Security of RFID protocols

Ton van Deursen

Supervisor:

Prof. Dr. S. Mauw (University of Luxembourg)

Daily advisor:

Dr. S. Radomirović (University of Luxembourg)

Printed by Ipskamp Drukkers, Enschede
Cover design by N.J.A. van Deursen

The author was employed at the University of Luxembourg and received support from the Ministry of Culture, Higher Education, and Research Luxembourg (reference BFR07/103) and the National Research Fund Luxembourg (references TR-PHD-BFR07-103 and EXT-BFR07-103) in the project "Security Protocols in Identity Management".

Printing of this thesis was supported by the National Research Fund, Luxembourg (reference FNR/11/AM4/58).

UNIVERSITÉ DU LUXEMBOURG

PhD-FSTC-2011-13
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 27/09/2011 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Ton Frederik Petrus VAN DEURSEN

Born on 8 June 1984 in Eindhoven (The Netherlands)

Security of RFID protocols

Dissertation defense committee

Dr. Gildas Avoine
Professor, Université Catholique de Louvain

Dr. Sjouke Mauw, dissertation supervisor
Professor, Université du Luxembourg

Dr. Saša Radomirović, vice-chairman
Université du Luxembourg

Dr. Mark D. Ryan
Professor, University of Birmingham

Dr. Peter Y.A. Ryan, chairman
Professor, Université du Luxembourg

Summary

Radio-frequency identification (RFID) is a technology that uses radio waves to exchange data between RFID readers and tags. The low manufacturing costs and small size as well as the lack of need of a power source make RFID tags useful in many applications, but also impose a strong need for secure RFID protocols.

We apply formal verification to analyze RFID protocols with respect to three security requirements: untraceability, authentication, and ownership transfer. Untraceability and authentication are the two most basic security requirements for RFID protocols. Informally, *untraceability* ensures that an attacker cannot recognize RFID tags with which he has communicated in the past. *Authentication* ensures that an attacker cannot impersonate a legitimate party. Finally, in dynamic environments where RFID tags are exchanged, sold, or traded, the owner of a tag may change. Secure *ownership transfer* ensures that ownership of the tag can be securely handed over to the new owner.

We then focus on the analysis of RFID systems of which the cryptographic keys can be recovered. Secrecy of cryptographic keys is often necessary for achieving authentication. If these cryptographic keys are known to the adversary, he may have access to data stored on RFID tags in the system. We develop a methodlogy to recover the memory structure from a set of memory dumps. This enables the formal analysis of the memory structure of RFID tags.

The first part of this thesis considers the analysis of untraceability of RFID protocols. We start by designing a formal syntax and semantics for security protocols. Within our formalization, one can precisely describe RFID protocols and systematically explore their execution traces. We define untraceability as a property on the traces of a protocol. Our definition requires that each trace in which the adversary could possibly deduce that a tag occurs twice, can also be understood as a trace in which two different tags occur.

The academic literature provides an abundance of RFID protocols designed to satisfy untraceability. We find untraceability flaws in a number of published RFID protocols. We introduce a categorization of these flaws comprising attribute acquisition attacks, man-in-the-middle attacks, insider attacks, compositionality attacks, and pseudonym-based attacks.

We then turn our attention to computational proof models for untraceability. We investigate the relation between two categories of computational proof models. The first category of computational proof models describes untraceability as the inability of the adversary to *distinguish* different tags. The second category defines untraceability as the attacker's inability to *predict* the messages sent by the tag. We show that the two classes of definitions are incomparable. Furthermore, we

show that a category of public-key based RFID protocols is susceptible to insider attacks. As a result, we extend an existing computational proof model with insider attackers. Finally, we propose a provably untraceable and authenticating RFID protocol based solely on elliptic-curve operations. As a basis of the protocol, we use an RFID protocol proposed by Vaudenay and the Cramer-Shoup encryption scheme.

The second part of this thesis is concerned with authentication of RFID protocols. We categorize authentication attacks into algebraic replay attacks, man-in-the-middle attacks, compositionality attacks, and cryptanalytic attacks. For each of these categories we give specific attacks on protocols from the academic literature.

The third part of this thesis deals with formalizing ownership in RFID systems as well as related security properties. Within our security protocol formalization, we define secure ownership, exclusive ownership, and secure ownership transfer. We use the concept of ownership to define desynchronization resistance. For all of these definitions, we show how they can be applied by exhibiting an attack on a published protocol.

The fourth part of this thesis describes the generic problem of recovering memory structures of systems. RFID tags often have a small portion of memory that can be used by the RFID application. One could repeatedly dump this memory and annotate each dump with the data suspected to be encoded on the tag. We define the carving problem as recovering the structure of the memory, based on this attributed dump set. We design and implement algorithms to find commonalities and dissimilarities and apply them to an RFID system deployed in Luxembourg.

Acknowledgments

A thesis does not just appear out of nothing. It is the result of several years of hard work. In the past four years, I have received support from many colleagues, family, and friends. They have helped me in one way or another for which I wish to thank them.

I owe my deepest gratitude to my daily advisor Saša Radomirović. The huge amount of time we spent together was always fun and productive. Saša has a passion for analyzing anything that can be analyzed. Aside from being an excellent researcher, he is also a great teacher. He taught me how to do research, how to write, how to give a talk, and how to teach. I could not have hoped for a better daily advisor than Saša, nor for the amount of time we worked together.

This thesis would not have been possible without Sjouke Mauw. I am amazed by how Sjouke leads his research group, by his devotion for formalizing anything that can be formalized, and by his ability to find time for everything. I am grateful for the time we worked together and for his scientific and moral support.

I thank the members of my defense committee, Gildas Avoine, Mark Ryan, and Peter Ryan for their time and valuable feedback. It was an honor to have had them in my defense committee. I would like to thank Alex Biryukov for taking place in my comité d'encadrement de thèse.

Sjouke's research group grew from five people when I started to thirteen people right now. I enjoyed our research seminars, lunches, movie nights, dinners, and coffee breaks. It is a great pleasure to thank my colleagues Baptiste Alcalde, Naipeng Dong, Wojtek Jamroga, Hugo Jonker, Barbara Kordy, Piotr Kordy, Simon Kramer, Matthijs Melissen, Tim Muller, Jun Pang, Georgios Pitsilis, Patrick Schweitzer, and Chenyi Zhang.

I am grateful to Jan Joris Vereijken for giving me the opportunity to stay at ING Direct Head Office. Furthermore, I would like to thank my colleagues Kino Verburg and Christopher Tong for the informative discussions we had and Martin Vonk for hosting me.

During my Ph.D., several master students finished a master's thesis under my co-supervision. They are Pim Vullers, Gerardo Iglesias, Xihui Chen, Davy Cox, and Cheng Xing. It was a pleasure to work with them and I am happy to see that they have found good jobs or Ph.D. positions.

I would like to thank my parents and brother for their support during critical periods, for keeping track of the Dutch news, and for being proud of me. Thanks to my friends for providing the necessary distractions and for their moral support.

Finally, I am forever indebted to my lovely wife Niki for her unconditional love,

time, support, and patience.

Ton van Deursen

Contents

1

Introduction

The last few decades have seen a vast number of electronic systems penetrate our daily lives. We use our cell phones to stay in contact with friends and family. Books can be bought from online book stores, paid through an online portal of the bank, and delivered the next day. Paper-based patient records have been replaced by electronic health records that provide medical staff with patient details whenever and wherever they are necessary. Even everyday items such as public transportation tickets have been replaced by electronic products. These technologies are meant to assist us in daily life, simplify tasks, and make our lives more enjoyable in general. At the same time, service providers embrace new technologies for their potential to increase efficiency, to reduce costs, and to bring new business opportunities.

Common to many recent technologies is that we depend on their security. We would not trust them if we knew that they were not secure, and thus we would not use them. Indeed, the market for cell phones that send text messages to all nearby burglars when you leave your house is arguably small. Any bank that allows criminals to drain bank accounts through its online portal would quickly lose its customers. People would be very skeptic toward electronic health records that can be accessed by their neighbors, their employer, insurance companies, financial institutes, and foreign governments. And finally, there is little use for public transportation tickets that everybody can forge without being noticed.

Unfortunately, we often hear about systems whose security is broken. In a typical week in spring 2011 the media reported on the following security incidents:

- Account details of 70 million online gamers were stolen. The details included e-mail addresses, date-of-births, home addresses, and credit card numbers.
- A centralized database for fingerprints of all Dutch citizens was canceled. The main reason was that the fingerprinting system incorrectly matched one in five fingerprints against the database.
- Three major mobile operating systems keep track of the physical location of the devices running their operating system. Periodically, these devices send the location data to the vendor without notifying the owner of the device.

These examples indicate that building secure systems is a very complex task. Fortunately, there are techniques to decide whether a system is secure before it is deployed. Formal verification is one of these techniques.

1.1 Formal verification

The purpose of formal verification is to prove or disprove a property of a system. The outcome of the analysis can be a confirmation of the security of a system by means of a proof. Alternatively, the insecurity of a system can be shown by revealing a flaw in the system. The first step in formal verification is to precisely specify the system to be analyzed, the environment in which the system is run, and the properties the system needs to satisfy. In particular, the following questions need to be answered unambiguously:

- *What does our system do?* In order to decide whether a system is secure, we first create an abstract model of our system. This model specifies the system and allows one to analyze properties of the system.
- *What do we mean by secure?* If we do not have a clear security requirement in mind, there will be no way of verifying whether a system satisfies it. We thus need to specify precisely what security guarantees we require of a system. A system relying on a fingerprinting scheme that incorrectly matches 20% of the fingerprints cannot be considered secure for most purposes. But, is the system secure if the number of incorrect matches is reduced to 1%, or do we require no false match at all? To consider another example, is leaking cell phone location data to a vendor a security breach? Many cell phone users would argue that way, but the vendor might disagree.
- *Who are we defending against?* It is much harder to defend a system against intelligence agencies than against a curious neighbor. Therefore, we need to identify the powers of the attacker. For instance, do we protect our system against outside attackers or also against legitimate users of the system? Do we protect our location data from phone manufacturers, or from everybody except our spouses?
- *What assumptions do we make about the environment?* To limit the scope of the design and analysis of a security system, we rely on properties of the building blocks of that system. These properties are called assumptions and they express the relevant characteristics of the building blocks. A common assumption is that nobody can decrypt a ciphertext without having the corresponding decryption key. Another assumption is that an honest user does not execute a malicious program.

After formalizing the system, the security requirement, the attacker's powers, and the assumptions, we can answer the most important question: *Does the model of the system satisfy the security requirement?* We can answer this question by giving a formal derivation or a mathematical proof of its (in)validity. If we can prove that the security requirement holds, then the model is secure under the stated assumptions. If we can refute the security requirement, we find an explicit weakness in the model. We can then fix the flaw and repeat the analysis.

A proof of security of the model of a system suggests that the system itself is secure. After all, our main goal is to prove security properties of systems. However, even if a model satisfies a security requirement, the actual system may be insecure.

To be able to apply formal verification, a model of a system abstracts away from (potentially relevant) properties of the system. Furthermore, a proof of security of the model relies on assumptions made by the verifier. One must, therefore, be convinced that the abstraction faithfully models the behavior of a system and that the assumptions made about the environment are realistic.

In this work, we use formal verification as a tool to analyze the security of radio-frequency identification (RFID) systems.

1.2 RFID systems

RFID is a technology that uses radio waves to exchange data between devices. The three main components of an RFID system are readers, tags, and a back-end. A tag is a small device attached to the object it is meant to identify. It consists of an integrated circuit with memory and processing capabilities and an antenna to receive and send signals. A tag is called active if it has its own power source and passive if it obtains power from the reader. Throughout this work, we concern ourselves with passive tags unless mentioned otherwise. A reader is a device that can detect the presence of RFID tags and communicate with them through radio waves. It has its own source of power and can communicate with the back-end. The back-end is a system that stores and processes information of tags and readers. The communication between back-end, RFID readers, and RFID tags is defined by RFID protocols.

Due to the miniaturization of electronic circuits, RFID tags can be incorporated in almost any other item. The smallest RFID tags are, in fact, smaller than sand particles. With prices starting at a few cents, RFID tags can be manufactured for a relatively low price. Due to their unique properties and the ability to communicate wirelessly without a clear line-of-sight, RFID systems have the potential of becoming ubiquitous.

The initial objective of RFID was identification of objects. Early applications of RFID systems include tracking systems for farm animals, library items, and airport baggage. Over the years, RFID tags have become more powerful and are used in applications that require more than mere identification. At present, RFID systems are found, for instance, in systems for public transportation ticketing, electronic toll collection, and building access control. Most countries issue passports with embedded RFID tags. RFID tags have even been implanted in humans.

1.3 Research question

Service providers that deploy RFID systems as well as the users of the system need to trust that the RFID system is secure. For instance, the service providers rely on the system to correctly identify a tag and on the inability of crooks to fabricate tags that impersonate real tags. Owners of RFID-tagged items expect that the tags they carry cannot be used to secretly track them. This is particularly relevant if users are not aware that RFID tags are embedded in their items or if they are forced to carry RFID-tagged items. For instance, in some countries citizens are

required by law to carry an identity document (with embedded RFID tag). If we realize that one can communicate with these tags from a distance without the owner noticing, we see that secretive tracking poses a realistic privacy threat.

To maximize their profits, RFID manufacturers wish to manufacture RFID tags as cheaply as possible. For this reason, as well as due to the absence of a power source on RFID tags, only very limited computational resources are available on passive RFID tags. This complicates the design of secure RFID systems.

Their low manufacturing costs and small size as well as the lack of need for a power source make RFID tags useful in many applications, but also impose a strong need for secure RFID systems. This leads to the main research question of this thesis.

Research question. *How can we apply formal verification to analyze the security of RFID systems?*

To answer this question we analyze the security of RFID protocols. RFID protocols define the communication between RFID readers, RFID tags, and the back-end. These protocols also provide a way for attackers to interact with the different components of an RFID system. Therefore, RFID protocols must be able to thwart attacks that violate the security of an RFID system.

We analyze RFID protocols with respect to three security requirements: untraceability , authentication, and ownership transfer. Untraceability and authentication are the two most basic security requirements for RFID protocols. Informally, *untraceability* ensures that an attacker cannot recognize RFID tags he has observed or communicated with in the past. *Authentication* ensures that agents can not be impersonated. Finally, in dynamic environments where RFID tags are exchanged, sold, or traded, the owner of a tag may change. Secure *ownership transfer* ensures that ownership of the tag can be securely handed over to the new owner. This requirement is only relevant in RFID systems in which tag ownership can change.

In particular, for each of the security requirements we wish to answer the following questions:

(a) *How do we formalize the security requirement?* A formal model is the first step towards understanding whether protocols satisfy a certain requirement. The model specifies protocols, protocol execution, and the adversary's capabilities. The intuitive understanding of the security requirement can then be transformed into a formal definition within this model. After finishing these steps, we can use the model to decide whether a protocol satisfies the security requirement.

