, which applies to cryptographic hash functions.

[37] For an extensive introduction to cryptographic hash functions, see Chapter 3.

[38] The hash function with the longest hash values known to date is SHA-512, which produces hash values of 512 bits long.

one can observe that

$$\lim_{|\Omega|\to\infty} \frac{|X| \cdot |Y|}{|\Omega|} = 0$$

which makes it abundantly clear that exploitation in the protocols of the sparsity of the sets will improve the communication complexity. The protocols which do exploit this sparsity have a 'yes' in the column 'domain compression' of Table 8.1 (they are: T-1, T-2, AgES-1 and AgES-2).[39]

The instrument that these protocols use in order to exploit the sparsity is a cryptographic hash function. Instead of communicating encrypted forms of the set items, the encrypted form of the hash value of the set items is communicated.

The use of domain compression has subtle but important implications for the *privacy* property of a protocol. First, let us repeat the definition of this property from Section 2.7. Suppose there are two principals, Alice who possesses X and Bob who possesses Y.

privacy The inputs X and Y are not mutually disclosed: Alice does not learn anything about Y except $f(X, Y)$, and Bob does not learn anything about X except $f(X, Y)$.

Domain compression implies that the privacy property of a protocol is *relaxed*. It remains the case that X and Y are not 'mutually disclosed', but in a weaker sense:

privacy′ The inputs X and Y are not mutually disclosed: for every item $y \in Y$ it holds that if Alice does not know y before the protocol, she does not know y after the protocol. Similarly for Bob: for every item $x \in X$ it holds that if Bob does not know x before the protocol, he does not know x after the protocol.

In the latter definition (of *privacy′*), it is not considered a violation of privacy if a principal can successfully mount a *dictionary attack* on the set of the other principal. In the former definition (of *privacy*), a successful dictionary attack is considered a problem.

Both definitions speak of 'nondisclosure', but this non-disclosure does not apply to exactly the same concepts. The difference between *privacy* and *privacy′* can also be explained using the distinction between the following two concepts:

the item itself the bit string which may be an element of X and/or Y

knowledge of the item the fact whether the item itself is part of the knowledge of Alice and/or Bob.

[39] In the context of sparse sets, it is appropriate to clarify an often misinterpreted result by Kalyanasundaram, Schnitger and Razborov [KS92, Raz92]. As it would distract too much from the 'story line' of this chapter, it is clarified in Appendix C.2.

In *privacy*, nondisclosure applies to both the item itself and the knowledge of the item. In *privacy'*, nondisclosure applies only to the item itself, and not to the knowledge thereof. The fact that in the definition of *privacy'*, nondisclosure does not apply to the knowledge of the item, does not mean that one can assume knowledge of the item is always disclosed. It means merely that in *privacy'*, knowledge of an item *may* be disclosed.

8.7 Conclusion

Gossiping without disclosing secrets has application in small social settings, but more importantly also in the comparison of police information (Section 8.1.1) and the exchange of the airline passenger data (Section 8.1.2).

Protocols for *knowledge authentication* are protocols that allow a player to prove possession of a secret to someone who also knows the secret, without disclosing the secret itself.

There are a number of variations of the problem, depending on the following properties:

- Does 'secret' mean that it is difficult to guess the string that represents secret (as with the string 'arkjjhhg bwr ufkng'), or does it mean that the player has attributed some stance to a commonly known string? (as with 'I voted for Pim Fortuyn'). We call the former *CIWLI without reference*, and the latter *CIWLI with reference* (see Section 8.2).

- How untrustful and untrustworthy are the players? (i.e., what *adversary model* is appropriate? see Section 8.3.)

- How many secrets need to be compared? Just one secret against one other secret (*1-to-1*), one secret against many other secrets (*1-to-many*), or many secrets against many secrets (*many-to-many*)? (see Section 8.4.)

- How many *possible secrets* exist? (i.e., what is the domain size $|\Omega|$?)
  How many *actual secrets* may exist? (i.e., what is the domain size $|\Phi|$?)
  (see Section 8.6)

There are many protocols which compute set relations in some secure manner (see Section 8.5, Table 8.1), but only a few of these protocols are suitable for *knowledge authentication* (see Table 8.2).

In the next two chapters, we will present our T-1 protocol for the 1-to-many case (Chapter 9) and our T-2 protocol for the many-to-many case (Chapter 10) . These are protocols secure in the malicious adversary model, for CIWLI without reference. These protocols use cryptographic hash functions in order to optimize the communication complexity.

Part IV

Protocols

The T-1 protocol, our solution for 1-to-many knowledge authentication, is presented. It uses cryptographic hash functions to 'point at' secrets, and to prove possession of secrets. The T-1 protocol is proven secure in our extended version of GNY logic.

Chapter 9

1-to-many Protocols (T-1)

In the previous chapter, the problem of 'comparing information without leaking it' has been extensively explained. In this chapter, we will present our T-1 protocol[1], which is the most efficient solution for the 1-to-many case. That is the case where one principal has only one secret in mind, and the other player any number of secrets. In the latter part of this chapter, we will prove the T-1 protocol correct using an extended version of GNY logic. The aims and requirements of the T-1 protocol can be illustrated with the following story:

> Victor is a secret agent, and keeping secret his intelligence has a high priority. However, his mission is to protect Peggy from great dangers, so when needed, protecting Peggy takes priority over keeping his information secret. Now he is confronted with the following situation: Victor does not know whether certain information I known to him, is also known to Peggy. ('Peggy is kindly invited for a dinner at the Mallory's place.')[2] Victor knows that Mallory is a very malicious person. If Peggy does know that she is kindly invited, Victor would like to send her a warning message ('Don't go there, it is a trap. You will get killed in case you go there.'). However, if Peggy has somehow not received the invitation I, Victor would like to keep his warning for himself, as well as his knowledge of Peggy's invitation. Therefore, Victor asks Peggy

[1] The name of the protocol stems from naming convention in Table 8.1: the author of the protocol (Teepe) and a number to distinguish it from other protocols by the same author.

[2] For clarity, this information could be possession of a computer file stating the invitation. This sets apart the matter whether the information is truthful.

> to prove her knowledge of the invitation. Only after the proof, Victor will disclose his warning to Peggy. In the protocol, Peggy does not learn whether Victor actually knew about the invitation, other than from his possible next actions, such as sending a warning.
>
> Peggy is willing to prove her knowledge of the invitation I, but only if she can make sure that Victor does not cheat on her, by actually finding out about the invitation because he tricks her into telling him that she has been invited. That is, she only wants to prove her knowledge of the invitation if Victor actually knew about the invitation beforehand.

Actually, this description only describes the first one of three possible configurations of the T-1 protocol:

1. The verifier initiates ('can you prove to me that you know I?')
2. The prover initiates ('I'll show you that I know I!')
3. Mutual proof: both players simultaneously prove to one another that they possess I.

A situation where such *mutual* verification could be used in real life is 'cautious gossip', such as gossiping about the Geertje's pregnancy (explained in the opening of the previous chapter).

In this chapter we will mainly focus on configuration 1, though we stress that the proof for configuration 1 can easily be modified to prove the protocols for the other cases.

From here on we will call pieces of information 'information blocks', or IBs for short. Here follows a somewhat more formal description of the story:

> Peggy has a certain IB I. If and only if Victor also possesses this IB I, she wants to prove her possession of it to Victor upon his request. Furthermore, Peggy need not know whether Victor indeed possesses IB I, in order to execute the protocol safely.

Thus, if Victor has the same IB, he can verify that Peggy indeed has it, but if Victor does not have the same IB, he does not learn anything.

9.1 Prerequisites

The T-1 family of protocols relies on some assumptions and uses some cryptographic primitives. Furthermore, we use some terminology in the protocol and its analysis. These prerequisites will be explained in this section.

The basic assumptions are that the communication channel cannot be modified by an adversary, and that it is authenticated. That is, the principals in the protocol know with whom they are communicating. Their communication

message	meaning
`ask`(h_1)	A principal asks the other player to look whether he knows a file which has the hash value h_1.
`halt`	A principal stops proving and/or verifying possession.
`challenge`(C)	A principal asks the other player to prove possession of a file, using the challenge C.
`prove`(h_2)	A principal proves possession of a file, by presenting the hash value h_2.

TABLE 9.1: Basic messages used in the T-1 protocol.

may be overheard, but not modified. To obtain such a communication channel, standard cryptographic techniques can be used, such as asymmetric cryptography.

The most important cryptographic primitive used is the non-incremental cryptographic hash function $H(\cdot)$, which has been explained extensively in Chapter 3. This function $H(\cdot)$ is publicly known and available to all protocol particupants and malicious parties alike. The most important properties of a non-incremental cryptographic hash function $H(\cdot)$ are:

- that it is easy to compute;
- that its inverse is not easy to compute: given $H(I)$, it is infeasible to infer any property of I; and
- that it is impossible to compute $H(I)$ from other inputs than I itself (I has to be present to compute $H(I)$).

In particular, for the non-incremental cryptographic hash function the *random oracle* model is used (see Section 3.3), which is roughly the assumption that the output of $H(\cdot)$ is indistinguishable from random noise.

This primitive is sufficient for the protocols, but it is possible to improve the computational complexity of the protocol if also some form of encryption is used (see Section 9.3). For our purposes, it does not really matter whether the encryption is symmetric or asymmetric, but where we are required to choose, we will choose asymmetric encryption. (Remember that it is not unlikely that asymmetric cryptography is used already to maintain the authenticity of the communication channel.)

In the protocols of the T-1 family, a number of basic messages is used. These basic messages are listed in Table 9.1. In these basic messages, the values h_1, h_2 and C occur. The first two are hash values, the latter is a piece of information with the sole purpose to be unpredictable to anybody but the sender of the message.

As explained in the introduction of this chapter, three configurations for the protocol exist. Two configurations are asymmetric in the sense that one

principal is only a prover, and the other principal is only a verifier.[3] In these configurations, we will refer to the principals with the names Peggy (P) for the *P*rover and Victor (V) for the *V*erifier. In the third configuration, both principals take both roles. In that configuration, we refer to the principals with the names Alice (A) and Bob (B). When we refer to a principal which could be any of the principals P, V, A or B, we use the name Q.

The abbreviations P, V, A and B are unique representations of the respective identities, such as a full name and birth date, or something like a passport number. An *information block* (IB) can be represented as a unique bit string I. The collection of IBs that a principal Q possesses is denoted KB_Q (thus, KB_P for Peggy, and so on).

In the 'simple' versions of the T-1 protocol, those that do not depend upon encryption, it is assumed that the two principals have agreed upon a commonly known secret nonce N beforehand. Here, a nonce is a piece of information with the sole purpose to be unpredictable to anybody but the participating principals. The nonce N functions as a secret key shared between the participating principals.[4] The set $I_Q\star$ is the set of IBs I in possession of principal Q, for which $H(I_Q, N)$ is equal to h_1. Thus, $I_Q\star = \{I_Q \in KB_Q | H(I_Q, N) = h_1\}$.

In the 'elaborate' versions of the T-1 protocol, those that do depend upon encryption, it is assumed that the principals have set up encryption keys in such a way that the initiator of the protocol can send messages in such a way that only the not-initiating (but participating) principal can read these messages. The opposite is not required: the non-initiating principal need not be capable of sending encrypted messages to the initiating principal. The set $I_Q\star$ is the set of IBs I in possession of principal Q, for which $H(I_Q)$ is equal to h_1. Thus, $I_Q\star = \{I_Q \in KB_Q | H(I_Q) = h_1\}$.

Now that the basic prerequisites are explained, we can introduce the 'simple' versions of the T-1 protocol in which no encryption is used.

9.2 Protocol Description (Simple, no Encryption)

There are three configurations of the T-1 protocol, as mentioned in the introduction of this chapter. Of these three configurations we will first show the 'simple' version, that is the version that does not depend upon encryption. These protocols are shown in Figures 9.1 (the verifier initiates), 9.2 (the prover initiates), and 9.3 (mutual proof).

A crucial step in the protocol is the computation of the set $I_Q\star$. By computing this set, a principal 'interprets' the other principal's $\texttt{ask}(h_1)$ message. The set $I_Q\star$ is the set of principal Q's IBs that match h_1. If a set $I_Q\star$ is empty,

[3] The difference between these two configurations is whether it is the prover or the verifier that initiates the protocol.

[4] Note that N does not function like the key in a keyed cryptographic hash. A keyed cryptographic hash offers hardly any guarantees in case the key is compromised. See Section 3.2 for more details.

1. Victor chooses an IB $I_V \in KB_V$ of which he wants to test Peggy's knowledge; Victor computes $h_1 = H(I_V, N)$; Victor computes $I_V\star \subseteq KB_V$; Victor generates a random challenge C
2. Victor sends Peggy the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C)$
3. Peggy computes $I_P\star \subseteq KB_P$
4. For each $I_{P_i} \in I_P\star$ of which Peggy is willing to prove her knowledge to Victor, the following happens:
   (a) Peggy computes $h_{2_i} = H(I_{P_i}, N, P, C)$
   (b) Peggy sends Victor the message $\texttt{prove}(h_{2_i})$
   (c) Victor verifies whether h_{2_i} is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_{P_i} equals the matching I_{V_j}, and thereby verifies that Peggy knows the matching I_{V_j}.
5. Peggy sends Victor the message `halt`
6. Victor concludes that no more $\texttt{prove}(h_{2_i})$ messages will follow

FIGURE 9.1: The T-1 protocol where the verifier initiates and no encryption is used

---

1. Peggy chooses an IB $I_P \in KB_P$ of which she wants to prove her knowledge to Victor; Peggy computes $h_1 = H(I_P, N)$
2. Peggy sends Victor the message $\texttt{ask}(h_1)$
3. Victor computes $I_V\star \subseteq KB_V$
4. if $I_V\star = \emptyset$, Victor sends Peggy the message `halt` and the protocol is halted
5. ($I_V\star \neq \emptyset$) Victor generates a random challenge C
6. Victor sends Peggy the message $\texttt{challenge}(C)$
7. Peggy computes $h_2 = H(I_P, N, P, C)$
8. Peggy sends Victor the message $\texttt{prove}(h_2)$
9. Victor verifies whether h_2 (received from Peggy) is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_P equals the matching I_{V_j}, and thereby verifies that Peggy knows I_{V_j}

FIGURE 9.2: The T-1 protocol where the prover initiates and no encryption is used

1. Alice chooses an IB $I_A \in KB_A$ of which she wants to prove her knowledge to Bob, and of which she wants to test Bob's possession; Alice computes $h_1 = H(I_A, N)$; Alice computes $I_A\star \subseteq KB_A$; Alice generates a random challenge C_A
2. Alice sends Bob the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C_A)$
3. Bob computes $I_B\star \subseteq KB_B$
4. If $I_B\star = \emptyset$, Bob sends Alice the message `halt` and the protocol is halted
5. ($I_B\star \neq \emptyset$) Bob generates a random challenge C_B
6. Bob sends Alice the message $\texttt{challenge}(C_B)$
7. Alice computes $h_{2_A} = H(I_A, N, A, C_B)$
8. Alice sends Bob the message $\texttt{prove}(h_{2_A})$
9. Bob verifies whether h_{2_A} (received from Alice) is equal to any $H(I_{B_i}, N, A, C_B)$, where $I_{B_i} \in I_B\star$ (locally computed). If they are equal, Bob concludes that I_A equals the matching I_{B_i}, and thereby verifies that Alice knows the matching I_{B_i} (which we will call I_B from here on)
10. If Bob is not willing to prove his knowledge of I_B to Alice, Bob sends Alice the message `halt` and the protocol is halted
11. (Bob is willing to prove his knowledge of I_B to Alice) Bob computes $h_{2_B} = H(I_B, N, B, C_A)$
12. Bob sends Alice the message $\texttt{prove}(h_{2_B})$
13. Alice verifies whether h_{2_B} (received from Bob) is equal to $H(I_A, N, B, C_A)$ (locally computed). If they are equal, Alice concludes that I_A equals I_B, and thereby verifies that Bob knows the matching I_A

FIGURE 9.3: The mutual T-1 protocol, where no encryption is used

principal Q has no IB to prove or verify knowledge of. If there is one IB in the set, the agent may prove or verify knowledge of this IB.

It is extremely unlikely that there will be more than one IB in the set $I_Q\star$. However, the protocol easily copes with the situation if it occurs.[5] If this protocol is widely adopted and applied, it can be expected that *somewhere* this

[5] If there is more than one IB in the set $I_Q\star$, an 'external' collision of the hash function has occurred [PvO95]. This is highly improbable, but not impossible. In such a case the principal wants to discriminate between the members of the set. He can do this by making sure his challenge C_Q yields a different hash $H(I_{Q_i}, N, P, C_Q)$ for each element $I_{Q_i} \in I_Q\star$.

Ensuring this is easy because it is extremely unlikely for two IBs I and I' that both $H(I)$ and $H(I')$ clash *and* that $H(I, C_Q)$ and $H(I', C_Q)$ clash as well. If this would not be extremely unlikely, this would be a very severe problem of the supposedly cryptographic hash function. In practice, principal Q may choose a C_Q at random and check for security's sake whether there are new clashes, and choose another C_Q if this would be the case.

This whole process of generating the challenge ensures that each possible h_2 corresponds to exactly one $I_{Q_i} \in I_Q\star$. In the figures, we summarize this process as 'generating a random challenge such that it discriminates'.

A	Hey! You know what?	A	Hey! You know what?
B	Huh, What?	B	Huh, What?
A	Well, you know, don't you?	A	Well, you know, don't you?
B	I don't know what you are talking about	B	Ahh, yeah, of course
A	Well, never mind	A	Thank you, goodbye

FIGURE 9.4: A rough paraphrase of the T-1 protocols. The above are two different conversations between A and B. On the left is the conversation which can be considered an equivalent of an unsuccessful protocol run. The conversation on the right can be considered an equivalent of a successful protocol run.

situation will occur. If the protocol could not handle this situation well, data corruption would be the result. Therefore, the ability to handle such unlikely situations still is an important feature.

Note that without the challenge C in the protocol, the prover could fool the verifier if the prover could somehow obtain h_1 and h_2 without ever knowing I. Therefore, the challenge C should be unpredictable to the prover, because it makes such a scenario infeasible. The challenge is there to prevent that the prover can store and present precomputed, stored values.

Without the nonce N in the protocol, any eavesdropper who happens to know I can analyze and interpret the protocol, which is undesirable. When the eavesdropper does not know the N, this analysis and interpretation is no longer possible. In the next section we further elaborate on eavesdroppers and their abilities to interpret messages of this protocol. In typical applications of one-way hashes, the input to the hash is more or less public knowledge. This protocol on the other hand exploits the fact that the input may *not* be publicly known. Successful completion depends on one of the players being able to 'invert' the one-way hash, since it knows the original input to the hash function.

To summarize, the protocol 'obscures' messages in such a way that only recipients with specific a priori knowledge can interpret the messages. A rough paraphrase[6] of the T-1 protocols can be found in Figure 9.4. It is very sketchy, but illustrates the non-intuitivity of the protocols in an intuitive way.

9.3 Making the Protocol More Efficient (Elaborate, Encryption)

The 'simple' version of the T-1 protocol, as presented in the previous section, has a constant communication complexity[7], which leaves no room for improvement. The computational complexity does leave some room for improvement.[8]

[6] With thanks to Marius Bulacu, who came up with this paraphrase.
[7] More precisely, constant for every secret of which possession is proven.
[8] For a basic introduction to complexity, consult Section 2.3.

1. Create the look-up table, with the columns *hash* and *IB location*. *IB location* is some information on how to locate the IB on the local system. (If IBs are files, this would typically be the file name.) Make the table efficiently searchable on at least the *hash* column.
2. For each IB $I_Q \in KB_Q$, compute $H(I_Q)$, and insert $(H(I_Q), \mathit{location}(I_Q))$ into the table. (Computing the hash value has a time complexity of $\mathit{size}(I_Q)$.)
3. With each modification of personal knowledge, update the look-up table:
   (a) For each added IB I_Q, insert $(H(I_Q), \mathit{location}(I_Q))$.
   (b) For each removed IB I_Q, remove $(H(I_Q), \mathit{location}(I_Q))$.
   (c) Consider each modified IB as an old IB to be removed, and a new IB to be added.

FIGURE 9.5: The initialisation and maintenance of the look-up table, needed by any non-initiating player of the protocol

The computationally most expensive part in the protocol is the computation of the set $I_Q\star$. The time complexity of this computation is $O(\mathit{size}(KB_Q) + |KB_Q|)$, where $\mathit{size}(KB_Q) = \sum_{I_Q \in KB_Q} \mathit{size}(I_Q)$, $\mathit{size}(I_Q)$ is the number of bits in I_Q, and $|KB_Q|$ is the number of items in KB_Q (i.e., the cardinality). Note that this time complexity essentially is the space required to store all IBs.

In this section we will show how we can improve the computational complexity of the protocol by shifting this computational load to a precomputation step which is only executed once. The improved protocol can be run any number of times without requiring this hefty computation.

This process of computing $I_Q\star$ can be divided into two steps:

1. Precomputing a look-up table of size $O(|KB_Q|)$ once, which can be used in all runs of the protocol which share the same nonce. Generating the look-up table still has computational complexity $O(\mathit{size}(KB_Q) + |KB_Q|)$.
2. Looking up received hashes h_1 in the table. When an efficient storage technique for the look-up table is used, this has a time complexity of only $O(\ln |KB_Q|)$.

If principal Q learns a new IB I_Q, the principal has to update his look-up table, which has a time complexity of $O(\ln |KB_Q| + \mathit{size}(I_Q))$. How to initialize and maintain the look-up table is described in Figure 9.5.

Computing a look-up table and performing the protocol once, has the same computational complexity as performing the protocol without any precomputations. Doing precomputations has two benefits. Firstly, the speed of execution of the protocol is much higher, because there are no expensive computations to wait for. Secondly, we can only re-use the look-up table as far as it is

1. Victor chooses an IB $I_V \in KB_V$ of which he wants to test Peggy's knowledge; Victor looks up $h_1 = H(I_V)$; Victor looks up $I_V\star \subseteq KB_V$; Victor generates a random challenge C
2. Victor sends Peggy the message $\{\texttt{ask}(h_1), \texttt{challenge}(C)\}_K$
3. Peggy decrypts the message from Victor and obtains the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C)$; Peggy looks up $I_P\star \subseteq KB_P$
4. For each $I_{P_i} \in I_P\star$ of which Peggy is willing to prove her knowledge to Victor, the following happens:
   (a) Peggy computes $h_{2_i} = H(I_{P_i}, P, C)$
   (b) Peggy sends Victor the message $\texttt{prove}(h_{2_i})$
   (c) Victor verifies whether h_{2_i} is equal to any $H(I_{V_j}, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_{P_i} equals the matching I_{V_j}, and thereby verifies that Peggy knows the matching I_{V_j}.
5. Peggy sends victor the message `halt`
6. Victor concludes that no more $\texttt{prove}(h_{2_i})$ messages will follow

FIGURE 9.6: The protocol where the verifier initiates and encryption is used

---

1. Peggy chooses an IB $I_P \in KB_P$ of which she wants to prove her knowledge to Victor; Peggy looks up $h_1 = H(I_P)$
2. Peggy sends Victor the message $\{\texttt{ask}(h_1)\}_K$
3. Victor decrypts the message from Peggy and obtains the message $\texttt{ask}(h_1)$; Victor looks up $I_V\star \subseteq KB_V$
4. if $I_V\star = \emptyset$, Victor sends Peggy the message `halt` and the protocol is halted
5. ($I_V\star \neq \emptyset$) Victor generates a random challenge C
6. Victor sends Peggy the message $\texttt{challenge}(C)$
7. Peggy computes $h_2 = H(I_P, N, P, C)$
8. Peggy sends Victor the message $\{\texttt{prove}(h_2)\}_K$
9. Victor decrypts the message from Peggy and obtains the message $\texttt{prove}(h_2)$; Victor verifies whether h_2 (received from Peggy) is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_P equals the matching I_{V_j}, and thereby verifies that Peggy knows I_{V_j}

FIGURE 9.7: The T-1 protocol where the prover initiates and encryption is used

1. Alice chooses an IB $I_A \in KB_A$ of which she wants to prove her knowledge to Bob, and of which she wants to test Bob's possession; Alice looks up $h_1 = H(I_A)$; Alice looks up $I_A\star \subseteq KB_A$; Alice generates a random challenge C_A
2. Alice sends Bob the message $\{\texttt{ask}(h_1), \texttt{challenge}(C_A)\}_K$
3. Bob decrypts the message from Alice and obtains the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C_A)$; Bob looks up $I_B\star \subseteq KB_B$
4. If $I_B\star = \emptyset$, Bob sends Alice the message `halt` and the protocol is halted
5. ($I_B\star \neq \emptyset$) Bob generates a random challenge C_B
6. Bob sends Alice the message $\texttt{challenge}(C_B)$
7. Alice computes $h_{2_A} = H(I_A, A, C_B)$
8. Alice sends Bob the message $\{\texttt{prove}(h_{2_A})\}_K$
9. Bob decrypts the message from Alice and obtains the message $\texttt{prove}(h_{2_A})$; Bob verifies whether h_{2_A} (received from Alice) is equal to any $H(I_{B_i}, A, C_B)$, where $I_{B_i} \in I_B\star$ (locally computed). If they are equal, Bob concludes that I_A equals the matching I_{B_i}, and thereby verifies that Alice knows the matching I_{B_i} (which we will call I_B from here on)
10. If Bob is not willing to prove his knowledge of I_B to Alice, Bob sends Alice the message `halt` and the protocol is halted
11. (Bob is willing to prove his knowledge of I_B to Alice) Bob computes $h_{2_B} = H(I_B, N, B, C_A)$
12. Bob sends Alice the message $\texttt{prove}(h_{2_B})$
13. Alice verifies whether h_{2_B} (received from Bob) is equal to $H(I_A, N, B, C_A)$ (locally computed). If they are equal, Alice concludes that I_A equals I_B, and thereby verifies that Bob knows the matching I_A

FIGURE 9.8: The symmetric protocol with encryption.

safe to re-use the nonce that was used to construct the look-up table. However, for each distinct nonce used, the player still needs to generate such a look-up table, which is by far the most expensive part of the T-1 protocols.

Therefore, we can improve dramatically on speed if we can find a way to safely re-use nonces, or to use no nonces at all. The reason to use nonces was to make sure we have semantic security with respect to any third party observing the conversation. Semantic security can also be achieved by means of encryption of some crucial parts of the protocol. The parts that need to be encrypted are those of which an eavesdropper could either infer the IB[9], or could verify the proof. To prevent inference of the IB, h_1 should be encrypted. To

[9] With 'infer', we mean 'properly guess'.

prevent verification of the proof, or the possibility to infer IB by a brute-force attack, at least one of C and h_2 should be encrypted. Since C and h_2 are always sent by opposing players, we may choose to encrypt the one sent by the player that also sent h_1, which is the player that initiated the protocol. Thus only the initiator needs to be able to send encrypted messages.

The adjusted ('elaborate') versions of the T-1 protocol are shown in Figures 9.6 (the verifier initiates), 9.7 (the prover initiates), and 9.8 (mutual proof).

By using encryption and no nonce (or a constant nonce), any responding player of the protocol needs to generate the look-up table *only once*. The need to establish a common nonce is no longer there, but the need for key exchange has come in its place. Since the protocol requires authentication, it may well be that key exchange is required anyway.

9.4 Correctness Proof in GNY Logic

In the remainder of this chapter, we will use GNY logic[10] to prove the T-1 protocols correct. To be precise, we will prove correct the configuration where the verifier initiates and no encryption is used (depicted in Figure 9.1). The configurations in which the prover initiates the protocol and where a mutual proof is exercised do not shed new light on the security analysis of the protocols. The proofs for these configurations can be obtained by slight adaptation of the proof for the configuration where the verifier initiates. The proof for the 'elaborate' version where encryption is used is very similar to the proof for the 'simple' version where no encryption is used. Therefore, we will only analyze the simple version.

Our proof uses *knowledge preconditions*, a special type of assumptions. These are explained in Section 9.4.1. The GNY idealization of the protocol and the precise protocol claims are given in Section 9.4.2. Then, in Section 9.4.3, we give an analysis of the protocol without encryption up to the point where the newly introduced inference rule **H2** is needed. A discussion on how to complete the proof, including the proof completion itself, is shown in Section 9.4.4.

In Section 9.1, it has been noted that the communication channel should be authenticated and cannot be modified by an adversary. The most important prerequisite is that the last message of the protocol is clearly bound to its sender, more precisely that the receiver can verify who is the sender. For our protocols, it is not really relevant in what way this authentication is established. To keep the proofs of the protocols as simple as possible, we simply assume a specific way of authentication of the sender. We choose a public-key signature for this. This choice is not essential and if we change this authentication method, the protocol proofs can easily be adjusted to reflect this.

It may seem counter-intuitive to prove a protocol that does not use encryption to be correct by assuming signatures, which essentially is a special case of

[10] GNY logic is summarized in Appendix B; our extension of GNY logic is explained in Chapter 6, and authentication logics in general are extensively explained in Chapter 4.

encryption. However we would like to stress that this is just the easiest way to prove the protocol correct. The issue is that we do not have to assume encryption for our protocols to work, but only sender authentication.

9.4.1 Knowledge Preconditions

A *knowledge precondition* is a special type of protocol assumption. First of all, it is an assumption which states that a particular principal has certain positive knowledge[11]. Moreover, a knowledge precondition is a neccessary condition for a protocol to end in an accepting state. For some protocols, it is useful to distinguish certain knowledge preconditions from the other protocol assumptions in order to analyse the protocol. This is because whether the protocol ends in in an accepting state should coincide with the conjunction of all distinguished knowledge preconditions. In the analysis of the T-1 protocol we use knowledge preconditions.