(b) *How do protocols fail to achieve the security requirement?* In the case of RFID protocols, there is a wide variety of protocol proposals in literature. Many of these protocols do not satisfy the desired security requirement. Analyzing how protocols fail to satisfy a security requirement gives insight in how to design secure protocols. Classifying the flaws into a number of categories results in a useful library of common mistakes.

(c) *How do different formalizations compare to each other?* There are different ways of formalizing a security requirement, protocol execution, and the adversary model. Comparing these definitions reveals how they relate to one

another. This shows, for instance, whether one definition of the security requirement is strictly stronger than another, or whether they are incomparable. It may also identify inaccurate definitions that should not be used for formal verification.

(d) *How do we design secure protocols?* The ultimate goal of understanding a security requirement is to be able to design protocols that satisfy the requirement. Therefore, we gain insight in how a security requirement can be reached by designing protocols and proving that they satisfy the security requirement.

We then turn our attention to the analysis of RFID systems of which the cryptographic keys can be recovered. Secrecy of cryptographic keys is often necessary for achieving authentication. If these cryptographic keys are known to the adversary, he may have access to data stored on RFID tags in the system. Our final step to answer the research question is to develop a structured way of recovering the memory structure from a set of memory dumps with a similar structure. This enables the formal analysis of the memory structure of RFID tags.

1.4 Thesis overview

This thesis is divided into four parts. Each of the three security requirements mentioned above covers one part: Part I covers untraceability, Part II covers authentication, and Part III covers ownership transfer. Part IV covers the generic problem of recovering memory structures of systems.

Each of the chapters in the first three parts answers one of the above questions about one of the three security requirements. We answer the questions for untraceability. For authentication, there exists a significant body of work in traditional protocol verification to answer the first and third question. For the other two questions, we focus on the challenges that are specific to RFID protocols. Finally, for ownership transfer we only answer the first question. The other three remain for future work. Table 1.1 summarizes the relation between security requirements, the above questions, and the chapters.

Table 1.1: Chapter contents (○: future work, ×: extant work)

	(a)	(b)	(c)	(d)
Untraceability	3	4	5	5.5
Authentication	×	6	×	5.5
Ownership transfer	7	○	○	○

We now summarize the specific contents of the chapters. The contributions are based on the author's research, which was actively supported by the involvement of the daily advisor Saša Radomirović and supervisor Sjouke Mauw.

- **Chapter 3: Formalization of RFID protocols and untraceability**

  We design a formal syntax and semantics for security protocols. Within our formalization, one can precisely describe RFID protocols and systematically

explore their execution traces. The novelty of the approach lies in the ability to model and analyze protocols that maintain a persistent state across different executions. Furthermore, the model captures RFID-specific properties such as the inability of tags to run multiple concurrent runs.

We define untraceability as a property on the traces of a protocol. Our definition requires that each trace in which the adversary could possibly deduce that a tag occurs twice, can also be understood as a trace in which two different tags occur.

This chapter is based on work with Sjouke Mauw and Saša Radomirović. The formal definition of untraceability has been published in [Wistp08] and the syntax and semantics have been published in [Esorics].

- **Chapter 4: Untraceability attacks**

  The academic literature provides an abundance of RFID protocols designed to satisfy untraceability. We find untraceability flaws in a number of published RFID protocols. We categorize these flaws into attribute acquisition attacks, man-in-the-middle attacks, insider attacks, compositionality attacks, and pseudonym-based attacks.

  This chapter is based on a series of papers [AIR, Wistp09, STM, RFIDSec] and a technical report [ePrint] with Saša Radomirović and on [Prime]. Attribute acquisition attacks were first described in [Wistp09], man-in-the-middle attacks in [ePrint, RFIDSec], insider attacks and compositionality attacks in [RFIDSec], and pseudonym-based attacks in [AIR, STM].

- **Chapter 5: Untraceability proof models**

  There exist a number of different computational proof models for untraceability. The first category describes untraceability as the inability of the adversary to *distinguish* different tags. The second category defines untraceability as the attacker's inability to *predict* the messages sent by the tag. We show that the two classes of definitions are incomparable.

  We show that a category of public-key based RFID protocols is susceptible to insider attacks. This result has impact on prior results about untraceability in the case where the fact whether a protocol run ended successfully is public information. We extend an existing computational proof model with insider attackers.

  We propose a provably untraceable and authenticating RFID protocol based solely on elliptic-curve operations. As a basis of the protocol, we use an RFID protocol proposed by Vaudenay and the Cramer-Shoup encryption scheme.

  This chapter is based on an article [IPL] and a paper [EuroPKI] with Saša Radomirović.

- **Chapter 6: Authentication attacks**

  We categorize authentication attacks into algebraic replay attacks, man-in-the-middle attacks, compositionality attacks and cryptanalytic attacks. For each of these categories we give specific attacks.

This chapter is based on a series of papers with Saša Radomirović [Wistp08, Wistp09, ePrint, RFIDSec] and on a case study with Xihui Chen and Jun Pang [ICFEM]. Algebraic replay attacks were first described in [Wistp09], compositionality attacks in [RFIDSec], and leakage attacks in [Wistp08, Wistp09].

- **Chapter 7: Formalization of ownership transfer**

  We extend the model of Chapter 3 to analyze ownership-related properties. To this end, we devise definitions for secure ownership and secure ownership transfer. We also use the concept of ownership to define desynchronization resistance.

  This chapter is based on a paper with Sjouke Mauw, Saša Radomirović, and Pim Vullers [Esorics].

- **Chapter 8: mCarve: Carving attributed dump sets**

  RFID tags often have a small portion of memory that can be used by the RFID application. One could repeatedly dump this memory and annotate each dump with the data suspected to be encoded on the tag. We define the carving problem as recovering the structure of the memory, based on this attributed dump set. We design algorithms to find commonalities and dissimilarities and apply them to an RFID system deployed in Luxembourg.

  This chapter is based on a paper with Sjouke Mauw and Saša Radomirović [Usenix]. Preliminary analysis results of the case study have been described in [Prime].

Part I

RFID protocols and untraceability

2

Preliminaries

This chapter presents the preliminary knowledge necessary to understand the thesis. In particular, we discuss security notions used in provable security and the basics of elliptic-curve cryptography.

2.1 Provable security

2.1.1 Pseudo-random function

A function $f(x)$ is said to be negligible if for all positive integers c, there exists an x_0 such that for all $x > x_0$ we have $|f(x)| < \frac{1}{x^c}$. Let n be a security parameter and let F_K for $K \in \{0,1\}^n$ be a family of functions from $\{0,1\}^{l_1(n)}$ to $\{0,1\}^{l_2(n)}$. The function family F_K is called a *pseudo-random function family* if

1. $F_K(x)$ is computable in polynomial time; and
2. if all polynomial-time algorithms have a negligible advantage in n of distinguishing an oracle simulating F_k (for a random $k \in \{0,1\}^n$) from a uniformly random function.

2.1.2 Message authentication codes

A *message authentication code* (MAC) is a keyed cryptographic hash function. Given a key k and a message m, the message authentication code $MAC_k(m)$ must be easy to compute. A MAC is secure against *existential forgery* if, given a message m, the adversary cannot create $MAC_k(m)$ without knowledge of k. A MAC is secure against *selective forgery* if, for a message m of his choice, the adversary cannot determine $MAC_k(m)$ without knowledge of k.

2.1.3 Indistinguishability and non-malleability

The standard security notion for a public-key cryptosystem is indistinguishability under adaptive chosen ciphertext attacks (IND-CCA2). Its definition is based on the following game [BDPR98]. The adversary starts by making arbitrary queries to a "decryption oracle", decrypting ciphertexts of his choice. Then the adversary chooses two messages m_0 and m_1 of the same length and sends them to an "encryption oracle". The encryption oracle chooses a random bit $b \in \{0,1\}$ and returns the encryption of m_b as a challenge to the adversary. The adversary is then given access to the decryption oracle, decrypting all ciphertexts of his choice except the

challenge. At the end of the game, the adversary guesses a bit $b' \in \{0, 1\}$. The adversary wins the game if $b = b'$.

A related security notion is that of non-malleability under adaptive chosen ciphertext attacks (NM-CCA2). Intuitively, a cryptosystem is non-malleable if the attacker cannot change a ciphertext in another ciphertext such that the underlying plaintexts are meaningfully related. In particular, it prevents an adversary from creating a second ciphertext with the *same* underlying plaintext. Non-malleability is formalized using the following game [BDPR98]. The adversary starts by making queries to a "decryption oracle". It then chooses a set of messages M. The encryption oracle chooses one of the messages at random, say x, and returns the encryption y of that message. The adversary can then query the decryption oracle on all ciphertexts of his choice except y. At the end of the game, the adversary outputs a vector of ciphertexts $\mathbf{y}$ and a relation R. Let $\mathbf{x}$ be the vector of plaintexts corresponding to the ciphertexts in $\mathbf{y}$. The adversary wins if the relation $R(x, \mathbf{x})$ holds with a probability significantly higher than with which $R(x', \mathbf{x})$ for some random $x' \in M$ holds.

We say that a cryptographic scheme is secure against adaptive chosen ciphertext attacks (CCA2) if the advantage of any polynomial-time adversary in winning the game is negligible.

If we modify the game so that the attacker does not have access to the decryption oracle after the challenge is issued, then we call a scheme secure against chosen ciphertext attacks (CCA1) or lunchtime attacks if the advantage of any polynomial-time adversary winning the game is negligible.

Bellare, Desai, Pointcheval, and Rogaway [BDPR98] showed that IND-CCA2 and NM-CCA2 are equivalent properties.

2.1.4 Indistinguishability in the multi-user setting

Bellare, Boldyreva, and Micali [BBM00] proposed a notion of IND-CCA2 in the multi-user setting. The idea behind the multi-user notion is that an adversary may be able to obtain encryptions of related (possibly the same) message(s) under different public keys. Also, the adversary may be able to obtain encryptions of related messages under the same public key.

The main concept in the IND-CCA-MU game is that of a left-right oracle LR. The left-right oracle $LR(m_0, m_1, b)$ returns the encryption of plaintext m_0 if the bit $b = 0$ and the encryption of m_1 if $b = 1$. At the start of the experiment a bit b is chosen at random. The value of b is only chosen once and is the same across all queries. The adversary has access to an encryption oracle that takes as input two message m_0 and m_1, the bit b, and a public key pk. It returns $\{m_b\}_{pk}$: the encryption of m_b under public key pk. The adversary may also query a decryption oracle on any ciphertext that was not the output of the encryption oracle. At the end of the experiment the adversary guesses the value of bit b. A scheme is IND-CCA-MU secure if the adversary has no non-negligible advantage in correctly guessing the bit b.

Bellare, Boldyreva, and Micali [BBM00, Theorem 1] have proven that any adversary that has negligible advantage in winning the IND-CCA2 experiment, also has

negligible advantage in winning the IND-CCA-MU experiment.

2.2 Elliptic-curve cryptography

Elliptic curve cryptography has recently become popular as a tool to build public key cryptosystems. Traditionally, RSA-based systems were used to implement public key cryptosystems. Both approaches are based on intractable mathematical problems and, therefore, provide provable security. However, for elliptic curve cryptography, smaller parameters (i.e. smaller key sizes) are possible with the same security as RSA-based systems. They are, therefore, within reach to be implemented on passive RFID tags.

Elliptic curves can be defined over various algebraic structures, such as the rationals or the reals. To be used for cryptography, elliptic curves are defined over finite fields $\mathbb{F}_p$ of large prime order p or over finite fields of the form $\mathbb{F}_{2^n}$. The advantage of using the latter type is that addition in the field is defined by bitwise exclusive or. Multiplication is defined by considering the bitstrings as the coefficients of polynomials and performing the multiplication of these polynomials modulo an irreducible polynomial of degree n.

We call (x, y) for $x, y \in \mathbb{F}_{2^n}$ a point. We consider all points that satisfy the equation

$$y^2 + xy + x^3 + ax^2 + b = 0, \tag{2.1}$$

for some $a, b \in \mathbb{F}_{2^n}$ and refer to them as points on the elliptic curve. The number of points N satisfying Equation (2.1) is approximately 2^n. A more precise estimate is given by Hasse's theorem:

$$|N - (2^n + 1)| \leq 2\sqrt{2^n}.$$

By defining a *point at infinity* $\mathcal{O}$ as a neutral element, a point addition operation can be defined. The group of points on the curve, together with $\mathcal{O}$ and the addition operation form an abelian group.

The coefficients a and b in Equation (2.1) ought to be chosen such that the elliptic curve group has a large cyclic subgroup of prime order p. An element P of order p can then be chosen as generator of that subgroup. One of the basic operations in elliptic curve cryptography is scalar multiplication. That is, given an integer n, compute $n \cdot P$. The opposite problem, i.e. given P and Q, find an integer n such that $nP = Q$ is known as the *discrete logarithm problem*. There exist groups for which the most efficient known algorithms for solving the discrete logarithm problem are exponentially slower than algorithms for scalar multiplication.

There are several problems that are derived from the discrete logarithm (DL) problem. In the following, let E be an elliptic curve and let P be a generator of a large subgroup of the points on E.

- The computational Diffie-Hellman (CDH) problem: Given random points aP, bP, compute abP.

- The decisional Diffie-Hellman (DDH) problem: Given random points aP, bP, and cP, decide whether $c = ab$ modulo the order of the group generated by P.

It follows that being able to solve DL means being able to solve CDH and being able to solve CDH means being able to solve DDH.

One can test whether a point lies on the elliptic curve by plugging it into Equation (2.1). The trace function can be used to determine whether a given x-coordinate is on the curve. The trace is a mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$:

$$T(x) = \sum_{i=0}^{n-1} x^{2^i}.$$

The trace is a linear operator, i.e. $T(a+b) = T(a) + T(b)$ for all $a, b \in \mathbb{F}_{2^n}$. Also, $T(x^2) = T(x)$ for all $x \in \mathbb{F}_{2^n}$. By setting $z = \frac{y}{x}$ and dividing by x^2, Equation (2.1) can be simplified to

$$z^2 + z + x + a + \frac{b}{x^2} = 0. \tag{2.2}$$

Equation (2.2) only has a solution in z if $T(x + a + \frac{b}{x^2}) = 0$. If that is the case and z is a solution to (2.2), then $z + 1$ is also a solution. Since $z = \frac{y}{x}$, if y is a solution for a given x for Equation (2.1), then so is $y + x$.

To map a bitstring u of length $n - l$ into the elliptic curve a simple try-and-increment method [Ica09] can be used. Let x_2 denote the integer x represented as l-bit string. Using the previously defined trace function, the algorithm can be defined as follows:

1. For $i = 0$ to $2^l - 1$ do
   (a) Set $x = u \| i_2$
   (b) If $T(x + a + \frac{b}{x^2}) = 0$ then solve Equation (2.1) for y and return (x, y)
2. return "fail"

3

Formalization of RFID protocols and untraceability

History has shown that designing protocols is a difficult and error-prone task and that formal verification of security properties is necessary [CJ97, Low96]. While traditional security properties such as authentication and secrecy have been studied thoroughly, untraceability has only become relevant with the introduction of traveling devices. It has typically been treated rather informally. In some cases, protocol designers argue untraceability of their protocols without providing a proper definition of untraceability.