In the T-1 protocol, an accepting state is a state in which the verifier is convinced of the knowledge of the prover, i.e., the verifier accepts. The knowledge preconditions of the T-1 protocol are that both the verifier and the prover know the secret. Thus, a correctness proof of the T-1 protocol in GNY logic (or more precisely, a derivation of the accepting protocol state) should critically depend on the truth value of the knowledge precondition. Any unsatisfied knowledge precondition should result in the impossibility of a GNY derivation of the accepting protocol state. This kind of proof requires a *completeness assumption*[12] about cryptographic hash functions.[13] Also, correct inference rules for cryptographic hash functions are required, most notably inference rule **H2**[14]. Our completeness assumption is:

> The rules **P4**, **I3** and **H2** capture all relevant properties of cryptographic hash functions.[15]

One of the major issues of the T-1 protocols is whether the prover can cheat by asking someone else to compute the proof in name of the prover, and just forward this proof. Making someone different from the prover compute the actual proof can be achieved by either a successful man-in-the-middle attack by the prover, or by a willing assistant of the prover which *does* have the knowledge referred to in the knowledge precondition. We should design protocols in such a way that successful man-in-the-middle attacks do not exist. Authentication logics such as GNY logic help in analyzing the existence of such attacks. However, we cannot fight willing assistants of provers. In some sense, this is also unnecessary, since the goal of the secret prover protocols is to test whether the prover has effective access to the secret, and one may reasonably claim that

[11] That is, a knowledge precondition cannot be of the form 'principal A does *not* know X'.
[12] Completeness assumptions are introduced in Section 6.1.2 (page 70).
[13] Cryptographic hash functions are extensively explained in Chapter 3.
[14] Rule **H2** is introduced in Section 6.2.1 on page 74.
[15] The rules **P4** and **I3** are introduced in Appendix B.2, on page 185.

the prover indeed has such access if she has an assistant who will perform computations on the secrets on behalf of the prover.

The T-1 protocols are protocols that by design should fail to complete if the knowledge precondition is not true at the start of a protocol run. Normally in GNY logic a protocol is proven correct if we can infer the desired end state using the assumptions, inference rule and communication steps. However, for a protocol to fail if an assumption is not met, means there should not exist proofs that do not depend on the critical assumptions. This leads to the possibly somewhat counterintuitive observation that *some GNY correctness proofs prove the incorrectness of a protocol*. Exactly the proofs that do not depend on all the knowledge precondition assumptions indicate that a protocol is incorrect. We call these proofs *invalidators*. Non-existence of invalidators can only be proven if we make a completeness assumption: we assume that we know all applicable inference rules, or at least all rules that can lead us to a proof of a protocol (some of which may be invalidators).

It should be noted that the absence of invalidators does not prove correctness of a protocol in a strict sense. Just as with normal authentication logics, it only shows that the protocol has passed a test of some not-so-obvious flaws.

9.4.2 Claims and GNY Idealization

In the T-1 protocol we have two participating principals, V the *V*erifier and P the *P*rover. We assert that, for any I, our protocols satisfy the following properties:

1. 'The verifier learns whether P knows I, iff the verifier knows I and the prover wants to prove her knowledge of I':
   $V \models P \ni I$ holds after the protocol run $\Longleftrightarrow$
   $P \ni I$ and $V \ni I$ hold before the protocol run,
   and P wants to prove possession of I before the protocol run.

2. 'Only the verifier learns whether anybody knows I by means of the protocol':
   For any principal Q, except V:

   (a) $Q \models P \ni I$ holds after the protocol run $\Longleftrightarrow$
   $Q \models P \ni I$ holds before the protocol run.

   (b) $Q \models V \ni I$ holds after the protocol run $\Longleftrightarrow$
   $Q \models V \ni I$ holds before the protocol run.

3. 'Nobody learns I by means of the protocol':
   For any principal Q,
   $Q \ni I$ holds after the protocol run $\Longleftrightarrow$
   $Q \ni I$ holds before the protocol run.

Here we should mention that all right-hand sides of the assetions should include 'or ... learns X by messages outside of the protocol', where X respectively reads I (1), $P \ni I$ (2a), $Q \ni I$ (2b), and I (3). Of course, a principal may

knowledge preconditions

A.1 $P \ni I$
A.2 $V \ni I$

assumptions

A.3 $P \ni P$ A.4 $V \ni P$ A.5 $P \ni -K$ A.6 $V \ni +K$ A.7 $V \models \overset{+K}{\mapsto} P$ A.8 $V \ni C$ A.9 $V \models \sharp(C)$ A.10 $P \ni N$ A.11 $V \ni N$

the protocol itself

1 $V \rightarrow P : H(I, N), C$
2 $P \rightarrow V : \{H(I, N, P, C)\}_{-K}$

claims See section 9.4.2

FIGURE 9.9: GNY idealization of the T-1 protocol where the verifier initiates and no encryption is used. The formal GNY language used is explained in Appendix B.

learn something by a message outside of the protocol. Learning in such a way has nothing to do with any property of the protocol, as it certainly not learned *by means of* the protocol.

We will prove these properties for the T-1 protocol where the verifier initiates and no encryption is used. The GNY idealization of this protocol is given in Figure 9.9. The $\Leftarrow$ part of property 1 will be proven in sections 9.4.3 and 9.4.4. The $\Rightarrow$ part of property 1 and property 3 will be proven in Section 9.4.5. Proving property 2 requires us to make some assumptions on the beliefs and possessions of an attacker. This will be done in Section 9.4.6.

9.4.3 The Easy Part of the Proof

With the GNY idealization given in Figure 9.9, we can analyze the T-1 protocol in a rather straightforward way. The first step is to apply the *protocol parser* to the idealized protocol, shown in Figure 9.10. As discussed in Sections 4.3 and 6.2.2, this gives for every communication step (step transition) in the protocol two statements. The first statement asserts that the sender possesses (can construct) the message he is sending, the second statement asserts what the receiver will see when he receives the message.

The protocol assumptions are given in Figure 9.9. Assumptions A.1 and A.2 express that the principals do indeed know the secret. Thus, these are the knowledge preconditions. Assumptions A.3 and A.4 reflect that both principals know the identity of P. Assumption A.5 expresses that the prover knows her private key, and assumption A.6 expresses that the verifier knows the corresponding public key. Assumption A.7 reflects that the verifier believes this public key indeed corresponds to the prover's private key. Assumptions A.8 and A.9 reflect respectively that the verifier knows his own challenge and that

protocol step	sender possession (precondition)	receiver learning (postcondition)
1	$V \ni (H(I,N),C)$	$P \lhd (*H(I,N),*C)$
2	$P \ni \{H(I,N,P,C)\}_{-K}$	$V \lhd *\{*H(I,N,P,C)\}_{-K}$

FIGURE 9.10: The output of the protocol parser for the T-1 protocol where the verifier initiates and no encryption is used.

the verifier believes its freshness. Assumptions A.10 and A.11 reflect that both principals know the nonce.

Just using these assumptions we can already infer a few lines which will be needed later on in the protocol. Namely the verifier can send message 1 of the protocol, and he can verify message 2 which the prover ought to send.

B.1	$V \ni (I,N)$	**P2**(A.2, A.11)
B.2	$V \ni H(I,N)$	**P4**(B.1)
B.3	$V \ni (H(I,N),C)$	**P2**(B.2, A.8)
B.4	$V \ni (I,N,P,C)$	**P2**(A.2, A.11, A.4, A.8)
B.5	$V \models \sharp(I,N,P,C)$	**F1**(A.9)
B.6	$V \ni H(I,N,P,C)$	**P4**(B.4)
B.7	$V \ni H(H(I,N,P,C))$	**P4**(B.6)
B.8	$V \models \phi(H(I,N,P,C))$	**R6**(B.7)

Now we start the actual protocol. The verifier sends a message to the prover. Thus, the verifier learns nothing new yet. The prover, however, can calculate the proof which she will send later on in message 2. The conveyed message is shown in line C.1.

C.1	$P \lhd (*H(I,N),*C)$	[1](B.3)
C.2	$P \lhd *H(I,N)$	**T2**(C.1)
C.3	$P \lhd *C$	**T2**(C.1)
C.4	$P \lhd C$	**T1**(C.3)
C.5	$P \ni C$	**P1**(C.4)
C.6	$P \ni (I,N,P,C)$	**P2**(A.1, A.10, A.3, C.5)
C.7	$P \ni H(I,N,P,C)$	**P4**(C.6)
C.8	$P \ni \{H(I,N,P,C)\}_{-K}$	**P8**(A.5, C.7)

The justification of line C.1 may require some explanation. On line C.1, it is reflected that the recipient of the first message in the protocol learns the message. The statement on line C.1 corresponds with the 'receiver learning' part of message 1 in the output of the protocol parser (Figure 9.10). The justification B.3 corresponds with the 'sender possession' part of message 1 in the output of the protocol parser.

9.4.4 Different Options to Complete the Proof

So far, the protocol analysis is plain and rather simple. Completing the proof from here on is not as straightforward as the easy part shown above. We will provide three options to complete the proof. The first two options are flawed,

and we will explain why. The third option we present is the 'correct one': it is not flawed.

1. There is a way to prove correctness in GNY logic of this protocol without introducing new inference rules. In that case, a rather appealing but weak assumption would have to be added:

   A.12 $V \models V \overset{N}{\leftrightarrow} P$

   This assumption states that V believes that only V and P know the secret N. Using this assumption, the proof goes as follows. The conveyed message is shown in line D.1.

D.1	$V \lhd *\{*H(I,N,P,C)\}_{-K}$	[2](C.8)
D.2	$V \lhd \{*H(I,N,P,C)\}_{-K}$	**T1**(D.1)
D.3	$V \lhd *H(I,N,P,C)$	**T6**(D.2, A.6)
D.4	$V \models P \mid\sim (I,N,P,C)$	**I3**(D.3, B.4, A.12, B.5)
D.5	$V \models P \ni (I,N,P,C)$	**I6**(D.4, B.5)
D.6	$V \models P \ni I$	**P3**(D.5)

   Note that in this proof, neither the identity of P, nor P's signature are used. Though the above proof is a correct GNY logic proof, it does not help us because it depends on assumption A.12. This assumption essentially states that the verifier should trust the prover on not disclosing the secret to someone else, since the verifier has no control over the truth value of this assumption. If the prover does disclose the secret, this opens up possibilities for a successful man-in-the-middle attack: the prover can use the same nonce with multiple different principals, and use a proof given by some principal Q to prove to principal V that she knows the secret, as long as the verifier V 'knows the name of Q' ($V \ni Q$).

   To see the man-in-the-middle attack easier, observe that a protocol which does not include the identity of P would have a virtually identical analysis in GNY logic. Even without $V \ni Q$, a verifier could be deceived in such a simplified protocol.

2. In order to prove correctness without relying on assumption A.12, we need new inference rules. In Section 6.2.1 we have discussed various rules, and we will apply them here. Using rule **H1**, we can finish the protocol proofs. The proof is as follows:

D'.1	$V \lhd *\{*H(I,N,P,C)\}_{-K}$	[2](C.8)
D'.2	$V \lhd \{*H(I,N,P,C)\}_{-K}$	**T1**(D'.1)
D'.3	$V \lhd *H(I,N,P,C)$	**T6**(D'.2, A.6)
D'.4	$V \models P \mid\sim (I,N,P,C)$	**H1**(D'.3, B.4)
D'.5	$V \models P \ni (I,N,P,C)$	**I6**(D'.4, B.5)
D'.6	$V \models P \ni I$	**P3**(D'.5)

   Note that in this proof, P's identity is not used. As discussed in Section 6.2.1, rule **H1** is dubious.

B.1	$V \ni (I, N)$	**P2**(A.2, A.11)
B.2	$V \ni H(I, N)$	**P4**(B.1)
B.3	$V \ni (H(I, N), C)$	**P2**(B.2, A.8)
B.4	$V \ni (I, N, P, C)$	**P2**(A.2, A.11, A.4, A.8)
B.5	$V \models \sharp(I, N, P, C)$	**F1**(A.9)
B.6	$V \ni H(I, N, P, C)$	**P4**(B.4)
B.7	$V \ni H(H(I, N, P, C))$	**P4**(B.6)
B.8	$V \models \phi(H(I, N, P, C))$	**R6**(B.7)
C.1	$P \lhd (*H(I, N), *C)$	[1](B.3)
C.2	$P \lhd *H(I, N)$	**T2**(C.1)
C.3	$P \lhd *C$	**T2**(C.1)
C.4	$P \lhd C$	**T1**(C.3)
C.5	$P \ni C$	**P1**(C.4)
C.6	$P \ni (I, N, P, C)$	**P2**(A.1, A.10, A.3, C.5)
C.7	$P \ni H(I, N, P, C)$	**P4**(C.6)
C.8	$P \ni \{H(I, N, P, C)\}_{-K}$	**P8**(A.5, C.7)
D″.1	$V \lhd *\{*H(I, N, P, C)\}_{-K}$	[2](C.8)
D″.2	$V \lhd \{*H(I, N, P, C)\}_{-K}$	**T1**(D″.1)
D″.3	$V \lhd *H(I, N, P, C)$	**T6**(D″.2, A.6)
D″.4	$V \models P \mid\sim *H(I, N, P, C)$	**I4**(D″.2, A.6, A.7, B.8)
D″.5	$V \models P \mid\sim (I, N, P, C)$	**H2**(D″.4, B.4)
D″.6	$V \models P \ni (I, N, P, C)$	**I6**(D″.5, B.5)
D″.7	$V \models P \ni I$	**P3**(D″.6)

FIGURE 9.11: GNY proof of the T-1 protocol where the verifier initiates and no encryption is used.

3. Using the well-justified rule **H2** from Section 6.2.1, we can also finish the protocol proofs. The proof is as follows:

D″.1	$V \lhd *\{*H(I, N, P, C)\}_{-K}$	[2](C.8)
D″.2	$V \lhd \{*H(I, N, P, C)\}_{-K}$	**T1**(D″.1)
D″.3	$V \lhd *H(I, N, P, C)$	**T6**(D″.2, A.6)
D″.4	$V \models P \mid\sim *H(I, N, P, C)$	**I4**(D″.2, A.6, A.7, B.8)
D″.5	$V \models P \mid\sim (I, N, P, C)$	**H2**(D″.4, B.4)
D″.6	$V \models P \ni (I, N, P, C)$	**I6**(D″.5, B.5)
D″.7	$V \models P \ni I$	**P3**(D″.6)

Note that changing this proof to use rule **H3** is trivial: P only needs to insert "I know" into its signed messages, and V only needs to verify that this token is indeed present in the message.

Thus, the whole GNY proof of the T-1 protocol is as in Figure 9.11. If we observe that the prover will only engage in the protocol if he wants to prove possession of i, we have proven the $\Leftarrow$ part of property 1 (stated in Section 9.4.2).

9.4.5 Proving principals do not learn too much

So far, of the properties stated in section 9.4.2, we have only proven the $\Leftarrow$ part of property 1. In this section, we will prove the the $\Rightarrow$ part of property 1, and we will prove property 3.

1. 'The verifier learns whether P knows I, iff the verifier knows I and the prover wants to prove her knowledge of I', $\Rightarrow$ part: We assume $V \models P \ni I$ holds after the protocol run (the verifier has been convinced of prover's possession of I).

   For the prover to be able to actually prove possession of I, she has to use it while constructing message 2. If she does not possess I, she cannot satisfy step C.6, which is necessary for C.8, which states message 2. This is because the only way in which the prover can obtain C.7 is by application of inference rule **I3**.[16] Thus, $P \ni I$ holds before the protocol run.

   If the prover would not want the verifier to possibly learn that the prover knows I, the prover would not have sent message 2. Thus, the prover wants to prove her knowledge of I.

   For the verifier to be able to verify the proof, he has to possess I as well. More specifically, the verifier has to verify whether the message he sees in line D''.3 (or equivalently, in D.3 or D'.3) equals the value the verifier computed at line B.6.[17] Thus, $V \ni I$ holds before the protocol run.

2. 'Nobody learns I by means of the protocol': We prove this by contradiction. Let us assume that principal Q does learn I by means of the protocol, and that by analyzing messages Q managed to reconstruct I. I itself is never conveyed except as an argument to a one-way hash function. Thus, Q managed to invert a one-way hash function. Obviously, this is impossible.

   In GNY logic, this is reflected in the inference rules of our completeness assumption (**P4**, **I3** and **H2**): in every rule in which something is inferred from a term which involves a term of the form $H(X)$ (i.e., rules **I3** and **H2**), the principal that learns something by means of the inference rule, must possess X as a condition for the inference rule to apply[18].

---

[16] See also the completeness assumption about cryptographic hash functions on page 134.

[17] Note that this protocol does not depend on the recognizability constraint of rule **I4**, as used in line D''.3 of the last proof. Even if the verifier can *always* recognize the message sent by the prover, as needed for verification of the signature, the verifier still cannot verify the proof, as the verifier has nothing to compare the message with. If we change the protocol to use rule **H3**, introduction of the "I know"-token will lead to immediate recognizability of the signed message. This will not invalidate the proof.

[18] In rule **I3**, P learns something as a result of observing $H(X, S)$, but only if $P \ni (X, S)$ is also satisfied, i.e., if P knows the protected secret already. Similarly, in rule **H2**, V learns something as a result of believing P conveyed $H(X, P)$, but only if $V \ni (X, P)$ is also satisfied, i.e., if V knows the protected secret already. Note that the principal names P and V in the rules **I3** and **H2** do not only apply to the principals P and V in the protocol, but to *any* principal.

Except that the protocol works, it is also very efficient. Both the verifier and the prover only need to perform a constant number of steps. The prover will, upon seeing $*H(I, N)$, look whether she has a matching secret I. Only after establishing that she actually does, she will start further computations. The bottleneck of course is recognizing an I which matches the sent $H(I, N)$. A principal can in fact generate a look-up table in advance, which stores for each possible I the corresponding $H(I, N)$ value. This is a one-time operation whose cost is proportional to the total size of all secrets that a player wants to be able to look up. This has to be done for each value of N the principal is interested in. If however the protocol which uses encryption is used, this dependency on N disappears.

9.4.6 Modeling the beliefs and possessions of an attacker

In the previous section we have shown that no principal can learn I itself from observing the protocol. However, we are also interested in anything that an eavesdropper *could* learn. Could an eavesdropper become convinced that the prover or the verifier knows I? This is what property 2 of section 9.4.2 is about. Or, less bad but still undesirable: could an eavesdropper learn about what secret I the protocol is run if she already knows the secret herself?

Let us assume that, at the start of the protocol, Eve the eavesdropper knows everything the participating principals know, except P's private key, the nonce N and the challenge C, but including the secret I:

E.1	$E \ni I$	E.3	$E \models \overset{+K}{\mapsto} P$	E.5	$E \models \sharp(C)$
E.2	$E \ni +K$	E.4	$E \ni P$		

In the course of the protocol, E will learn both $\{H(I, N, P, C)\}_{-K}$, C and $H(I, N)$. Since Eve does not know N, she will never be able to infer what secret I the protocol is run about, since in all messages where I is communicated, it is 'mixed' with N in a one-way hash function. For the same reason Eve cannot verify P's proof. Thus, all three values Eve learns are indistinguishable from random noise (as per the *random oracle model*, see Section 3.3). In the case of the protocol that uses encryption instead of a nonce, E will learn $\{H(I), C\}_{+K}$ and $\{H(I, P, C)\}_{-K}$. E cannot decrypt the first message, and therefore never learns C, which is needed to be able to interpret $\{H(I, P, C)\}_{-K}$.

An eavesdropper knowing everything except private keys and the shared nonce does not learn anything sensible from observing the protocol. This is a strong result. One of its implications is that N may be known to any principal who is either (1) trusted by the verifier, or (2) not capable of intercepting messages from or to any principal using the nonce N.

One last question is whether one of the participants could be a passive attacker. In that case, the attacker would also possess N. For the case the attacker is the verifier, the proof is trivial, since the goal of the protocol is that the verifier *does* learn. For the case where the attacker is the prover, the prover *will* indeed learn what secret the protocol is about. However, the prover will not learn whether the verifier really possesses I: the verifier might have learned

$H(I, N)$ from someone else.

9.5 Conclusion

In this chapter, we have presented our T-1 protocol. Let us return to Victor, the secret agent who wants to protect Peggy from great danger (the story which opened this chapter, page 123). Peggy is invited by Mallory, but will get killed by Mallory if she accepts the invitation. Peggy is reluctant to disclose the invitation to Victor. Let us see how the T-1 protocol handles this situation.

1. *How do Victor and Peggy make sure the invitation is not disclosed in case the other does not know of the invitation?*

   Peggy and Victor only send cryptographic hash values of the invitation. From a cryptographic hash value, one cannot infer the pre-image (Section 3.2). Thus, nobody can learn the invitation from the communicated messages.

2. *How do Victor and Peggy establish about what secret they are communicating?*

   Victor sends Peggy the cryptographic hash value of the invitation I. If Peggy knows the invitation, she recognizes the hash value.

3. *How does Victor become convinced of Peggy's knowledge of the invitation?*

   Victor asks Peggy to present a cryptographic hash value of the invitation and a *challenge* C chosen by Victor. Only if Peggy has the invitation, she will be able to construct the requested cryptographic hash value.

4. *How is a man-in-the-middle attack prevented?*

   In the pre-image of the hash value that must convince of possession, the identity of the prover must be incorporated.

   In a man-in-the-middle attack, Peggy would try to deceive Victor by initiating a concurrent protocol with someone else (say, Cecil) and passing the messages from Cecil to Victor.

   If Peggy sends Cecil's messages to Victor, these hash values will be constructed with Cecil's identity, and not Peggy's. Victor expects hash values in which Peggy's identity P has been incorporated. He can detect it if this is not the case. Thus, if Peggy mounts a man-in-the-middle attack, Victor will not be convinced.

5. *How do Peggy and Victor make sure that eavesdroppers cannot learn anything from the protocol?*

   There are two solutions:

   - They use a commonly agreed, secret nonce N, which is incorporated into every pre-image before the hash is computed. As the eavesdropper does not know the nonce, the eavesdropper cannot learn anything.

- They use encryption to hide the hash values from which the eavesdropper could learn something.

6. *What if one of the principals does not know the invitation?*

   In that case the ignorant principal sees bits which he/she cannot distinguish from random noise. The ignorant principal may continue the protocol, but will not convince the other principal.

7. *What if not Victor, but Peggy wants to initiate the protocol?*

   In that case, Peggy sends the first message of the protocol. All configurations of the protocol are listed in Figures 9.1–9.3 (using a nonce N), and Figures 9.6–9.8 (using encryption).

8. *What would an actual protocol run look like, if we omit all the technicalities?*

   Look at Figure 9.4 on page 129.

The T-1 protocol is proven correct in our extended version of GNY logic.[19] The T-1 protocol works for knowledge authentication[20] in the 1-to-many case: where one of the principals 'points at' a secret, and the other principal looks whether he/she knows the same secret. In the next chapter, the T-1 protocol is generalized to the many-to-many case.

[19] GNY logic is summarized in Appendix B, and our extensions are explained in Chapter 6.
[20] Knowledge authentication is introduced in Chapter 8.

The T-2 protocol, our solution for many-to-many knowledge authentication, is presented. It is a parallel composition of the T-1 protocol. It relies on prefix trees made from hash values to optimize communication complexity. The average-case communication complexity is experimentally estimated.

Chapter 10

Many-to-many Protocols (T-2)

In the previous chapter, we presented our T-1 protocol, which is a solution for 'comparing information without leaking it' in the 1-to-many case. That is the case where one principal has only one secret in mind, and the other player any number of secrets. The T-1 protocol can be generalized to the many-to-many case: the case where two principals each have a large number of secrets and wish to determine the intersection of their secrets. In this chapter, we will present and explain our generalization, the T-2 protocol.[1]

The T-2 protocol offers the same security guarantees as the T-1 protocol, because the T-2 protocol can be seen as a group of parallel runs of the T-1 protocol, and the T-1 protocol is secure under parallel composition. The T-2 protocol is, for cooperative principals, very efficient in terms of communication complexity and computational complexity.

Moreover, the T-2 protocol offers *fairness*: the players can make sure that they only prove possession of secrets for which they receive a reciprocal proof in return.

10.1 Using Prefix Trees for Efficiency

The T-2 protocol heavily relies on prefixes and prefix trees. To grasp the working of the protocol, it is important to understand some properties of prefix trees, and how this relates to *set intersection*. This section will offer the necessary background on this subject.

[1] Discussions with Sieuwert van Otterloo have been of critical value for the work that is reported in this chapter. The results reported up to and including Section 10.3 are joint work.

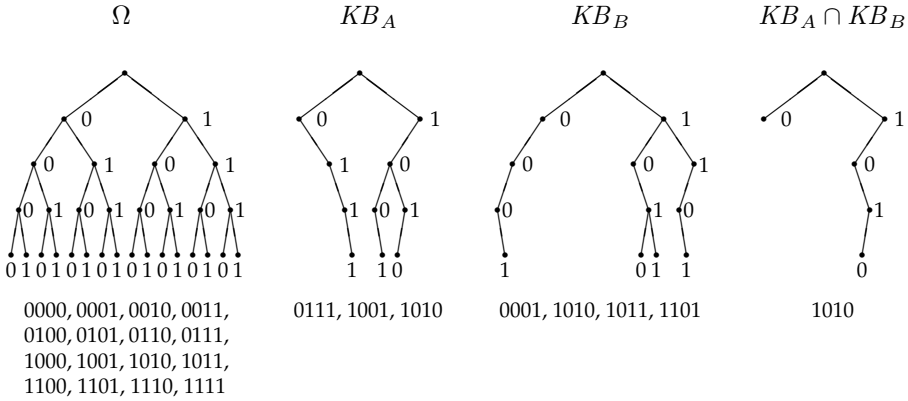


FIGURE 10.1: Sets KB_A, KB_A represented as binary hash value prefix trees. In this toy example, the length of the hash value is 4 bits. Leaf nodes at depth 4 represent full hash values. The prefix trees of the full domain Ω and the intersection $KB_A \cap KB_B$ are also shown. Below every prefix tree, the list of hash values in the tree is shown.

Consider two principals, Alice and Bob, with knowledge bases KB_A and KB_B. A simple way to generalize the T-1 protocol to the many-to-many case, is to repeat the protocol $|KB_A|$ times: in every run of the T-1 protocol, another single secret of Alice is compared with all Bob's secrets. For every secret of Alice, l bits (where l is the length of the hash value, a security parameter) have to be exchanged. A typical message would be: 'Bob, do you have the secret with hash `1ab1dabb2c90231e3182b15ffcacd732`?'[2] It turns out that if a specific secret of Alice is not a secret of Bob, the communication of l bits is largely a waste of communication capacity.

In the T-1 protocol, one player 'points at a secret', by sending a message of the form `ask`(h_1). The T-2 protocol differs from the T-1 protocol in that the process of pointing at secrets is performed interactively and iteratively by both principals. Instead of pointing at a secret by presenting a bit string of length k, very short bit strings are presented. A typical message would be: 'Bob, do you happen to have any secrets with hash prefix `1a`?'[3] Both Alice and Bob may in fact have many secrets with this hash prefix. If it is the case that Bob does not have any secret with hash prefix `1a`, he can say so to Alice: 'Alice, I do not have any secrets with hash prefix `1a`!'[4] As a result of such 'refusal' message, Alice knows that stating the full hash values which start with `1a` is a waste of time.

The hash values of a set of secrets can be represented in a binary prefix tree. Figure 10.1 shows the binary prefix trees for the toy example where the length of the hash value is four bits, Alice has three secrets, and Bob has four secrets.

The way that the T-2 protocol optimizes the communication complexity is

[2] Technically, the message that is exchanged is '`ask(1ab1dabb2c90231e3182b15ffcacd732)`'.
[3] Technically, the message that is exchanged is '`ask(1a)`'.
[4] Technically, the message that is exchanged is '`refuse(1a)`'.

	encoding of the prefix tree				
depth	1	2	3	4	encoding
tree	01	0101	01010101	0101010101010101	size (bits)
Ω	11	1111	11111111	1111111111111111	30
KB_A	11	0110	0111	010110	16
KB_B	11	1011	10 0110	01 1101	18
$KB_A \cap KB_B$	11	0010	01	10	10

TABLE 10.1: Binary encoding of some hash prefix trees. The depth corresponds to the depth of the corresponding tree (see Figure 10.1). The two bits at depth 1 signify the two possible branches at depth 1, the four bits at depth 2 signify the possible two branches at depth 2, and so on. Whitespace is inserted in the encodings to align the bits of the encoding with the table header.

by constructing (almost) precisely the prefix tree that belongs to the set $KB_A \cap KB_B$ (an *intersection prefix tree*). The efficiency of a protocol that does just this depends on the size of an intersection prefix tree (counted in nodes), and on the way an intersection prefix tree is encoded. Determining the expected size of an intersection prefix tree is far from trivial, and we will devote Section 10.4 to this.

Creating an efficient encoding scheme for a binary tree (without labels) is relatively simple: A binary tree can be represented in $(1 + 2 \cdot \#\text{nodes})$ bits. One bit is used to encode whether the root node exists. Then, for every existing node, two bits are used: one bit for each of the two branches that may spring from it. The bit is 0 if there is no such branch, and 1 if there is such a branch. There are a few options to order the bits; we choose for 'breadth-first' order: that is the order in which a breadth-first search algorithm would visit the nodes.[5] The binary strings that stem from this encoding scheme are self-delimiting.

The binary hash prefix trees we use are a specific subset of all possible binary trees, because the depth of the binary hash prefix trees is bounded by the size l of the hash values. This means that binary hash prefix trees can be encoded even more efficiently: the nodes that are at depth l cannot have any branches springing from it.[6] Therefore nodes at depth l need zero bits for their encoding. Also, when we assume that binary hash prefix trees are nonempty, we can omit the bit for the root node. Using this coding scheme, a binary hash prefix tree with maximum depth l can be encoded in this number of bits:

$$2 \cdot (\#\text{nodes} - \#\text{nodes at depth } l)$$

Table 10.1 illustrates our binary hash prefix tree coding scheme for the sets given in Figure 10.1.

[5] This is sometimes called 'level-order traversal'.
[6] We adhere to the convention that we consider the root node of a tree to be at depth 0.

message	meaning
`ask`(p_1)	A principal asks the other player to look whether he has any secrets which have a hash value that starts with p_1
`refuse`(p_1)	A principal tells the other player he will not prove possession of any secrets whose hash value starts with p_1
`challenge`(h_1, C)	A principal asks the other player to prove possession of a secret with hash value h_1, using the challenge C
`prove`(h_1, p_2)	A principal partially proves possession of a secret, identified by h_1, by presenting the prefix p_2.

TABLE 10.2: Basic messages used in the T-2 protocol. The prefixes can be of any length between 0 and l (where l is the length of the hash value, a security parameter).

10.2 Specification of the T-2 Protocol

The prerequisites of the T-2 protocol are the same as the prerequisites of the T-1 protocol, given in Section 9.1. Here, we will mainly describe the parts of the T-2 protocol that differ from the T-1 protocol.

The T-2 protocol is a protocol for two principals, which we will call Alice (A) and Bob (B). They have their respective knowledge bases KB_A and KB_B. They want to determine the intersection $KB_A \cap KB_B$ without leaking any information on the items outside of the intersection $KB_A \cap KB_B$. Here we assume Alice and Bob want to mutually prove possession of the items in the intersection $KB_A \cap KB_B$, but the protocol can easily be adjusted to allow for a unidirectional proof. Also, we adopt the setting that A and B have established a shared secret nonce N, and that no encryption is used. The protocol can easily be adjusted to encryption instead of a nonce.