In this chapter, we develop a formal model for security protocol analysis. The model allows protocol designers and verifiers to describe a protocol in an unambiguous way. Subsequently, the behavior of the protocol (i.e. all ways in which the protocol can be executed) can be systematically derived. We define untraceability as a property on all possible behaviors of the protocol. We define security requirements related to ownership transfer in Chapter 7.

Intuitively, a protocol is untraceable if an attacker cannot recognize a tag he has seen before. In other words, a protocol is untraceable if the adversary cannot observe any difference between protocol executions of the same tag and protocol executions of different tags. We formalize untraceability as a requirement on all behaviors of the protocol. We then apply our formal model and untraceability definition to show untraceability of one protocol and traceability of another.

3.1 Basic concepts

We start by explaining the main concepts used in our formal model after which we formalize each of these concepts.

A *protocol* consists of a number of *roles*. Each of these roles describes the steps that an *agent* is expected to carry out. There are different sorts of agents that can be involved in protocols, such as computers, humans, or RFID tags. One execution of a protocol role by an agent is called a *run*.

The role specifies the *messages* that need to be sent and received. These messages are, for instance, encryptions or cryptographic hashes of simpler terms. Among the *basic terms*, we find agent names, system-wide constants (such as the natural numbers), and cryptographic keys. An important type of basic term is the *nonce*, short for 'number-used-once'. Nonces are fresh, unpredictable terms that can be used to ensure that the messages in which they are used are not predictable.

Protocols embody several constructs to define the control flow in an execution. The *events* describe the actions performed by an agent executing a role. An agent can execute *read* and *send* events, in order to read messages from or send messages to the network. There are several ways in which these events can be composed. For instance, one can specify that events have to be executed in sequence, or that one of two events must be executed. The agents in the system have two kinds of memory. The *temporary* memory contains variables whose values are only accessible to a single run of that agent. The *persistent* memory contains variables whose values are shared across all runs of that agent. We call a protocol *stateless* if it does not update any persistent variable during protocol execution, or *stateful* if it does. We assume that RFID tags can run only one protocol execution at a time, but RFID readers can run different executions concurrently.

We consider an *asynchronous communication* model where messages are not instantly received after they are sent. Following Dolev and Yao [DY83], we assume that the *adversary* (sometimes called intruder) controls the messages that are being exchanged. This means that he can modify messages, block messages, eavesdrop on messages, and inject messages. When describing an explicit attack on a protocol, we refer to one or more *attackers* that carry out the attack. The adversary is thus an idealization of the capabilities that a real-world attacker (or set of attackers) might have. Any agent that is not malicious is called *honest* and it runs the protocol exactly as specified. If an agent does not receive the message he expects to receive according to his role specification, he simply does not continue the execution.

A *security requirement* formalizes a security or privacy goal of the protocol. It describes a property that the protocol must enforce when executed by honest agents. An example of a security requirement is secrecy, stating that the adversary cannot deduce a certain message. Another security requirement is untraceability, requiring that an adversary cannot recognize an agent he has previously observed. We call a protocol *secure* with respect to a security requirement if the adversary has no means to invalidate it. If the adversary can invalidate a claim, we call the protocol *flawed* or vulnerable.

Central to our model is the concept of *adversary knowledge*. It contains the messages that the adversary has received in the past and all the public knowledge. By combining messages, the adversary can derive new terms. For instance, if the adversary knows a ciphertext and the corresponding decryption key, he can derive the plaintext. We adopt the *perfect cryptography* assumption, stating that a ciphertext leaks no information about the plaintext if the adversary does not have the decryption key. Furthermore, cryptographic hash functions are assumed to be perfect. That is, the cryptographic hash of a message does not leak any information about the message.

3.2 Syntax: protocol specification

3.2.1 An RFID protocol

In this chapter, we use the protocol by Ha, Moon, Nieto, and Boyd [HMNB07] as a running example. We call the protocol *HMNB* after the last names of the authors. The HMNB protocol is an RFID protocol that aims to mutually authenticate RFID tag and reader, keep the tag untraceable, and resist a particular form of denial-of-service attacks, known as desynchronization attacks. We give formal definitions of untraceability in Section 3.5, of authentication in Chapter 6, and of desynchronization resistance in Chapter 7. Furthermore, the HMNB protocol has been designed with limited computational requirements on tags in mind employing a hash function as the only cryptographic primitive.

The protocol assumes that all tags T have an identifier ID. This identifier is only known to the reader R and tag T. It is updated at the end of a successful protocol execution. Thus the protocol is stateful. The reader also stores the hash of the ID in HID and the value of ID before the last update in ID'. Therefore, for a system with n tags, the reader stores n tuples (ID, ID', HID). The tag keeps track of whether its last run ended successfully or not. For this purpose, the tags use a variable S. If the last run ended successfully, the value of S is 0, otherwise 1.

We assume that before the reader starts its protocol execution, it does not know the identity of the tag it is about to communicate with. By matching the first message received from a tag against the list of tuples, the reader can identify the tag. If this procedure is successful, the reader continues the protocol execution and we say that the reader "accepts" the tag. Otherwise, the reader halts the protocol execution and we say that the reader "rejects" the tag.

Table 3.1: Reader's verification and update procedure in the HMNB protocol

Tag response	Update
$h(ID), nt$	$ID' := ID$; $ID := h(ID, nr)$; $HID := h(ID)$
$h(ID, nt, nr), nt$	$ID' := ID$; $ID := h(ID, nr)$; $HID := h(ID)$
$h(ID', nt, nr), nt$	$ID := h(ID', nr)$; $HID := h(ID)$
other	reject tag

We represent protocols graphically using message sequence charts. Every message sequence chart shows the role names, framed, near the top of the chart. Above the role names, the role's secret terms are shown. Actions, such as nonce generation, computation, verification of terms, and assignments are shown in boxes. Messages to be sent and expected to be received are specified above arrows connecting the roles. Other conditions that need to be satisfied are shown in diamond boxes.

The HMNB protocol is depicted in Figure 3.1. The protocol starts with the reader challenging the tag with a nonce nr. The tag then generates a nonce nt. The response of the tag depends on the value of S. In case the previous run ended successfully, the tag responds with $(h(ID), nt)$. In case it did not end successfully, the value of S is 1 and the tag responds with $(h(ID, nt, nr), nt)$. In either case,

the tag sets S to 1.

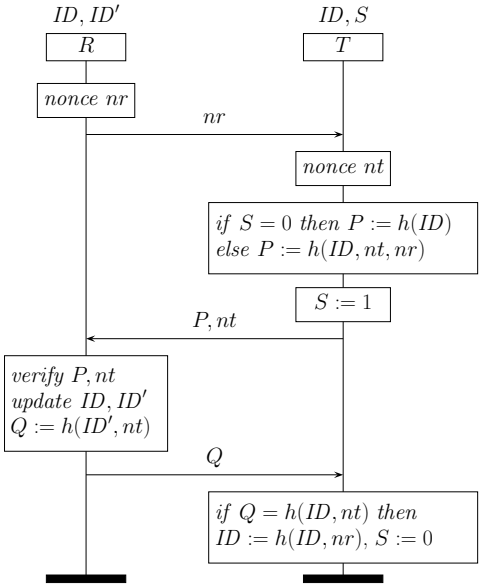


Figure 3.1: The HMNB protocol

The reader first tries to identify the tag by searching whether the response, aside from the nonce nt, matches any of the stored values of HID. This search can be implemented in constant time if the reader stores the tuples in a hash table. If this match is not successful, the reader iterates through its tuples for a value ID or ID' for which $h(ID, nt, nr)$ or $h(ID', nt, nr)$ equals the first part of the response. This search can be implemented in linear time in the number of tags n. If any of the searches is successful, the reader accepts the tag and continues the protocol execution. If this is not the case, the reader rejects the tag and halts. If the reader accepts the tag, it updates the information for the tag according to Table 3.1 and sends $h(ID', nt)$ to the tag. Finally, if the message received by the tag matches $h(ID, nt)$, it replaces ID by $h(ID, nr)$ and sets S to 0.

Since the value HID is only meant to improve the efficiency and does not have a security purpose, we leave it out of the protocol specification.

3.2.2 Terms

Messages to be sent over the network are constructed by a term algebra. The smallest elements of this term algebra are called *basic terms*. Basic terms can be composed to construct *complex terms*. The basic terms are separated into six sets:

- *Agent* contains the names of the agents that are allowed to execute protocol roles.
- *Const* contains all constants such as natural numbers, bitstrings, and strings.
- *Nonce* contains nonces; values that are freshly generated for each protocol execution.
- *Func* contains all function symbols.

- *Var* contains all variable names.
- *RoleName* contains all role names.

During protocol execution, values can be assigned to variables. Hence, variables represent values that may be different across different protocol executions. There are two different disjoint sets of variables:

- Var_T: The set of temporary variables. These variables are local to one particular run of the protocol. Initially, no value is assigned to temporary variables. A value can be assigned to a temporary variable only once during a protocol execution. After assignment the value can no longer be changed. At the end of the protocol execution, the value is lost.
- Var_P: The set of persistent variables. These variables are shared across all protocol runs of the same agent.

There is one special variable $\theta \in Var_T$ that is used to denote the identifier of a run. This variable is used to disambiguate nonces from different runs. A fresh value is assigned to θ when a role is instantiated. Table 3.2 contains representative examples for each of the above set of constants and variables.

Table 3.2: Symbol sets.

Symbol set	Description	Representative examples
$Agent$	Agent names	a, b, r, t
$Nonce$	Nonces	na, nb, nr, nt
$Const$	Constants	$0, 1, 2, 3$
$Func$	Functions	h
$RoleName$	Role names	R, T
Var_P	Persistent variables	ID, k
Var_T	Temporary variables	$NR, NT, V, W, x, y, \theta$

Complex terms can be constructed by pairing terms, denoted by $(\_, \_)$, encrypting a term by another term, denoted by $\{\_\}_\_$, or applying a function $f \in Func$ to a term, denoted by $f(\_)$. We assume that pairing is right-associative and leave out superfluous brackets to simplify terms. Unless stated otherwise, encryption is assumed to be deterministic. Probabilistic encryption can be modeled by deterministic encryption by including the randomness in the encryption. The set of all terms, *Term*, is defined as follows.

Definition 3.1 (Term)**.** *Terms are constructed using the following term algebra.*

$$Term ::= Agent \mid Const \mid Nonce(\theta) \mid RoleName \mid Var_P \mid Var_T \mid Func(Term) \mid (Term, Term) \mid \{Term\}_{Term}$$

Example 3.2. *As an example, we construct the tag response for the tag role of the HMNB protocol. Let $h \in Func$ be a function and let $ID \in Var_P$ be a persistent variable. The nonce nr is generated by the reader and is, therefore, a temporary variable for T: $NR \in Var_T$. Finally, let $nt \in Nonce$. Then the tag response is $(h(ID, nt(\theta), NR), nt(\theta))$.*

To recover the plaintext from an encryption, the corresponding decryption key must be known. The encryption key and decryption key are inverses of each other. We use the unary function symbol $\cdot^{-1}$ to denote the inverse of a term. We require that $\cdot^{-1}$ is its own inverse, i.e. for all terms t we have $(t^{-1})^{-1} = t$. For symmetric key encryption, we require that encryption and decryption key are the same, i.e. $k = k^{-1}$. Throughout this thesis, we associate $pk(a)$ with the public key of agent a and $sk(a)$ with the secret key. Therefore, for asymmetric encryption the public key is the inverse key of the secret key: $pk(a)^{-1} = sk(a)$. Note that, in general, asymmetric encryption schemes have the property that for any agent a, given $pk(a)$, it is infeasible to compute $sk(a)$.

3.2.3 Events

Agents can perform two types of events. A *send* event specifies that an agent sends a message to the network. A *read* event specifies that an agent receives a message from the network. The parameter of both send and read events is the message that the agent sends or reads.

There are two types of variable assignments in our protocol specification language. Temporary variables occur in the messages of an agent's read event. Assignments to temporary variables are carried out while an agent processes the message that he received and are called *implicit*. In contrast, assignments to persistent variables are *explicit* and can be combined with a read or send event of the agent. Assignments to persistent variables are denoted by $v := t$ for $v \in Var_P$ and $t \in Term$. We write $v_1 := t_1; \ldots; v_n := t_n$ for the sequential assignment of terms $t_1 \ldots t_n$ to $v_1 \ldots v_n$ for any number n. Formally, the set of assignments is defined by:

$$Assignment ::= Var_P := Term \mid Assignment;\ Assignment$$

Events can be composed in three ways. *Sequential composition*, denoted by $(\_ \cdot \_)$, specifies consecutive execution of events, while *alternative composition*, denoted by $(\_ + \_)$, models branching. *Conditional branching*, denoted by $(\_ \lhd \_ \rhd \_)$, chooses the left branch if the value in the middle is true and the right branch otherwise. We use parentheses around event compositions to allow for disambiguation of the composed event. In order to simplify role specifications, we assume that sequential composition $(\_ \cdot \_)$ binds stronger than alternative composition $(\_ + \_)$ and alternative composition binds stronger than conditional branching $(\_ \lhd \_ \rhd \_)$. We leave out superfluous parentheses whenever no confusion can arise. Formally, the set of events Ev is defined as follows.

$$\begin{aligned} Ev ::= {} & send(Term) \mid send(Term)[Assignment] \mid \\ & read(Term) \mid read(Term)[Assignment] \mid \\ & Ev + Ev \mid Ev \cdot Ev \mid Ev \lhd Term = Term \rhd Ev \end{aligned}$$

Example 3.3. *The tag role in the HMNB protocol consists of three events: a read event, a send event, and another read event. In the second read event, the tag expects to receive the message* $h(ID, nt(\theta))$*. Upon receipt of the message, the tag assigns the value* $h(ID, NR)$ *to the permanent variable* ID *and the value* 0 *to* S*. The*

specification of the second read event of the tag is thus $read(h(ID, nt(\theta)))[ID := h(ID, NR); S := 0]$.

$$
\begin{array}{ll}
ProtSpec & \rightarrow ProtName(RoleName) = RoleSpec \mid \\
 & \quad ProtSpec, ProtSpec \\
RoleSpec & \rightarrow (\mathcal{P}(Nonce), \mathcal{P}(Var_T), \mathcal{P}(Var_P), Ev \cdot \bot) \\
Ev & \rightarrow send(Term) \mid \\
 & \quad send(Term)[Assignment] \mid \\
 & \quad read(Term) \mid \\
 & \quad read(Term)[Assignment] \mid \\
 & \quad (Ev + Ev) \mid \\
 & \quad (Ev \cdot Ev) \\
 & \quad (Ev \triangleleft Term = Term \triangleright Ev) \mid \\
Assignment & \rightarrow Var_P := Term \mid \\
 & \quad Assignment; Assignment \\
Term & \rightarrow Agent \mid \\
 & \quad Const \mid \\
 & \quad Nonce(\theta) \mid \\
 & \quad Var_T \mid \\
 & \quad Var_P \mid \\
 & \quad Func(Term) \mid \\
 & \quad (Term, Term) \mid \\
 & \quad \{Term\}_{Term}
\end{array}
$$

Figure 3.2: Grammar for protocol specifications

3.2.4 Protocols

A *protocol specification* is a mapping of *role names* to *role specifications*. A role specification consists of a declaration of the nonces and variables used by that role and the events defining the messages that an honest agent sends and expects to read, when executing the role. For technical reasons, we append the symbol $\bot$ to the list of events to denote that a protocol execution has finished.