Both players know one another's names, A and B. They have established a common shared secret nonce N, and computed the hash $H(I_Q, N)$ for each $I_Q \in KB_Q$. The length of the hash values used in the protocol is l bits. These hash values are stored in a binary prefix tree, with the corresponding files at the leaf nodes. Since hash values have an equal length, all leaf nodes in the prefix tree are at depth l.

The basic messages that are exchanged in the T-2 protocol are listed in Table 10.2.[7] Note that sending a message of the form `refuse`(p_1) is an explicit refusal: the principal that sends this message states that he does not have any

[7] It is easy to see that the basic messages of the T-1 protocol are special cases of these messages:

- `ask`(h_1) is a special case of `ask`(p_1) where the length of p_1 is equal to l.
- `halt` is a special case `refuse`(p_1) where the p_1 is of zero length.
- `challenge`(C) is s special case of `challenge`(h_1, C) where h_1 is equal to the h_1 of the previously sent `ask`(h_1) message.
- `prove`(h_2) is a special case of `prove`(h_1, p_2) where h_1 is equal to the h_1 of the previously sent `ask`(h_1) message and the length p_2 is equal to l.

secrets with hash prefix p_1 of which he is willing to prove possession in this protocol run. Conversely, sending a message of the form $\texttt{ask}(p_1)$ is *not* an explicit confirmation: the principal that sends this message may have not a single secret with hash prefix p_1 of which is he willing to prove possession in the current protocol run.

One might think that sending a message of the form $\texttt{ask}(p_1)$ *should* be an explicit confirmation, but this is neither necessary nor sensible. It is not necessary because the only *convincing* confirmation is a $\texttt{prove}(h_1, p_2)$ message where p_2 meets certain criteria. It is not sensible because one cannot verify whether the player sending the $\texttt{ask}(p_1)$ message actually has a message with hash prefix p_1.

A run of the T-2 protocol consists of many subprotocols. There are two types of subprotocols: one for determining the approximation $KB_{AB?}$ of the intersection $KB_A \cap KB_B$ (described in Section 10.2.1), and one for performing the mutual proof of possession of the elements in the intersection $KB_A \cap KB_B$ (described in Section 10.2.2).

In the text, ϵ, p, p_Q, h, h_n and s denote binary strings, and $\cdot$ denotes string concatenation. The empty string is denoted as ϵ. When applied to numbers, $\cdot$ denotes multiplication. The length (in bits) of the hash values is l, which is the same for all hash values. The length (in bits) of the challenges is l_c, which is the same for all challenges.

10.2.1 Subprotocol for Determining Intersection

The subprotocol for determining the approximation $KB_{AB?}$ of the intersection of KB_A and KB_B takes a recursive divide-and-conquer approach. By means of messages, the domain of possible mutually owned files is divided into a number of smaller domains, and for each of these domains a new 'subprotocol' is started. By means of recursion, these domains get smaller, and eventually the players either state that they do not have files within the domain, or the domain contains only a single file. In the latter case, a subprotocol for proving possession is started (described in Section 10.2.2).

A subprotocol is started by a player sending an `ask` message, which is typically $\texttt{ask}(\epsilon)$. When a player (let us say Alice) sends a message $\texttt{ask}(p)$ with $\text{length}(p) < l$, the other player (let us say Bob) is obliged to respond with a set of messages R_p such that Bob gives a full account of the domain denoted by p. Roughly, for every part of the domain, Bob has to tell whether he has secrets with the corresponding hash prefix. The amount of detail in the response of Bob is hardly restricted. The only constraint for Bob is that if Bob does not refuse having any secrets in the domain, his description has to be *more* detailed than the question of Alice. *How much more* detailed it will be, is up to Bob to decide. (A special case of the protocol, which we will introduce shortly, is where the amount of extra detail is fixed.)

More formally, a response set is a set $R_p = \{M | M = \texttt{ask}(p \cdot s) \text{ or } M = \texttt{refuse}(p \cdot s)\}$, such that there is a set S_p of binary strings which satisfies the following properties:

$$S_p = S_p^{\texttt{ask}} \cup S_p^{\texttt{refuse}}, \tag{10.1}$$
$$S_p^{\texttt{ask}} = \{s | \texttt{ask}(p \cdot s) \in R_p\}, \tag{10.2}$$
$$S_p^{\texttt{refuse}} = \{s | \texttt{refuse}(p \cdot s) \in R_p\}, \tag{10.3}$$
$$S_p^{\texttt{ask}} \cap S_p^{\texttt{refuse}} = \emptyset, \tag{10.4}$$
$$\forall s \in S_p^{\texttt{ask}} : \text{length}(s) \geq 1, \tag{10.5}$$
$$\forall s \in S_p^{\texttt{refuse}} : \text{length}(s) \geq 0, \tag{10.6}$$
$$\forall s \in S_p : \text{length}(s) \leq (l - \text{length}(p)), \tag{10.7}$$
$$\forall s \in S_p : \neg\exists s' \in S_p : \exists s'' \in \{0,1\}^* : s = s' \cdot s'', \tag{10.8}$$
$$\forall h \in \{0,1\}^l : \exists s \in S_p : \exists s'' \in \{0,1\}^* : h = s \cdot s''. \tag{10.9}$$

Thus, the binary string p is suffixed with a number of strings s. Some of the suffixes correspond to `ask` messages (10.2), and some to `refuse` messages (10.3). There are no suffixes which are used in an `ask` message *and* in a `refuse` message simultaneously (10.4). Suffixes which are part of an `ask` message have minimum length one (10.5), and suffixes which are part of a `refuse` message have minimum length zero (10.6).

When all suffixes are taken together into S_p (10.1) the following hold: Every suffix is length-bounded by $l - \text{length}(p)$ (i.e., $\text{length}(p \cdot s) \leq l$) (10.7). There are no two suffixes such that one suffix is a prefix of the other suffix (10.8). Every binary string of length l has a prefix in S_p (10.9).

Note that within a `refuse` message, s may be of zero length, but not in an `ask` message. If an s of zero length would be allowed in an `ask` message, there would be no guarantee that the T-2 protocol would ever terminate: every `ask` message could be answered with exactly the same message, creating an infinite loop.

Because both players know what messages they send, and all messages sent are assumed to arrive, both players can detect whether the protocol has terminated, i.e., whether all obligations have been met. The most efficient protocol run possible for determining the intersection is a run in which both players only send $\texttt{ask}(p_1)$ messages if they indeed possess secrets whose hash value has the prefix p_1, and send $\texttt{refuse}(p_1)$ messages otherwise. This strategy can however not be enforced.[8]

To illustrate how a collection of subprotocols establishes the intersection of two knowledge bases, let us suppose that hash values are only four bits long, A and B have restricted themselves to strings s of length 1. Moreover, suppose that A possesses files with prefixes 0111, 1001 and 1010, and B possesses files with prefixes 0001, 1010, 1011 and 1101. These sets correspond with the sets KB_A and KB_B in Figure 10.1. Within this context, Table 10.3 shows how the protocol may develop.

[8] That is, enforcing such a strategy would result in a far less efficient protocol, while guaranteeing protocol efficiency would be the purpose of enforcing the strategy.

step	player	p	messages (R_p)	message meaning
1	A		{ask(ϵ) }	I've got some secrets whose prefix is ϵ.
2	B	ϵ	{ask(0) ,	I have got some secrets with prefix 0
			ask(1) }	and some with prefix 1.
3	A	0	{refuse(00) ,	I do not have secrets with prefix 00,
			ask(01) }	but I do have some with prefix 01;
		1	{ask(10) ,	moreover, I have secrets with prefix 10,
			refuse(11) }	but I don't have any with prefix 11.
4	B	01	{refuse(010) ,	} Sorry, no secrets with prefix 01 here,
			refuse(011) }	
		10	{refuse(100) ,	also no secrets with prefix 100,
			ask(101) }	but indeed some with 101 here.
5	A	101	{ask(1010) ,	Some secrets with prefix 1010,
			refuse(1011)}	but none with 1011 here.

TABLE 10.3: A sample run of interleaved subprotocols for establishing the intersection. In the message meaning column, 'some' should be read as 'zero or more', and 'secret with prefix' should be read as 'secret with hash value with prefix'.

It can be seen that A and B in turn increase the prefixes in their messages by one bit. Every ask(p_1) message obliges the opposing player to respond. More specifically, step 2 is a response to step 1, step 3 contains two responses to step 2, step 4 contains two responses to step 3, and finally step 5 is a response to the last part of step 4. Step 5 should lead to a subprotocol for mutually proving possession of the file with prefix 1010. In course of the protocol run, A has said she does not possess files whose hashes start with either 00, 11 or 1011, and B has said he does not possess files whose prefixes start with either 010, 011 or 100. Thus, 9/16 of the full hash domain has been refused by A, and 6/16 by B. There remains 1/16 of the domain of which neither player has refused possession, though the subprotocol for determining intersection has terminated. This means that this remaining 1/16 part of the domain denotes possible candidates $KB_{AB?}$ for actual list intersection. For each element in the remaining set, a subprotocol for proving possession must be invoked. In this case, the remaining set contains only one hash value, 1010.

The protocol run shown in Table 10.3 can also be depicted as a binary tree that grows as the protocol proceeds. Figure 10.4 shows this binary tree. The binary tree after protocol execution (the rightmost one) closely resembles the hash value prefix tree that belongs to $KB_A \cap KB_B$, shown at the right of Figure 10.1.

The protocol states shown in Figure 10.4 can also be depicted as colored surfaces, which is done in Figure 10.2. The whole surface denotes the whole set Ω. As the protocol progresses, the structure of the surface becomes finer. Light gray blocks denote parts of the domain for which Alice has sent a refuse mes-

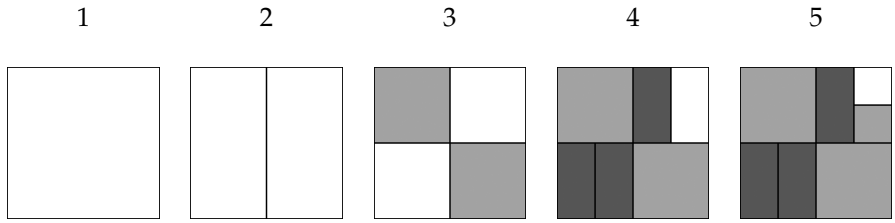


FIGURE 10.2: Interleaved subprotocols for establishing the intersection, shown as a colored surface, with $l = 4$, $|KB_A| = 3$, $|KB_B| = 4$, $|KB_A \cap KB_B| = 1$. The protocol run depicted here corresponds with the protocol run shown in Table 10.3 and Figure 10.4. See the text for explanation.

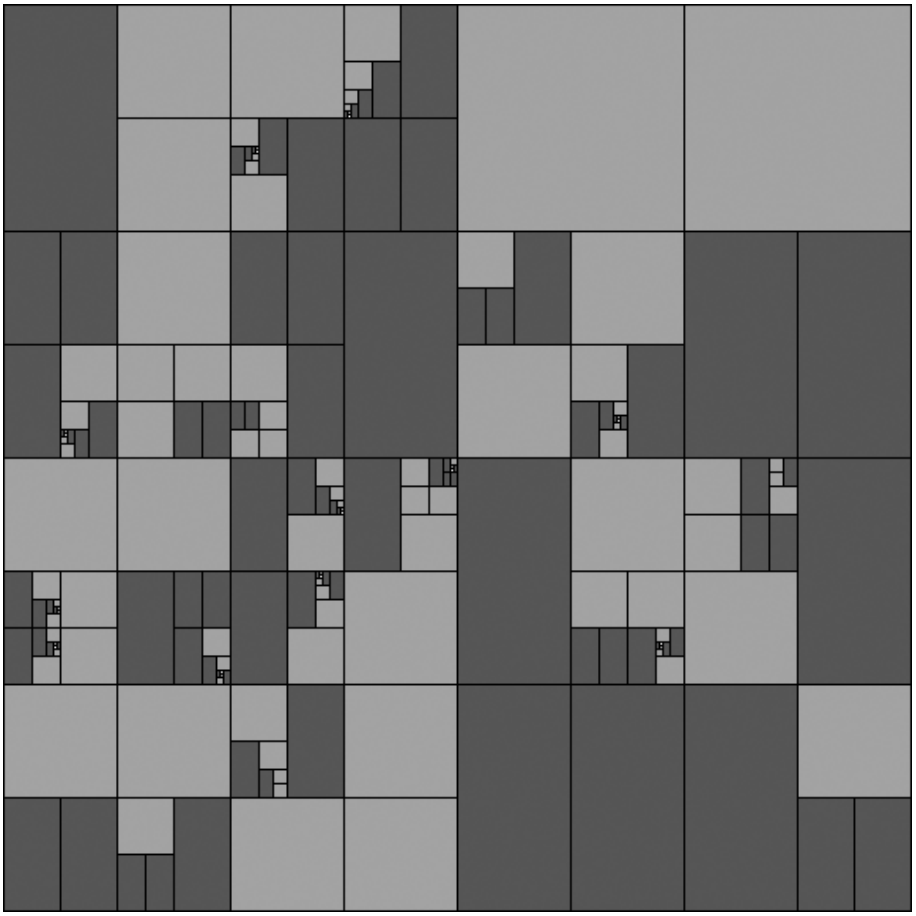


FIGURE 10.3: Interleaved subprotocols for establishing the intersection, shown as a colored surface, with $l = 16$, $|KB_A| = 40$, $|KB_B| = 40$, $|KB_A \cap KB_B| = 10$. See the text for explanation.

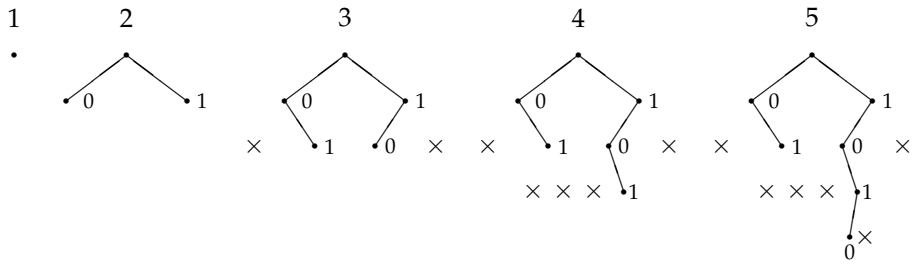


TABLE 10.4: Interleaved subprotocols for establishing the intersection, shown as a growing binary tree. The protocol run depicted here corresponds with the protocol run shown in Table 10.3. Every '×' corresponds with a `refuse` message. Every node corresponds with an `ask` message. The leftmost tree, which contains only the root node, corresponds with the protocol state after the first message ($\texttt{ask}(\epsilon)$). The rightmost tree corresponds with the protocol state after completion of the protocol, when $KB_{AB?} = \{1010\}$ has been established.

sage. Dark gray blocks represent parts of the domain for which Bob has sent a `refuse` message. White blocks represent parts of the domain for which neither Alice nor Bob has sent a `refuse` message. The sizes of the blocks correspond with the proportion of the domain that has been `refused` in the corresponding message.

For a first impression of how the protocol scales up, and how this affects the protocol state, Figure 10.3 shows the final state of a protocol where $l = 16$, $|KB_A| = 40$, $|KB_A| = 40$, $|KB_A \cap KB_B| = 10$, all suffixes s are of length 1 and all hash values are randomly chosen.[9]

While meeting their protocol obligations, the participants have a lot of freedom, of which we mention a few important aspects:

1. It is not a protocol violation to send $\texttt{ask}(p_1)$ messages if the player actually does *not* have any secret $I_V \in KB_V$ of which p_1 is a hash prefix. The player may 'act as if' he has some secrets which he in fact does not have.

2. The set R_p (sent in response to a message $\texttt{ask}(p)$) does not have to be sent at once. It may be sent in parts, interleaved with other response sets $R_{p'}$. Parts of R_p may even be sent only after the opposing player has performed some moves. This also means that the exact details of R_p can be chosen in response to the opposing player's future moves.

3. The length of the string s may be longer than 1. Thus, a player can choose to surrender multiple bits of information to the opposing player within one step.

These freedoms allow participants in the protocol to choose between a lot of different strategies. In Section 10.3 we will elaborate on various strategies.

[9] Because the hash values come from a cryptographic hash function, it is safe to assume such a random distribution.

The approximation $KB_{AB?}$ of the intersection $KB_A \cap KB_B$ is the set of hash values p for which one of the players has sent a message $\texttt{ask}(p)$ with $\text{length}(p) = l$. It is guaranteed that every secret of which both Alice and Bob are willing to mutually prove possession, has a corresponding hash value in the set $KB_{AB?}$. However, there is no guarantee that there are no bogus hash values in the set $KB_{AB?}$: hash values for which either Alice or Bob does not know a corresponding secret.[10] For every member of the set $KB_{AB?}$ a subprotocol for proving possession has to be executed to determine whether the hash value is bogus or not.

10.2.2 Subprotocol for Proving Possession

When the subprotocols for determining intersection have been completed, Alice and Bob have constructed a set $KB_{AB?}$ of hash values for which it is claimed that both Alice and Bob possess a secret with the corresponding hash values. This set can be transformed into the set $KB_A \cap KB_B$ by application of a subprotocol for proving possession to every element $h_1 \in KB_{AB?}$. If a subprotocol is convincing for a principal Q, this principal considers the secret $I \in KB_Q$ for which $h_1 = H(I, N)$ holds to be an element of the intersection $KB_A \cap KB_B$. Thus, the subprotocol for proving possession is a sifting on the set $KB_{AB?}$.

An instance of the subprotocol for proving possession is invoked when within a subprotocol for determining intersection, a message $\texttt{ask}(p)$ has been sent, with $\text{length}(p) = l$. This 'prefix' p actually contains a complete hash value h_1. This hash value h_1 refers to a unique secret I, and an instance of the subprotocol is purely dedicated to one specific secret. Essentially, the two players give one another a NP-complete puzzle which they can only solve feasibly if they indeed possess the secret. The solution is again a hash value, but now for both players the hash value is different: the solutions to the puzzles prove possession of the same file, but the puzzles and the answers themselves differ. Alice has to compute and show $h_A = H(I, N, A, C_B)$, whereas Bob has to compute and show $h_B = H(I, N, B, C_A)$.

For running an instance of the subprotocol for proving possession, a principal has to maintain seven state variables, which are listed in Table 10.5.

Every message of the subprotocol contains h_1, to distinguish the messages of the current subprotocol from other subprotocols. In the beginning of the subprotocol, C_A and C_B are exchanged without further ado. Using these challenges, both players can compute h_2^A and h_2^B. Next, the players in turn send one another prefixes of their proof messages m_A and m_B. In every step, the prefix must be longer than the prefix sent in the previous step. If in the end of the protocol it turns out that $m_A = h_A$ and $m_B = h_B$, the players have indeed mutually proven possession of the file denoted by h.

[10] In extremely rare cases, it might happen that Alice and Bob want to mutually prove possession of different secrets which have the same hash value. Such hash values are also included in $KB_{AB?}$.

variable	meaning
h_1	the hash value that denotes the secret for which the subprotocol has been invoked, $h_1 = H(I, N)$
C_A	the challenge chosen by Alice
C_B	the challenge chosen by Bob
h_2^A	the hash value that constitutes Alice's proof, $h_2^A = H(I, N, A, C_B)$
h_2^B	the hash value that constitutes Bob's proof, $h_2^B = H(I, N, B, C_A)$
p_A	the proof message disclosed by Alice
p_B	the proof message disclosed by Bob

TABLE 10.5: State variables in a subprotocol for proving possession

More formally, we arrive to the following description of the subprotocol for proving possession:

- When a principal Q sends a message `ask`(p), with $\text{length}(p) = l$, the player is obliged to also send a message `challenge`(h_1, C_Q) with $h_1 = p$. The challenge C_Q is of fixed length and its contents may be freely chosen by the sender Q.

- When a principal Q receives a message `challenge`(h_1, C), and he has not yet sent a message `challenge`(h_1, C_Q) (that is, a challenge for the same value of h_1), this player is obliged to send two messages: `challenge`(h_1, C_Q) where C_Q may be freely chosen by the sender, and `proof`(h_1, p_Q) where p_Q is a binary string with $\text{length}(p_Q) \leq l$.

- When a player receives a message `proof`(h_1, p), with $\text{length}(p) \leq l$, then he has to respond with a message `proof`(h_1, p_Q), with $p_Q = p'_Q \cdot s$, where p'_Q is the last prefix the principal Q has disclosed within the current subprotocol (or ϵ if the player has not yet sent a `proof` message within the current subprotocol). There is a required minimum length of p_Q, which depends on p. If $\text{length}(p) = l$, then p_Q should be of length l as well. Otherwise, it should be the case that $\text{length}(p_Q) > \text{length}(p)$. The subprotocol terminates when both players have sent a prefix of length l.

What actually happens within this subprotocol, is that the players in turn disclose a few bits of the value they must compute, the solution to their NP-hard puzzle. In each step, the prefix shown must be at least one bit longer than the last one shown by the opposite player.

Similar to the subprotocol for determining intersection, the hash values are disclosed in parts, more specifically in ever longer prefix strings. As in the protocol for determining intersection, it is important to appreciate what information is actually transferred within the messages. The bit strings p_A and p_B may be equal to h_2^A and h_2^B respectively, but this is not required. Of course, if these values are (pairwise) not equal, this implies that the secrets that correspond to h_1 will not be considered an element of $KB_A \cap KB_B$.

step	player	messages	message meaning
1	A	{ask(1010), challenge(1010, 0110)}	Let's run the protocol for 1010, I challenge you, $C_A = 0110$.
2	B	{challenge(1010, 1110), proof(1010, 1)}	I challenge you, $C_B = 1110$, the first bit of my proof is 1.
3	A	{proof(1010, 01)}	The first two bits of my proof are 01.
4	B	{proof(1010, 110)}	The first three bits of my proof are 110.
5	A	{proof(1010, 0100)}	The four bits of my proof are 0101.
6	B	{proof(1010, 1101)}	The four bits of my proof are 1101.

TABLE 10.6: A sample run of the subprotocol for proving possession. The hash value secret for which possession can be proven in this protocol is 1010. For completeness, the ask message leading to the protocol is included in the first step.

To illustrate how a single subprotocol works, let us again suppose that hash values are only four bits long, and A has sent ask(1010) to B. Table 10.6 shows how the protocol may develop. At the end of the protocol run the full 'proofs' p_A and p_B are known: $p_A = 0100$ and $p_B = 1101$. Whether these 'proofs' are convincing depends on whether $p_A = h_2^A$ and $p_B = h_2^B$.

Again, the principals have a lot of freedom in how they act in the protocol:

1. The players are not obliged to actually send prefixes of the actual proof of possession; they are allowed so send any information they like, as long as they adhere to the syntactical rules.

2. The players may increase the length of their prefixes faster than the minimum requirement.

These options leave room for many strategies. Assuming that the players want to prove possession of the secret to one another and want to make sure they get a reciprocal proof in return, there is an optimal strategy which is very simple. As soon as both challenges are known, compute h_A and h_B. As long as the other player's shown prefix is a prefix of the proof he should send, reply 'truthfully' by sending the prefix of your own proof, which is only one bit longer than the other player's last prefix. As soon as the other player's shown prefix is not a prefix of the expected proof, stop sending parts of your own proof, but append random noise to your previously sent prefixes until the protocol terminates.

This strategy ensures that if the other player chooses to abort the protocol, the other player only has an advantage of one bit. Also, if the other player does not know the secret I_V, your proof of possession of the secret is not communicated to the other player (or at most one bit of it). When both players adopt this strategy, they will obtain a mutual proof when both players possess the secret I_V in question. As such, this strategy guarantees *fairness* (see page 23).

10.3 Making the Protocol Efficient by Restrictions

The protocol description leaves a lot of freedom to principals participating in the protocol (see the lists on pages 153 and 156). This means that the principals can develop and use various strategies while engaging in the T-2 protocol. For example, a *reluctant* strategy is to never send `refuse` messages. A *cooperative* strategy is to send send as many `refuse` messages as possible, while 'merging' `refuse` messages of adjacent prefixes.[11]

Due to this large amount of freedom in the protocol specification it is impossible to give precise measurements of the communication complexity of the protocol. Moreover, the freedom itself increases the communication complexity: many possible protocol actions imply the need for many bits to encode one single protocol action. In this section, we will impose restrictions on the protocol that allow (1) efficient encoding of the protocol actions and (2) precise computations and measurements of the communication complexity (given in Section 10.4).

The restrictions we impose are the following:

1. It is assumed Alice starts the protocol with the message $\texttt{ask}(\epsilon)$
2. In the subprotocols for determining intersection:
   (a) All suffixes s are of length 1.
   (b) All sets R_p are sent in order of increasing length of p.
   (c) All sets R_p with equal length of p are sent in the lexicographical order of p.
3. In the subprotocols for proving possession:
   (a) The challenges are sent as soon as possible in the lexicographical order of h_1.
   (b) For every h_1, the first $\texttt{proof}(h_1, p_Q)$ message, p_Q is of length 1. These first proofs are sent as soon as possible, but after all challenges have been sent.
   (c) All suffixes s are of length 2, except the last suffix, which is of length 1.
   (d) All messages $\texttt{proof}(h_1, p_Q)$ are sent in order of increasing length of p_Q.
   (e) All messages $\texttt{proof}(h_1, p_Q)$ with equal length of p_Q are sent in the lexicographical order of h_1.

[11] That is, it is not the sheer number of messages that counts, but the proportion of the domain Ω for which a principal sends `refuse` messages. If this proportion is maximal, the strategy is called *cooperative*.

messages (R_p)	encoding
$\{\texttt{refuse}(p \cdot 0), \texttt{refuse}(p \cdot 1)\}$	00
$\{\texttt{refuse}(p \cdot 0), \texttt{ask}(p \cdot 1)\}$	01
$\{\texttt{ask}(p \cdot 0), \texttt{refuse}(p \cdot 1)\}$	10
$\{\texttt{ask}(p \cdot 0), \texttt{ask}(p \cdot 1)\}$	11

TABLE 10.7: Encoding for sets R_p where $\forall s\colon |s| = 1$ and p may be omitted. Every `refuse` is encoded as a 0, and every `ask` as a 1.

These restrictions have a huge impact on the protocol runs: The first message message can be omitted (1). Bob (Alice) sends only sets R_p where the length of p is odd (even) (2a). The principals send all their sets R_p out as soon as possible (2b), which reduces the number of communication steps for determining intersection down to l. The principals send their sets R_p in a strictly imposed order (2c).

The challenges are sent in a strictly imposed order (3a). Bob (Alice) sends only proofs p_A (p_B) where the length of p_A (p_B) is odd (even) (3b and 3c). The principals send all their proofs p_Q out as soon as possible (3b and 3d), which reduces the number of communication steps for proving possession down to $l + 1$. The principals send their proofs p_Q in a strictly imposed order (3e).

As a result, the total number of communication steps in a protocol run where possession is proven is $2 \cdot l + 1$. If the set $KB_{AB?}$ is empty, the total number of communication steps is at most l. Moreover, as a result of the imposed order, it is always possible to reconstruct for which prefix p or hash value h_1 a message is bound to arrive. Thus, sending the prefix p or the hash value h_1 itself is redundant.

In the subprotocols for determining intersection, the suffixes s are of length 1, and the prefix p can be omitted from the message sets R_p. As a result, we can encode every set R_p in only two bits. This is shown in Table 10.7.

In the subprotocols for proving possession, h_1 can also be omitted because of the ordering of the messages. Also, sending the full prefix p_Q in every message is redundant, because a large part of the prefix has already been sent in a previous message. It is only needed to send the suffix s.

Using these coding schemes, the protocols shown in Tables 10.3 and 10.6 can be merged into one single protocol run, shown in Table 10.8. It is a run of the restricted T-2 protocol on the sets $KB_A = \{0111, 1001, 1010\}$ and $KB_B = \{0001, 1010, 1011, 1101\}$. The principals are cooperative, they send as many `refuse` messages as possible. The column 'message' denotes the actual communicated bits. Observe that from the bits in the 'message' column, it is possible to reconstruct the columns p and h_1, and with help from Table 10.7 it is possible to reconstruct the decoded messages. Moreover, observe that the first twelve bits communicated in the protocol (110110000110) correspond exactly to the binary encoding (explained in Section 10.1) of the prefix tree that is constructed in the protocol (shown at the right of Figure 10.4).

step	player	p	h_1	message	decoded message
0	A				$\{\texttt{ask}(\epsilon)\}$
1	B	ϵ		11	$\{\texttt{ask}(0), \texttt{ask}(1)\}$
2	A	0		01	$\{\texttt{refuse}(00), \texttt{ask}(01)\}$
		1		10	$\{\texttt{ask}(10), \texttt{refuse}(11)\}$
3	B	01		00	$\{\texttt{refuse}(010), \texttt{refuse}(011)\}$
		10		01	$\{\texttt{refuse}(100), \texttt{ask}(101)\}$
4	A	101		10	$\{\texttt{ask}(1010), \texttt{refuse}(1011)\}$
			1010	0110	$\texttt{challenge}(1010, 0110)$
5	B		1010	1110	$\texttt{challenge}(1010, 1110)$
			1010	1	$\texttt{proof}(1010, 1)$
6	A		1010	01	$\texttt{proof}(1010, 01)$
7	B		1010	10	$\texttt{proof}(1010, 110)$
8	A		1010	00	$\texttt{proof}(1010, 0100)$
9	B		1010	1	$\texttt{proof}(1010, 1101)$

TABLE 10.8: A sample protocol run of the restricted T-2 protocol, efficiently encoded. Only the bits in the column 'message' are communicated. From it, the column 'decoded message' can be reconstructed.

10.4 Determining Communication Complexity

Now that we have fully explained the T-2 protocol and its restricted version, we want to establish the communication complexity of the restricted T-2 protocol. Some steps in this process are simple, some are rather complicated. Our approach is simple: we compute the simple parts of the complexity, and we perform some experiments to estimate the complicated parts of the complexity.

Although in the previous section, we have restricted the freedom of the principals dramatically, the principals still have room for various strategies. We have already mentioned the two most important strategies: the *cooperative* and the *reluctant* strategy. When applied to the restricted T-2 protocol, these strategies are implemented as follows:

cooperative Choose the sets R_p in such a way that the number of ones in the encoding of R_p shown in Table 10.7 is minimized. This leads to an intersection prefix tree of minimal size.

reluctant Choose the sets R_p in such a way that the number of ones in the encoding of R_p shown in Table 10.7 is maximized. This leads to an intersection prefix tree of maximal size.