Let *ProtName* be a set of protocol names and *RoleName* be a set of role names. The syntax of our security protocol specification language is depicted in Figure 3.2.

Example 3.4. *The HMNB protocol consists of two roles: a reader role* R *and a tag role* T*. The tag role contains a nonce* $nt \in Nonce$ *and has a temporary variable* $NR \in Var_T$ *to store the nonce received from the reader. It has a persistent variable* $ID \in Var_P$ *for its current identifier and* $S \in Var_P$ *to store whether the previous run ended successfully. The full specification of the tag role is:*

$$
\begin{array}{l}
HMNB(T) = \\
\quad (\{nt\}, \{NR\}, \{ID, S\}, \\
\quad read(NR) \cdot \\
\quad (send(h(ID), nt(\theta))[S := 1] \triangleleft S = 0 \triangleright send(h(ID, nt(\theta), NR), nt(\theta))[S := 1]) \cdot \\
\quad read(h(ID, nt(\theta)))[ID := h(ID, NR); S := 0] \cdot \bot)
\end{array}
$$

The reader role R contains a nonce $nr \in Nonce$ and a temporary variable $NT \in Var_T$. There is a temporary variable $P \in Var_T$ to store the message that the reader receives from the tag. If the first condition, $P = h(ID)$, holds the reader responds with $h(ID, NT)$. If it does not hold, it continues with the second condition, etc. If none of the three conditions holds, the tag responds with the constant "reject". We thus have $reject \in Const$. The reader keeps track of the current and previous identifier of the tag in persistent variables $ID, ID' \in Var_P$. The specification is as follows.

$$
\begin{aligned}
&HMNB(R) = \\
&(\{nr\}, \{NT, P\}, \{ID, ID'\}, \\
&send(nr(\theta))\cdot \\
&read(P, NT)\cdot \\
&send(h(ID, NT))[ID' := ID; ID := h(ID, nr(\theta))] \triangleleft P = h(ID) \triangleright (\\
&\ send(h(ID, NT))[ID' := ID; ID := h(ID, nr(\theta))] \triangleleft P = h(ID, NT, nr(\theta)) \triangleright (\\
&\ \ send(h(ID', NT))[ID := h(ID', nr(\theta))] \triangleleft P = h(ID', NT, nr(\theta)) \triangleright (\\
&\ \ \ send(reject)))) \cdot \bot)
\end{aligned}
$$

3.3 System model

If an agent starts an execution of a protocol role, a *run* is created. A run is identified by a *run identifier* chosen from the set of natural numbers. A run contains the *agent name* of the agent executing the run. It also contains the list of *events* to be executed by the agent. During execution, events are removed from this list. Finally, a *temporary variable assignment* is maintained to keep track of the values that are assigned to the temporary variables. A run is thus defined by:

$$Run = \mathbb{N} \times Agent \times Ev \times (Var_T \to Term)$$

Aside from a temporary variable assignment, agents maintain persistent knowledge that is shared across all runs of the agent. We call this the *persistent variable assignment*, and define it as follows:

$$Pva = Agent \to (Var_P \to Term)$$

For our analysis we assume that the adversary has full control over the network. That is, he can see every message, he can modify messages, he can block messages, and he can inject messages. We keep track of the messages obtained by the adversary by adding them to the *adversary knowledge*. The adversary knowledge is a monotonically increasing set of terms observed by the adversary.

The system *state* is defined by the runs, the persistent variable assignment, and the adversary knowledge:

$$State = \mathcal{P}(Run) \times Pva \times \mathcal{P}(Term)$$

The agents and the adversary can combine terms from their knowledge and derive new terms. For instance, given a ciphertext and the decryption key, the plaintext

can be recovered. We call the process of deriving a term from a set of terms *knowledge inference*.

Definition 3.5 (Knowledge inference)**.** *We say that a set K infers a term t (denoted by $K \vdash t$) if t can be obtained from K by repeatedly applying the following rules:*

$$\begin{aligned} K \vdash t_1 \wedge K \vdash t_2 &\Rightarrow K \vdash (t_1, t_2) && \text{(pair)} \\ K \vdash (t_1, t_2) &\Rightarrow K \vdash t_1 && (\text{unpair}_1) \\ K \vdash (t_1, t_2) &\Rightarrow K \vdash t_2 && (\text{unpair}_2) \\ K \vdash t_1 \wedge K \vdash t_2 &\Rightarrow K \vdash \{t_1\}_{t_2} && \text{(encrypt)} \\ K \vdash \{t_1\}_{t_2} \wedge K \vdash t_2^{-1} &\Rightarrow K \vdash t_1 && \text{(decrypt)} \\ K \vdash t &\Rightarrow K \vdash h(t) && \text{(hash)} \end{aligned}$$

Example 3.6. *Let $K = \{\{a\}_k, k^{-1}\}$ be the adversary knowledge. Then the adversary can derive $h(a)$ by applying the* decrypt *and* hash *rules.*

A *substitution* σ is a mapping of variables to ground terms. We write $a \mapsto b \in \sigma$ if $\sigma(a) = b$. The domain of a substitution is defined by $dom(\sigma) = \{a \in Var \mid \exists_{b \in Term}\ \sigma(a) = b\}$ and the range is defined by $rng(\sigma) = \{b \in Term \mid \exists_{a \in Var}\ \sigma(a) = b\}$. We denote by $\sigma[b/a]$ the substitution σ modified such that $\sigma(a) = b$ and unmodified otherwise. The application of $\sigma(t)$ for any term t is defined by replacing all variables v in t for which $v \in dom(\sigma)$ by $\sigma(v)$. The composition $\sigma \circ \sigma'$ of two substitutions σ and σ' is defined by $\sigma \circ \sigma'(t) = \sigma(\sigma'(t))$ for all terms t. Substitutions extend to events in the obvious way.

3.4 Semantics: protocol execution

In this section we describe how, through instantiation of variables, an abstract role specification can be transformed into an execution by an agent, called a *run*. Furthermore, we define how the interleaved execution of a collection of runs defines the behavior of a system.

The behavior of the system is defined as a transition relation on system states. The derivation rules are of the form

$$\frac{C}{S \xrightarrow{l} S'},$$

expressing that a system in state S may execute l and continue to state S' if condition C is satisfied. A state transition from a state S to S' is the conclusion of applying these rules. In this way, starting from an initial state, we can derive all possible behavior of a system executing a set of protocols.

The semantics describe which conditions have to be satisfied for an agent to execute a single event. We separate the derivation rules into two categories. The *agent rules* express under which conditions an agent may execute a read or send event, as well as start or end a protocol run. Agent rules can be composed in several ways to model protocol flow, expressed by the *composition rules*. By adopting this two-layer approach we can keep the number of rules to a minimum, at the cost of having a slightly more complicated model. We first describe the composition rules and then the agent rules.

3.4.1 Composition rules

The composition rules model the conditions that are induced by the event *composition* of the protocol. In the following, let $e, e_1, e_2 \in Ev$ be events, a be an atomic (i.e. send or read) event, and $x, y \in Term$ be terms. We introduce the event $\checkmark$ to denote successful execution of an event. The rules in Figure 3.3 describe the semantics for composing events. The sequential composition rule (*seq*) specifies that an atomic event a followed by any event e can always be executed. As specified by the *exec* rule, an atomic event a can always be executed. If two events e_1 and e_2 are composed using alternative composition, $e_1 + e_2$, then either of the branches can be executed ($choice_1$ and $choice_2$). Finally, given two events e_1 and e_2 and two terms x and y, the conditional branching statement $e_1 \lhd x = y \rhd e_2$ can execute either of the branches. The left branch e_1 can be chosen if $x = y$ ($cond_1$) and the right branch e_2 can be chosen if $x \neq y$ ($cond_2$). We take $x = y$ to mean that x is syntactically equivalent to y and $x \neq y$ that x is not syntactically equivalent to y.

$$\text{exec}\ \frac{}{a \xrightarrow{a} \checkmark} \qquad \text{seq}\ \frac{a \xrightarrow{a} \checkmark}{a \cdot e \xrightarrow{a} e}$$

$$\text{choice}_1\ \frac{e_1 \xrightarrow{a} e_1'}{e_1 + e_2 \xrightarrow{a} e_1'} \qquad \text{choice}_2\ \frac{e_2 \xrightarrow{a} e_2'}{e_1 + e_2 \xrightarrow{a} e_2'}$$

$$\text{cond}_1\ \frac{e_1 \xrightarrow{a} e_1'}{e_1 \lhd x = x \rhd e_2 \xrightarrow{a} e_1'} \qquad \text{cond}_2\ \frac{e_2 \xrightarrow{a} e_2' \quad x \neq y}{e_1 \lhd x = y \rhd e_2 \xrightarrow{a} e_2'}$$

Figure 3.3: Composition rules

Example 3.7 (Composition rules)**.** *Let* $e = (send(1) \lhd 0 = 1 \rhd send(2)) \cdot \bot \in Ev$ *be an event. The following derivation shows that* e *can do a* $send(2)$ *event resulting in* $\bot$*. Note that the rule* $cond_2$ *needs to be applied since* 0 *is not syntactically equivalent to* 1*.*

$$seq\ \frac{cond_2\ \dfrac{exec\ \dfrac{}{send(2) \xrightarrow{send(2)} \checkmark} \quad 0 \neq 1}{send(1) \lhd 0 = 1 \rhd send(2) \xrightarrow{send(2)} \checkmark}}{(send(1) \lhd 0 = 1 \rhd send(2)) \cdot \bot \xrightarrow{send(2)} \bot}$$

3.4.2 Agent rules

The agent rules describe the effect of the execution of an agent on his *run* and on the *state*. We define rules for creating a run, terminating a run, sending a message, and reading message.

Creating a run. The *create* rule models the start of a protocol execution by an agent. The environment in which RFID protocols are run imposes two restrictions on run creation. First, there are two clearly separated sets of agents: tags and

readers. Unless stated otherwise, we assume that tag agents can only execute the tag role of an RFID protocol and reader agents can only execute the reader role. Second, an RFID tag can only run one simultaneous protocol execution. Before a tag can start a run, its previous execution must have finished or been terminated. Readers can run multiple concurrent protocol executions. In order to faithfully model RFID communication, the semantics of the create rule must allow both restrictions to be enforced. Otherwise, the semantics would allow one to derive attacks that cannot be executed in a deployed RFID system.

The create rule creates a run with a *fresh* run identifier f and adds it to the set of active runs. Let $S = \langle A, \sigma, I\rangle \in \mathit{State}$ be a system state with active runs A. Recall that an active run $a = (f, n, e, v) \in A$ consists of a run identifier f, an agent name n, a list of events e, and a temporary variable assignment v. We use the function $\mathit{runids}(A)$ to extract the run identifiers of the set of runs A.

$$\mathit{runids}(A) \equiv \{f \mid (f, n, e, v) \in A\}.$$

We introduce the function $\mathit{agents}(R)$ to denote the set of agents that can execute a role R. This allows modeling that certain agents can execute a role, while others can not. We assume two types of agents: agents that cannot run concurrent protocol executions ($\mathit{concur} = 0$) and agents that can ($\mathit{concur} = 1$). We further assume the existence of a function *concur* of type $\mathit{Agent} \mapsto \{0, 1\}$ that captures for each agent whether they can run concurrent protocol executions. In our setting, for RFID tag agents t we have $\mathit{concur}(t) = 0$ and for RFID readers r we have $\mathit{concur}(r) = 1$.

For a run to be created, the following two conditions have to be satisfied; (1) The agent must belong to the set of agents that can execute a role, and (2) the agent must be able to execute runs concurrently, or it must not have an unfinished run at the time of run creation. Unfinished runs have an event list not equal to $\bot$. We say that an agent n is *allowed* to execute role R in a state with runs A if the following condition is satisfied.

$$\begin{aligned}\mathit{allowed}(n, A, R) \equiv\ & n \in \mathit{agents}(R) \wedge \\ & (\mathit{concur}(n) = 1 \vee n \notin \{m \mid (f, m, e, v) \in A \wedge e \neq \bot\})\end{aligned}$$

The new active run is a tuple containing the run identifier f, the agent name n, the events of the role, and the initial temporary variable assignment. We assume the existence of a function $\mathit{events} : \mathit{RoleName} \to \mathit{Ev}$ that takes a protocol name and a role name and returns its corresponding list of events. The initial temporary variable assignment maps the run identifier variable to its fresh value ($\theta \mapsto f$).

The operational semantics for the create rule are given by the following rule:

$$\text{create}\ \frac{f \in \mathbb{N} \qquad f \notin \mathit{runids}(A) \qquad \mathit{allowed}(n, A, R)}{\langle A, \sigma, I\rangle \xrightarrow{\mathit{create}(f,R)} \langle A \cup \{(f, n, \mathit{events}(R), \{\theta \mapsto f\})\}, \sigma, I\rangle}$$

Terminating a run. A run of an agent can be terminated at all times: right after it is created, after executing a number of events, or after all events have been

executed. Upon termination of a run, the event list is replaced with $\perp$. The rule is as follows:

$$\text{end}\frac{a = (f, n, e, \rho) \in A}{\langle A, \sigma, I\rangle \xrightarrow{end(f)} \langle A\backslash\{a\} \cup \{(f, n, \perp, \rho)\}, \sigma, I\rangle}$$

Sending a message. If an agent executes a *send* event, a message gets added to the adversary knowledge. The message is obtained from the event by applying the temporary variable assignment ρ and persistent variable assignment σ to the parameter of the send event. Upon executing the send event, it is removed from the list of events for that run.

A send event can be accompanied by a list of persistent variable assignments of the form $x := c$. We denote by $\overrightarrow{x} := \overrightarrow{c}$ the sequential assignment of a list of values c to a list of variables x of the same length. The persistent variable assignment of *all* agents is represented by σ. If a send event is accompanied by a list of assignments, σ must be updated for *one* agent. For readability we use σ_n to denote the persistent variable assignment for agent n. Given an assignment $\overrightarrow{x} := \overrightarrow{c}$, the substitution σ_n for agent n must be updated to $\sigma_n[\overrightarrow{c}/\overrightarrow{x}]$. Therefore, the persistent variable assignment σ must be updated to $\sigma[n \mapsto \sigma_n[\overrightarrow{c}/\overrightarrow{x}]]$.

$$\text{send}\frac{\begin{array}{c}\sigma_n\rho(e) \xrightarrow{send(\sigma_n\rho(m))[\overrightarrow{x}:=\overrightarrow{c}]} \sigma_n\rho(e') \\ a = (f, n, e, \rho) \in A \qquad \sigma' = \sigma[n \mapsto \sigma_n[\overrightarrow{c}/\overrightarrow{x}]]\end{array}}{\langle A, \sigma, I\rangle \xrightarrow{send(f,\sigma_n\rho(m))} \langle A\backslash\{a\} \cup \{(f, n, e', \rho)\}, \sigma', I \cup \{\sigma_n\rho(m)\}\rangle}$$

Example 3.8 (Agent rules)**.** *In this example, we show how the agent rules and composition rules are combined to derive that an agent can do a* send *event. We assume that there is an agent $t \in Agent$ with an active run with runid $\theta = 3$. We further assume that the term 0 is assigned to the persistent variable S for agent t, i.e. $\sigma_t(S) = 0$. Then we can derive:*

$$send\frac{\begin{array}{c}(send(1) \triangleleft 0 = 1 \triangleright send(2)) \cdot \perp \xrightarrow{send(2)} \perp \\ a = (3, t, (send(1) \triangleleft S = 1 \triangleright send(2)) \cdot \perp, \rho) \in A\end{array}}{\langle A, \sigma, I\rangle \xrightarrow{send(3,2)} \langle A\backslash\{a\} \cup \{(3, t, \perp, \rho)\}, \sigma, I \cup \{2\}\rangle}$$

We can complete the derivation by importing the derivation of Example 3.7.