Both Alice and Bob can independently choose their strategy. There are more strategies than the cooperative and the reluctant strategies, but all other strategies fall complexity-wise 'in between' the complexities of these two strategies.

strategy		upper bound		
Alice	Bob	size of the binary rep. of the prefix tree	$\lvert KB_{AB?}\rvert$	communication complexity (bits)
coop.	coop.	$2 \cdot l \cdot \min(\lvert KB_A\rvert, \lvert KB_B\rvert)$	$\lvert KB_A \cap KB_B\rvert$	$2 \cdot l \cdot \min(\lvert KB_A\rvert, \lvert KB_B\rvert)$ $+2 \cdot (l + l_c) \cdot \lvert KB_A \cap KB_B\rvert$
coop.	rel.	$4 \cdot \lvert KB_A\rvert \cdot l$	$2 \cdot \lvert KB_A\rvert$	$2 \cdot \lvert KB_A\rvert \cdot (3 \cdot l + l_c)$
rel.	coop.	$4 \cdot \lvert KB_B\rvert \cdot l$	$2 \cdot \lvert KB_B\rvert$	$2 \cdot \lvert KB_B\rvert \cdot (3 \cdot l + l_c)$
rel.	rel.	$2 \cdot (2^l - 1)$	2^l	$2^{l+1} \cdot (1 + l + l_c) - 2$

TABLE 10.9: The worst case communication complexity for the restricted T-2 protocol, depending on the strategies of Alice and Bob. The communication complexity, shown at the right, is the size of the binary representation of the prefix tree plus $2 \cdot (l + l_c) \cdot \lvert KB_{AB?}\rvert$.

Therefore, it is sufficient to analyze these two strategies to form an impression of how the communication complexity depends on the strategies chosen.

- If both Alice and Bob use the cooperative strategy a prefix tree of minimal size is constructed. In the case of the running example of this chapter, this corresponds to the tree shown at the right of Figure 10.4.
- If both Alice and Bob use the reluctant strategy, a prefix tree spanning the full domain Ω is constructed. In the case of the running example of this chapter, this corresponds to the tree shown at the left of Figure 10.1. The size of this tree in binary encoding is $2 \cdot (2^l - 1)$.
- If Alice uses the cooperative strategy and Bob the reluctant strategy, a prefix tree is constructed that closely matches the hash value prefix tree of Alice. In the case of the running example of this chapter, this corresponds to the tree shown under KB_A in Figure 10.1. The size of this tree in binary encoding is bounded by $4 \cdot \lvert KB_A\rvert \cdot l$.
- The case where Alice uses the reluctant strategy and Bob uses the cooperative strategy is symmetric to the previous case.

The total communication complexity of a protocol run is the sum of the communication complexities of the subprotocols for determining intersection and the subprotocols for proving possession. The former is precisely the size of the binary encoding of the constructed tree; the latter is precisely $2 \cdot (l + l_c) \cdot \lvert KB_{AB?}\rvert$, where l is the length of the hash values in bits, and l_c is the length of the challenges in bits.

The worst case communication complexities can be calculated rather easily. The results are shown in Table 10.9. The only case for which the worst case communication complexity is not so trivial is the case where both principals use the cooperative strategy. The biggest prefix tree that can be constructed in this setting occurs in case the prefix trees corresponding to KB_A and KB_B overlap almost completely, in which case the size of the prefix tree of the intersection is just a little below $2 \cdot l \cdot \min(\lvert KB_A\rvert, \lvert KB_B\rvert)$. This is however extremely

unlikely to actually happen, as the prefix trees belonging to the sets KB_A and KB_B have a uniform random distribution.

For the settings where at least one of the principals uses the reluctant strategy, the average communication complexity is equal to the worst case communication complexity. The average case communication complexity for the case where both principals use the cooperative strategy can be expected to be much lower than the worst case.

Observe that both principals can force the communication complexity to be at most $2 \cdot |KB_Q| \cdot (3 \cdot l + l_c)$ by using the cooperative strategy.

The average case communication complexity for the case where both principals use the cooperative strategy can be derived mathematically, but this is very complicated.[12] So instead, we did some experiments to estimate the communication complexity in this setting.

The communication complexity for the T-2 protocol consists of two contributions:

- *communication resulting from secrets that* are *shared*
  This communication is heavily influenced by $|KB_{AB?}|$. For cooperative principals $|KB_{AB?}|$ will be equal[13] to $|KB_A \cap KB_B|$.
  - *communication in subprotocols for determining intersection*
    This is bounded by $2 \cdot l \cdot |KB_{AB?}|$.
  - *communication in subprotocols for proving possession*
    This is exactly $2 \cdot (l + l_c) \cdot |KB_{AB?}|$ bits.
- *communication resulting from secrets that* are not *shared*
  This is only communication in the subprotocols for determining intersection. We estimated this experimentally.

When these contributions are added up, there will be a little bit of double counting, as some communication in the subprotocols for determining intersection is due to both shared secrets and not-shared secrets.

To estimate the communication resulting from secrets that are not shared, we performed an experiment in ten different conditions. The conditions differ in the sizes of the sets KB_A and KB_B, and these are shown in Table 10.10. In every condition, $l = 256$ and $KB_A \cap KB_B = \emptyset$. The hash values corresponding to the sets KB_A and KB_B were taken randomly from the domain 2^l where every element had an equal probability. For every condition, the experiment was performed 1000 times, and the number of bits communicated is recorded.

The results of the experiment are shown in Figure 10.4 and Table 10.11. Table 10.11 reports for each of the conditions the minimum and maximum observations, the median, the average and the standard deviation, and also the

[12] We have not yet succeeded in establishing a formula which expresses the communication complexity in which we have sufficient confidence. Our gratitude goes to various people who have tried to help us in finding this formula, most notably to Gerard te Meerman.

[13] There is a negligible chance that $|KB_{AB?}|$ is larger than $|KB_A \cap KB_B|$; in that case one or more *collisions* of the hash function must have occurred.

condition	$\|KB_A\|$	$\|KB_B\|$	$\frac{\|KB_A\|}{\|KB_B\|}$
1	1	1	1
2	1	10	0.1
3	1	100	0.01
4	1	1000	0.001
5	10	10	1
6	10	100	0.1
7	10	1000	0.01
8	100	100	1
9	100	1000	0.1
10	1000	1000	1

TABLE 10.10: The ten conditions of the experiment to estimate the average communication complexity of the restricted T-2 protocol with cooperative principals. Every condition can be identified by two of the three variables $|KB_A|$, $|KB_B|$ and $\frac{|KB_A|}{|KB_B|}$.

bounds of the interval in which the middle 95% of the observations lie. Figure 10.4 is a density (local frequency) plot of the data.[14]

At first glance, the only conclusion that can be drawn is that larger sets lead to more communication. At closer observation, one can see that there are peaks at approximately 5,5, 55, 550 and 5500 bits that correspond to conditions where $\frac{|KB_A|}{|KB_B|} = 1$, in increasing order of $|KB_A|$. Similarly, there are peaks at approximately 13,6, 136 and 1360 bits that correspond to conditions where $\frac{|KB_A|}{|KB_B|} = 0.1$. Also, there are peaks at approximately 23,4 and 234 bits that correspond to conditions where $\frac{|KB_A|}{|KB_B|} = 0.01$. This suggests two findings:

1. An increase of a factor 10 in the sizes of both $|KB_A|$ and $|KB_B|$ leads to an increase of a factor 10 of the communicated bits.

2. The fraction $\frac{|KB_A|}{|KB_B|}$ influences the number of communicated bits.

To investigate these hypotheses, we divide the communication by the sum of the set sizes, which gives us Figure 10.5 and Table 10.12. Figure 10.5 is a density plot[15] of the communicated bits divided by $|KB_A|+|KB_B|$, and Table 10.12 gives descriptive statistics of the data (similar to Table 10.11). The conditions have been re-ordered to highlight the structure that can be seen in Figure 10.5.

The findings are very clear:

[14] Technically, Figure 10.4 is not a density plot. It is a density plot of the base 10 logarithm of the observations with modified labels at the x-axis. The labels at the x-axis are 10^x where it should technically read x. In this way, the data plotted is easy to read, while the 'visual surface' for each distribution is equal.

[15] As opposed to Figure 10.4, Figure 10.5 is a 'true' density plot. The x-axis is in linear scale, and the surfaces below the lines are both 'visually' and mathematically equal (i.e., equal to 1).

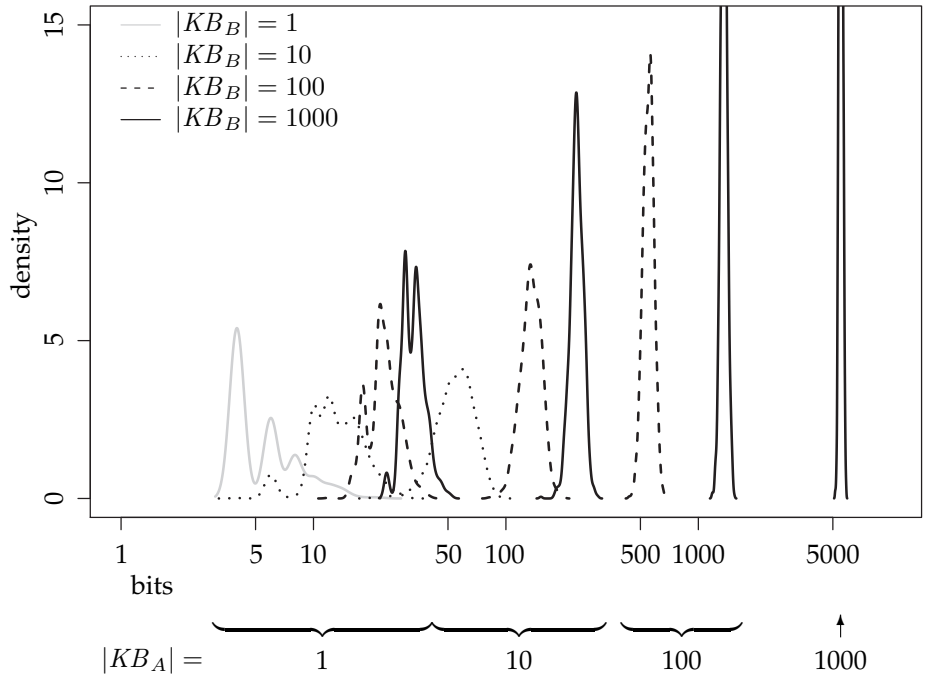


FIGURE 10.4: The number of communicated bits in the restricted T-2 protocol with cooperative participants, shown as a compressed density plot.[14] For every distribution, $|KB_A|$ can be found by finding its peak, and looking straigt down to where either a brace or an arrow is found. At the other side of the brace or arrow, $|KB_A|$ is printed.

$\|KB_A\|$	$\|KB_B\|$	min	-95	med	$+95$	max	avg	stdev
1	1	4	4	4	14	22	5,97	2,77
1	10	4	6	12	24	30	13,67	4,24
1	100	12	16	22	34	40	23,47	4,40
1	1000	24	24	34	44	52	33,38	4,61
10	10	20	34	56	78	94	55,67	11,75
10	100	82	104	136	168	196	135,82	16,95
10	1000	152	200	234	268	300	233,84	18,25
100	100	438	484	552	620	660	551,16	35,26
100	1000	1180	1256	1356	1470	1534	1358,29	54,01
1000	1000	5142	5298	5510	5736	5860	5508,95	110,98

TABLE 10.11: Descriptive statistics of the number of communicated bits in the restricted T-2 protocol with cooperative participants. For every condition, 1000 experiments were done. Shown are the minimum and maximum observations (min, max), the bounds of the interval where the middle 95% of the observations lie (-95, $+95$), the median andaverage (med, avg), and the standard deviation (stdev).

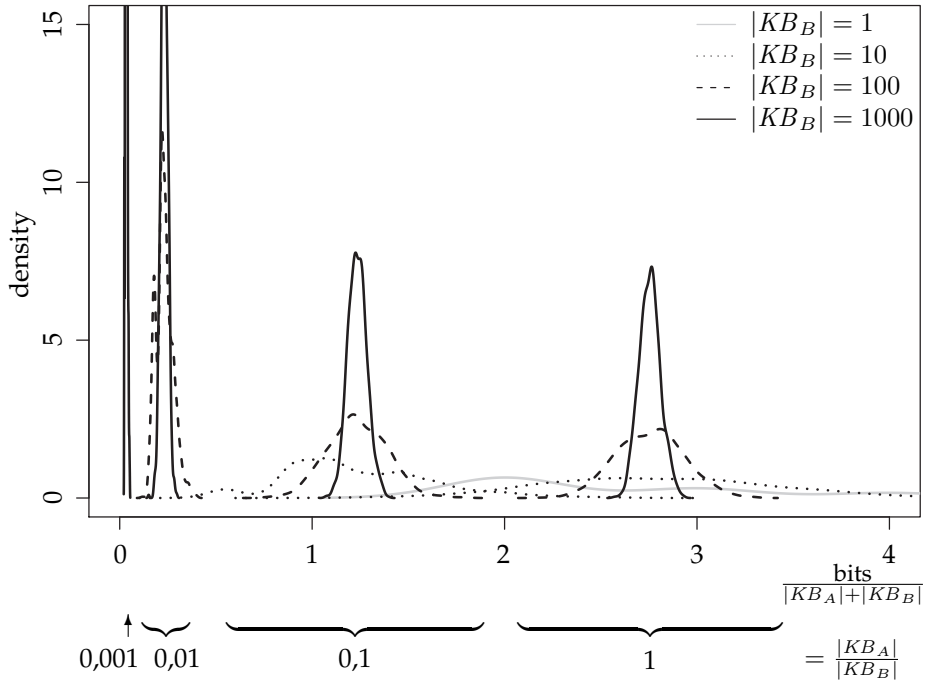


FIGURE 10.5: The number of communicated bits per compared secret in the restricted T-2 protocol with cooperative participants, shown as a density plot. For every distribution, $\frac{|KB_A|}{|KB_A|}$ can be found looking straight beneath its peak.

$\frac{\lvert KB_A\rvert}{\lvert KB_B\rvert}$	$\lvert KB_B\rvert$	min	-95	med	$+95$	max	avg	stdev
0,001	1000	0,024	0,024	0,034	0,044	0,052	0,034	0,005
0,01	100	0,119	0,158	0,218	0,317	0,396	0,232	0,044
0,01	1000	0,151	0,198	0,232	0,265	0,297	0,232	0,018
0,1	10	0,364	0,545	1,091	2,182	2,727	1,243	0,386
0,1	100	0,745	0,945	1,236	1,527	1,782	1,234	0,154
0,1	1000	1,073	1,142	1,233	1,337	1,395	1,235	0,049
1	1	2,000	2,000	2,000	7,000	11,000	2,986	1,386
1	10	1,000	1,700	2,800	3,900	4,700	2,783	0,588
1	100	2,190	2,420	2,760	3,090	3,300	2,756	0,176
1	1000	2,571	2,649	2,755	2,868	2,930	2,755	0,055

TABLE 10.12: Descriptive statistics of the number of communicated bits per compared secret in the restricted T-2 protocol with cooperative participants. For every condition, 1000 experiments were done. Shown are the minimum and maximum observations (min, max), the bounds of the interval where the middle 95% of the observations lie (-95, $+95$), the median and average (med, avg), and the standard deviation (stdev).

protocol	upper bound on average communication complexity (bits)
iterated T-1	$(l+1)\cdot\|KB_A\| + (2\cdot l + 2\cdot l_c)\cdot\|KB_A \cap KB_B\|$
restricted T-2 with cooperative principals	$2,76\cdot\|KB_A \cup KB_B\| + (4\cdot l + 2\cdot l_c)\cdot\|KB_A \cap KB_B\|$

TABLE 10.13: Bounds on average communication complexities of the T-1 and the T-2 protocol.

1. For every fraction $\frac{|KB_A|}{|KB_B|}$, the average amount of communicated bits per $|KB_A|+|KB_B|$ is practically identical. In the worst case, where $\frac{|KB_A|}{|KB_B|} = 1$, the average communication complexity is approximately $2,76$ bits per $|KB_A| + |KB_B|$.

2. When the fraction $\frac{|KB_A|}{|KB_B|}$ decreases (i.e., the difference between $|KB_A|$ and $|KB_B|$ grows), the average amount of communicated bits per $|KB_A| + |KB_B|$ decreases.

3. As $|KB_B|$ (and $|KB_A|$) grow larger, the standard deviation clearly declines.

This is good news. The first finding essentially means that the communication complexity of the restricted T-2 protocol with cooperative principals is linear in $|KB_A|+|KB_B|$. The second finding means that if $|KB_A|$ and $|KB_B|$ are of dissimilar size, the efficiency of the protocol increases; the average communication complexity is always below (approximately) $2,76\cdot 2\cdot\max(|KB_A|,|KB_B|)$. The third finding entails that the protocol scales up very well: as the set sizes increase the actual communication complexity for a particular run will be very close to the expected communication complexity (which is linear in $|KB_A| + |KB_B|$).

Table 10.13 gives a first impression of the average communication complexity of the restricted T-2 protocol with cooperative principals. Also, the communication complexity of the alternative, iteration of the T-1 protocol, is shown.

For the T-1 protocol, the actual communication complexity is exactly the formula given. The principals could still optimize the complexity, if they first establish whether $|KB_A|$ or $|KB_B|$ is smaller, and change roles in case $|KB_A| > |KB_B|$. With this optimization, the term $(l+1)\cdot|KB_A|$ can be replaced with $(l+1)\cdot\min(|KB_B|,|KB_B|)$, at the cost of an extra term that signifies the communication complexity of the protocol that determines whether $|KB_A| > |KB_B|$ is the case.

For the restricted T-2 protocol with cooperative principals, with $|KB_A| = |KB_B|$ the average communication complexity is slightly lower than the formula given (due to double counting of some communicated bits). If $|KB_A| \neq |KB_B|$, the average communication complexity improves even more. When $\frac{\min(|KB_A|,|KB_B|)}{\max(|KB_A|,|KB_B|)} = 0,1$, the factor 2,76 reduces to approximately 1,24. When

$\frac{\min(|KB_A|,|KB_B|)}{\max(|KB_A|,|KB_B|)} = 0,01$, the factor 2,76 reduces to approximately 0,23. The precise relation between the fraction and the factor remains subject to future research. Nevertheless, it is not too hard to see that the factor decreases so fast that the average communication complexity of the T1 protocol is always larger than the average communication complexity for the restricted T-2 protocol with cooperative principals.[16]

10.5 Conclusion

In this chapter, we have generalized the T-1 protocol into the T-2 protocol that does many-to-many intersection. Thus, using the T-2 protocol, the following is possible, in an efficient way[17]:

- Airline carriers can allow the authorities of (for example) the United States of America to check whether wanted terrorists are on board of the airplane without disclosing the identities of the non-criminals.
- Police officers can compare the electronic dossiers of their investigations without relying on a trusted third party (i.e., the VROS, see Section 1.5).
- Intelligence agencies can compare their secrets without disclosing them.

In cases where the set sizes $|KB_A|$ and $|KB_B|$ are public information, the T-2 protocol is just as secure as the T-1 protocol. The T-2 protocol works by means of two principals disclosing hash value prefixes, in increasing length. In this way, they can efficiently and adaptively survey the complete domain of possible secrets.

The T-2 protocol leaves room for various strategies to be used by the principals. Due to the security of the protocol, there is no informational benefit in choosing one strategy over the other. The chosen strategy influences the communication complexity. If the T-2 protocol is restricted in a particular way, efficient coding is used, and both principals use a particular strategy, then the protocol satisfies the *fairness condition*.

[16] There are two ways in which this can be seen:

1. If iterated T-1 would be faster than T-2, this would mean that in the restricted T-2 protocol with cooperative principals, the branches of the tree that correspond with the KB_A would 'escape' the intersection prefix tree at a depth greater than l. That would mean that many collisions of the cryptographic hash function would occur. The chance of this happening is negligible, and is certainly nowhere close to average case behavior.
2. Observe that the communication complexity of the T-2 protocol where Alice uses the cooperative strategy and Bob the reluctant strategy (shown in Table 10.9), is equal to the communication complexity for the case where both principals use the cooperative strategy and $KB_B = \Omega$ (Bob holds the whole domain). That communication complexity, $2 \cdot |KB_A| \cdot (3 \cdot l + l_c)$ is only larger than $(l+1) \cdot |KB_A| + (2 \cdot l + 2 \cdot l_c) \cdot |KB_A \cap KB_B|$ by a constant factor. (If $l = l_c$, the factor is $8/5$). But as KB_B is only a sparse subset of Ω, this factor will be defeated.

[17] See Section 8.1 for a detailed description of these application areas.

Let us see how the T-2 protocol works:

1. *How are the individual secrets of the players protected?*

   The T-2 protocol is a parallel composition of the T-1 protocol. See Section 9.5 for a summary of why the T-1 protocol protects the secrets of the players.

2. *Is the number of secrets that a player possesses hidden from the other player?*

   No. If a player wants to infer how many secrets the other player has, and the other player plays the *cooperative* strategy[18], then the first player can play the *reluctant* strategy to find out how many secrets the other player has. However, when both players play the reluctant strategy, the communication complexity (and thus the run time) of the T-2 protocol is exponential.

   In the analysis of the T-2 protocol, one should consider the set sizes $|KB_A|$ and $|KB_B|$ to be public knowledge.

3. *How can the players enforce fairness?*

   The proofs of possession are disclosed bit by bit, turn by turn. Both players know which bits should be sent by the other player. As soon as the bit sent by the other player is different from the expected bit, a player stops sending his own proof, and starts sending random noise bits. In this manner, the advantage that a player can get over the other player is limited to one bit of the proof.

4. *How is the communication complexity optimized?*

   The hash values of the secrets that are only known to one player, are only partially communicated. Of these hash values, only a prefix is communicated that is just long enough for the 'ignorant' player to determine he does not possess a secret corresponding to the hash value prefix.

   Moreover, all hash values and prefixes are compressed by representing them in a tree structure.

5. *How many communication steps are required?*

   When the players have a secret in common, the number of communication steps is $2 \cdot l + 1$, where l is the length of the hash values in bits. When the players have no secrets in common, the number of communication steps depends on the number of secrets they possess; in practice the number of communication steps will be much smaller than l.

6. *How many bits need to be communicated?*

   For every secret that is possessed by only one player, on average at most three bits are communicated. For every secret that is possessed by both players, $4 \cdot l + 2 \cdot l_c$ bits are communicated, where l is the length of the hash values in bits, and l_c is the length of the challenge.

[18] The cooperative and reluctant strategy are explained in Section 10.4.

Part V

Conclusion

Chapter 11

Conclusion

In this thesis, research is reported that is of both fundamental and practical value. Fundamentally, we show that a number of theoretical problems can be solved. Our solutions to these fundamental problems can be applied to resolve practical real-world problems.

In the following sections, we will summarize our results and their relevance. Directions for future research are also given.

11.1 Information Designators

Linking information which stems from various sources, information integration, is by itself a difficult problem. Even within 'relaxed' conditions where no data has to be kept secret, linking existing information sources consistently and reliably is hard. In practice, information integration does not enjoy relaxed conditions: the information is differently encoded, inconsistent and asynchronously updated. To cope with these conditions, current techniques for information integration take essentially the approach to expose as much information as deemed possibly helpful.

If information has to be kept secret, however, it seems that one faces the choice between either not integrating the information, or sacrificing confidentiality. Of course, if one has a secret, the best way to keep it secret is to tell it to nobody. But if one *has* to tell it to some people, one would certainly like those people to protect the secret. Current information integration technology cannot offer such guarantees.

In Chapter 7 we analyze this problem and identify two causes for this problem:

1. When information is integrated across databases, this is done by literally

copying information from one database to the other (or a process which is to some extent equivalent to this). We call this the *raw data* problem.

2. The ontologies of the databases that are to be integrated overlap. With the integration of the ontologies, insufficient care is taken to identify which contributor 'owns' (is responsible for) the information that overlaps.

We propose a solution to these problems which may seem very simple, but has in effect intricate, if not dramatic effects. The solution is to *never* replicate raw data, and to *always* refer to the original author ('owner') of the information. When phrased in a slogan, it becomes:

> Don't propagate, but link!

Instead of using raw data, *information designators* are used. An information designator is a pseudonym for a piece of information. Owners of information may use any number of pseudonyms for any piece of information. They can precisely control the extent to which others can use the information designators to reason about and recombine the information.

The information designator is a new concept, and is not an instantly applicable technique. But as a proof of concept, it demonstrates that linking information and protecting it against dissemination can go hand in hand.

The information designator approach needs extension in future research in the following ways:

- The information designator has to be fleshed out. Prototype production systems have to be built in order to reveal yet unknown intricacies of the approach. Basic understanding of precisely what bottlenecks will appear when the information designator systems are scaled up have to be identified and addressed.
- For practical 'real-world' deployment, an elegant way has to be found for incorporating information that is not stored using the information designator approach.

11.2 Knowledge Authentication

When one wants to compare two pieces of information, it may seem that it is necessary to have the two pieces of information available at hand. Consider the following problem "Comparing Information Without Leaking It" (CIWLI) [FNW96]:

> Two players want to test whether their respective secrets are the same, but they do not want the other player to learn the secret in case the secrets do not match.

In Chapter 8, we identify a number of variations of the problem, depending on the following properties:

- How untrustful and untrustworthy are the players? (i.e., what *adversary model* is appropriate?)
- How many *possible secrets* exist? (i.e., what is the domain size $|\Omega|$?)
- How many secrets need to be compared? Just one secret against one other secret (*1-to-1*), one secret against many other secrets (*1-to-many*), or many secrets against many secrets (*many-to-many*)?
- Does 'secret' mean that it is difficult to guess the string that represents secret (as with the string 'arkjjhhg bwr ufkng'), or does it mean that the player has attributed some stance to a commonly known string? (as with 'I voted for Pim Fortuyn'). We call the former *CIWLI without reference*, and the latter *CIWLI with reference*.

We argue that for CIWLI, the only appropriate adversary model is the *malicious adversary model*. In other adversary models, the adversary may infer the secret by using a feasible amount of computation power. Of the many protocols for CIWLI that exist in literature, only a few use the malicious adversary model.

We observe that for all protocols for CIWLI that exist in literature, the communication complexity (measured in bits) contains a factor $\ln|\Omega|$ or worse, which renders these protocols infeasible for comparing secrets from large domains (for example domains containing all possible files which are 16 megabyte large).

We present two new protocols, T-1 and T-2. Both protocols assume a malicious adversary, and solve CIWLI without reference. The term Ω does not occur in their communication complexity, which means that the protocols remain feasible when the domain of possible secrets Ω is huge.

The T-1 protocol, presented in Chapter 9, solves the 1-to-many case, with a communication complexity of only $O(1)$. We prove the T-1 protocol correct using an extended version of GNY logic.

The T-2 protocol, presented in Chapter 9, solves the many-to-many case, and can be seen as a parallel composition of the T-1 protocol. It has an average case communication complexity of

$$c_1 \cdot |KB_A \cup KB_B| + c_2 \cdot |KB_A \cap KB_B|$$

where KB_A (KB_B) is the set of secrets possessed by player A (B), the constant c_1 has an upper bound $c_1 < 3$ and the constant c_2 depends on chosen security parameters. This is particularly efficient, as every extra secret of A or B (which is not mutually shared) results in a communication increase of on average less than three bits. This complexity has not yet been formally derived, but experiments point strongly to the above relation.

In future research on knowledge authentication, the following issues need to be addressed:

- The theoretical framework of knowledge authentication may benefit from further development and consolidation.
- The T-1 protocol is currently only formally analyzed using our extended version of GNY logic. Appropriate would be additional analysis using other methodologies, such as strand spaces [THG98, THG99], spi calculus [AG99] or Datta-Derek-Mitchell-Pavlovic logic [DDMP03].
- The communication complexity of the T-2 protocol has been established experimentally. Though the results point in a very positive direction, they do not provide strong formal guarantees. For formal guarantees on the communication complexity, the communication complexity has to be formally derived.

11.3 Hash Functions and Authentication Logics

Both knowledge authentication and information designators use cryptographic hash functions in new, unprecedented ways. In common applications of cryptographic hash functions, the pre-image of a particular hash value is not considered to be secret. In our applications, the pre-image is often secret, while the corresponding hash value is not secret.

For our applications, we need cryptographic hash functions which satisfy an uncommon property, namely that they are *non-incremental*. A cryptographic hash function is non-incremental, if it is always necessary to have the full pre-image at hand to compute the hash value of this pre-image. None of the current standard cryptographic hash functions is non-incremental, but one can construct a non-incremental cryptographic hash function quite easily from any Merkle-Damgård cryptographic hash function, such as SHA-512.

At various places in the literature, it is assumed that the possession of a hash value counts as a proof of the corresponding pre-image, but this not the case. We show that BAN logic, a highly influential method for analyzing security protocols, relies on this false assumption. As a result, BAN logic is not 'sound': it is possible to derive false beliefs from true ones. As such, we demonstrate that properly modeling cryptographic primitives can be very difficult.

We extend GNY logic, a particular authentication logic, to properly model cryptographic hash functions. We prove correctness of the T-1 protocol using GNY logic.

The following issues with cryptographic hash functions require future research:

- The concept of *non-incrementality* for cryptographic hash functions is in need of a formal definition. Given that the formal definition of a cryptographic hash functions itself is already rather cumbersome (see Sections 3.2 and 3.3), the exercise of defining non-incrementality will probably be very difficult.

- The false assumption that possession of a hash value counts as a proof of the corresponding pre-image has trickled through some parts of the literature on computer security. Results that rely on this assumption may turn out to be incorrect. Literature in which the false assumption is made needs to be identified and the results in these publications need verification.

  In particular, the SET protocol [MV97a, MV97b, MV97c] and its analysis in BAN logic [AvdHdV01] need close re-examination.

11.4 Relevance to the Privacy Debate

We have demonstrated that for a number of problems, confidentiality (read: privacy protection) and availability (read: fighting terrorism) can go hand in hand. A number of techniques has been developed:

Methods The *information designator* is a solution which demonstrates that linking databases does not imply the abundant dissemination of sensitive information. On the contrary, if information designators are used, linking databases can enhance confidentiality.

Protocols *Knowledge authentication*, as exemplified in the T-1 and T-2 protocols, provides solutions demonstrating that comparing information for equality (a simple and elementary action) can be done without disclosing the information.

The methods and protocols demonstrate that for linking and comparing information, the information does not need to be disclosed. Thus, for linking or comparing secrets without disclosing them, there is no longer a need for a trusted third party, which is a gain. For application domains where it is not possible to find a trusted third party, our contributions offer solutions which were impossible before.

Our results warrant an existential statement: in relevant cases, it *is* possible to reconcile information exchange and confidentiality. Thus, the idea that there is an intrinsic *trade-off* between information exchange and confidentiality is wrong and misleading. This is relevant to the privacy debate, since the goal of the privacy debate is to find a balance in this supposed trade-off.

It may take a long time before the techniques presented in this thesis are applied to the issues of the privacy debate. For one thing, policy makers must understand the basic properties of our presented solutions and the possible future solutions. We do not cherish any illusions about this. The personal experience of the author is that policy makers often have an abominable knowledge of IT, information systems and epistemic logic[1], and that the knowledge of 'IT

[1] Epistemic logic is roughly the logic of knowledge about other people's knowledge. It analyzes constructs like 'I know that you know it, but you do not know that', which are essential if one wants to protect information against inappropriate dissemination. [FHMV95, MvdH95]

consultants' of privacy and related security issues is similarly depressing. In our opinion, when a policy maker or IT consultant states that it is necessary that privacy is sacrificed for some righteous task, this likely expresses either ignorance, unwillingness or insufficient priority.[2]

Not *all* privacy problems which are caused by anti-terrorism activities can be solved with the solutions offered in this thesis, only *some* of them. There is no reason to suppose that this thesis has exhausted all solutions for reconciliation. Future research by us and others may provide many more results which help to reconcile information exchange and confidentiality.

In general, security and cryptography research has mainly focused on facilitating a situation in which there are only *good guys* and *bad guys*. In this situation, the bad guys need to be avoided, and need to be kept ignorant while the good guys can be almost fully trusted. In practice, one considers only very few of the organizations one needs to interact with as unequivocally *good guys*. Thus, we need security solutions and cryptographic methods for interacting with *so-so guys*: those not intrinsically bad, but not to be trusted more than strictly necessary. Such solutions and methods are essential for addressing privacy issues.

[2] Of course, there is nothing wrong with a policy maker who assigns only a humble priority to the issue of privacy protection, when he clearly acknowledges this.

Part VI

Appendices

'BAN logic' does not uniquely identify one single article in which it is introduced. The relation between the many versions of the 'BAN paper' that exist is explained. We survey the major critisims of BAN logic in the literature.

Appendix A

Remarks to Authentication Logics

A.1 A Taxonomy of Versions of the BAN Paper

(Referred to on pages 47 and 56.)

The seminal paper "A Logic of Authentication" has a respectable number of versions. Its precursor, "Authentication: A Practical Study in Belief and Action" was presented at the second conference on Theoretical Aspects of Reasoning About Knowledge in March 1988 [BAN88, 18 pages]. Then, there is the DEC technical report, which was published in February 1989 and revised in February 1990 [BAN89a, 49 pages]. In April 1989, the work was submitted to the Royal Society of London, which published it in December 1989 [BAN89b, 39 pages]. Also in December 1989, a revised version of the article was presented on the twelfth ACM Symposium on Operating Systems Principles, which was also published in the ACM SIGOPS Operating Systems Review [BAN89c, 13 pages]. This led to a paper in the ACM Transactions on Computer Systems in February 1990 [BAN90a, 19 pages]. In May 1994, an appendix to the DEC technical report was published [BAN94, 10 pages].

The most notable distinction between these versions is that in the ACM-published versions and the DEC appendix, the notation of many operators has changed from symbols (e.g. $\models$) to linguistic terms (e.g. **believes**). These versions refer to the DEC technical report for full reference. The DEC technical report and the Royal Society version [BAN89a, BAN89b] should be considered the most complete versions, due to their size and the fact that these papers are most often used in self-references of the authors. Martín Abadi considers the Royal Society version the most definite one (on his homepage). These two versions of the article contain a Section 12, "On Hashing", which introduces

and discusses the inference rule essential in Chapter 5 of this thesis. These two versions also contain a Section 13, "Semantics", which defines the partial semantics for BAN logic, used in Sect. 5.6 of this thesis.

A.2 A Short Survey of Critisisms on BAN Logic

(Referred to on page 54.)

The main criticisms of BAN logic regard the following properties of the logic:

1. the semantics,
2. the notion of belief,
3. the protocol idealization method,
4. the honesty assumption, and
5. the incompleteness.

These weaknesses of BAN logic have not been a reason to abandon the way of thinking introduced by Burrows, Abadi and Needham. Many researchers have tried to 'repair' BAN logic. The following pages we will give a short survey of the critiques and how and where they have been addressed in literature.

The first two critiques of the list above are, in more detail:

1. *BAN logic has no (well-defined) semantics*
   It is not really clear what the constructs in the logic actually represent.[1]
2. *BAN logic does not distinguish between possession and belief*
   Normally, one would like to distinguish between possessing a sentence (e.g. "Clapton is God") and believing such a sentence.

One of the first attempts to fix BAN logic was by Abadi and Tuttle, who introduced AT logic [AT91], which has a slightly better semantics than the original BAN logic. A second attempt was by Gong, Needham and Yahalom, who introduced GNY logic [GNY90]. GNY logic distinguishes between possession and belief, but has a semantics just about as poor as BAN logic.

Authentication logics with the goal to have a well-defined semantics *and* a clear distinction between possession and belief are VO logic [vO93], SVO logic [SvO94, SvO96], AUTLOG [KW94, WK96] and SVD logic [Dek00]. There are many crossbreeds of these logics, and the logics have heavily influenced one another. Often, it is relatively easy to translate protocols and formulae between these logics.

Except for 'repairs' of the original BAN logic, extensions have also been available in abundance. Here we will mention just a few of the extensions.

[1] For a compact treatment on the value of semantics for authentication logics, consult [Syv91].

Gaarder and Snekkenes added time-related concepts to BAN logic [GS91], and Syverson did something similar for AT logic [Syv93]. Kailar and Gligor extended BAN logic to make it less dependent on *jurisdiction*[2] [GKSG91, KG91].

Analyzing a protocol by hand is a tedious task, and some of the logics have been designed in such a way that computer-aided analysis can be performed. In particular, AUTLOG is implemented in PROLOG [KW94], and SVD has an implementation in the Isabelle theorem prover [Dek00][3]. Also, model checkers have been used for protocol analysis, most notably by Lowe who found a major error in the Needham-Schroeder Public-Key protocol (NSPK) in this way [Low96].

Authentication logics have been tied to radically different approaches for analyzing security protocols. One of them is the strand-space methodology [THG98, THG99]. Van Oorschot used Strand-spaces to create yet another semantics [Syv00], and Jacobs used it to create a new logic with an accompanying implementation in the theorem prover PVS [Jac04][4].

The semantics and the notion of belief have not been the only aspects inviting criticism. The remaining three important avenues of criticism are:

3. *BAN logic has a rather vague 'protocol idealization method'*
   The process of translating a protocol into the language of the logic is poorly described and depends on 'intuitions an intentions'. Clearly, a rigorous description would be better.

4. *BAN logic assumes honest principals*
   Within BAN logic, it is impossible to model principals which state lies. This limits the kind of protocol flaws that can be found using BAN logic, as lying can sometimes be relatively easy and computationally cheap.[5]

5. *BAN logic is incomplete*
   There are protocol errors which BAN logic fails to identify; in fact, the class of such protocol errors contains some rather obvious errors.

Critiques on the protocol idealization method of BAN logic have appeared in [MB93, WK96]. In general, one can say that this problem has been addressed in almost all authentication logics. Criticism number 4 (honesty), stated in [GKSG91], boils down to criticism number 2 (possession and belief), and has been resolved in various ways in most authentication logics.

Incompleteness of BAN logic has been demonstrated by Nessett [Nes90], who showed a protocol which has a very obvious flaw, which cannot be de-

[2] Jurisdiction is the concept that a specific principal in a protocol has designated authority over some statements. Making BAN logic less dependent on jurisdiction therefore widens the class of protocols that can be analyzed using BAN logic.

[3] For background on Isabelle, consult [NPW02].

[4] For background on PVS, consult [ORSvH95].

[5] Thus, BAN logic supposedly is a logic in the Honest-But-Curious (HBC) attacker model, instead of in the Dolev-Yao threat model.

tected in BAN logic[6]. In some circumstances, incompleteness of BAN logic is also due to incorrect protocol idealization [BM94].

Solving the problem of incompleteness of authentication logics is difficult for a number of reasons. One of the challenging problems is that while strengthening the logic, the modeled inference capabilities (computational resources) of principals should remain the same. Thus, the inference capabilities of the principals should be constrained (as in [ABV01]). This approach has led to preliminary completeness results for authentication logics [CD05b, CD05a].

Though the original BAN logic has serious limitations, the way of reasoning is useful. The way of reasoning should not be abandoned because of the flaws in the original logic [HPvdM03]. Moreover, recent results show that it is possible to create a computational justification of authentication logics [AR02].

[6] Burrows, Abadi and Needham replied to this critique essentially by stating that they never claimed this would be the case [BAN90b]. Moreover, they felt the urge to humiliate Nessett by calling one of his assumptions "absurd" and questioning his "wit of a man to notice" [BAN90b, page 40].

In this thesis, we use an extension of GNY logic to analyze the T-1 protocol. We summarize the formal language and inference rules of GNY logic, as presented in [GNY90]. Parts of the language we do not use are omitted.

Appendix B

Summary of GNY Logic

(Referred to extensively from Chapters 4, 6 and 9.)

In this appendix, we summarize the formal language and inference rules of GNY logic [GNY90]. Parts of the logic that we do not use are omitted.

B.1 Formal Language

A formula is a name used to refer to a bit string, which would have a particular value in a protocol run. Let X and Y range over formulae, and $+K$ and $-K$ over public keys and private keys respectively. The following are also formulae:

(X, Y)	conjunction of two formulae. We treat conjunctions as sets with properties such as associativity and commutativity.
$\{X\}_{+K}$	public-key (asymmetric) encryption.
$\{X\}_{-K}$	private-key (asymmetric) signature We assume a cryptosystem for which $\{\{X\}_{+K}\}_{-K} = X$ holds (i.e., encryption), and for which also $\{\{X\}_{-K}\}_{+K} = X$ holds (i.e., signatures, e.g. RSA [RSA78]).
$H(X)$	the hash value of X obtained by application of a strongly collision-free cryptographic hash function (CRHF).
$*X$	a not-originated-here formula. A formula X is a not-originated-here formula if a principal receives X without having sent X itself before. Thus, a formula is a not-originated-here formula for a principal P, if it is not a replay of one of P's previously sent messages.

Assertions reflect properties of formulae. Let P and Q be principals. The following are basic assertions:

$P \lhd X$	P *is told* formula X. P receives X, possibly after performing some computation such as decryption.
$P \ni X$	P *possesses*, or is capable of possessing, formula X. At a particular state of a run, this (X) includes all the formulae P has been told, all the formulae he started the session with, and all the ones he has generated in that run. In addition P possesses, or is capable of possessing, everything that is computable from the formulae he already possesses.
$P \mid\sim X$	P *once conveyed* formula X. X can be a message itself or some content computable from such a message, i.e., a formula can be conveyed implicitly.
$P \models \sharp(X)$	P *believes*, or is entitled to believe, that formula X is *fresh*. That is, X has not been used for the same purpose at any time before the current run of the protocol.
$P \models \phi(X)$	P *believes*, or is entitled to believe, that formula X is *recognizable*. That is, P would recognize X if P has certain expectations about the contents of X before actually receiving X. P may recognize a particular value or a particular structure.
$P \models P \overset{S}{\leftrightarrow} Q$	P *believes*, or is entitled to believe, that S is a suitable *secret* for P and Q. They may properly use it to mutually prove identity. They may also use it as, or derive from it, a key to communicate. S will never be discovered by any principal except P and Q.
$P \models \overset{+K}{\mapsto} Q$	P *believes*, or is entitled to believe, that $+K$ is a suitable *public key* for Q, i.e., the matching secret key $-K$ will never be discovered by any principal except Q.

Let C range over assertions. The following are also assertions:

$P \models C$	P *believes*, or is entitled to believe, that formula C *holds*.
C_1, C_2	conjunctions. We treat conjunctions as sets with properties such as associativity and commutativity.

B.2 Inference Rules

We repeat all inference rules from [GNY90] used in this thesis. The rule names correspond to the names used in the original article. The first letter of the name intends to reflect the category of the rule: **T** is about being **t**old, **P** about **p**ossession, **F** about **f**reshness, **R** about **r**ecognizability, and **I** about message

interpretation. Some of the inference rules (**P2**, **F1**, **I3**) have more allowable conclusions than used in this thesis. These unused conclusions are omitted.

An inference rule that applies to formula X also applies to $*X$, though not necessarily vice versa. If $\frac{C1}{C2}$ is an inference rule then for any principal P so is $\frac{P \models C1}{P \models C2}$.

T1	$\frac{P \lhd *X}{P \lhd X}$	Being told a 'not-originated-here' formula is a special case of being told a formula.
T2	$\frac{P \lhd (X, Y)}{P \lhd X}$	Being told a formula implies being told each of its concatenated components.
T6	$\frac{P \lhd \{X\}_{-K},\ P \ni +K}{P \lhd X}$	If a principal is told a formula encrypted with a private key and he possesses the corresponding public key then he is considered to have also been told the decrypted contents of that formula. This rule only holds for public-key systems with the property $\{\{X\}_{-K}\}_{+K} = X$ (e.g. RSA [RSA78]).
P1	$\frac{P \lhd X}{P \ni X}$	A principal is capable of possessing anything he is told.
P2	$\frac{P \ni X,\ P \ni Y}{P \ni (X, Y)}$	If a principal possesses two formulae then he is capable of possessing the formula constructed by concatenating the two formulae.
P3	$\frac{P \ni (X, Y)}{P \ni X}$	If a principal possesses a formula then he is capable of possessing any one of the concatenated components of that formula.
P4	$\frac{P \ni X}{P \ni H(X)}$	If a principal possesses a formula then he is capable of possessing the hash value of that formula obtained by application of a strongly. collision-free (collision resistant) cryptographic hash function (CRHF)
P8	$\frac{P \ni -K,\ P \ni X}{P \ni \{X\}_{-K}}$	If a principal possesses a formula and a private key then he is capable of possessing the decryption of that formula with that key (i.e., the cryptographically signed formula).
F1	$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)}$	If a principal believes a formula X is fresh, then he is entitled to believe that any formula of which X is a component is fresh.

R6 $\dfrac{P \ni H(X)}{P \models \phi(X)}$ If P possesses a formula $H(X)$, then he is entitled to believe that X is recognizable.

I3 $$\frac{P \lhd *H(X,S), \quad P \ni (X,S), \quad P \models P \overset{S}{\leftrightarrow} Q, \quad P \models \sharp(X,S)}{P \models Q \mid\sim (X,S)}$$

Suppose that for principal P all of the following hold: (1) P receives a formula consisting of the hash value of X and S marked with a not-originated-here sign; (2) P possesses S and X; (3) P believes that S is a suitable secret for himself and Q; (4) P believes that either X or S is fresh. Then P is entitled to believe that Q once conveyed the formula X concatenated with S.[1]

I4 $$\frac{P \lhd \{X\}_{-K}, \quad P \ni +K, \quad P \models \overset{+K}{\mapsto} Q, \quad P \models \phi(X)}{P \models Q \mid\sim X, \quad P \models Q \mid\sim \{X\}_{-K}}$$

If P sees a signed message $\{X\}_{-K}$, knows the public key $+K$, knows the corresponding private key $-K$ belongs to Q, and recognizes X to be a message, then P is entitled to believe that Q once conveyed the signed message $\{X\}_{-K}$, and thus also once conveyed the message X itself.

I6 $\dfrac{P \models Q \mid\sim X, \quad P \models \sharp(X)}{P \models Q \ni X}$ If P believes that Q once conveyed formula X and P believes that X is fresh, then P is entitled to believe that Q possesses X.

[1] It should be noted that rule **I3** as given here differs slightly from the definition in [GNY90], which uses the notation $\langle S \rangle$ in some places instead of S to denote that S is used for identification. This is non-essential and only syntactic sugar. For readability of the proofs, these brackets have been omitted throughout this thesis.

There are many articles closely related to knowledge authentication which deserve some detailed comments because they contain minor (or major) flaws, while the comments are outside of the scope of Chapter 8 (Knowledge Authentication). These comments are bundled here.

Appendix C

Remarks to Knowledge Authentication

C.1 The 'French Approach'

(Referred to on page 115.)

In Table 8.1, a list of protocols is shown. The protocols presented in [DQB95, QBA+98a, QBA+98b, QAD00, Ber04, CC04] are omitted from this table. These omitted protocols are mainly protocols developed in the medical domain, and it is rather difficult to qualify these protocols without being harsh and impolite. In short: it seems that the thought that the mere application of cryptography would solve all problems prevented a clear formulation of the threats for which the protocols should offer a solution. We acknowledge that these are very strong claims. Nevertheless, they seem appropriate. Some examples:

- In [QBA+98a, QBA+98b] quality assesment of the protocols is performed, quoting figures on sensitivity and specificity. This type of assesments implies that privacy and validity (as defined in Section 2.7 and summarized at the start of this section) are not considered for granted.
- In [QBA+98a, QAD00] the term "cryptology" is used where 'cryptography' is supposedly intended.
- In [Ber04] a protocol is devised that should be "zero-knowledge" but it is never explained what zero-knowledge actually means, nor contains the paper any reference to a paper about zero-knowledge.
- In [CC04] it is stated that the use of a MAC can prevent a dictionary attack. This is only the case when one assumes the principals who know

the key to the MAC are essentially honest — this makes the whole excercise of devising a secure protocol for computing set relations useless.

These publications sometimes use terms like "minimal knowledge", which can be considered a confession that some knowledge (other than the set sizes) is leaked. It suggests that zero-knowledge is impossible.

C.2 On the Probabilistic Communication Complexity of Set Intersection

(Referred to on page 118.)

In the context of sparse sets, it is appropriate to clarify an often misinterpreted result by Kalyanasundaram, Schnitger and Razborov (from here on: KSR) [KS92, Raz92][1]. For example, in [FNP04] it is claimed that the result of KSR implies that the lower bound for communication complexity of the secure computation of set intersection is at least proportional in $|n|$. All men are mortal ([KS92, Raz92]), therefore Socrates is mortal ([FNP04]), so it seems.

KSR show that the probabilistic communication complexity of disjointness is $\Theta(n)$ (where $n = |\Omega|$)[2]. This result applies to a problem which is more general than the problems described in Chapter 8. In particular:

1. The result of KSR applies to the communication complexity of a *problem*, and not to the communication complexity of a *particular protocol*. In particular, they assume the principals have unlimited computational power. Thus, the *efficiency* property as mentioned at the start of Section 8.5 does not apply to the computational resources, only to the communication resources.

2. The problem analyzed by KSR does not include *privacy* or *validity* concerns: the principals are implicitly assumed to be honest, and do not care whether their 'private' information is disclosed. One could say the honest adversary model is assumed.

3. The problem analyzed by KSR applies to the *disjointness* problem (in terms of Figure 8.2: f_{disj}), and not to the *intersection* problem (f_{int}). The intersection problem is more difficult than the disjunction problem[3].

[1] The result is oringinally published by Kalyanasundaram and Schnitger in [KS92]. It has been greatly simplified by Razborov [Raz92].

[2] It is difficult to use consistent notation which is not misleading. As we have defined our domain of set items to be Ω, we might as well write $\Theta(|\Omega|)$, but Ω has also a meaning in complexity. Therefore, we introduce n. We admit this notation is far from optimal.

[3] If, for some X and Y one knows $f_{\text{int}}(X, Y)$, one can easily infer $f_{\text{disj}}(X, Y)$. Observe that

$$f_{\text{disj}}(X,Y) = \begin{cases} 1 & \text{if} \quad f_{\text{int}}(X,Y) = \emptyset \\ 0 & \text{if} \quad f_{\text{int}}(X,Y) \neq \emptyset \end{cases}$$

The opposite is not the case: one cannot always infer $f_{\text{int}}(X, Y)$ from $f_{\text{disj}}(X, Y)$.

4. In the problem analyzed by KSR, no assumptions are made on the set sizes.

It is tempting to project these results to the problems described in Chapter 8. As the result by KSR is more general, these results would, so it seems, also apply to the problems analyzed in Chapter 8.

The generality of the result by KSR is misleading. The first three observations listed above seem to warrant a projection of the results of KSR to secure computation of set relations. The fourth observation, that there is no assumption on the set sizes, prevents projection of their results to secure computation of set relations. With extra assumptions on the set sizes, it is possible to construct protocols which are more efficient than the lower bound derived by KSR. In particular, if the sets cover only a sparse fraction of the domain Ω, it is possible to devise protocols whose communication complexity is below $|\Omega|$. And in fact, the protocols presented in [FNP04] are an example of this.

It has to be admitted that [KS92] gives opportunities to misinterpret the result. The abstract starts with:

> "It is shown that, for inputs of length n, the probabilistic (bounded error) communication complexity of *set intersection* is $\Theta(n)$."

With "inputs of length n", the authors tacitly mean 'inputs of length n *if encoded in a specific manner, namely ...*'. Moreover, where they write "set intersection", the authors actually mean 'intersection cardinality > 0', which is the dual of *set disjointness* (see Figure 8.2 on page 110).

Therefore, the result of KSR *does not* imply that the lower bound for communication complexity of the secure computation of set intersection is at least proportional in $|n|$.

C.3 Fuzzy Private Matching

In [FNP04], on pages 16 and 17, a problem closely related to knowledge authentication is presented: Private Fuzzy Matching. The problem is not only to find exact matches, but also 'close matches', which they define well. The protocol they present on page [FNP04, page 17] is however incorrect. This problem has been identified by Łukasz Chmielewski. In [CH06], a corrected protocol, and other protocols for this problem are presented.

The T-1 protocol has a prototype implementation as a Java application. It is explained how this prototype is operated. The prototype allows one to stress-test the T-1 protocol.

Appendix D

The Secret Prover

With use of the T-1 protocol, it is possible to prove possession of arbitrary secrets (Chapter 9). The 'Secret Prover' is a prototype implementation of the T-1 protocol. With it, people can prove possession of files, and explore how the protocol works. The Secret Prover is a Java application[1], which can be downloaded from `http://www.teepe.com/phdthesis/demo` . Java 1.4 or newer is required, which is available for Mac OS, Windows and many Unices, including Linux.

In this appendix, we will talk you through the whole application, in such a way that you can run the protocol yourself. No deep understanding of the T-1 protocol is required. Running the protocol hands-on, with some help from the screenshots printed in this chapter, may even help to gain some basic understanding of the T-1 protocol.

The Secret Prover has the following features:

- Execution of all three configurations of the protocol (the verifier initiates, the prover initiates, mutual proof).
- Support for the following hash algorithms: MD5, SHA-1, SHA-256, SHA-384 and SHA-512.[2]
- Support for using a nonce, or encryption. The following encryption algorithms are supported: DES, Triple DES, Blowfish.
- The transport mechanisms used is TCP/IP. (i.e., it works over the internet.)
- There is no authentication of the identity of the principals in the protocol. The authenticity of the communication channel is also not guaranteed.

[1] The current version number is 0.03.

[2] For SHA-256, SHA-384 and SHA-512, Java 1.5 is required. If one side of the protocol tries to use one of these hash algorithms, while it is not supported at the other side of the protocol, this is detected and a warning is given.

- Multiple proof messages can be sent per protocol.[3]
- Protocols can be interleaved.
- It is possible to perform 'fake' actions in the protocol.
- It is possible to trim down ('mutilate') hash values to a few bits, to simulate the effect of hash collisions on the protocol.

If you want to run the prototype yourself and experiment with it, you should read on from here. If on the other hand you only want so look how it works, without running anything yourself, you may skip Section D.1, and if you are very impatient, you might skip Section D.2 as well.

D.1 Starting Up and Connection Control

Once you have downloaded the application file you can start it up. On Unix-like operating systems, this is done by typing

```
# java -jar Prover.jar &
```

at the console. In more graphical environments, like Mac OS X, the file can simply be 'double-clicked'. Once the application has launched, you will see appear the main application window (Figure D.1).

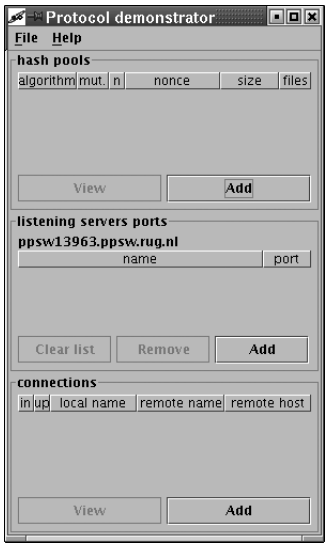


FIGURE D.1:
Main application window.

The main application window consists of four parts, from top to bottom:

A menu bar From here you can stop the application or open the splash screen.

Hash pools This is where pre-computed hash values are stored, it will be described in Section D.2.

Listening server ports Here all connection listeners are managed. When you open a connection listener, someone can contact you by opening a connection to the specified host and port.

Connections Here all connections are managed. You may contact somebody, and the connection will be displayed here. Also, if you have a connection listener open, if somebody contacts you through the listener, the connection will show up here.

[3] This is a slight generalization of the T-1 protocol. In the T-1 protocol, if the prover is also the initiator, he can only send one single h_2 value. In the prototype, an initiating prover may send multiple h_2 values. This generalization has been implemented to ease the process of experimenting with the protocol.

FIGURE D.2: Opening a connection listener.

FIGURE D.3: Filling in a name.

D.1.1 Opening a Connection Listener

The TCP/IP protocol works using 'meeting places'. A meeting place is a combination of a computer (a host, an IP address) and a port (a number). The owner of a computer may set up a meeting place by listening at the spot of the number. Anybody else contacting the host at the port will get connected. For any connection, one of the participants has to set up such a 'meeting place'.

By clicking on 'Add' in the 'Listening server ports' pane of the main application window (Figure D.1), you can set up such a meeting place. The host is already set to reflect your computer's IP address, but you do have to provide a port number and a name (Figure D.2). The port number is set to a reasonable default, but you really should fill in the name field. This name is not essential to setting up a connection, but it will be used in the protocols later on.

Throughout the examples I will use the name 'Wouter' (Figure D.3). Later on in the example, a second player will be introduced, and for the second player I will use the name 'Kathy'. In the text I will sometimes refer to Kathy or Wouter. Obviously, you are free to use other names. You should however take note which of the names you use correspond to the roles of Kathy and Wouter, because it may help you to understand the explanation.

If the port for some technical reason or another cannot be used as a meeting place, you will be informed so, and in that case you should use another port number and try again.

Now you may by whatever means you see fit inform somebody else to contact you at the specified host and port. The host is the name or IP address of your computer. For your convenience, this is displayed in the 'Listening server ports' pane of the main application window. The port number is the number you just specified.

Instruct some friend or colleague to run the software as well. The best and most convincing way to see how it works is to run the protocol with somebody else. However, if you lack somebody to play the game with, or don't want to bother anybody, you may connect yourself to the server and the port. That is what we're going to do in the next section.

D.1.2 Making a Connection

By clicking 'Add' in the 'Connections' pane of the main application window (Figure D.1), you get the 'connection' window shown in Figure D.4. Since

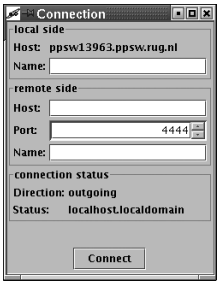


FIGURE D.4: Making a connection.

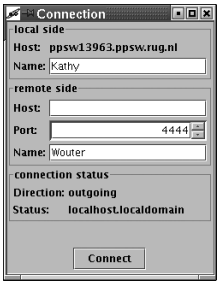


FIGURE D.5: Filling in connection details.

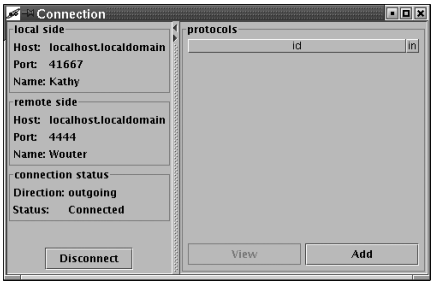


FIGURE D.6: An initiated connection (outgoing).

we are connecting to ourselves, we should distinguish between our own two 'egos'. A good way to easily distinguish these is to put all windows concerning ego #1 (Wouter) on the left hand side of the screen, and all windows concerning ego #2 (Kathy) of the right hand side of the screen. (Now drag the connection window to the right hand side of your screen.)

In the top of the connection window, you see the 'local side' pane: this is about who you (the connecting side) are. Choose some name for your ego #2 and fill it in. Below that, you see the 'remote side pane': this is what meeting place (host and port) you are connecting to, and who you expect to find there (Figure D.5). If you leave empty the host field, it will connect to the local computer. (And we have just opened a listener on the local computer on port 4444, so this will all work out.) As soon as you click 'Connect', the software will try and make the connection for you. The window will 'grow' and transform into the window shown in Figure D.6.

When you have done this, you will see that some more windows have popped up. Most importantly, a window has popped up that belongs to the listening port (Figure D.7). If you would not have connected to yourself, but to somebody on another host (i.e., another computer), the window would have come up there.

Also, two 'authentication' windows have popped up (Figure D.8), which inform both sides of the connection that some authentication should take place.[4]

You may click on 'OK' to close the authentication windows.

Of course, it may happen that the person responding on the meeting place is not the person you expected to meet there. In that case an 'authentication mismatch' warning is given (Figure D.9).[5]

[4] The T-1 protocol requires an authenticated communication channel. We haven't implemented this authentication, because it would complicate the setup process even more, while the authentication is not essential for demonstrating the protocol. Of course, the authentication is essential in the sense that without it, anybody can claim to be anybody else, which is clearly undesirable. Any production implementation should obviously include authentication of the players and of the communication channel.

[5] Since no real authentication is done in this prototype, this warning is mainly cosmetic.

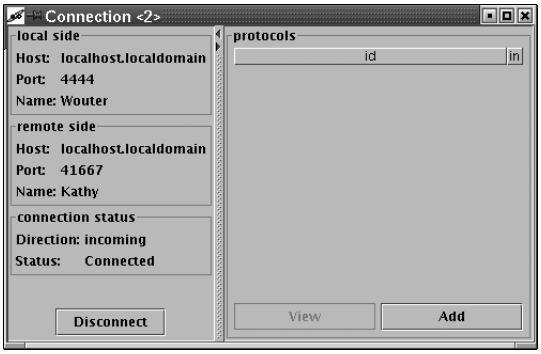


FIGURE D.7:
Receiving a connection (incoming).

FIGURE D.8:
An authentication warning.

FIGURE D.9:
An authentication mismatch.

If all has gone well, the main application window will look like Figure D.10:

Listening server ports In this pane a line has appeared which informs you the port is listening for new connections. You may select ports and remove/close them using the buttons in the pane.