Reading a message. The *read* rule defines when an agent can receive and interpret a message from the network. Since we represent the network as the adversary knowledge, any message that the adversary can infer from his knowledge can be read by an agent. The role specification of an agent specifies the *pattern* of the message that the agent receives. The message received from the network must, therefore, be of the same format as the pattern. Upon receiving a message, the agent updates his temporary variable assignment by assigning a value to all variables in the pattern. We define two auxiliary predicates, *Rd* and *Match*, to decide whether a message is readable and how the temporary variable assignment of the agent should be updated.

A received term m is readable with respect to a pattern p if there is a substitution ρ that makes them syntactically equivalent. Furthermore, every subterm of the received message must be inferable from the agent's knowledge or from the received message itself. We first give the formal definition of the *subterm* operator. It is used to decompose a term into the terms from which it was constructed.

Definition 3.9 (Subterms)**.** *Let $t, t_1, t_2 \in Term$ be terms. The subterm operator is defined as follows.*

$$\begin{array}{lll} t \sqsubseteq t & t_1 \sqsubseteq (t_1, t_2) & t_2 \sqsubseteq (t_1, t_2) \\ t_1 \sqsubseteq \{t_1\}_{t_2} & t_2 \sqsubseteq \{t_1\}_{t_2} & t \sqsubseteq h(t) \end{array}$$

Readability is then defined as follows.

Definition 3.10 (Readability)**.** *Let $m, p \in Term$, $K \in \mathcal{P}(Term)$, and $\rho(p) = m$. Then, we define readability of term m by*

$$Rd_K(\rho, m, p) \equiv \forall_{a \sqsubseteq p}\ K \cup \{m\} \vdash \rho(a) \vee K \cup \{m\} \vdash \rho(a)^{-1}.$$

The intuition behind the definition is that in case the agent receives a complex term, all of its subterms must be interpretable by the agent. The agent can use his knowledge to interpret the subterms, but also the message that he receives. This is essential, for instance, if the message is the pairing of an encryption and the decryption key.

Example 3.11 (Readability predicate)**.** *We assume that $k \neq k^{-1}$ and $l = l^{-1}$. The following table illustrates the readability predicate on several examples. For instance, the last line should be read as follows. Let $K = \emptyset$ be a knowledge set and $p = (\{V\}_W, W)$ be a pattern with variables V and W. The subterms of p are $V, W, \{V\}_W$, and $(\{V\}_W, W)$. For a substitution $\rho = \{V \mapsto a, W \mapsto l\}$ the message $m = (\{a\}_l, l)$ is readable since $\{(\{a\}_l, l)\} \vdash a$, $\{(\{a\}_l, l)\} \vdash l$, and $\{(\{a\}_l, l)\} \vdash \{a\}_l$.*

K	ρ	p	m	$Rd(K, \rho, p, m)$
$\emptyset$	$V \mapsto \{a\}_k$	V	$\{a\}_k$	true
k^{-1}	$V \mapsto a$	$\{V\}_k$	$\{a\}_k$	true
$\emptyset$	$V \mapsto a$	$\{V\}_k$	$\{a\}_k$	false
k	$V \mapsto a$	$\{V\}_k$	$\{a\}_k$	false
a, k	$V \mapsto a$	$\{V\}_k$	$\{a\}_k$	true
a	$V \mapsto a$	$h(V)$	$h(a)$	true
$\emptyset$	$V \mapsto a$	$h(V)$	$h(a)$	false
$\emptyset$	$V \mapsto a, W \mapsto l$	$(\{V\}_W, W)$	$(\{a\}_l, l)$	true

The *match* predicate defines how the temporary variable assignment for the current run must be extended. Given a pattern p and a message m, the match predicate fixes a minimal substitution ρ that maps every variable in p to a ground term, such that $\rho(p) = m$.

Definition 3.12 (Match)**.** *Let $m, p \in Term$, $K \in \mathcal{P}(Term)$, and ρ be a substitution. Let $vars(p)$ denote the set of variables in a pattern p. Then the match predicate is defined by*

$$Match_K(\rho, m, p) \equiv m = \rho(p) \wedge dom(\rho) = vars(p).$$

We now have all ingredients to define the read rule:

$$\text{read}\frac{\begin{array}{cc}\sigma_n\rho(e) \xrightarrow{read(m)[\overrightarrow{x}:=\overrightarrow{c}]} \sigma_n\rho(e') & a = (f, n, e, \rho) \in A \qquad I \vdash m \\ Match(\rho', m, p) & Rd_{rng(\rho)\cup rng(\sigma_n)}(\rho', m, p)\end{array}}{\langle A, \sigma, I\rangle \xrightarrow{read(f,m)} \langle A\backslash\{a\} \cup \{(f, n, e', \rho' \circ \rho)\}, \sigma[n \mapsto \sigma_n[\overrightarrow{c}/\overrightarrow{x}]], I\rangle}$$

3.4.3 System behavior

The above semantics define a labeled transition system describing the behavior of a system. The labels represent the protocol steps performed by the agents. Formally,

$$\begin{aligned} Label = \{&send(f, m), read(f, m), create(f, R), end(f) \mid \\ &m \in Term, f \in \mathbb{N}, R \in RoleName\}. \end{aligned}$$

The behavior of a protocol P is defined by its traces. A trace is a tuple of a list of states $s_0, \ldots, s_n$ and a list of labels $t_1, \ldots t_n$, such that $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} s_n$. In other words, a trace is a derivation starting from state s_0 representing one possible interleaving of protocol executions. The length of the trace, denoted by $|(s_0, \ldots, s_n, t_1, \ldots, t_n)|$, is n. Traces are defined by:

$$\begin{aligned} Traces = \{(s_0, \ldots, s_n, t_1, \ldots, t_n) |\ & s_0 \ldots s_n \in State, \\ & t_1, \ldots t_n \in Label, \\ & \forall_{1 \le i < n}\ s_i \xrightarrow{t_i} s_{i+1}, \\ & s_0 = \langle \emptyset, \sigma, I\rangle\}, \end{aligned}$$

where I is the initial adversary knowledge and σ is the initial persistent variable assignment.

Example 3.13 (Trace)**.** *In this example, we explore a trace of the HMNB protocol in which a reader and a tag have a successful protocol execution. We assume that* $0, ID_0 \in Const$*. Let* σ, σ', ρ *and* ρ' *be the following substitutions:*

$$\begin{aligned} \sigma &= \{r \mapsto \{ID \mapsto ID_0\}, t \mapsto \{ID \mapsto ID_0, S \mapsto 0\}\} \\ \sigma' &= \{r \mapsto \{ID \mapsto h(ID_0, nr(1), ID' \mapsto ID_0\}, \\ &\quad\ t \mapsto \{ID \mapsto h(ID_0, nr(1)), S \mapsto 0\}\} \\ \rho &= \{\theta \mapsto 1, NT \mapsto nt(2)\} \\ \rho' &= \{\theta \mapsto 2, NR \mapsto nr(1)\} \end{aligned}$$

Moreover, let s_0 *and* s_{10} *be states:*

$$\begin{aligned} s_0 &= \langle \emptyset, \sigma, \emptyset\rangle \\ s_{10} &= \langle \{(1, r, \bot, \rho), (2, t, \bot, \rho')\}, \sigma', \{nr(1), h(ID_0), nt(2), h(ID_0, nt(2))\}\rangle \end{aligned}$$

The following trace describes one interleaving of a reader run and a tag run.

$$s_0 \xrightarrow{create(1,R)} s_1 \xrightarrow{create(2,T)} s_2 \xrightarrow{send(1,nr(1))} s_3 \xrightarrow{read(2,nr(1))} s_4 \xrightarrow{send(2,(h(ID_0),nt(2)))} s_5 \xrightarrow{read(1,(h(ID_0),nt(2)))} s_6 \xrightarrow{send(1,(h(ID_0,nt(2),nt(2)))} s_7 \xrightarrow{read(2,(h(ID_0,nt(2),nt(2)))} s_8 \xrightarrow{end(1)} s_9 \xrightarrow{end(2)} s_{10}$$

3.5 Untraceability

An RFID system satisfies *untraceability* if an attacker cannot infer whether different actions were performed by the same tag or by two different tags. If untraceability is not satisfied, an attacker can attribute different actions to one (possibly unknown) tag. By linking one of these actions to a person that carries the tag the attacker effectively traces that person.

Consider a building access system based on RFID-tagged employee badges. When a user wants to open the door he swipes his RFID tagged badge across the RFID reader. After successful execution of a protocol between reader and tag, the door is unlocked and the user is granted access. The simplest RFID protocol one can think of is the protocol in which a reader queries a tag and the tag responds with its identifier. The reader then contacts a database server to verify whether the person carrying the tag identifier should be granted access and subsequently unlocks the door. It is essential for the protocol that tag identifiers are secret and not known to the adversary.

In our analysis of RFID protocols we make several simplifying assumptions. One of these assumptions is that we consider the communication between the reader and the database server to be secure. This means that the adversary's powers are restricted to the protocol messages exchanged between the tag and the reader. We, therefore, only model the protocol between the tag and the reader. Also, passive tags cannot initiate a protocol execution, since they need to draw power from the messages sent by the reader. For our analysis, we only model communication that actually carries protocol messages and leave out empty reader messages.

We now present four RFID protocols that can be used in a building access control system (Figure 3.4). The purpose of these protocols is to build our intuition about untraceability of RFID protocols. In the next section, we make this intuition more precise.

The message sequence chart in Figure 3.4a depicts the RFID protocol for building access described above. The protocol consists of one message in which the RFID tag transmits its identifier to the RFID reader. The protocol is not untraceable: any time the tag executes a run of the protocol it broadcasts its identifier. An eavesdropping adversary can easily recognize tags by eavesdropping on all communication on the network. If he observes an identifier twice, he can infer that the same tag was present twice.

In the second protocol (Figure 3.4b), the tag does not send its identifier in plain text, but rather a cryptographic hash of the identifier. Upon receiving the mes-

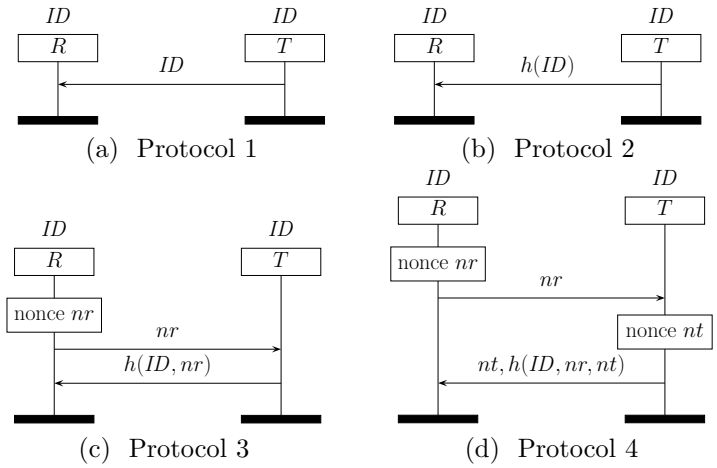


Figure 3.4: Traceable and untraceable protocols

sage, the reader can go through his list of tag identifiers, compute the hash of every identifier and compare it against the received message. Even though an eavesdropping attacker can no longer recover the identifier of the tag, the protocol is still not untraceable. Since we assume cryptographic hash functions to be perfect, we know that $h(ID) = h(ID')$ if and only if $ID = ID'$. The attacker can thus observe that two runs of the protocol were performed by the same tag simply by comparing messages sent by tags.

The third protocol (Figure 3.4c) adds a reader challenge to the protocol. The challenge consists of a nonce, freshly generated by the reader. The tag responds by pairing his identifier and the nonce and returning the cryptographic hash of the pairing. Due to the reader's randomness in the tag responses, an eavesdropping adversary is not able to link two tag responses. However, an adversary can observe one protocol execution with messages nr and $h(ID, nr)$. If he *replays* the nonce nr to another tag, he gets the response $h(ID', nr)$. We have $h(ID, nr) = h(ID', nr)$ if and only if $ID = ID'$. This gives the attacker a procedure to recognize a tag that he previously observed: the adversary can trace a tag by repeatedly challenging tags with the same nonce and comparing the responses.

Finally, the fourth protocol (Figure 3.4d) adds a tag-generated nonce nt to the response. This ensures that the tag response always contains randomness generated by the tag. The attacker cannot link any two executions of the tag. Therefore, the protocol is untraceable.

3.5.1 Defining untraceability

We define untraceability as a trace property of a role in a protocol. We consider traces in which an agent executes the same role twice. A protocol is untraceable if for any such trace, there exists a trace that is indistinguishable to the adversary, in which the role was executed by two different agents. Before presenting the definition of untraceability, we first define the concepts of *linkability*, *reinterpretation*, and *indistinguishability*.

For ease of notation we define the agent of a run with run identifier f in system state $S = \langle a, \sigma, I \rangle$ to be:

$$agent_S(f) = n \quad \equiv \quad \exists_{a \in A}\, a = (f, n, e, v)$$

Let (s, t) be a trace. The function *role* returns the rolename of a given run with run identifier f:

$$role_{(s,t)}(f) = R \quad \equiv \quad \exists_{1 \leq i \leq |(s,t)|}\, t_i = create(f, R)$$

Definition 3.14 (Linkability of runs)**.** *Two runs f and f' of a trace (s, t) of length l are linked, denoted by $L_{(s,t)}(f, f')$, if and only if they are executed by the same agent:*

$$L_{(s,t)}(f, f') \quad \equiv \quad s_l = (A, \sigma, I) \wedge agent_{s_l}(f) = agent_{s_l}(f')$$

The notion of reinterpretation has been introduced by Garcia, Hasuo, Pieters, and Van Rossum [GHPvR05]. We use it to express that subterms of a message can be substituted by other terms if the adversary is not able to read (or interpret) these subterms. All terms that the adversary can interpret remain unchanged. Any reinterpretation must, therefore, satisfy the following requirements. Basic terms such as agent names, nonces, and constants can be interpreted by the adversary. Hence, any reinterpretation *must* map such a term to itself. A ciphertext can be mapped to another term if (1) the adversary cannot decrypt it, or (2) the adversary does not know the plaintext and the encryption key. If the adversary can interpret the structure of a message, the reinterpretation must maintain it. Therefore, in case a message is a pairing of two other messages, its reinterpretation must be a pairing of the reinterpretation of the two messages. A hash of a message which is known to the adversary, must be mapped to the hash of the reinterpretation of that message. The same condition holds for any message to which a function has been applied.

Definition 3.15 (Reinterpretation)**.** *A map π from terms to terms is called a reinterpretation under knowledge set K if it and its inverse π^{-1} satisfy the following conditions:*

$$\begin{array}{ll}
\pi(m) = m & \text{if } m \in Agent \cup Nonce(\mathbb{N}) \cup Const \\
\pi(m) = (\pi(m_1), \pi(m_2)) & \text{if } m = (m_1, m_2) \\
\pi(\{m\}_k) = \{\pi(m)\}_k & \text{if } K \vdash k^{-1} \\
 & \text{or } K \vdash m \wedge K \vdash k \\
\pi(f(m)) = f(\pi(m)) & \text{if } K \vdash m \\
 & \text{or } f \text{ is not a hash function.}
\end{array}$$

Reinterpretations of labels are defined in the following way:

$$\begin{array}{ll}
\pi(send(f, m)) & = send(f, \pi(m)) \\
\pi(read(f, m)) & = read(f, \pi(m)) \\
\pi(create(f, R)) & = create(f, R) \\
\pi(end(f)) & = end(f)
\end{array}$$

K	m	$\pi(m)$
$\emptyset$	a	a
$\{k^{-1}\}$	$\{a\}_k$	$\{a\}_k$
$\{a\}$	$\{a\}_k$	$\{b\}_l$
$\emptyset$	$(\{a\}_k, h(\{a\}_k))$	$(h(b), h(h(b)))$

Example 3.16. *Let $K \in \mathcal{P}(Term)$ be a set of knowledge and $m \in Term$ be a ground term. The following table presents the reinterpretation function on several examples.*

Note that for the first and second example, the reinterpretation is unique, while for the third and fourth example infinitely many reinterpretations exist. In the fourth example, the reinterpretation must always satisfy the condition that the second part of the pair is a hash of the first part of the pair.

We use reinterpretations to define indistinguishability of traces. Two traces are indistinguishable to the adversary, if the adversary cannot see any meaningful difference between the two traces, based on the knowledge he has.

Definition 3.17 (Indistinguishability of traces)**.** *Let (s,t) and (s',t') be traces of length l and I the adversary knowledge in state s_l. Trace (s,t) is indistinguishable from trace (s',t'), denoted by $(s,t) \sim (s',t')$, if there is a reinterpretation π under I, such that $\pi(t_i) = t'_i$ for all $1 \leq i \leq l$.*

We can now formally define untraceability. Untraceability is the property that for every trace of a protocol in which two runs are linked, there is a trace that is indistinguishable to the adversary in which these two runs are not linked.

Definition 3.18 (Untraceability)**.** *A protocol is untraceable with respect to role R if*

$$\forall_{(s,t)\in Traces} \\ \forall_{f\neq f'\in\mathbb{N}} L_{(s,t)}(f,f') \wedge role_{(s,t)}(f) = role_{(s,t)}(f') = R \Rightarrow \\ \exists_{(s',t')\in Traces}\ (s,t) \sim (s',t') \wedge \neg L_{(s',t')}(f,f').$$

3.5.2 An untraceable protocol

Feldhofer, Dominikus, and Wolkerstorfer [FDW04] have shown that it is possible to implement the symmetric cipher AES on an RFID tag. In their paper, they discuss two simple protocols for unilateral and mutual authentication. These protocols are based on the ISO/IEC 9798-2 standard [ISO08]. The *unilateral* authentication protocol is not untraceable. In this section, we prove that the *mutual* authentication protocol *is* untraceable.

Figure 3.5 shows a graphical representation of the protocol specification. The protocol assumes that every pair of reader R and tag T shares a unique key k_T. These shared keys are initially not part of the adversary's knowledge. The reader initiates the protocol by sending a freshly generated nonce nr to the tag. The tag generates a nonce nt, encrypts the pair (nr, nt) under the shared key k_T, and sends it to the reader. The reader decrypts the message using the same shared key,

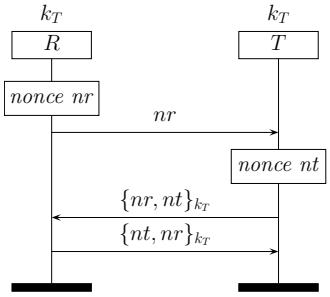


Figure 3.5: An untraceable mutual authentication protocol.

reverses the order of the two nonces, encrypts the message under the shared key, and sends it to the tag.

Theorem 3.19. *The mutual authentication protocol of ISO/IEC 9798-2, depicted in Figure 3.5, is untraceable with respect to role T.*

Proof. We notice first that k_T and nt remain secret throughout the protocol execution. This can be easily verified with an automated tool.

Let (s, t) be a trace with two runs with run identifiers $f \neq f'$. We assume that they are both executions of role T: $role_{(s,t)}(f) = role_{(s,t)}(f')$. We consider the case where f and f' were executed by the same agent, i.e. $L_{(s,t)}(f, f')$. For every such trace, there must exist an indistinguishable trace $(s', t') \sim (s, t)$ in which f and f' were executed by different agents, i.e. $\neg L_{(s',t')}(f, f')$.

For ease of notation, we set $agent_{s_l}(f) = agent_{s_l}(f') = agent_{s'_l}(f) = Alice$ and $agent_{s'_l}(f') = Bob$. The general idea of the proof is that the trace (s', t') can be constructed from (s, t) by replacing all occurrences of *Alice* in run f' by *Bob*. We make this more precise below and motivate that the adversary cannot distinguish (s, t) from (s', t').

Since we are verifying untraceability for role T, we may assume that the agent executing T is trusted, i.e. that it executes all read and send events according to the specification. By definition, there are $f', i \in \mathbb{N}$ such that the trace (s, t) contains the event where $\{nr(i), nt(f')\}_{k_{Alice}}$ is sent.

We consider the map π with the following properties:

$$\begin{array}{lll} \pi(\{x, nt(f')\}_{k_{Alice}}) = \{x, nt(f')\}_{k_{Bob}} & & \text{for any } x, \\ \pi(\{nt(f'), x\}_{k_{Alice}}) = \{nt(f'), x\}_{k_{Bob}} & & \text{for any } x, \\ \pi(m) = m & & \text{elsewhere.} \end{array}$$

Note that π is a reinterpretation under the adversary's knowledge, by Definition 3.15 and secrecy of k_T.

Let $t' = \pi(t)$. We show that (s', t') is a valid trace. The only difference between t and t' occurs in messages containing the nonce $nt(f')$. By construction, the changes produce a valid run for Bob while keeping the reader's run valid. It follows from the secrecy of nt and k_T that any further occurrence of $nt(f')$ must be in $\{nr(i), nt(f')\}_{k_{Bob}}$ or $\{nt(f'), nr(i)\}_{k_{Bob}}$. Since $nr(i)$ is produced by run i of R, no other run of R will accept the former message. Similarly, since $nt(f')$ is produced by run f' of Bob, no other run of Bob will accept the latter message.

Finally, for run f we have $agent_{s_l}(f) = agent_{s'_l}(f)$ thus $\neg L_{(s',t')}(f, f')$. □

3.5.3 A traceable protocol

In this section, we revisit the HMNB protocol of our running example. Using our framework we can show that the protocol is not untraceable.

In the HMNB protocol, the tag's response depends on the state S of the tag at the start of the protocol execution. If $S = 0$ the tag responds with $h(ID), nt$ and otherwise the tag responds with $h(ID, nr, nt), nt$. Because the adversary does not know ID, he can not conclude from the tag's response in which state the tag was. He may, however, take advantage of the reader's ability to distinguish between the two states. If the tag was in state $S = 0$ at the beginning of the protocol, the reader cannot verify whether the value of the nonce nt has changed during transmission. Thus, an accidental or malicious modification of nt does not result in a rejection of the tag's response by the reader. The reader completes its run by sending the third message of the protocol. If the tag was in state $S = 1$, the reader uses nt, its own nonce nr, and ID to compute a hash value and compare it with the received one. In this case, a modification of nt can be detected and leads to a rejection of the tag's response and an early termination of the protocol execution by the reader.

We exploit this weakness to find a trace that cannot be reinterpreted. Choose two tags a_0 and a_1 and create a trace in which a_0 is put into state $S = 1$. The adversary can do this by challenging a_0 with any value and terminating the protocol before sending the third message. The adversary then performs a man-in-the-middle attack. He obtains a challenge from the reader and sends it to the tag to obtain a response. He then replaces the nonce provided by the tag with a different value and submits the response to the reader. If the reader accepts the response, the tag was in state $S = 0$, hence the selected tag is a_1. If the reader rejects the response, the tag was in state $S = 1$, hence the selected tag is a_0. Figure 3.6 depicts the trace for the rejection case.

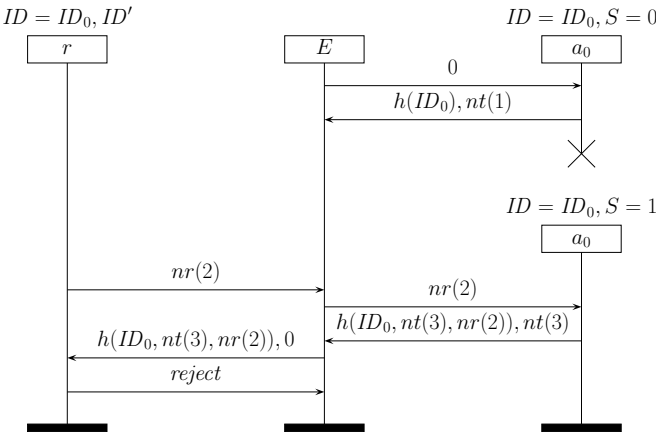


Figure 3.6: Attack on HMNB

To prove that the HMNB protocol is not untraceable, it suffices to show one trace of the protocol in which a tag executes two runs of the protocol. For this trace,

there must not exist an indistinguishable trace in which the runs are executed by different agents.

Theorem 3.20. *If all tags initially are in state $S = 0$, then the HMNB protocol does not satisfy untraceability.*

Proof. Let $s_0 \in State$ be an initial system state in which all tags have value 0 assigned to their state variable S. That is, $\sigma_{a_i}(S) = 0$ for all tags a_i. The following trace represents the attack depicted in Figure 3.6.

$$s_0 \xrightarrow{t_0=create(1,T)} s_1 \xrightarrow{t_1=read(1,0)} s_2 \xrightarrow{t_2=send(1,(h(ID_0),nt(1)))} s_3 \xrightarrow{t_3=create(2,R)} s_4 \xrightarrow{t_4=create(3,T)} s_5 \xrightarrow{t_5=send(2,nr(2))} s_6 \xrightarrow{t_6=read(3,nr(2))} s_7 \xrightarrow{t_7=send(3,(h(ID_0,nt(3),nr(2)),nt(3)))} s_8 \xrightarrow{t_8=read(2,(h(ID_0,nt(3),nr(2)),0))} s_9 \xrightarrow{t_9=send(2,reject)} s_{10} \xrightarrow{t_{10}=end(2)} s_{11} \xrightarrow{t_11=end(3)} s_{12}$$

The trace contains two runs of tag a_0 with run identifiers 1 and 3. Therefore, runs 1 and 3 are linked: $L_{(s,t)}(1,3)$. We need to show that there does not exist a reinterpretation function π that gives an indistinguishable trace for which runs 1 and 3 are not linked.

Following Definition 3.15, any reinterpretation π has to satisfy $\pi(0) = 0$, $\pi(nt(1)) = nt(1)$, $\pi(nr(2)) = nr(2)$, $\pi(nt(3)) = nt(3)$, and $\pi(reject) = reject$. Additionally, for $t'_7 = send(3,(v,w))$ and $t'_8 = read(2,(x,y))$, π must be such that $v = x$.

For any valid indistinguishable trace (s', t'), the reader must reject the tag in run 2. Let a_1 be the tag executing run 3. By construction, a_1 is only rejected in run 2 if its variable S is assigned the value 1. Initially, S is assigned 0 for all tags a_i. The state variable S must, therefore, be changed as a result of a_1 executing run 1 of trace (s', t'). Hence, $L_{(s',t')}(1,3)$ for all valid indistinguishable traces (s', t').

Therefore, the HMNB protocol does not satisfy untraceability. □

3.6 Related work

Arapinis, Chothia, Ritter, and Ryan [ACRR10] analyze untraceability in the applied pi calculus. They give definitions for *strong unlinkability* and *weak unlinkability*. Their definitions concern traces in which two messages from different sessions were sent by the same agent. For *weak unlinkability*, for any such trace there must exist another trace of the system that is indistinguishable to the adversary; in this other trace, the messages must be sent by different agents. Indistinguishability of traces is defined as static equivalence of processes [AC06]. *Strong unlinkability* is defined as a bisimulation relation on processes. It is satisfied if the process in which the original protocol can be executed by an arbitrary number of agents is bisimilar to the process in which an agent never executes the protocol more than once. This bisimulation relation is implied by observational equivalence in the applied pi calculus and can sometimes be automatically verified using the ProVerif tool [Bla01]. Strong unlinkability is strictly stronger than weak unlinkability. Thus, when analyzing a protocol for weak unlinkability, it is useful to first verify whether it satisfies

strong unlinkability. The absence of attacks on the strong variant implies the weak variant. However, if strong unlinkability is not satisfied then there does not necessarily exist a practical attack on untraceability of the protocol. Our definition of untraceability resembles weak unlinkability.

Brusò, Chatzikokolakis, and Den Hartog [BCdH10] define untraceability in a formal model based on the applied pi calculus. They assign tag interfaces to RFID tags through which communication with the tags is possible. The essential feature of the model is that a single tag can have multiple interfaces. A priori, the attacker cannot discover whether two tag interfaces belong to one tag, or to two different tags. Their definition of *unlinkability* requires that the process in which the attacker communicates with two different interfaces of the *same tag* is indistinguishable from the process with two interfaces of two *different tags*. Two processes are defined to be indistinguishable if they are observationally equivalent. Due to the choice of using observational equivalence as a security property and the applied pi calculus to model the protocols, parts of the protocol analysis can be performed using the ProVerif tool [Bla01]. The protocol model takes two RFID-specific features into account: (1) the limitation that RFID tags can only execute one session at a time and (2) the statefulness of RFID protocols. Both properties are modeled by an auxiliary channel, acting like a buffer, to store variables.

The notions of reinterpretation (Definition 3.15) and trace indistinguishability (Definition 3.17) are based on the work by Garcia, Hasuo, Pieters, and Van Rossum [GHPvR05]. They define a formal framework for the analysis of anonymous communication protocols. Central to their work is the concept of anonymity sets: the set of agents among which a given agent cannot be identified. They give epistemic logic-based definitions for sender anonymity (the receiver of a message does not know who sent the message), unlinkability (the sender and receiver of a message cannot be linked), and plausible deniability (an agent can show that he did not know that he possessed a message). Finally, they analyze two anonymity protocols in their framework.