Connections In this pane, the two sides of the connection are shown. If you would be connecting to somebody on another host, only one line would have shown up. If you click the 'View' button, the window containing the connection details will be shown (either Figure D.6 or D.7).[6]

Congratulations! you have passed the 'boring' part of the prototype! In the next section, we will pre-compute hash values, and store them in *hash pools*.

FIGURE D.10:
Main application window, with connections.

[6] The details shown in the connections pane are as follows:

- The columns 'local name', 'remote name' and 'remote host' are pretty self-explanatory.
- The column 'in' shows whether it is an incoming connection (you have set up a listener/meeting place and somebody went there), or whether it is an outgoing connection (you went to a meeting place that somebody else has set up). Checked means it is an incoming connection, unchecked means it is an outgoing connection.
- The column 'up' is checked if the connection is up, that is, it has not been terminated in some way or another.

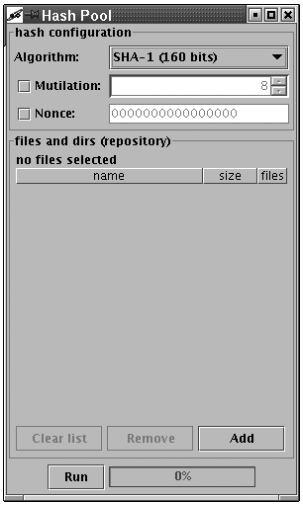


FIGURE D.11:
A new hash pool window.

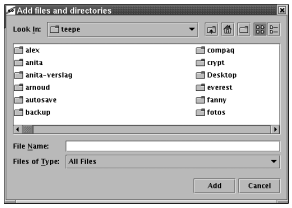


FIGURE D.12:
Adding files to a hash pool.

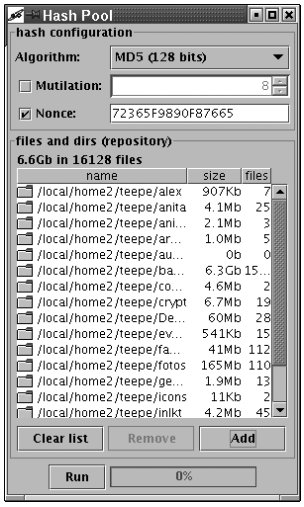


FIGURE D.13:
A hash pool with files added.

D.2 Managing Hash Pools

The protocol uses a hash value (h_1) to refer to the file of which knowledge is to be proven. When you initiate the protocol, you can compute the hash value as easy as that. If, however, you are on the responding side of the protocol, you have to find out what file the hash value h_1 refers to. That is what hash pools are for. A hash pool is a collection of precomputed mappings from hash value to file and vice versa.

A hash pool has some settings, which determine the exact way the hashes are computed from the files. These settings are:

Algorithm The hash algorithm used to compute the hash. Currently MD5, SHA-1, SHA-256, SHA-384 and SHA-512 are available.

Mutilation All hash algorithms generate a value of a certain number of bits. If we want to stress-test the protocol for what happens if we have (for example) collisions, we can easily enforce collisions by trimming down the hash values to a limited number of bits.

Obviously, in any production environment, you don't want to mutilate your hash values.

Nonce If the protocol uses a nonce (and thus no encryption), a nonce must be used in the process of computing the hash values. In the prototype, nonces are 64 bits long, but they could be of any sufficiently large size.

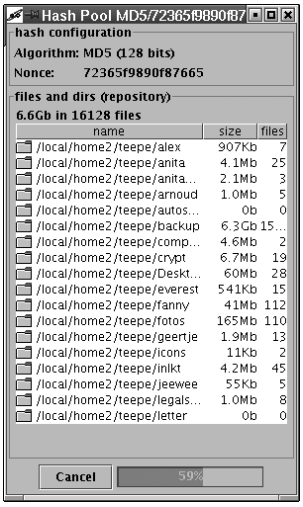


FIGURE D.14: Computation of hash values.

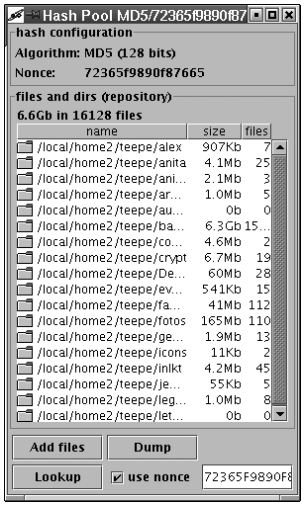


FIGURE D.15: A ready hash pool.

FIGURE D.16: Adding files to an existing hash pool.

To create a hash pool from scratch, press the 'Add' button in the 'hash pools' pane of the main application window (Figures D.1 and D.10). It will display a window in which the described settings can be set (Figure D.11).

Next to some settings, a hash pool obviously has a file list. Using the 'Add' button, files can be selected for inclusion in the hash pool (Figure D.12). You can also add directories. In that case, the program will recursively add all files in the directory (Figure D.13). Only after pressing the 'Run' button, the computation of the hash values starts (Figure D.14). Depending on the total volume of the files, this may take some time. A progress bar will give you a precise approximation of the progress.

After having done the precomputations, the hash pool window (Figure D.15) serves two main purposes:

1. Displaying the list of added files and directories;

2. Allowing you to add more files and directories to the hash pool.

The first purpose is obvious, the second almost as much. After pressing 'Add files' in a hash pool window which has finished computing, it will pop up a new window in which all settings are adopted from the existing hash pool, and you can add new files (Figure D.16). After pressing 'Run', the additional precomputations will be performed and the files will be added to the hash pool. Any protocols bound to this hash pool will be updated automatically.

If you run the protocol, you may indeed use the 'Add files' function quite a lot. That will be described in the Section D.3.3.

D.3 Running the Protocol

There are three configurations of the T-1 protocol, and for each of the configurations, a protocol with and a protocol without encryption exists. As an initiator of the protocol, one can choose to use encryption or use a nonce (two mutually exclusive options), and one can choose whether the initiator only proves, both proves and verifies, or only verifies (three mutually exclusive options). The combination of these choices gives a total of six possible protocols. All six blends of the protocol can be performed with the prototype.

When running a protocol, one has to keep track of several properties of the protocol, such as the file the protocol is about, what hash function is used, what nonce is used, and so on. All protocol windows display this information. When one initiates a protocol, one can also manipulate these attributes. Therefore, before explaining all protocol actions, we will briefly describe all protocol properties, and how they are shown in the protocol window.

In a protocol window, one can see from top to bottom the following things:

chat session The full communication history of the protocol. This displays exactly what bytes are sent along the TCP/IP connection regarding the current protocol run. Messages sent by you are prepended with 'SEND:', and messages sent by the remote side (the person you're communicating with) are prepended with 'RECEIVE:', for clarity.

Protocol configuration Here the security settings of the protocol are set or displayed. It is a superset of the options of a hash pool.

If you initiate the protocol, this is where you choose to either use a nonce, or use encryption. If you choose to use a nonce, you must give one, and if you choose to use encryption, you must choose an encryption algorithm and a key.

If you are not the initiator, this is where you must fill in either the nonce or the key, depending on the protocol type the initiator has chosen.

Your role Whether you are to prove, you are to verify or to prove and verify. As an initiator, you can set this parameter.

The big fingerprint/subject Here the subject of the protocol is identified. That is: the initiator chooses a file, h_1 will be computed. The initiator will see both the file name and h_1, the responder will only see h_1.

Here is also an option to 'fake', this will be explained later on. For now, you may ignore this option.

A button A multi purpose button. Its purpose depends on the state the protocol is in. While the protocol is being set up, the initiator of the protocol can fire up the protocol by pressing 'Initiate', upon which the communication will commence.

FIGURE D.17:
A new protocol window for the initiator.

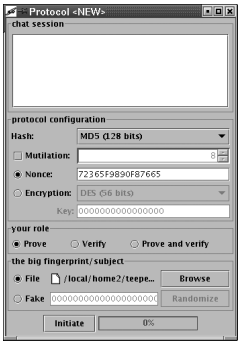


FIGURE D.18:
A protocol window, configured by the initiator.

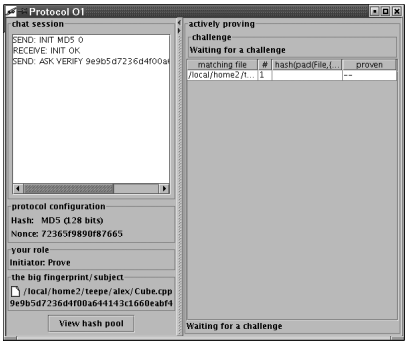


FIGURE D.19:
A protocol window of the initiator for a protocol that has started.

The responder on the other hand, will see 'Commit ...' when the protocol has just started. Pressing the button, the responder can confirm the nonce or key he has entered.

After this, both sides will see here 'View hash pool'. Each protocol run has an associated hash pool, which can be inspected and extended using this button. It will display the corresponding hash pool window.

The association between hash pool and protocol run is not exclusive: one hash pool may be bound to more than one protocol run.

All these options may seem somewhat dazzling, but that will wear away quickly. In the next section, hopefully most questions will vanish.

D.3.1 Initiating a Protocol

Remember, we had set up a connection from Kathy to Wouter. We also had made a hash pool with the MD5 algorithm and the nonce 72365F9890F87665. Kathy (e.g. you) may now press the 'Add' button in the connection window (Figure D.7), and in the window that pops up (Figure D.17) select the hash algorithm, state that she wants to use a nonce, fill in the nonce, say she wants to prove (only), and use the 'Browse' button to select a file (it will look like Figure D.18).

Finally she presses 'Initiate' and the protocol is fired up. Along firing up the protocol, the window is enlarged (Figure D.19). Don't worry, we'll explain what you'll see in the added part later on, in step 3.3.

In case the responding side does not know or support the chosen hash and encryption algorithms, you will be informed so and may choose to try again using other algorithms. (Java 1.4 only supports MD5 and SHA-1, Java 1.5 also supports SHA-256, SHA-384 and SHA-512).

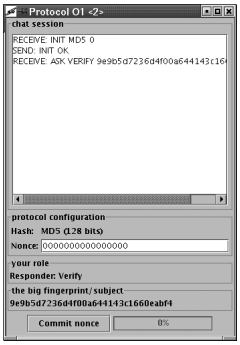


FIGURE D.20: A new protocol window for the responder.

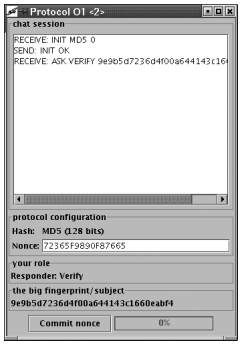


FIGURE D.21: The responder has filled in the nonce.

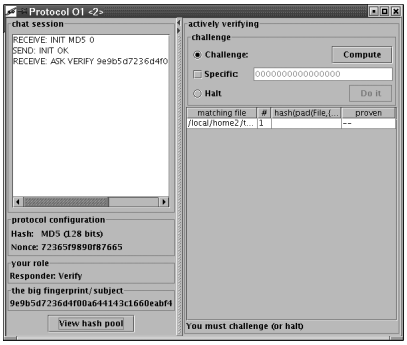


FIGURE D.22: The responder has committed the nonce.

D.3.2 Responding to a Protocol

Okay, Kathy has just initiated a protocol to Wouter. This has the result that at Wouter's side, a window has popped (Figure D.20). All protocol attributes are shown, except a crucial part. This crucial part is the nonce, if a nonce is used, and if encryption is used, the key. The responder should provide this information.

When a nonce is used, the nonce influences which hash pool is used. Filling in the wrong nonce will result in not finding the file corresponding to the received hash value.

When encryption is used, filling in the wrong key will result in not being able to decrypt the received hash value. This will be detected, and you will be prompted to try another key instead.

For now, Wouter should somehow know the nonce 72365F9890F87665 and fill it in (Figure D.21), and commit it. After committing, the window is enlarged (Figure D.22), similarly to the enlargement on the initiator's side.

D.3.3 A Side Note on Hash Pools

After initiating a protocol or committing the key or nonce to a protocol, the prototype automatically finds out what hash pool settings should be used for the current protocol. If such a hash pool already exists, it will take it. If such a hash pool does not yet exist, it will create it on the fly, with no files added yet.

Then, for the initiator, the hash value of the chosen file is computed and added to the hash pool, or looked up if it already was in the hash pool.

Strictly spoken, the initiator does not need a hash pool, only the responder does. There may however be situations in which it is desirable for the initiator to maintain a hash pool as well. (For example, if the initiator will be a responder in another protocol run, keeping a hash pool will save him computations.)

The responder will use the hash pool to try to figure out what file the initiator is talking about. If the responder already has done precomputations (i.e., created the hash pool and filled it with files), this will be inferred automagically. The responder may however always, during execution of the protocol, add files to the hash pool. This means that if the initiator has not done any precomputations, he is free to do these computations right on the spot, while the protocol has already started.

D.3.4 Challenging

There is another side note to make before we get to making a challenge. The screenshots and explanations you see and read here investigate the case where the initiator is the proving side, and the responder is the verifying side. If the initiator has chosen another option in the 'role' field (Figure D.17), this would all be different. From here on we talk about provers and verifiers. It is just a mere coincidence that the prover also is the initiator, and nothing more.

Now, Wouter should verify Kathy's possession of the file, so he should go and do this. First step would be to look in the hash pool for a matching file, and if no file was found, Wouter could try and add more files to the hash pool to see whether he finds the matching file. In case of this example, we have already done precomputations, and so Wouter has an educated guess which file it is.

To verify Kathy's possession of the file, Wouter has to challenge Kathy. By pressing 'Compute' (Figure D.22), a challenge will be generated such that it discriminates within $I\star$. In fact, this is just making up some random challenge and test that it does indeed discriminate within $I\star$, if it doesn't, it just tries again with another random challenge.[7]

You may also choose the challenge yourself. In that case check the 'Specific' box, and fill in a challenge (Figure D.22). Then press 'Compute'. This option should not be used in production environments. It allows to test and verify that nobody can perform a man-in-the-middle attack.

Currently, all challenges are 64 bits long. This is not due to any limitation in the protocol, but just because we wanted to limit the number of options to the user. (There are enough options to choose from already.) The size of the challenge should be big enough to make it infeasible for an adversary to try all possible challenges.

You may remember from Chapter 9, that if the verifier is also the responder, he may also halt the protocol. This should typically be done when the responder has no matching file or does not wish to prove possession of the matching file. Therefore, you see that there actually is the choice to either challenge, or to halt. To confirm the choice, press 'Do it' (Figure D.23). After this confirmation the other side will be informed accordingly. For demonstration purposes, let Wouter actually challenge Kathy, and press 'Do it' (it will look like Figure D.24).

[7] Little exercise: Imagine what would happen if we would have thousands of files in our hash pool and would be trimming our hash values to only 3 bits.

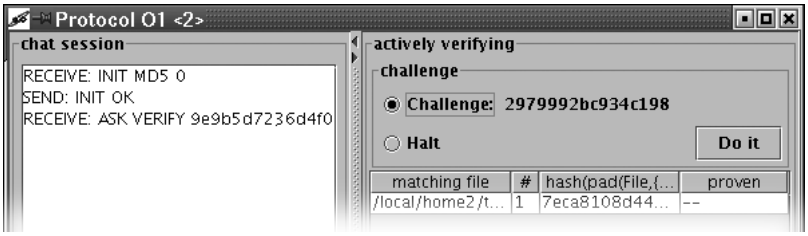


FIGURE D.23: The verifier chooses whether he will halt the protocol. For the chosen challenge, the required hash value h_2 has been computed.

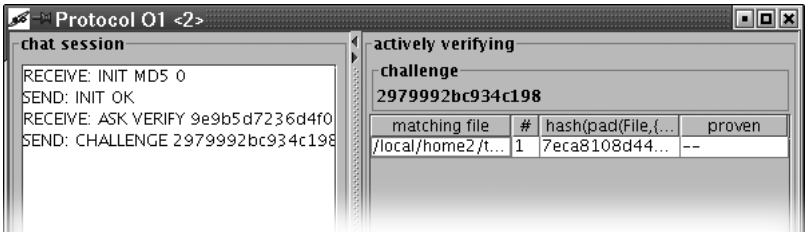


FIGURE D.24: The verifier has challenged the prover.

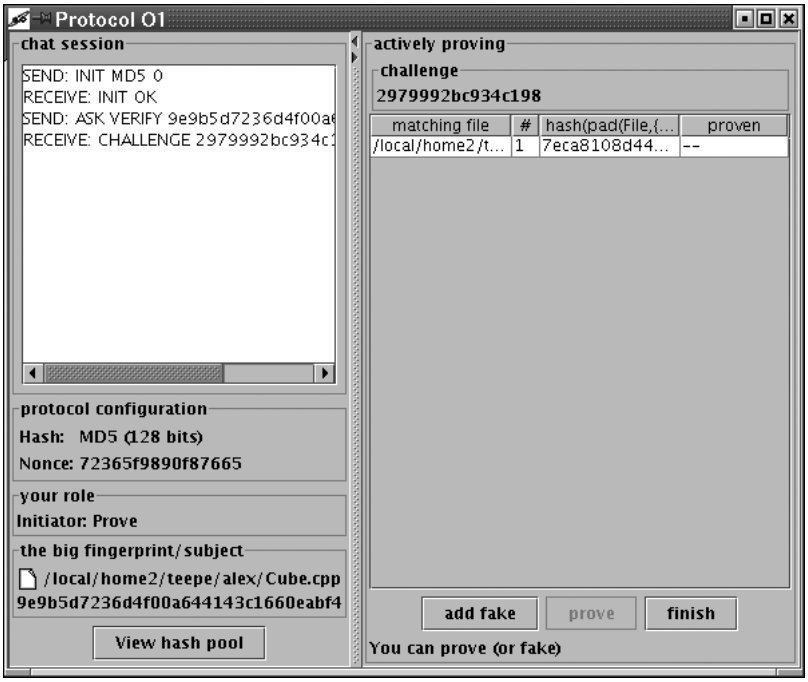


FIGURE D.25: The prover has received a challenge. With this challenge, the required hash value h_2 has been computed.

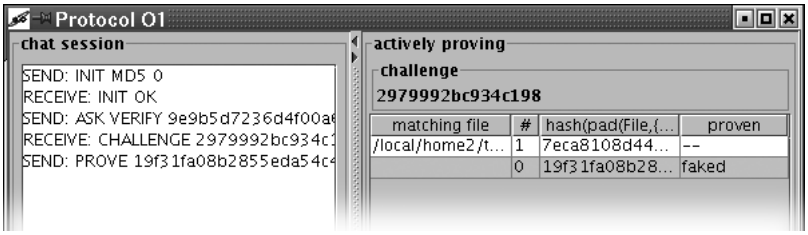


FIGURE D.26: The prover sends some fake hash value h_2.

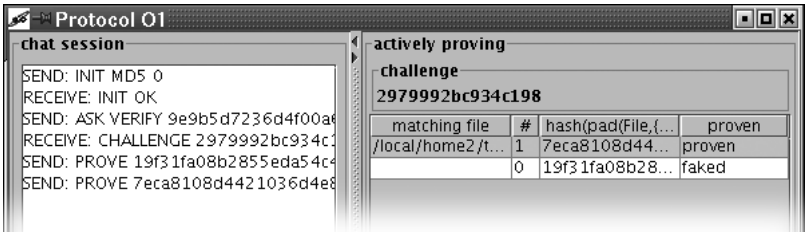


FIGURE D.27: The prover sends a genuine hash value h_2.

D.3.5 Proving

When the prover has received a challenge, the main protocol window (which looked like Figure D.19), will have the challenge filled in, and the value h_2 computed (Figure D.25).

The prover may send any number of h_2 hash values, and should then inform the verifier that no more hash values will be sent.

The h_2 hash values the prover sends would in an optimal case always be truthful. However, we want to demonstrate that sending untruthful h_2 hash values will not result in the verifier becoming convinced. Therefore, the prover has two options for sending a h_2 hash value:

1. Just make up some untruthful h_2 hash value, and send it. To do this, the prover presses the 'add fake' button, which will add a fake file to $I\star$ with a random h_2 hash value. This 'hash value' may be edited. After this, select the fake file and press 'prove' (Figure D.26).

2. Choose a matching file from $I\star$, and sent the corresponding hash value h_2 (is has already been computed). To do this, the prover selects one or more files from the list, and presses the 'prove' button (Figure D.27).

When the prover has performed actions as he sees fit, he can inform the verifier that he is finished. This is done by pressing the 'finish' button (it will look like Figure D.28).

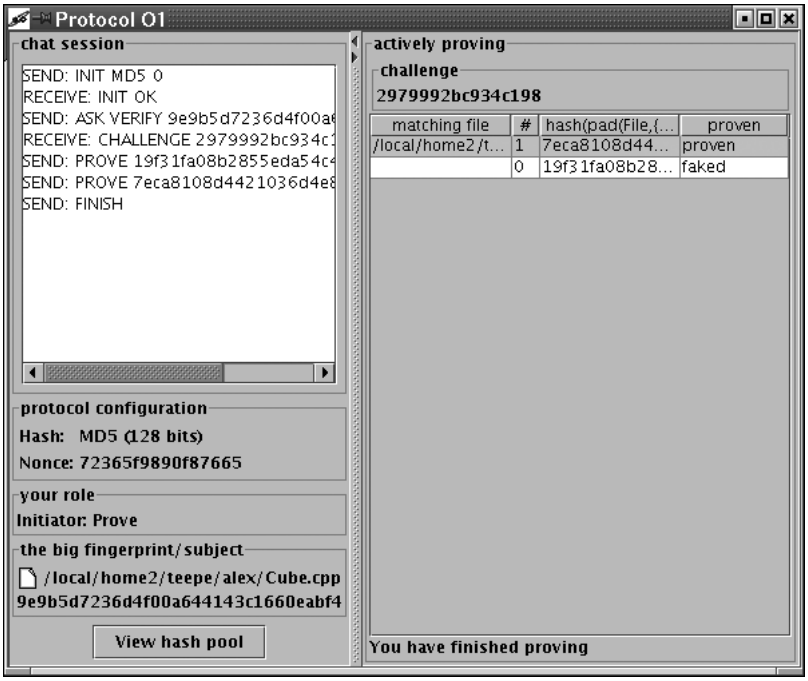


FIGURE D.28: The prover hash halted the protocol (no more proofs will follow).

D.3.6 Verifying

After the verifier has challenged the prover, he is left with the task of the actual verification of the h_2 messages sent by the prover.

The prover can send a number of h_2 hash values, and then inform that no more h_2 hash values will follow. Each incoming h_2 hash value is looked up in the set $I\star$. If there is no matching file, there are two possible interpretations:

1. The prover has a file which gives the same h_1 hash value as the file the verifier has. This is increasingly unlikely as the hash size gets bigger. With a hash size of 128 bits, this is negligibly unlikely. (Assumed that the used hash function is a sufficiently good one.)

2. The prover has just made up some h_2 hash value.

Since the first option is so unlikely, if a hash value h_2 is received with no matching file in $I\star$, it will be marked as 'faked' (Figure D.29).

If a matching file is found, this file will be marked as 'proven', and this will be shown to the verifier (Figure D.30).

Finally, when the prover informs the verifier that he is finished, the verifier can finalize his interpretation of the received hashes. If the file was marked as 'proven', the prover both possessed the file and wanted to prove it to the

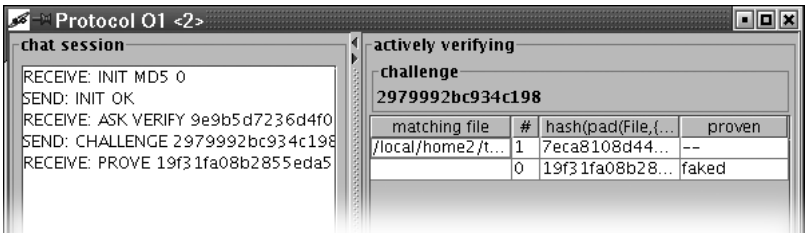


FIGURE D.29: The verifier receives an unexpected value of h_2. It is marked as 'fake'.

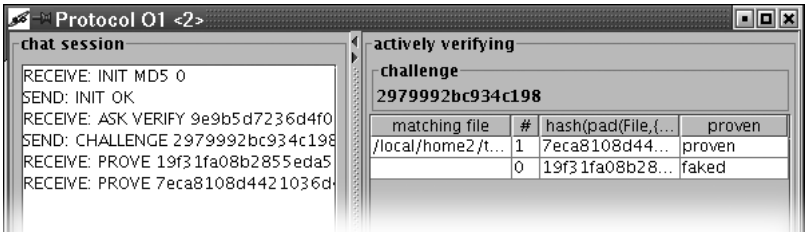


FIGURE D.30: The verifier receives the h_2 he expected. The corresponding file is marked as 'proven'.

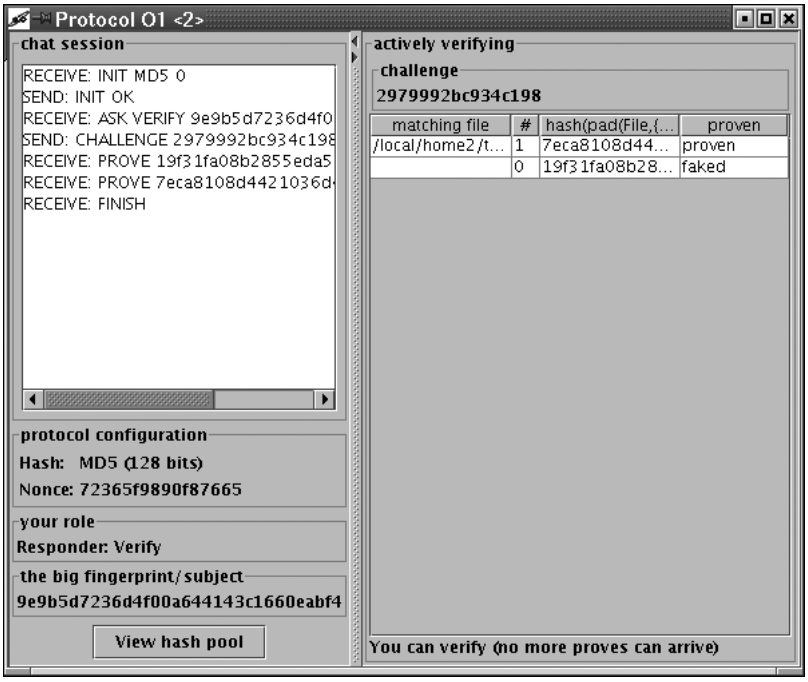


FIGURE D.31: The verifier has been informed that the prover has halted the protocol. (No more proofs will follow)

verifier. If no such mark has been made, the prover either did not possess the file, or did possess the file but did now want to prove this to the verifier.

D.3.7 Faking

The prototype can demonstrate that the T-1 protocol is secure in the malicious adversary model. The prototype facilitates all kinds of malicious behavior:

h_1 The initial hash value h_1 can be set manually. In the initialization of the protocol (Section D.3.1), instead of selecting a specific file to run the protocol with, one can choose 'Fake', and fill in the h_1 value directly. The button 'Randomize' is at ones disposal to help you make up some random value for h_1.[8]

C The challenge the verifier sends to the prover, should be made up at random, which is what the prototype can do. However, a malicious player may want to challenge in a specific way. While challenging (Section D.3.4), one can check the 'Specific' box, in which case the malicious player can fill in the challenge he wants to use.[9]

h_2 The response h_2 hash value the prover sends, can also be faked. This has been demonstrated in Sections D.3.5 and D.3.6.

Using these features, a malicious adversary can perform actions which are syntactically correct, but not intended. This kind of actions will never result in a non-malicious player being misled. That is, a non-malicious player will never be sneaked into believing that a malicious player possesses a file while he does not possess the file.

D.4 Closing

After having played around, a moment in time might arrive in which a user of the prototype decides to temporarily stop playing around with the software. Specifically for those users, the 'Exit' action is implemented. It is found in the 'File' menu of the main application window.

Exiting the program will result in all information in the hash pools being lost. If at a later moment one wants to rerun the protocols, one will need to rebuild the hash pools from scratch.

In case one only wants to stop communicating with some other player, one can disconnect the other player by pressing the 'Disconnect' button in the connection window (Figure D.7). All non-terminated protocols using the connection will obviously stop.

[8] Using this feature, one can try to claim possession of a file which one does not have. The protocol will however never let you successfully prove you do indeed possess the claimed file.

[9] Using this function, the malicious player might try to perform a man-in-the-middle attack. He will not succeed, because he will need to present a hash value h_2 with the given challenge and also a name, and such a hash will never be computed by another trustworthy user.

Appendix E

Notation

E.1 Symbols

$\sim$	once conveyed (GNY logic, BAN logic)
$\vdash$	derivable within a logic
$\models$	observable within a model
$\equiv$	believes (GNY logic, BAN logic)
$\overset{\cdot}{\mapsto}$	public key (GNY logic, BAN logic)
$\overset{\cdot}{\leftrightarrow}$	shared secret (GNY logic, BAN logic)
$\sharp(\cdot)$	fresh (GNY logic, BAN logic)
$\phi(\cdot)$	recognizable (GNY logic)
$*$	not-originated-here (GNY logic)
$\lhd$	is told (GNY logic, BAN logic)
$\ni$	possesses (GNY logic)
$\odot$	combining function (used in the randomize-then-combine paradigm)
$\oplus$	bitwise exclusive or
$\lvert \cdot \rvert$	the cardinality (number of elements) of a set
$\{\cdot\}$	a set
$\{\cdot\}.$	encryption; $\{M\}_K$ denotes the message M encrypted under symmetric key K; $\{M\}_{+K}$ denotes the message M encrypted under public key $+K$; $\{M\}_{-K}$ denotes the message M signed under private key $-K$

E.2 Letters

Ω	the domain of all possible secrets
Φ	the compressed domain of all possible secrets (Section 8.6)
A	the principal Alice
$\mathscr{B}$	the set of beliefs ($\models$) of a principal (BAN logic)
B	the principal Bob
c	a constant factor in a formula
C	either: • the principal Cecil (trustworthy) or Charlie (untrustworthy) • a challenge (a message or bit string constructed for the sake of being unpredictable)
D	a designator
E	the principal Eve (the evil eavesdropper)
ϵ	the empty string
ε	an error margin
$f(\cdot)$	a compression function (used in the Merkle-Damgård paradigm)
$g(\cdot)$	a randomizing function (used in the randomize-then-combine paradigm)
h	a hash value; in the T-1 and T-2 protocols: h_1 is the hash value that 'points at' a secret h_2 is the hash value that 'proves possession of' a secret
$H(\cdot)$	a non-keyed cryptographic hash function
I	a specific secret; I_Q is a secret of principal Q ($I_Q \in KB_Q$)
$I\star$	a set of specific secrets; $I_Q\star = \{I_Q \in KB_Q \mid H(I_Q, N) = h_1\}$; or $I_Q\star = \{I_Q \in KB_Q \mid H(I_Q) = h_1\}$
k	a security parameter (a measure of the strength of a cryptographic function)
K	a symmetric key
$-K$	a private key (the corresponding public key is $+K$); $-K_A$ denotes the private key of A (etc.)
$+K$	a public key (the corresponding private key is $-K$); $+K_A$ denotes the public key of A (etc.)
KB	a set of IB's possessed by an principal; KB_A denotes the set of IB's of A (etc.)
l	the length of a bit string
ln	the logarithm (with base 2)
M	a message
$\mathscr{M}$	the set of messages seen ($\lhd$) by a principal (BAN logic)
$MAC(\cdot,\cdot)$	a keyed cryptographic hash function (also called message authentication code, or MAC); $MAC(K, M)$ denotes message M hashed under key K
N	a nonce (a message constructed for the sake of being fresh)
p	a prefix of a hash value

	p_1 is the prefix of h_1 p_2 is the prefix of h_2
P	the principal Peggy (the prover)
$\mathcal{P}(\cdot)$	the *power set*, the set of all possible subsets; $\overset{<\infty}{\mathcal{P}}$ is the set of all possible *finite* subsets; $\overset{<\infty}{\mathcal{P}}(\{0,1\}^*)$ is the set of all possible finite sets of finite bit strings
Q	a principal which may be either Alice, Bob, Cecil, Eve, Peggy or Victor
s	a global state in a protocol run; s_Q is a local state of principal Q (BAN logic)
S	either: • a secret (either private or shared) (GNY logic) • a set of binary strings (used in the T-2 protocol)
t	a timestamp or time interval (see Section 7.4)
V	the principal Victor (the verifier)

Bibliography

At the right are the page numbers on which the sources are referred.

[ABV01] Rafael Accorsi, David Basin, and Luca Viganò. Towards an awareness-based semantics for security protocol analysis. *Electronic Notes in Theoretical Computer Science*, 55(1), 2001. 71, 182

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004. 88, 92

[AES03] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 86–97, New York, NY, USA, 2003. ACM Press. 114, 115

[AF90] Martín Abadi and Joan Feigenbaum. Secure circuit evaluation, a protocol based on hiding information from an oracle. *Journal of Cryptology*, 2(1):1–12, February 1990. 24

[AF04] Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004. 102

[AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999. 174

[AIR01] Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, Berlin / Heidelberg, 2001. Springer. 22

[And93] Ross Anderson. The classification of hash functions. In *Proceedings of the IMA Conference in Cryptography and Coding*, 1993. 29, 33, 34, 36

[AR02] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002. 18, 47, 66, 182

[AT91] Martín Abadi and Mark Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, Montreal, August 1991. 47, 66, 180

[AvdHdV01] Nesria Agray, Wiebe van der Hoek, and Erik P. de Vink. On BAN logics for industrial security protocols. In Barbara Dunin-Kęplicz and Edward Nawarecki, editors, *Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems*, pages 29–36, Cracow, 2001. 44, 48, 66, 175

[AvH04] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, 2004. 84

[Bac02] Adam Back. Hashcash - a denial of service counter-measure. Technical report, hashcash.org, August 2002. 43

[BAN88] Michael Burrows, Martín Abadi, and Roger Needham. Authentication: A practical study in belief and action. In M. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 325–342, 1988. 63, 179

[BAN89a] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Equipment Corporation Systems Research Center, February 28 1989. revised on February 22, 1990. 47, 55, 56, 57, 58, 61, 62, 63, 65, 179

[BAN89b] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 426(1871):233–271, December 1989. 44, 47, 55, 56, 57, 58, 61, 62, 63, 65, 179

[BAN89c] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM SIGOPS Operating Systems Review (Proceedings of the 12th ACM Symposium on Operating Systems Principles)*, 23(5):1–13, December 1989. 179

[BAN90a] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990. 179

[BAN90b] Michael Burrows, Martín Abadi, and Roger Needham. Rejoinder to Nessett. *ACM SIGOPS Operating Systems Review*, 24(2):39–40, April 1990. 182

[BAN94] Michael Burrows, Martín Abadi, and Roger Needham. A scope of a logic of authentication. appendix to DEC SRC research report 39, Digital Equipment Corporation Systems Research Center, May 13, 1994. 56, 179

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988. 113

[BCLL91] Gilles Brassard, Claude Crépeau, Sophie Laplante, and Christian Léger. Computationally convincing proofs of knowledge. In C. Choffrut and M. Jantzen, editors, *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science*, pages 251–262, 1991.

[BDG88] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity*. Monographs on Theoretical Computer Science. Springer-Verlag, Berlin / Heidelberg, 1988. 20

[Ber04] Jules J. Berman. Zero-check, a zero-knowledge protocol for reconciling patient identities across institutions. *Archives of Pathology and Laboratory Medicine*, 128(3):344–346, 2004. 115, 187

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 103–112, Chicago, Illinois, 2–4 1988. 105

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge.

In E.F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420, Berlin, 1993. Springer-Verlag. 105

[BGG94] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: the case of hashing and signing. In Y.G. Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, Berlin, 1994. Springer-Verlag. . 38, 40, 45

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography with application to virus protection. In *Proceedings of the 27th Annual Symposium on the Theory of Computing*. ACM, 1995. 38, 45

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yan. On the (im)possibility of obfuscating programs. Technical Report TR01-057, Electronic Colloquium on Computational Complexity, 2001. 37

[BGR95] Mihir Bellare, Roch Guerin, and Phillip Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In D. Coppersmith, editor, *Advances in Cryptology - CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28, Berlin, 1995. Springer-Verlag. 39

[Bin92] Ken Binmore. *Fun and Games: a Text on Game Theory*. DC Heath & Company, Lexington, MA, 1992. 5

[BKK95] Pieter A. Boncz, Fred Kwakkel, and Martin L. Kersten. High performance support for OO traversals in Monet. In *Proceedings British National Conference on Databases (BNCOD96)*, volume 1094 of *Lecture Notes in Computer Science*, pages 152–169, Berlin, 1995. Springer-Verlag. 100

[BM94] Colin Boyd and Wenbo Mao. On a limitation of BAN logic. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 240–247, Berlin, 1994. Springer-Verlag. 182

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In W. Fumy, editor, *Advances in Cryptology- EUROCRYPT 97 Proceedings*, volume 1233. Springer-Verlag, 1997. 37, 38, 39

[Bon02] Pieter A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002. 100

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, 1993. 36

[BST01] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001. 107, 113, 114, 116

[CC04] Tim Churches and Peter Christen. Some methods for blindfolded record linkage. *BMC Medical Informatics and Decision Making*, 4(9), June 2004. 115, 187

[CD05a] Mika Cohen and Mads Dam. A completeness result for BAN logic. In *Prococeedings of Methods for Modalities 4*, Berlin, December 2005. 47, 66, 182

[CD05b] Mika Cohen and Mads Dam. Logical omniscience in the semantics of ban logics. In Andrei Sabelfeld, editor, *Proceedings of the Foundations of Computer Security '05 — FCS'05*, pages 121–132, Chicago, 2005. 47, 66, 182

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of 30th Annual ACM Symposium on the Theory of Computing*, pages 209–218. ACM, 1998. 37

[CGKS98] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–982, November 1998. 100

[CH06] Łukasz Chmielewski and Jaap-Henk Hoepman. Fuzzy private matching. in submission, 2006. 189

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981. 99

[Cha85] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985. 99

[Cha92] David Chaum. Achieving electronic privacy. *Scientific American*, pages 96–101, 1992. 99

[CK85] George P. Copeland and Setrag N. Khoshafian. A decomposition storage model. In S.B. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin*, pages 268–279, New York, NY, USA, 1985. ACM Press. 100

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 131–140, Dallas, 1998. 43

[Cod70] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 100

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. 35

[Dam88] Ivan B. Damgård. Collision free hash functions and public key signature schemes. In D. Chaum and W.L. Price, editors, *Advances in Cryptology - EUROCRYPT '87*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216, Berlin, 1988. Springer-Verlag. 42

[Dam90] Ivan B. 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, Berlin, 1990. Springer-Verlag. 38

[Dam97] Ivan B. Damgård. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *Journal of Cryptology*, 10(3):163–

194, July 1997. 43

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD160: A strengthened version of RIPEMD. In *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82, 1996. 38, 40

[DDMP03] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. Secure protocol composition. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*, pages 11–23, New York, NY, USA, 2003. ACM Press. 174

[Dek00] Anthony H. Dekker. C3PO: a tool for automatic sound cryptographic protocol analysis. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop — CFSW-13*, pages 77–87. IEEE Computer Society Press, 2000. 47, 66, 180, 181

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. 29, 42, 55

[DH05] AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, Spring 2005. 82, 84

[vD03] Hans P. van Ditmarsch. The russian cards problem. *Studia Logica*, 75:31–62, 2003. 107

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, New York, NY, USA, 2002. ACM Press. 100

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In E.F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Berlin, 1993. Springer-Verlag. 43

[DPP94] Ivan B. Damgård, Torben Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In D.R. Stinson, editor, *Advances in Cryptology - CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 250–265, Berlin, 1994. Springer-Verlag. 43

[DQB95] Liliane Dusserre, Catherine Quantin, and Hocine Bouzelat. A one way public key cryptosystem for the linkage of nominal files in epidemiological studies. *Medinfo*, 8(1):644–647, 1995. 115, 187

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983. 23, 48

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, June 1985. 22

[ESAG02] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. Privacy preserving mining of association rules. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, July 2002. 99

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, May 1973. 43

[FFS87] Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 210–217, New York, NY, USA, June 1987. ACM Press.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, June 1988.

[FGR92] Joan Feigenbaum, Eric Grosse, and James A. Reeds. Cryptographic protection of membership lists. *Newsletter of the International Association for Cryptologic Research*, 9(1):16–20, 1992. 99, 112

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995. 47, 175

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *Proceedings of the Twentsixth Annual ACM Symposium on Theory of Computing*, pages 554–563. ACM Press, 1994. 24

[FLW91] Joan Feigenbaum, Mark Y. Liberman, and Rebecca N. Wright. Cryptographic protection of databases and software. In Joan Feigenbaum and Michael Merritt, editors, *Distributed Computing and Cryptography*, volume 2, pages 161–172, 1991. 99, 112

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 2037 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, 2004. Springer-Verlag. 111, 114, 116, 188, 189

[FNS75] Horst Feistel, W.A. Notz, and J. Lynn Smith. Some cryptographic techniques for machine-to-machine data communications. *Proceedings of the IEEE*, 63(11):1545–1554, 1975. 43

[FNW96] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996. 99, 106, 107, 114, 115, 172

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishability and witness hiding protocols. In *Proceedings of the 22nd Annual Symposium on the Theory of Computing*, pages 416–426, New York City, 1990. ACM Press.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley, 2003. 19, 46

[Get63] Edmund L. Gettier. Is justified true belief knowledge? *Analysis*, 23:121–123, 1963. 105

[GH05] Flavio D. Garcia and Jaap-Henk Hoepman. Off-line karma: A decentralized currency for peer-to-peer and grid applications. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security: Third International Conference, ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 364–377, Berlin / Heidelberg, 2005. Springer. 43

[GK05] Michael Grüninger and Joseph B. Kopena. Semantic integration through invariants. *AI Magazine*, 26(1):11–20, Spring 2005. 82, 84

[GKSG91] Virgil D. Gligor, Rajashekar Kailar, Stuart G. Stubblebine, and Li Gong. Logics for cryptographic protocols — virtues and limitations. In *Proceedings of the IEEE Computer Security Foundations Workshop IV (CFSW IV)*, pages 219–226, Los Alamitos, 1991. IEEE Computer Society Press. 181

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, Rhode Island, 1985. 24, 25, 105

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proofs. *Journal of the ACM*, 38:691–729, 1991. 26

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248, Los Angeles, 1990. IEEE Computer Society Press. 47, 50, 52, 66, 72, 75, 180, 183, 184, 186

[Gol02] Oded Goldreich. Zero-knowledge twenty years after its invention. Technical report, Department of Computer Science, Weizmann Institute of Science, 2002. 25, 26, 105

[GS91] Klaus Gaarder and Einar Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3(2):81–98, January 1991. 181

[GSG99] Stefanos Gritzalis, Diomidis Spinellis, and Panagiotis Georgiadis. Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications*, 22(8):697–709, May 1999. 48

[Gua98] Nicola Guarino. Formal ontology and information systems. In Nicola Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems*, pages 3–15, Trento, Italy, June 1998. IOS Press. 100

[Gut01] Joshua D. Guttman. Key compromise, strand spaces, and the authentication tests. *Electronic Notes in Theoretical Computer Science*, 47:1–21, 2001. 49

[Gut02] Joshua D. Guttman. Security protocol design via authentication tests. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 92–103, Los Alamitos, 2002. IEEE Computer Society Press. 49

[Hei05] Dorothee Heisenberg. *Negotiating Privacy: The European Union, the United States and Personal Data Protection*. Lynne Rienner Publishers, 2005. 104

[Hel61] Joseph Heller. *Catch-22*. Simon & Schuster, 1961.

[Hid04] Jan-Willem Hiddink. Informatie als waardegoed. Master's thesis, Rijksuniversiteit Groningen, August 2004. 90

[HIM+04] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The Piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, July 2004. 88

[HM84] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In Tiko Kameda, Jayadev Misra, Joseph Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 50–61. ACM, ACM Press, 1984. 56

[HM90] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. 56

[Hoa69] C.A.R. (Tony) Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969. 52

[HPvdM03] Joseph Y. Halpern, Riccardo Pucella, and Ron van der Meyden. Revisiting the foundations of authentication logics. Manuscript, 2003. 66, 182

[HS06] Theo Hooghiemstra and Dirk Schravendeel. Burger service nummer werkt. *NRC Handelsblad*, page 7, June 27 2006. 15

[HT07] Marc Hooghe and Wouter Teepe. Party profiles on the web. *New Media & Society*, to appear, 2007. 10

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996. 39

[Jac04] Bart Jacobs. Semantics and logic for security protocols. Manuscript, September 2004. 181

[Jac05] Bart Jacobs. Select before you collect. *Ars Aequi*, 54(12):1006–1009, December 2005. 6, 16

[JLS02] Stanislaw Jarecki, Patrick Lincoln, and Vitaly Shmatikov. Negotiated privacy (extended abstract). In *Proceedings of the International Symposium of Software Security (ISSS)*, pages 96–111, 2002. 99

[JY96] Markus Jakobsson and Moti Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 186–200, Berlin, 1996. Springer-Verlag. 107, 113, 114, 116

[KG91] Rajashekar Kailar and Virgil D. Gligor. On belief evolution in authentication protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop IV (CFSW IV)*, pages 103–116, Los Alamitos, 1991. IEEE Computer Society Press. 181

[KM05] Aggelos Kiayias and Antonina Mitrofanova. Testing disjointness of private datasets. In *Proceedings of Financial Cryptography 2005*, 2005. 113, 114, 115

[KM06] Aggelos Kiayias and Antonina Mitrofanova. Syntax-driven private evaluation of quantified membership queries. In *Proceedings of Applied Cryptography and Network Security 2006*, 2006. 109, 111, 114, 115

[Koo03] Barteld Kooi. *Knowledge, Chance and Change*. PhD thesis, Institute for Logic, Language and Communication, 2003. xv

[KP98] Joe Kilian and Erez Petrank. Identity escrow. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 169–185, Berlin, 1998. Springer-Verlag. 99

[KS92] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. 118, 188, 189

[KS04] Lea Kissner and Dawn Song. Private and threshold set-intersection. Technical Report CMU-CS-04-182, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2004. 111, 114, 116

[Kus97] Eyal Kushilevitz. *Communication Complexity*. Cambridge University Press, 1997. 21

[KW94] Volker Kessler and Gabriele Wedel. AUTLOG – an advanced logic of authentication. In *Proceedings of the 7th Computer Security Foundations Workshop (CSFW'94)*, pages 90–99, Los Alamitos, 1994. IEEE Computer Society Press. 47, 66, 180, 181

[Lip04] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In Chi Sung Liah, editor, *Advances in Cryptology - ASIACRYPT 2003*, Lecture Notes in Computer Science, pages 416–433, Berlin, 2004. Springer-Verlag. 22

[LLM05] Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. Private itemset support counting. In Wenbo Mao, Javier Lopez, and Guilin Wang, editors, *Information and Communications Security: 7th International Conference, ICICS 2005*, volume 3783 of *Lecture Notes in Computer Science*, pages 97–111, Berlin / Heidelberg, December 2005. Springer-Verlag. 111, 114, 116

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166, Berlin, 1996. Springer-Verlag. 76, 181

[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 36–47, Berlin, 2000. Springer-Verlag. 98

[MB93] Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop VI (CFSW VI)*, pages 147–158, Los Alamitos, 1993. IEEE Computer Society Press. 181

[Mer90a] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238, Berlin, 1990. Springer-Verlag. 42

[Mer90b] Ralph C. 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, Berlin, 1990. Springer-Verlag. 38

[Mil75] Gary L. Miller. Riemann's hypothesis and tests for primality. In *Proceedings of seventh annual ACM symposium on Theory of computing*, pages 234–239, New York, NY, USA, 1975. ACM Press. 21

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay — a secure two-party computation system. In *Proceedings of Usenix Security 2004*, 2004. 24

[Mom06] Laurens Mommers. Burger service nummer levert weinig service en veel risico's. *NRC Handelsblad*, page 7, May 30 2006. 15

[Moo05] Chris Mooney. *The Republican War on Science*. Basic Books, 2005. 5

[MV97a] Mastercard and Visa. *The SET Standard Book 1: Business Description, Version 1.0*. SETCO, May 31 1997. 66, 175

[MV97b] Mastercard and Visa. *The SET Standard Book 2: Programmer's Guide, Version 1.0*. SETCO, May 31 1997. 66, 175

[MV97c] Mastercard and Visa. *The SET Standard Book 3: Formal Protocol Definitions, Version 1.0*. SETCO, May 31 1997. 66, 175

[MvdH95] John-Jules Ch. Meyer and Wiebe van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press, 1995. 47, 175

[Nat92] National Institute of Standards and Technology (NIST). Proposed federal information processing standard for secure hash standard. *Federal Register*, 57(21):3747–3749, 1992. 40

[Nat02] National Institute of Standards and Technology (NIST). Secure hash standard. *Federal Information Processing Standards*, 180(2):1–71, 2002. 30, 40

[Nat04] National Institute of Standards and Technology (NIST). Secure hash standard, change notice 1. *Federal Information Processing Standards*, 180(2):72–79, 2004. 40

[Nes90] Dan M. Nessett. A Critique of the Burrows, Abadi and Needham Logic. *ACM SIGOPS Operating Systems Review*, 24(2):35–38, April 1990. 47, 66, 181

[NNR99] Moni Naor, Yael Naor, and Omer Reingold. Applied kid cryptography or how to convince your children you are not cheating. *Journal of Craptology*, 0(1), 1999. 26

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 245–254, New York, 1999. ACM Press. 112, 114, 116

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002. 181

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing. (May 15–17 1989: Seattle, WA, USA)*, pages 33–43, New York, 1989. ACM Press. 35

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In E.F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes*

in Computer Science, pages 31–53, Berlin, 1993. Springer-Verlag. 34

[vO93] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols (extended abstract). In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 232–243, New York, November 1993. ACM Press. 47, 66, 180

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, Cambridge, MA, 1994. 5

[ORSvH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995. 181

[Orw49] George Orwell. *Nineteen Eighty-Four*. Secker & Warburg, London, 1949.

[OYGB04] Christine M. O'Keefe, Ming Yung, Lifang Gu, and Rohan Baxter. Privacy-preserving data linkage protocols. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 94–102, New York, NY, USA, 2004. ACM Press. 115

[Pre93] Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, January 1993. 29

[Pre98] Bart Preneel. Cryptographic primitives for information authentication – state of the art. In Bart Preneel and Vincent Rijmen, editors, *State of the Art and Evolution of Computer Security and Industrial Cryptography*, volume 1528 of *Lecture Notes in Computer Science*, pages 50–105, Berlin, 1998. Springer-Verlag. 29

[Pre05] Bart Preneel. Hash functions: past, present and future. Invited Lecture at ASIACRYPT 2005, December 2005. 36

[PvO95] Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In D. Coppersmith, editor, *Advances in Cryptology - CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 1–14, Berlin, 1995. Springer-Verlag. 128

[QAD00] Catherine Quantin, François-André Allaert, and Liliane Dusserre. Anonymous statistical methods versus cryptographic methods in epidemiology. *International Journal of Medical Informatics*, 60(2):177–183, November 2000. 115, 187

[QBA+98a] Catherine Quantin, Hocine Bouzelat, François-André Allaert, Anne-Marie Benhamiche, Jean Faivre, and Liliane Dusserre. Automatic record hash coding and linkage for epidemiological follow-up data confidentiality. *Methods of Information in Medicine*, 37(3):271–277, September 1998. 115, 187

[QBA+98b] Catherine Quantin, Hocine Bouzelat, François-André Allaert, Anne-Marie Benhamiche, Jean Faivre, and Liliane Dusserre. How to ensure data security of an epidemiological follow-up:quality assessment of an anonymous record linkage procedure. *International Journal of Medical Informatics*, 49(1):117–122, March 1998. 115, 187

[QQQ+90] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guil-

lou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Tom Berson. How to explain zero-knowledge protocols to your children. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631, Berlin, 1990. Springer-Verlag. 26

[Rab78] Michael O. Rabin. Digitalized signatures. In R.A. DeMillo, R.J. Lipton, D.P. Dobkin, and A.K. Jones, editors, *Foundations of Secure Computation*, pages 155–166. Academic Press, New York, 1978. 42

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, February 1980. 21

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981. 22

[Raz92] Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, December 1992. 118, 188

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001. 82

[RCF04] Pradeep Ravikumar, William W. Cohen, and Stephen E. Fienberg. A secure protocol for computing string distance metrics. In *Proceedings of the Workshop on Privacy and Security Aspects of Data Mining*, pages 40–46, November 2004. 115

[Rei95] Raymond Reiter. On specifying database updates. *Journal of Logic Programming*, 25(1):53–91, 1995. 88

[Riv92] Ronald L. Rivest. The MD5 message-digest algorithm. Technical Report RFC 1321, IETF Network Working Group, 1992. 40

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. 96, 183, 185

[SC01] Paul Syverson and Iliano Cervesato. The logic of authentication protocols. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design: Tutorial Lectures*, number 2171 in Lecture Notes in Computer Science, pages 63–136. Springer-Verlag, 2001. 48, 54

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, New York, 1996. 19, 43, 56

[Sch98] Claus Peter Schnorr. The black-box model for cryptographic primitives. *Journal of Cryptology*, 11(2):125–140, March 1998. 37

[Sin99] Simon Singh. *The Code Book*. Doubleday Books, 1999. 19

[SOL06] SOLV, Mosho client care & communications. *SOLV FIVE+*. SOLV Attorneys, May 2006. 15

[Spa05] Karin Spaink. *Medische geheimen*. Nijgh & Van Ditmar, 2005. 16

[Sus06] Ron Suskind. *The One Percent Doctrine*. Simon & Schuster, 2006. 5

[SvO94] Paul Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14–28. IEEE Com-

puter Society Press, May 1994. 47, 51, 66, 180

[SvO96] Paul Syverson and Paul C. van Oorschot. A unified cryptographic protocol logic. Report 5540-227, Center for High Assurance Computer Systems, Naval Research Laboratory (NRL CHACS), 1996. 47, 66, 180

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000. 100

[Syv91] Paul Syverson. The value of semantics for the analysis of cryptographic protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop IV (CFSW IV)*, pages 228–229, Los Alamitos, 1991. IEEE Computer Society Press. 180

[Syv93] Paul Syverson. Adding time to a logic of authentication. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 97–101, New York, November 1993. ACM Press. 181

[Syv00] Paul Syverson. Towards a strand semantics for authentication logic. In Stephen Brookes, Achim Jung, Michael Mislove, and Andre Scedrov, editors, *Electronic Notes in Theoretical Computer Science*, 20, 2000. 66, 181

[Tee99] Wouter Teepe. Privacy-gerichte workflowanalyse, een verkenning aan de hand van color-x. Master's thesis, Rijksuniversiteit Groningen, December 1999. 10

[Tee04] Wouter Teepe. New protocols for proving knowledge of arbitrary secrets while not giving them away. In Sieuwert van Otterloo, Peter McBurney, Wiebe van der Hoek, and Michael Wooldridge, editors, *Proceedings of the First Knowledge and Games Workshop*, pages 99–116, Liverpool, July 2004. Department of Computer Science, University of Liverpool. 10

[Tee05a] Wouter Teepe. Een classificatie van persoonlijke partijprofielen een analyse vanuit de kennistechnologie. *Samenleving en Politiek*, pages 2–12, March 2005. 10

[Tee05b] Wouter Teepe. Integrity and dissemination control in administrative applications through information designators. *International Journal of Computer Systems Science & Engineering*, 20(5):377–386, September 2005. 10

[Tee05c] Wouter Teepe. Wetenschap kan conflict met Amerika oplossen. *het Financieele Dagblad*, page 7, August 15 2005. 10

[Tee06a] Wouter Teepe. BAN logic is not 'sound', constructing epistemic logics for security is difficult. In Barbara Dunin-Kęplicz and Rineke Verbrugge, editors, *Proceedings of Formal Approaches to Multi-Agent Systems 2006*, pages 79–91, August 2006. 10

[Tee06b] Wouter Teepe. Proving possession of arbitrary secrets while not giving them away, new protocols and a proof in GNY logic. *Synthese*, 149(2):409–443, March 2006. 10

[TH05] Wouter Teepe and Marc Hooghe. Interactief internetgebruik in tijden van verkiezingskoorts een analyse van de gebruikers van "wij kiezen partij voor u" in 2003 en 2004. *Samenleving en Politiek*, pages 73–88, March 2005. 10

[THG98] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *IEEE Symposium on Security and Privacy*, 1998. 174, 181

[THG99] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2-3):191–230, 1999. 174, 181

[Tom88] Martin Tompa. Zero knowledge interactive proofs of knowledge (a digest). In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 1–12, New York, NY, USA, 1988. ACM Press.

[Tsu92] Gene Tsudik. Message authentication with one-way hash functions. In *Proceedings of IEEE INFOCOM 1992*, pages 2055–2059, Los Angeles, 1992. IEEE Computer Society Press. 41

[TvdRO02] Wouter Teepe, Reind P. van de Riet, and Martin Olivier. Workflow analyzed for security and privacy in using databases. In Bhavani Thuraisingham, Reind P. van de Riet, Klaus R. Dittrich, and Zahir Tari, editors, *Data and Application Security, Development and Directions*, volume 73 of *IFIP International Federation for Information Processing*, pages 271–282, Boston, 2002. Springer. 10

[TvdRO03] Wouter Teepe, Reind P. van de Riet, and Martin Olivier. Workflow analyzed for security and privacy in using databases. *Journal of Computer Security*, 11(3):353–363, 2003. 10, 75, 99

[TW87] Martin Tompa and Heather Woll. Random self reducibility and zero knowledge interactive proofs of possession of information. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 472–482, 1987.