The formal model developed in Sections 3.1 through 3.4 is an extension of the operational semantics for security protocols by Cremers and Mauw [CM05, Cre06b]. Our model is different from their model in several ways:

- In the model by Cremers and Mauw the adversary is a parameter of the system. Their system state includes a *send buffer* and a *read buffer*. The adversary capabilities are formalized by semantic rules for the adversary that act on these buffers. For instance, the capability of an adversary to eavesdrop a certain message is modeled by a rule that moves a message from the send buffer to the read buffer and adds it to the adversary knowledge. Since we assume a Dolev–Yao adversary with full power over the network, we do not have to separate the read buffer and the send buffer. Instead, we model sending a message by adding it to the adversary knowledge and allow agents to read any message from the adversary knowledge. The disadvantage of this approach is that the agent semantics have to be adapted if we want to model a weaker adversary.

- Cremers and Mauw make a distinction between role terms and run terms to separate the specification from the execution level. Role terms are trans-

formed to run terms by applying an instantiation function. In our model, the only difference between the specification and execution level is that at specification level, terms can contain variables, while at execution level they cannot. The side-effect of this choice is that it allows protocol specifications to contain agent names.

- Our model incorporates persistent variables to allow agents to store and update terms that are maintained across different runs. This enables modeling stateful protocols. As a consequence, we can no longer verify readability of terms at the syntactic level, but only at the time of execution of the protocol.
- The model by Cremers and Mauw only allows to compose events sequentially. We have extended the model by adding alternative composition and conditional branching.
- In our model, the sender and receiver of messages are not explicit in the traces. The reason for this it that in RFID protocols, tags and readers do not know who their communication partner is at the start of the protocol execution. Furthermore, for untraceability it is important that the sender of a message remains secret. To make sender and receiver explicit in the traces, they can be added to the protocol specification, imitating the Cremers and Mauw model.
- We do not have explicit *create*, *end*, and *claim* events in the protocol specification. Instead, we assume that new runs can always be created and always be terminated. We have no explicit claim events for the reason that it is not observable by the adversary that at a certain point in the trace a security property is satisfied.

3.7 Conclusion

We have developed a formal framework for analyzing RFID protocols. The syntax allows users to describe a protocol in an unambiguous way. The protocol behavior can then be systematically derived as described by the semantics. Security properties such as authentication, secrecy, untraceability, and ownership transfer can be modeled as requirements on the traces of a protocol.

A protocol is untraceable if for any trace in which the adversary observes the same tag twice, there is a trace in which two different tags appear. This second trace should be indistinguishable to the adversary, i.e. the adversary cannot see any meaningful difference between the two traces. We have formalized untraceability based on the notion of reinterpretation as introduced by Garcia, Hasuo, Pieters, and Van Rossum [GHPvR05]. We have then applied our model and definition to existing RFID protocols. First, we have proven that the mutual authentication protocol of ISO/IEC 9798-2 [ISO08] is untraceable. Second, we have proven that the stateful RFID protocol by Ha, Moon, Nieto, and Boyd [HMNB07] is traceable.

4

Untraceability attacks

There are three main complicating factors in the design and verification of untraceable RFID protocols. First, untraceability has only relatively recently been studied. Several (incomparable) definitions of untraceability have been proposed in literature (see Chapter 5). A protocol proven untraceable in one model, may thus be traceable in another model. Second, many protocols are designed under the restriction that an RFID tag is not powerful enough to perform public-key cryptography. Designing untraceable protocols under this restriction requires non-standard techniques that may lead to unforeseen flaws. Finally, theoretical results show that it is impossible to design an efficient symmetric-key protocol that satisfies authentication and untraceability based on symmetric keys (see Section 4.7.2).

For these reasons, many RFID protocols in literature have untraceability flaws. In this chapter, we describe new attacks on the untraceability of a number of RFID protocols. Based on the attack strategy of the adversary, we classify these attacks into five categories. These five categories represent the most common attacks on untraceability. As such, the categorization provides a library of attacks for protocol designers to refer to, so as to minimize the chance that these flaws reappear in new protocol proposals.

Note that we do not give a formal description of the attacks in any of the proof models. Instead, we give a general high-level description of the attacks. Nevertheless, these descriptions are sufficient to reconstruct a formal description of the attack in any of the proof models and thus formally prove traceability of the protocols.

4.1 Overview

Perhaps the least complicated type of attacks to carry out in practice is the *attribute acquisition attack* (Section 4.2). In such an attack the attacker interacts with tags and combines the messages to derive a unique attribute for a tag. This unique attribute serves as an identifier by which the attacker can trace the tag. If the attacker can derive the unique attribute from the messages of two seemingly different tags, then he knows that the two tags are actually the same. Frequently, only a few interactions with a tag are needed to execute the attack. In practice, an attacker would install one of his rogue readers in a place where he expects to observe many people carrying RFID tags. He then traces tags by interacting with them and computing their unique attributes.

In a *man-in-the-middle attack* (Section 4.3) the attacker positions himself in the communication channel between the legitimate reader and legitimate tag. He can

choose to forward messages, but also selectively block or modify messages. One additional difficulty in designing untraceable RFID protocols is that the attacker may have the possibility to observe whether a reader accepts a session with a tag as valid or not. The attacker can exploit this capability in a man-in-the-middle attack by modifying a message and then observing whether the tag is accepted or rejected. Man-in-the-middle attacks are harder to execute than most attribute acquisition attacks, since the attacker needs to have a simultaneous connection with a genuine reader and a tag.

Insider attacks (Section 4.4) are a type of attacks that appear in traditional communication protocols as well as in RFID protocols. In an insider attack, the attacker is assumed to possess the key material of one or more legitimate RFID tags in the system. The attacker abuses such an insider tag to trace a tag that he has not corrupted. RFID systems are usually closed systems, meaning that the adversary does not have access to the key material of any legitimate tag. However, a sufficiently motivated attacker may be able to insert one of his own tags into the system or corrupt a legitimate tag to obtain its key material.

Several RFID protocols consist of a set of more than one protocol executable by the tag. Each of these protocols is meant to achieve one or more goals. If different protocols share key material, then the attacker may be able to combine messages from different protocols to trace a tag. We call these attacks multi-protocol attacks, or *compositionality attacks* (Section 4.5). Depending on the particular flaw, an attacker can either interact with just the tag, or with the tag and a reader. In the former case, it is easy to perform the attack in practice, in the latter case it is more complicated.

Our last type of attacks results from the frequent use of pseudonyms instead of unique identifiers for RFID protocols. Pseudonym-based protocols are stateful and may be subject to *pseudonym-based attacks* (Section 4.6). Pseudonym-based attacks exploit the adversary's knowledge of the state in which a tag is to trace a tag. An example of this attack is the attack on the HMNB protocol in the previous chapter. Other untraceability attacks exist if an attacker can desynchronize the pseudonym value of the reader and tag or exploit a weakness in the pseudonym-update procedure. Most pseudonym-based attacks are hard to perform in practice since they require an attacker to control all communication between all legitimate tags and readers. Therefore, they generally are of limited impact.

4.2 Attribute acquisition attacks

A necessary condition for tag untraceability is that an adversary, who has observed a particular tag once, must not be able to recognize the tag as being the same tag in the future. To refine this condition, we call a term that the adversary can derive from one or more runs of a tag and which identifies the tag to the adversary, a *unique attribute* of the tag. The absence of unique attributes derivable by the adversary is essential for untraceability of the protocol. In case the adversary is able to compute a unique attribute, then we refer to the adversary's steps to arrive at such a term as the *attribute acquisition attack*.

A simple unique attribute can be found in protocols where the tag's answer to a

challenge c is merely a function $f_k(c)$ of the challenge and a secret (or collection of secrets) k and does not involve any nonce created by the tag. In this case, c is under the adversary's control, k is unique to the tag, and the adversary learns $f_k(c)$ after one round of communication with the tag. Thus for constant c chosen by the adversary, $f_k(c)$ is a unique attribute of the tag whose secret is k.

To prevent traceability in protocols that employ a challenge-response mechanism, the tag typically includes a nonce in its response. This ensures that when twice challenged with the same challenge c, the tag will respond with two different values. Not surprisingly, this does not necessarily prevent unique attributes from existing in a protocol.

Let c denote a reader challenge, r the tag's nonce, k the tag's secret, and $f_k(c, r)$ the tag response to challenge c. Then a unique attribute for that tag exists if (for n polynomial in the security parameter) the adversary can find challenges $c_1 \dots c_n$ and an efficiently computable function $g(\cdot)$ whose output does not depend on the tag's nonces $r_1, \dots r_n$:

$$g(c_1, f_k(c_1, r_1), \dots, c_n, f_k(c_n, r_n)) = \tilde{g}_k(c_1, \dots, c_n).$$

We call $\tilde{g}_k(c_1, \dots, c_n)$ the unique attribute.

The existence of an efficiently computable function g implies a straightforward way to trace tags. The attacker challenges a tag T with $c_1 \dots c_n$ and records the responses. He then computes g giving him the unique attribute $\tilde{g}_k(c_1, \dots, c_n)$. In a later stage, the attacker repeats the same procedure for tag T' and obtains $\tilde{g}'_{k'}(c_1, \dots c_n)$. If the two attributes are equal, the attacker knows that T and T' were the same tag, otherwise they were different tags.

4.2.1 The Kim, Choi, and Lee protocol

We show a new attribute acquisition attack on the protocol proposed by Kim, Choi, and Lee [KCL07] depicted in Figure 4.1. The protocol is designed to authenticate a tag T to a reader R. Each tag has an identifier ID and a key k, both known to the reader. The reader initiates the protocol by generating a fresh random value c. Upon receipt of the query c, the tag generates a nonce r. It then computes the bitwise exclusive-or ($\oplus$) of its identity ID and r as well as the exclusive-or of r and the cryptographic hash of c and k. The response is then sent to the reader and verified.

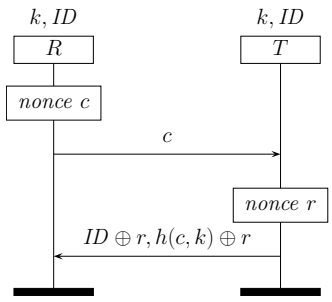


Figure 4.1: The Kim, Choi, and Lee protocol

The exclusive-or function has the following algebraic properties. For any terms a, b, and c and a constant term 0:

$$\begin{array}{ll} a \oplus a = 0 & a \oplus b = b \oplus a \\ a \oplus 0 = a & (a \oplus b) \oplus c = a \oplus (b \oplus c) \end{array} \tag{4.1}$$

An attribute acquisition attack can be carried out by an attacker that repeatedly queries tags with the same challenge a. We represent the tags secret by the tuple (k, ID) and represent the tag's response to query c by $f_{(k,ID)}(c, r) = (ID \oplus r, h(c, k) \oplus r)$. We define the function g by $g(x, (y, z)) = y \oplus z$. Then a unique attribute is defined by $g(a, f_{(k,ID)}(a, r)) = g(a, (ID \oplus r, h(a, k) \oplus r) = ID \oplus r \oplus h(a, k) \oplus r$. By repetitive application of Equations (4.1), we have $\tilde{g}_{(k,ID)} = ID \oplus h(a, k)$. Finally, for any $k', ID' \neq k, ID$ we have $\tilde{g}_{(k,ID)}(a) \neq \tilde{g}_{(k',ID')}(a)$.

4.2.2 The EC-RAC protocol

The construction of public-key protocols is interesting for several reasons. Public-key protocols aim to maintain untraceability against strong attackers. It has been shown that if the adversary is able to corrupt tags symmetric-key protocols cannot be untraceable [Vau07].

To maintain untraceability, tags cannot simply send their identity to the reader. Instead, the reader must learn the identity of the tag from the messages in the authentication protocol. However, if these messages are encrypted under a symmetric key (or if they are hashed) the reader must find the appropriate key by traversing its table of tag keys and trying each key to interpret the received message. If the number of tags in a system is large an attacker could perform timing attacks against a protocol [EA11]. Public-key protocols prevent this type of attack by enabling scalable tag lookup procedures on the reader's side. Based on the previous observation, Damgård and Pedersen have shown that in a system relying on symmetric keys, either untraceability, security, or scalability has to be sacrificed [DP08].

EC-RAC is a set of protocols aimed at providing untraceable tag authentication. The early EC-RAC protocols are among the first published and implemented asymmetric-key RFID protocols. There are a number of publications describing the protocols that belong to the EC-RAC protocol family. To distinguish the various revisions of EC-RAC, we call the original publication by Lee, Batina, and Verbauwhede EC-RAC I [LBV08]. The first revision, by the same authors, is called EC-RAC II [LBV09]. The second and third revision are by Lee, Batina, Singelée, and Verbauwhede, and are called EC-RAC III [LBSV10a] and EC-RAC IV [LBSV10b] respectively.

EC-RAC I is a challenge-response protocol initiated by the reader. EC-RAC II through IV are commitment-challenge-response protocols initiated by the tag. In this section we show a new attribute acquisition attack on EC-RAC I, while in the next sections we show various types of new attacks on EC-RAC II through IV.

The EC-RAC I protocol (Figure 4.2) is based on a fixed, system-wide elliptic curve over a finite field. The points P, $Y = yP$, x_1P, x_2P on the elliptic curve are publicly known and the scalar y is only known to the reader. The scalars x_1, x_2

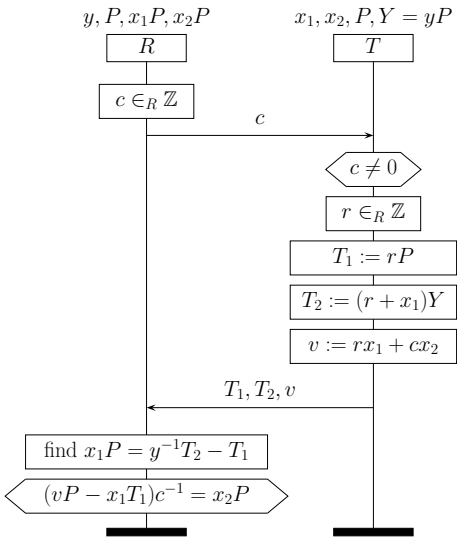


Figure 4.2: EC-RAC.

are unique to each tag and x_1 is known to the reader. The elliptic curve is assumed to have been chosen such that the computational Diffie-Hellman problem is hard. That is, given only the points xP, yP, and P on the elliptic curve, it is hard to compute xyP.

In the protocol, the reader challenges the tag with a random number $c \neq 0$ to which the tag responds with two points $T_1 = rP$, $T_2 = (r+x_1)Y$ on the elliptic curve and a scalar $v = rx_1+cx_2$. Using this information, the reader can compute $y^{-1}T_2-T_1 = x_1P$ to obtain the identity of the tag and then compute $(vP - x_1T_1)c^{-1} = x_2P$ to authenticate the tag.