[UG96] Mike Uschold and Michael Grüninger. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, June 1996. 100

[WC81] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. 35, 115

[WK96] Gabriele Wedel and Volker Kessler. Formal semantics for authentication logics. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *Computer Security — ESORICS 96: 4th European Symposium on Research in Computer Security Rome*, number 1146 in Lecture Notes in Computer Science, pages 219–241, Berlin, 1996. Springer-Verlag. 47, 61, 66, 180, 181

[WSI03] Yodai Watanabe, Junji Shikata, and Hideki Imai. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 71–84, Berlin, 2003. Springer-Verlag. 108

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology - EURO-*

CRYPT 2005, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35, Berlin, 2005. Springer-Verlag. 40

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36, Berlin, 2005. Springer-Verlag. 40

[Yao79] Andrew C. Yao. Some complexity questions related to distributed computing. In *Proceedings of the eleventh annual ACM symposium on Theory of Computing*, pages 209–213, New York, NY, USA, 1979. ACM Press. 21

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, Los Angeles, 1982. IEEE Computer Society Press. 23, 24, 107, 108

[Yao86] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, Los Angeles, 1986. IEEE Computer Society Press. 24, 108

Author Index

A

Abadi, Martín . . . . . [AF04], [AF90], [AG99], [AR02], [AT91], [BAN88], [BAN89a], [BAN89b], [BAN89c], [BAN90a], [BAN90b], [BAN94]
Accorsi, Rafael . . . . . [ABV01]
Adleman, Leonard . . . . . [RSA78]
Agrawal, Rakesh . [AES03], [ESAG02]
Agray, Nesria . . . . . [AvdHdV01]
Aiello, Bill . . . . . [AIR01]
Allaert, François-André . . . . [QAD00], [QBA+98a], [QBA+98b]
Alonso, Gustavo . . . . . [ACKM04]
Anderson, Ross . . . . . [And93]
Antoniou, Grigoris . . . . . [AvH04]

B

Back, Adam . . . . . [Bac02]
Balcázar, José Luis . . . . . [BDG88]
Barak, Boaz . . . . . [BGI+01]
Basin, David . . . . . [ABV01]
Baxter, Rohan . . . . . [OYGB04]
Bellare, Mihir . . . . . [BG93], [BGG94], [BGG95], [BGR95], [BM97], [BR93]
Benhamiche, Anne-Marie [QBA+98a], [QBA+98b]
Berman, Jules . . . . . [Ber04]
Bernstein, Philip . . . . . [RB01]
Berson, Tom . . . . . [QQQ+90]
Binmore, Ken . . . . . [Bin92]
Blum, Manuel . . . . . [BFM88]
Boncz, Pieter . . . . . [Bon02], [BKK95]
Bosselaers, Antoon . . . . . [DBP96]
Boudot, Fabrice . . . . . [BST01]
Bouzelat, Hocine . . . . . [DQB95], [QBA+98a], [QBA+98b]
Boyd, Colin . . . . . [BM94], [MB93]
Brassard, Gilles . . [BCC88], [BCLL91]
Burrows, Michael . . . . . [BAN88], [BAN89a], [BAN89b], [BAN89c], [BAN90a], [BAN90b], [BAN94]

C

Canetti, Ran . . . . . [CGH98], [CMR98]
Carter, Lawrence . . . [CW79], [WC81]
Casati, Fabio . . . . . [ACKM04]
Cervesato, Iliano . . . . . [SC01]
Chaum, David . . . . [BCC88], [Cha81], [Cha85], [Cha92]
Chmielewski, Łukasz . . . . . [CH06]
Chor, Benny . . . . . [CGKS98]
Christen, Peter . . . . . [CC04]
Churches, Tim . . . . . [CC04]
Codd, Edgar . . . . . [Cod70]
Cohen, Mika . . . . . [CD05a], [CD05b]
Cohen, William . . . . . [RCF04]
Copeland, George . . . . . [CK85]
Crépeau, Claude . [BCC88], [BCLL91]

D

Dam, Mads . . . . . [CD05a], [CD05b]
Damgård, Ivan . . . [Dam88], [Dam90], [Dam97], [DPP94]
Datta, Anupam . . . . . [DDMP03]
Dekker, Anthony . . . . . [Dek00]
Derek, Ante . . . . . [DDMP03]
Díaz, Josep . . . . . [BDG88]
Diffie, Whitfield . . . . . [DH76]
van Ditmarsch, Hans . . . . . [vD03]
Doan, AnHai . . . . [DH05], [DMDH02]
Dobbertin, Hans . . . . . [DBP96]
Dolev, Danny . . . . . [DY83]
Domingos, Pedro . . . . . [DMDH02]
Dusserre, Liliane [DQB95], [QAD00], [QBA+98a], [QBA+98b]
Dwork, Cynthia . . . . . [DN93]

E

Even, Shimon . . . . . [EGL85]
Evfimievski, Alexandre . . . . . [AES03], [ESAG02]

F

Fagin, Ronald . . [FHMV95], [FNW96]
Faivre, Jean . [QBA+98a], [QBA+98b]
Feige, Uriel . . . . . [FFS87], [FFS88], [FKN94], [FS90]
Feigenbaum, Joan . . [AF90], [FLW91], [FGR92]
Feistel, Horst . . . . . [Fei73], [FNS75]
Feldman, Paul . . . . . [BFM88]
Ferguson, Niels . . . . . [FS03]
Fiat, Amos . . . . . [FFS87], [FFS88]
Fienberg, Stephen . . . . . [RCF04]
Fournet, Cédric . . . . . [AF04]
Freedman, Michael . . . . . [FNP04]

G

Gaarder, Klaus . . . . . [GS91]
Gabarró, Joaquim . . . . . [BDG88]
Garcia, Flavio . . . . . [GH05]
Gehrke, Johannes . . . . . [ESAG02]
Georgiadis, Panagiotis . . . . . [GSG99]
Gettier, Edmund . . . . . [Get63]
Gligor, Virgil . . . . . [GKSG91], [KG91]
Goldreich, Oded . . . [BG93], [BGG94], [BGG95], [BGI+01], [CGH98], [CGKS98], [EGL85], [GMW91], [Gol02]
Goldwasser, Shafi [BGG94], [BGG95], [GMR85]
Gong, Li . . . . . [GKSG91], [GNY90]
Gordon, Andrew . . . . . [AG99]
Grüninger, Michael . . [GK05], [UG96]
Gritzalis, Stefanos . . . . . [GSG99]
Grosse, Eric . . . . . [FGR92]
Gu, Lifang . . . . . [OYGB04]
Guarino, Nicola . . . . . [Gua98]
Guerin, Roch . . . . . [BGR95]
Guillou (whole family) . . . [QQQ+90]
Guttman, Joshua . . . [Gut01], [Gut02], [THG98], [THG99]

H

Halevi, Shai . . . . . [CGH98]
Halevy, Alon . . . [DH05], [DMDH02], [HIM+04]
Halpern, Joseph [FHMV95], [HM84], [HM90], [HPvdM03]
van Harmelen, Frank . . . . . [AvH04]
Heisenberg, Dorothee . . . . . [Hei05]
Heller, Joseph . . . . . [Hel61]
Hellman, Martin . . . . . [DH76]
von Henke, Friedrich . . . . [ORSvH95]
Herzog, Jonathan . [THG98], [THG99]
Hiddink, Jan-Willem . . . . . [Hid04]
Hoare, Tony . . . . . [Hoa69]
van der Hoek, Wiebe . . [AvdHdV01], [MvdH95]
Hoepman, Jaap-Henk . . . . . [GH05], [CH06]
Hooghe, Marc . . . . . [HT07], [TH05]
Hooghiemstra, Theo . . . . . [HS06]

I

Imai, Hideki . . . . . [WSI03]
Impagliazzo, Russell . . . . . [BGI+01], [IN96]
Ishai, Yuval . . . . . [AIR01]
Ives, Zachary . . . . . [HIM+04]

J

Jacobs, Bart . . . . . [Jac04], [Jac05]
Jakobsson, Markus . . . . . [JY96]
Jarecki, Stanislaw . . . . . [JLS02]

K

Kailar, Rajashekar [GKSG91], [KG91]
Kersten, Martin . . . . . [BKK95]
Kessler, Volker . . . . . [KW94], [WK96]
Khoshafian, Setrag . . . . . [CK85]
Kiayias, Aggelos . . . . [KM05], [KM06]
Kilian, Joe . . . . . [FKN94], [KP98]
Kissner, Lea . . . . . [KS04]

Kooi, Barteld . . . . . [Koo03]
Kopena, Joseph . . . . . [GK05]
Kuno, Harumi . . . . . [ACKM04]
Kushilevitz, Eyal . [CGKS98], [Kus97]
Kwakkel, Fred . . . . . [BKK95]

L

Lamport, Leslie 42
Laplante, Sophie . . . . . [BCLL91]
Laur, Sven . . . . . [LLM05]
Léger, Christian . . . . . [BCLL91]
Lempel, Abraham . . . . . [EGL85]
Liberman, Mark . . . . . [FLW91]
Lincoln, Patrick . . . . . [JLS02]
Lindell, Yehuda . . . . . [LP00]
Lipmaa, Helger . . . . [Lip04], [LLM05]
Lowe, Gavin . . . . . [Low96]

M

Machiraju, Vijay . . . . . [ACKM04]
Madhavan, Jayant . . . . . [DMDH02], [HIM+04]
Malkhi, Dahlia . . . . . [MNPS04]
Mao, Wenbo . . . . . [BM94], [MB93]
Mastercard . . . . . [MV97a], [MV97b], [MV97c]
Merkle, Ralph . . . [Mer90a], [Mer90b]
van der Meyden, Ron . . . [HPvdM03]
Meyer, John-Jules . . . . . [MvdH95]
Micali, Silvio . . . . [BFM88], [GMR85], [GMW91]
Micciancio, Daniele [BM97], [CMR98]
Mielikäinen, Taneli . . . . . [LLM05]
Miller, Gary . . . . . [Mil75]
Mitchell, John . . . . . [DDMP03]
Mitrofanova, Antonina . . . . . [KM05], [KM06]
Mommers, Laurens . . . . . [Mom06]
Mooney, Chris . . . . . [Moo05]
Mork, Peter . . . . . [HIM+04]
Moses, Yoram . . . [FHMV95], [HM84], [HM90]

N

Naor, Moni . . . . . [DN93], [FKN94], [FNW96], [IN96], [NNR99], [NP99], [NY89]
Naor, Yael . . . . . [NNR99]
Needham, Roger [BAN88], [BAN89a], [BAN89b], [BAN89c], [BAN90a], [BAN90b], [BAN94], [GNY90]
Nessett, Dan . . . . . [Nes90]
Nipkow, Tobias . . . . . [NPW02]
Nisan, Noam . . . . . [MNPS04]
Nissim, Kobbi . . . . . [FNP04]
NIST . . . . . [Nat92], [Nat02], [Nat04]
Notz, W. . . . . [FNS75]

O

Okamoto, Tatsuaki . . . . . [Oka93]
O'Keefe, Christine . . . . . [OYGB04]
Olivier, Martin . . . . . [TvdRO03]
van Oorschot, Paul . [vO93], [PvO95], [SvO94], [SvO96]
Orwell, George . . . . . [Orw49]
Osborne, Martin . . . . . [OR94]
Owre, Sam . . . . . [ORSvH95]

P

Paulson, Lawrence . . . . . [NPW02]
Pavlovic, Dusko . . . . . [DDMP03]
Pedersen, Torben . . . . . [DPP94]
Perrig, Adrian . . . . . [SWP00]
Petrank, Erez . . . . . [KP98]
Pfitzmann, Birgit . . . . . [DPP94]
Pinkas, Benny . . . . . [FNP04], [LP00], [MNPS04], [NP99]
Preneel, Bart . . . . . [DBP96], [Pre05], [Pre93], [Pre98], [PvO95]
Pucella, Riccardo . . . . . [HPvdM03]

Q

Quantin, Catherine . . . . . [DQB95], [QAD00], [QBA+98a], [QBA+98b]
Quisquater (whole family) [QQQ+90]

R

Rabin, Michael . . . . . [Rab78], [Rab80], [Rab81]
Rackoff, Charles . . . . . [GMR85]

Rahm, Erhard . . . . . [RB01]
Ravikumar, Pradeep . . . . . [RCF04]
Razborov, Alexander . . . . . [Raz92]
Reeds, James . . . . . [FGR92]
Reingold, Omer . . [AIR01], [CMR98], [NNR99]
Reiter, Raymond . . . . . [Rei95]
van de Riet, Reind . . . . . [TvdRO03]
Rivest, Ronald . . . . . [Riv92], [RSA78]
Rogaway, Phillip . . . [AR02], [BGR95], [BR93]
Rubinstein, Ariel . . . . . [OR94]
Rudich, Steven . . . . . [BGI+01]
Rushby, John . . . . . [ORSvH95]

S

Sahai, Amit . . . . . [BGI+01]
Schneier, Bruce . . . . . [FS03], [Sch96]
Schnorr, Claus . . . . . [Sch98]
Schoenmakers, Berry . . . . . [BST01]
Schravendeel, Dirk . . . . . [HS06]
Sella, Yaron . . . . . [MNPS04]
Shamir, Adi . [FFS87], [FFS88], [FS90], [RSA78]
Shankar, Natarajan . . . . . [ORSvH95]
Shikata, Junji . . . . . [WSI03]
Shmatikov, Vitaly . . . . . [JLS02]
Singh, Simon . . . . . [Sin99]
Smith, Lynn . . . . . [FNS75]
Snekkenes, Einar . . . . . [GS91]
SOLV Attorneys . . . . . [SOL06]
Song, Dawn . . . . . [KS04], [SWP00]
Spaink, Karin . . . . . [Spa05]
Spinellis, Diomidis . . . . . [GSG99]
Srikant, Ramakrishnan . . . . . [AES03], [ESAG02]
Stubblebine, Stuart . . . . . [GKSG91]
Suciu, Dan . . . . . [HIM+04]
Sudan, Madhu . . . . . [CGKS98]
Suskind, Ron . . . . . [Sus06]
Syverson, Paul . . . . . [SC01], [SvO94], [SvO96], [Syv91], [Syv93], [Syv00]

T

Tatarinov, Igor . . . . . [HIM+04]
Teepe, Wouter . . . . . [HT07], [Tee04], [Tee05a], [Tee05b], [Tee05c], [Tee06b], [Tee06a], [Tee99], [TH05], [TvdRO03]
Thayer Fábrega, Javier . . . . . [THG98], [THG99]
Tompa, Martin . . . . . [Tom88], [TW87]
Traoré, Jacques . . . . . [BST01]
Tsudik, Gene . . . . . [Tsu92]
Tuttle, Mark . . . . . [AT91]

U

Uschold, Mike . . . . . [UG96]

V

Vadhan, Salil . . . . . [BGI+01]
Vardi, Moshe . . . . . [FHMV95]
Viganò, Luca . . . . . [ABV01]
de Vink, Erik . . . . . [AvdHdV01]
Visa . . . . [MV97a], [MV97b], [MV97c]

W

Wagner, David . . . . . [SWP00]
Wang, Xiaoyun . . . . [WY05], [WYY05]
Watanabe, Yodai . . . . . [WSI03]
Wedel, Gabriele . . . . [KW94], [WK96]
Wegman, Mark . . . . . [CW79], [WC81]
Wenzel, Markus . . . . . [NPW02]
Wigderson, Avi . . . . . [GMW91]
Winkler, Peter . . . . . [FNW96]
Winternitz, Robert 42
Woll, Heather . . . . . [TW87]
Wright, Rebecca . . . . . [FLW91]

Y

Yahalom, Raphael . . . . . [GNY90]
Yan, Ke . . . . . [BGI+01]
Yao, Andrew . . . . . [DY83], [Yao79], [Yao82], [Yao86]
Yin, Yiqun . . . . . [WYY05]
Yu, Hongbo . . . . . [WY05], [WYY05]
Yung, Ming . . . . . [OYGB04]
Yung, Moti . . . . . [JY96], [NY89]

SIKS Dissertation Series

1998-1 Johan van den Akker (CWI) *DEGAS — An Active, Temporal Database of Autonomous Objects*
1998-2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*
1998-3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*
1998-4 Dennis Breuker (UM) *Memory versus Search in Games*
1998-5 E.W.Oskamp (RUL) *Computerondersteuning bij Straftoemeting*
1999-1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products*
1999-2 Rob Potharst (EUR) *Classification using decision trees and neural nets*
1999-3 Don Beal (UM) *The Nature of Minimax Search*
1999-4 Jacques Penders (UM) *The practical Art of Moving Physical Objects*
1999-5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*
1999-6 Niek J.E. Wijngaards (VU) *Re-design of compositional systems*
1999-7 David Spelt (UT) *Verification support for object database design*
1999-8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*
2000-1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*
2000-2 Koen Holtman (TUE) *Prototyping of CMS Storage Management*
2000-3 Carolien M.T. Metselaar (UVA) *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief*
2000-4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*
2000-5 Ruud van der Pol (UM) *Knowledge-based Query Formulation in Information Retrieval*
2000-6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*
2000-7 Niels Peek (UU) *Decision-theoretic Planning of Clinical Patient Management*
2000-8 Veerle Coupé (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*
2000-9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*
2000-10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*
2000-11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*
2001-1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*
2001-2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*
2001-3 Maarten van Someren (UVA) *Learning as problem solving*
2001-4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*
2001-5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*
2001-6 Martijn van Welie (VU) *Task-based User Interface Design*
2001-7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*
2001-8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*
2001-9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

2001-10 Maarten Sierhuis (UVA) *Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*
2001-11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*
2002-01 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*
2002-02 Roelof van Zwol (UT) *Modelling and searching web-based document collections*
2002-03 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*
2002-04 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*
2002-05 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents*
2002-06 Laurens Mommers (UL) *Applied legal epistemology; Building a knowledge-based ontology of the legal domain*
2002-07 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*
2002-08 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*
2002-09 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*
2002-10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*
2002-11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*
2002-12 Albrecht Schmidt (UVA) *Processing XML in Database Systems*
2002-13 Hongjing Wu (TUE) *A Reference Architecture for Adaptive Hypermedia Applications*
2002-14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*
2002-15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*
2002-16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*
2002-17 Stefan Manegold (UVA) *Understanding, Modeling, and Improving Main-Memory Database Performance*
2003-01 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*
2003-02 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*
2003-03 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*
2003-04 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*
2003-05 Jos Lehmann (UVA) *Causation in Artificial Intelligence and Law — A modelling approach*
2003-06 Boris van Schooten (UT) *Development and specification of virtual environments*
2003-07 Machiel Jansen (UVA) *Formal Explorations of Knowledge Intensive Tasks*
2003-08 Yongping Ran (UM) *Repair Based Scheduling*
2003-09 Rens Kortmann (UM) *The resolution of visually guided behaviour*
2003-10 Andreas Lincke (UvT) *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*
2003-11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*
2003-12 Roeland Ordelman (UT) *Dutch speech recognition in multimedia information retrieval*
2003-13 Jeroen Donkers (UM) *Nosce Hostem — Searching with Opponent Models*
2003-14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*
2003-15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*
2003-16 Menzo Windhouwer (CWI) *Feature Grammar Systems — Incremental Maintenance of Indexes to Digital Media Warehouses*
2003-17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*
2003-18 Levente Kocsis (UM) *Learning Search Decisions*
2004-01 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*

2004-02 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*
2004-03 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*
2004-04 Chris van Aart (UVA) *Organizational Principles for Multi-Agent Architectures*
2004-05 Viara Popova (EUR) *Knowledge discovery and monotonicity*
2004-06 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*
2004-07 Elise Boltjes (UM) *Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes*
2004-08 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise*
2004-09 Martin Caminada (VU) *For the Sake of the Argument; explorations into argument-based reasoning*
2004-10 Suzanne Kabel (UVA) *Knowledge-rich indexing of learning-objects*
2004-11 Michel Klein (VU) *Change Management for Distributed Ontologies*
2004-12 The Duy Bui (UT) *Creating emotions and facial expressions for embodied agents*
2004-13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*
2004-14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*
2004-15 Arno Knobbe (UU) *Multi-Relational Data Mining*
2004-16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*
2004-17 Mark Winands (UM) *Informed Search in Complex Games*
2004-18 Vania Bessa Machado (UVA) *Supporting the Construction of Qualitative Knowledge Models*
2004-19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*
2004-20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*
2005-01 Floor Verdenius (UVA) *Methodological Aspects of Designing Induction-Based Applications*
2005-02 Erik van der Werf (UM) *AI techniques for the game of Go*
2005-03 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*
2005-04 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*
2005-05 Gabriel Infante-Lopez (UVA) *Two-Level Probabilistic Grammars for Natural Language Parsing*
2005-06 Pieter Spronck (UM) *Adaptive Game AI*
2005-07 Flavius Frasincar (TUE) *Hypermedia Presentation Generation for Semantic Web Information Systems*
2005-08 Richard Vdovjak (TUE) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*
2005-09 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*
2005-10 Anders Bouwer (UVA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*
2005-11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering — A Decentralized Approach to Search*
2005-12 Csaba Boer (EUR) *Distributed Simulation in Industry*
2005-13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*
2005-14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*
2005-15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*
2005-16 Joris Graaumans (UU) *Usability of XML Query Languages*
2005-17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*
2005-18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*
2005-19 Michel van Dartel (UM) *Situated Representation*
2005-20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*
2005-21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*
2006-01 Samuil Angelov (TUE) *Foundations of B2B Electronic Contracting*
2006-02 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*
2006-03 Noor Christoph (UVA) *The role of metacognitive skills in learning to solve problems*

2006-04 Marta Sabou (VU) *Building Web Service Ontologies*
2006-05 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*
2006-06 Ziv Baida (VU) *Software-aided Service Bundling — Intelligent Methods & Tools for Graphical Service Modeling*
2006-07 Marko Smiljanic (UT) *XML schema matching — balancing efficiency and effectiveness by means of clustering*
2006-08 Eelco Herder (UT) *Forward, Back and Home Again — Analyzing User Behavior on the Web*
2006-09 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*
2006-10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*
2006-11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*
2006-12 Bert Bongers (VU) *Interactivation — Towards an e-cology of people, our technological environment, and the arts*
2006-13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*
2006-14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign — towards a Theory of Requirements Change*
2006-15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*
2006-16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*
2006-17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*
2006-18 Valentin Zhizhkun (UVA) *Graph transformation for Natural Language Processing*
2006-19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*
2006-20 Marina Velikova (UvT) *Monotone models for prediction in data mining*
2006-21 Bas van Gils (RUN) *Aptness on the Web*
2006-22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*
2006-23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*
2006-24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*
2006-25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*
2006-26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*
2006-27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*
2006-28 Borkur Sigurbjornsson (UVA) *Focused Information Access using XML Element Retrieval*
2007-01 Kees Leune (UvT) *Control and Service-Oriented Architectures*
2007-02 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality — A Formal Approach*

About the Author

Wouter Teepe was born in Darmstadt, Germany on February 23, 1977, and holds the Dutch nationality. In January 2000, he obtained his Master's degree at the University of Groningen (*Technische Cognitiewetenschap*). Since then, Wouter Teepe has been affiliated to the department of Artificial Intelligence of the University of Groningen. First only as a lecturer in Knowledge Systems, and since June 2002 also as a PhD student. He finished working on his PhD thesis in August 2006.

Wouter Teepe has two main research interests. The first reseach interest is security, privacy and databases, which is reflected in the thesis at hand.

The other research interest is in online expert systems for voting advice during elections ("party profile websites"). Since 1998, Wouter Teepe has built such expert systems for *Planet Internet* (an internet service provider), *Kennisnet* (a portal site for schools), *Politicsinfo* (a portal site about politics) and *De Standaard* (a leading newspaper in Belgium).

Wouter Teepe is an editor of the Journal of Information Technology & Politics (`http://www.jitp.net`).

Selected Publications

[Tee99] Wouter Teepe. Privacy-gerichte workflowanalyse, een verkenning aan de hand van color-x. Master's thesis, Rijksuniversiteit Groningen, December 1999.

[TvdRO03] Wouter Teepe, Reind P. van de Riet, and Martin Olivier. Workflow analyzed for security and privacy in using databases. *Journal of Computer Security*, 11(3):353–363, 2003.

[Tee05b] Wouter Teepe. Integrity and dissemination control in administrative applications through information designators. *International Journal of Computer Systems Science & Engineering*, 20(5):377–386, September 2005.

[Tee06b] Wouter Teepe. Proving possession of arbitrary secrets while not giving them away, new protocols and a proof in GNY logic. *Synthese*, 149(2):409–443, March 2006.

[Tee06a] Wouter Teepe. BAN logic is not 'sound', constructing epistemic logics for security is difficult. In Barbara Dunin-Kęplicz and Rineke Verbrugge, editors, *Proceedings of Formal Approaches to Multi-Agent Systems 2006*, pages 79–91, August 2006.

[HT07] Marc Hooghe and Wouter Teepe. Party profiles on the web. *New Media & Society*, to appear, 2007.

Samenvatting

Het thema van de bescherming van persoonsgegevens is actueler dan ooit. Een paradoxale eigenschap van persoonsgegevens is dat zowel het geheim houden, als het uitwisselen ervan gedaan kan worden onder het argument van 'veiligheid'. Het geheim houden van persoonsgegevens verhoogt veiligheid doordat deze gegevens niet misbruikt kunnen worden. Het uitwisselen van persoonsgegevens verhoogt veiligheid omdat het opsporingsdiensten helpt criminelen en terroristen te vangen. Zowel de argumenten voor het geheim houden van gegevens, als die voor het uitwisselen van gegevens zijn valide.

Het probleem is helder: het uitwisselen van gegevens en het geheim houden van gegevens lijkt niet, of althans moeilijk, samen te kunnen gaan.

Dit is niet alleen een probleem in de discussie tussen de voorvechters van privacy en de voorstanders van verregaande opsporingsbevoegdheden. Ook opsporingsdiensten zelf worstelen met het spanningsveld van uitwisseling versus geheimhouding: het is makkelijker een subject (bijvoorbeeld een verdachte) in de gaten te houden als hij of zij daar niet op beducht is. Wanneer het subject weet dat hij onderwerp van onderzoek is, kan hij of zij bijvoorbeeld mogelijk bezwarende bewijzen vernietigen. Hoe meer mensen binnen een opsporingsorganisatie weet hebben van een lopend onderzoek, hoe groter de kans is dat er gelekt wordt naar het subject. Aan de andere kant, hoe meer mensen binnen opsporingsdienst weet hebben van een lopend onderzoek, hoe meer mensen kunnen meehelpen met dat onderzoek.

Het hoofddoel van dit proefschift is om te onderzoeken of het mogelijk is om oplossingen voor dit spanningsveld te vinden. De resultaten van het onderzoek zijn van fundamentele en praktische waarde. Aan de fundamentele kant laten we zien, dat een een aantal problemen überhaupt oplosbaar is. Aan de praktische kant laten we zien dat deze oplossingen niet slechts theoretisch zijn, maar ook zonder al te veel problemen kunnen worden toegepast om bestaande, praktische problemen op te lossen. De oplossingen die gepresenteerd worden in dit proefschift bieden beleidsmakers de ruimte om de bescherming van privacy enerzijds, en het uitwisselen van persoonsgegevens voor terrorismebestrijding anderzijds, goed samen te laten gaan. In plaats van of/of, is er de mogelijkheid voor en/en, als de beleidsmakers het willen.

Enige relativering is hierbij wel op zijn plaats. Niet *alle* problemen rondom privacy en uitwisseling van persoonsgegevens kunnen worden opgelost, slechts *enkele*. Er is echter geen enkele reden om te veronderstellen dat dit proefschrift de mogelijkheden om dit type problemen op te lossen, heeft uitgeput. Dit proefschift is slechts een begin: we laten zien dat het überhaupt mogelijk is dit type problemen op te lossen; toekomstig onderzoek kan het palet van oplossingen verder uitbreiden.

In hoofdstuk 1 bespreken we de context van het onderzoek: welke maatschappelijke kwesties zijn er, waarbij zowel geheimhouding als uitwisseling van persoonsgegevens kan geschieden onder het motto van 'veiligheid'? Wat hebben deze kwesties gemeen? Ook wordt er van een praktijktoepassing uit de doeken gedaan hoe deze uitwisseling op het moment georganiseerd is. Deze toepassing is het uitwisselen van opsporingsinformatie tussen regiokorpsen van de Nederlandse politie. Verder besteden we aandacht aan de vraag of centrale opslag van persoonsgegevens wenselijk is. Uiteraard wordt in hoofdstuk 1 de structuur van dit proefschrift nader uitgelegd.

Omdat de lezer wellicht niet vertrouwd is met de theoretische achtergronden die te maken hebben met 'computerbeveiliging', bevat hoofdstuk 2 een bondige samenvatting en uitleg van veel begrippen en onderzoeksvelden waar dit proefschrift veelvuldig naar verwijst. Daaronder valt een aantal relatief brede onderwerpen, zoals encryptie, authorisatie, authenticatie, complexiteitstheorie en probabilistische algorithmen, maar ook een aantal vrij specialistische onderwerpen, zoals *oblivious transfer*, *secure multiparty computation* en *zero-knowledge proofs*.

De *cryptografische hashfunctie* heeft een dermate belangrijke rol in dit proefschrift, dat hoofdstuk 3 in zijn geheel gewijd is aan het bespreken van de eigenschappen van de cryptografische hashfunctie. Kort gezegd is een cryptografische hashfunctie een gereedschap om een soort vingerafdruk te maken van een blok gegevens. Deze vingerafdruk kan gebruikt worden om de informatie te identificeren of te herkennen, maar verklapt verder niets over die gegevens. Technisch gezien heeft het nogal wat voeten in de aarde om precies te definiëren wat een cryptografische hashfunctie is, en hoe je er eentje zou moeten maken. In dit kader bespreken we *niet-incrementaliteit*, een nieuwe, optionele eigenschap van cryptografische hashfuncties.

Een ander belangrijk gereedschap zijn *authenticatielogica's*, dat zijn logica's waarmee je *cryptografische protocollen* kunt analyseren. In hoofdstuk 4 wordt uitgelegd hoe authenticatielogica's in elkaar zitten en hoe je ze kunt gebruiken. De oudste authenticatielogica is BAN-logica, en andere authenticatielogica's, zoals GNY-logica, zijn afgeleid van BAN-logica.

In hoofdstuk 5 laten we zien dat BAN-logica een fout bevat. Deze fout in BAN-logica is dat cryptografische hashfuncties niet juist gemodelleerd worden. Dit demonstreren we aan de hand van een vrij eenvoudig protocol. Door deze fout blijkt het mogelijk om binnen BAN-logica om uit ware feiten onjuiste feiten af te leiden, wat natuurlijk zeer ongewenst is. Andere authenticatielogica's, zoals GNY-logica, bevatten deze fout niet.

Authenticatielogica's zijn niet altijd helemaal geschikt om bepaalde protocollen te analyseren. Ook de protocollen die in dit proefschrift worden geanalyseerd kunnen niet zonder meer geanalyseerd worden. Om een analyse toch mogelijk te maken, breiden we in hoofdstuk 6 de GNY-logica zó uit, dat het geschikt is voor onze doeleinden. De modellering van bepaalde eigenschappen van cryptografische hashfuncties wordt toegevoegd aan GNY-logica. Ook maken we een aantal zaken in GNY-logica preciezer, waardoor een nauwkeuriger analyse van protocollen mogelijk is.

Het daadwerkelijke beschermen van (persoons)gegevens is het onderwerp van hoofdstuk 7. Het beschermen van persoonsgegevens wordt moeilijk gemaakt door de wens vele informatiebronnen en databases aan elkaar te koppelen. Een gebruikelijke manier om die koppeling tot stand te brengen is het royaal toegang geven tot databases. Wij stellen een geheel andere aanpak voor, waarbij informatie juist zoveel mogelijk geïsoleerd wordt. Deze isolatie helpt bij de bescherming van de gegevens, maar óók bij het efficiënt koppelen van de databases. Centraal in onze aanpak staat de *information designator*, een soort pseudoniem voor informatie.

Kennisauthenticatie, het vergelijken van gegevens (geheimen) zonder deze te lekken is het onderwerp van hoofdstuk 8. Er wordt in kaart gebracht waaraan een communicatieprotocol moet voldoen om de vraag 'Ken jij de geheimen die ik ken?' correct te kunnen beantwoorden — zonder dat de geheimen zelf vrijgegeven worden, natuurlijk. Een aantal subtiele maar op essentiële punten verschillende variaties van dit probleem worden uitgelegd. Een belangrijke variatie is of *één* geheim wordt vergeleken met *vele* geheimen ('1-to-many'), of dat *vele* geheimen worden vergeleken met *vele* geheimen ('many-to-many').

In hoofdstuk 9 presenteren we het nieuwe T-1 protocol, dat op zeer efficiënte wijze 1-to-many kennisauthenticatie implementeert. Het T-1 protocol maakt intensief gebruik van cryptografische hashfuncties. Er is een variant waarin alleen maar cryptografische hashfuncties gebruikt worden, en een efficiëntere variant waarin ook encryptie gebruikt wordt als bouwsteen. Het T-1 protocol wordt geanalyseerd in de uitgebreide versie van GNY-logica.

Het T-2 protocol, dat we presenteren in hoofdstuk 10, is een nieuw en efficiënt protocol voor many-to-many kennisauthenticatie. Hiermee is het mogelijk de overlap tussen twee lijsten te bepalen, zonder dat bekend wordt wat er *buiten* het overlappende deel zit. Het T-2 protocol is een parallelle compositie van het T-1 protocol waarbij een aantal optimalisaties is toegepast. De belangrijkste optimalisatie is het gebruik van *prefix trees* om de hoeveelheid te communiceren bits te verkleinen.

Hoofdstuk 11 is de conclusie van dit proefschrift, waarin alle resultaten van het onderzoek netjes op een rij worden gezet, en aanbevelingen voor vervolgonderzoek worden gedaan.