Hence, EC-RAC I is a challenge-response protocol with challenge c and a response that can be written as $f_{(x_1,x_2)}(c, r) = rP, (r + x_1)yP, rx_1 + cx_2$. The points P and yP are constant and publicly known. To find a unique attribute, the adversary needs to find challenge terms $c_1, \ldots, c_n$ and functions g and $\tilde{g}$ such that $g(c_1, f_{(x_1,x_2)}(c_1, r_1), \ldots, c_n, f_{(x_1,x_2)}(c_n, r_n)) = \tilde{g}_{(x_1,x_2)}(c_1, \ldots, c_n)$, where $\tilde{g}$ does not depend on the tag's random numbers $r_1, \ldots, r_n$.

If we write $f_{(x_1,x_2)}(c, r) = T_1, T_2, v$ as in the protocol specification, then

$$g(c, f_{(x_1,x_2)}(c, r), c, f_{(x_1,x_2)}(c, r')) = \frac{T_1 - T_1'}{v - v'} = x_1^{-1}P$$

depends only on x_1. Thus $\tilde{g}_{(x_1,x_2($c, c') = x_1^{-1}P$ is a unique attribute. Finally, for any other tag with secrets x_1' and x_2', we have $\tilde{g}_{(x_1,x_2)}(c, c') \neq \tilde{g}_{(x_1',x_2')}(c, c')$

From the definition of the function g, it is now easy to obtain the attribute acquisition attack. The adversary challenges the tag twice with the same value c. The information received from the tag in the two runs can be used to compute the term $x_1^{-1}P$ as follows. Observe that $v - v' = (r_1 - r_1')x_1$ and $T_1 - T_1' = (r_1 - r_1')P$, thus, multiplying $T_1 - T_1'$ with the inverse of $v - v'$ modulo the order of the elliptic curve, the attacker obtains $x_1^{-1}P$.

Note that after executing this attack once, it suffices for the adversary to challenge any given tag only once with the previously used value c to determine whether the presented tag is equal to the tag identified by $x_1^{-1}P$.

A similar attribute acquisition attack was independently found by Bringer, Chabanne, and Icart [BCI08]. The authors observe that for any two protocol executions, the following equations hold:

$$cv' - c'v = (cr' - c'r)x_1$$
$$cT_1' - c'T_1 = (cr' - c'r)P$$

The attacker may then combine these two equations to obtain $x_1^{-1}P$ and proceed as described above.

4.3 Man-in-the-middle attacks

In this section, we assume that an attacker can observe whether a protocol execution was completed successfully. Attackers with this capability are sometimes called *wide* and attackers without it *narrow* [Vau07]. The assumption of wide attackers is reasonable in some settings. Consider, for instance, an RFID system that is used for building access. In such a system, the fact that a door opens indicates that the authentication protocol between the tag and the reader was carried out successfully.

The first man-in-the-middle attack on RFID protocols was on the HB$^+$ protocol of Juels and Weis [JW05]. The protocol uses the binary inner product and *xor* operator to prove knowledge of a key while keeping it secret. The attack by Gilbert, Robshaw, and Sibert [GRS05] breaks secrecy of a tag's key by first modifying the messages exchanged between reader and tag, then observing whether the reader accepts or rejects the tag, and finally using the observed information to set up and solve a system of linear equations. Knowledge of the key allows an attacker to trace tags in the system.

We now study man-in-the-middle attacks in which a wide attacker selectively modifies messages. Our first attack is a traceability attack on a public-key protocol and our second attack is on a symmetric-key protocol.

In this section, we show a man-in-the-middle attack on EC-RAC IV by Lee, Batina, Singelée, and Verbauwhede [LBSV10b]. The attack is also applicable to EC-RAC II and EC-RAC III.

EC-RAC IV assumes a group of points on an elliptic curve. The publicly known point P is chosen from a subgroup of prime order of the points on the curve. The point yP can be considered as the RFID reader's public key, y being a scalar only known to the RFID reader. RFID tags store a secret scalar x. The corresponding public key is xP and is used by the reader to identify a tag.

The protocol follows a commitment-challenge-response structure. It sends out a random point on the elliptic curve $T_2 = bP$ which serves as a commitment. The RFID reader challenges the tag with a random integer s upon which the tag answers with a point $T_3 = (b + h(s)x)Y$. The function $h(z)$ in EC-RAC IV denotes the x-coordinate of the point zP. Since our man-in-the-middle attack does not exploit any algebraic property of the function h, we simply assume that h is a cryptographic

hash function. The reader computes $(y^{-1}T_3 - T_2)h(s)^{-1}$ which equals xP. The protocol is given in Figure 4.3.

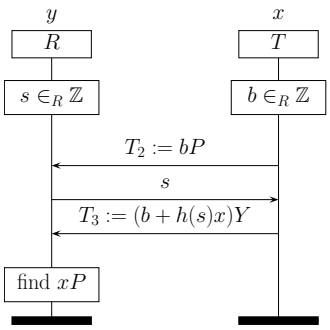


Figure 4.3: The EC-RAC IV protocol

The idea of the scheme is that anybody able to produce the correct response can also compute a particular secret. Thus successful completion of the protocol constitutes a proof of knowledge for the secret. A moment's thought shows that knowledge of the points xY allows an agent to authenticate itself as the tag whose public key is xP. Thus the intractability of the computational Diffie-Hellman problem is necessary in order for the schemes to provide tag authentication.

We first show how an attacker can modify the first and third message of *any* valid protocol execution without the tag being rejected. The attacker modifies the first message T_2 to $T_2 + \delta P$ and the third message T_3 to $T_3 + \delta yP$. At the end of the execution, the reader computes $xP = y^{-1}(T_3 + \delta yP) - (T_2 + \delta P)h(s)^{-1}$. The right-hand side of the equation equals $(y^{-1}T_3 + \delta P - T_2 + \delta P)h(s)^{-1} = (y^{-1}T_3 - T_2)h(s)^{-1}$. Thus, for any scalar δ, adding δP to the first message and δyP to the third message of a protocol execution between a legitimate reader and tag never leads to a rejection of the tag.

The attacker can trace a tag by eavesdropping on two protocol executions between a tag and a reader. In these executions he observes the messages T_2, s, T_3 and T_2', s', T_3', where $T_3 = (b + h(s)x)Y$ and $T_3' = (b' + h(s')x')Y$. In order to decide whether the two executions were carried out by the same tag, the attacker needs to be able to decide whether $x = x'$. We define α and β as follows.

$$\begin{aligned} \alpha &= h(s)T_2' + h(s')T_2 \\ &= (h(s)b' + h(s')b)P \\ \beta &= h(s)T_3' - h(s')T_3 \\ &= h(s)(b' + h(s')x')Y - h(s')(b + sx)Y \\ &= (h(s)b' + h(s')b)Y + h(s)h(s')(x' - x)Y \end{aligned} \tag{4.2}$$

It now follows that for $s \neq s'$, we have $y\alpha = \beta$ if and only if $x = x'$.

To find out whether this is the case, i.e. whether the two executions were carried out by the same tag, the adversary uses a communication between *any legitimate tag* and a reader as an oracle, as shown in Figure 4.4. As a man-in-the-middle, the attacker modifies the first message by adding α to it and the third message by adding β to it. The reader will accept the tag if and only if $y\alpha = \beta$. Hence, the reader accepts the tag if and only if $x = x'$. The attacker can use this decision

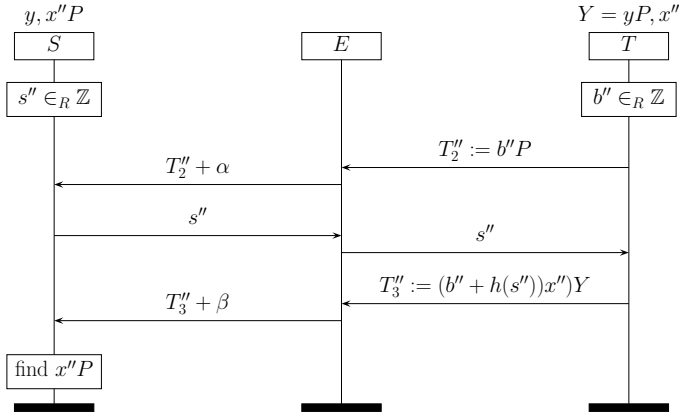


Figure 4.4: Man-in-the-middle attack on EC-RAC IV

procedure to find out whether any two protocol executions were carried out by the same tag.

A similar man-in-the-middle attack on the EC-RAC III protocol was found by Fan, Hermans, and Vercauteren [FHV10].

4.3.1 The Di Pietro and Molva protocol

Di Pietro and Molva [DM07] designed an RFID protocol aimed at identifying and authenticating a tag efficiently while keeping the tag untraceable. The protocol is based on a lightweight hash function called dpm.

The majority function $M(a, b, c)$ of three bits a, b, and c is defined by

$$M(a, b, c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c).$$

For any bitstring of size ℓ, where ℓ is a multiple of 3, the function $\mathrm{dpm} : \{0,1\}^\ell \to \{0,1\}$ is defined as the parity of the majority functions of consecutive bit-triplets. Its output is therefore one bit.

$$\mathrm{dpm}(x_0, \ldots x_{\ell-1}) = \bigoplus_{i=0}^{(\ell/3)-1} M(x_{3i}, x_{3i+1}, x_{3i+2}).$$

The protocol assumes that every tag and reader share a key k. The reader initiates the protocol by sending its name and a random value r_0 to the tag. The tag then generates q random bitstrings r_i and computes $\alpha_i = k \oplus r_i$ for $1 \leq i \leq q$. It also computes V by concatenating $\mathrm{dpm}(r_i)$ for every random bitstring. Finally, ω is set to $h(k, r_0, r_1, k)$. All terms are sent to the reader. The reader can find a particular tag's key k with the help of the bitstrings α_i and values $\mathrm{dpm}(r_i)$ by going through all the keys in its database and iteratively excluding the impossible ones, namely those for which $\mathrm{dpm}(k \oplus \alpha_i) \neq \mathrm{dpm}(r_i)$. It is expected that each α_i reduces the number of possible keys by approximately one half. At last, the reader uses ω to uniquely identify the correct key and authenticate the tag. The last message of the protocol allows the tag to authenticate the reader. The protocol is depicted in Figure 4.5.

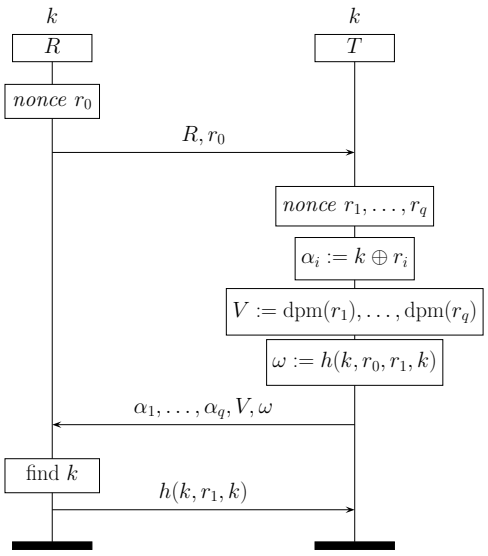


Figure 4.5: The Di Pietro and Molva protocol

The bitlength of k and the random numbers is a fixed constant ℓ which the protocol designers suggest to be 117. For a system with n tags, the suggested value of q is $2 \log n$.

In the following we show that over several runs, the protocol leaks $\frac{2\ell}{3}$ bits of k. Let $x = x_1 x_2 \cdots x_\ell$ be a bit string of length ℓ and let $\bar{x}_i$ denote the complement of the bit x_i. It is easy to see that $M(\bar{x}_1, x_2, x_3) = M(x_1, x_2, x_3)$ if and only if $x_2 = x_3$. Analogous equations hold for the complements of x_2 and x_3. It follows that

$$\mathrm{dpm}(\bar{x}_1, x_2, x_3, \ldots) = \mathrm{dpm}(x_1, x_2, x_3, \ldots) \Leftrightarrow x_2 = x_3, \tag{4.3}$$

again with analogous equations for any other bit of x.

The adversary can take advantage of property (4.3) as follows. Suppose the adversary intercepts the tag's message, flips the first bit of $\alpha_2 = r_2 \oplus k$ to obtain $\tilde{\alpha}_2$ and forwards the modified message to the reader. If the second and third bit of r_2 are equal, then $\mathrm{dpm}(k \oplus \tilde{\alpha}_2) = \mathrm{dpm}(k \oplus \alpha_2) = \mathrm{dpm}(r_2)$. In this case, the reader will still be able to find the correct key k and answer the tag with the third message of the protocol. However, if the second and third bit of r_2 are not equal, then $\mathrm{dpm}(k \oplus \tilde{\alpha}_2) \neq \mathrm{dpm}(r_2)$ and the reader will remove the key k from the list of possible keys. No other key will pass the verification with ω, thus the reader will not answer with the third message. The adversary can therefore distinguish the two cases.

It follows that by selectively flipping bits of α_2 an adversary may, after several protocol executions, determine for each consecutive bit triplet of k which bits are equal to each other. In other words, the adversary may determine the bits of k up to complements of consecutive bit-triplets.

This information can be used to reduce the complexity of computing all bits of k to a brute force search of a space whose size is the cubic root of the full key space. For the parameters of the system suggested by the designers of the protocol, this brute force search becomes feasible (2^{39} keys).

To break untraceability, the brute force search is not necessary. The probability

that two keys are equal up to complements of consecutive bit-triplets is vanishingly small. This follows from the fact that the shared keys k are chosen uniformly at random. If we assume that the number of tags n in the system is small compared to 2^ℓ, then the probability that for a given tag, there are one or more tags indistinguishable by the above method is approximately $1 - (1 - \frac{1}{2^{2\ell/3}})^n$. If we choose $\ell = 117$ as suggested by the authors and assume that there are $n = 2^{16}$ tags in the system, then the probability to find one or more tags which would be indistinguishable from a given tag is approximately $2.17 \cdot 10^{-19}$.

4.4 Insider attacks

Insider attacks are a major source of security breaches of computer systems. Some estimates show that as much as 70% - 90% of the security breaches are caused by insiders [Gol98]. One can think of various scenarios for insider attacks. For instance, a malicious merchant may want to cheat one of his customers, a disgruntled employee may want to inflict damage on his employer's assets, or a legitimate user of a system could be compromised and used in an attack against another user. The latter is the case when a computer system is infected with malware or trojan horses and used to attack another, more important, system. Common to all insider attacks is that the adversary abuses the credentials and knowledge of one compromised user to violate a particular security goal of *another* user.

Many cryptographic protocols achieve security in the absence of insider attackers, but fail to achieve their security goals when insider attackers are present. A well-known example is the the Needham-Schroeder protocol [NS78], which was first proven to be secure [BAN89], but later shown to be flawed in the presence of insider attackers [Low96]. It is therefore not surprising that standard frameworks for security protocol analysis assume that the adversary controls one or more malicious users in the system [Low98, Bla01, Cre06b].

To perform an insider attack, the adversary needs the key material stored in one legitimate tag. Since RFID tags are often used as hardware tokens, the users of RFID tags usually have no access to the key material. However, one can think of several practical scenarios for the adversary to acquire the key material:

- The manufacturer of RFID tags may be compromised or malicious.
- RFID tags are often not sufficiently tamper resistant. If the adversary is a user of the RFID system he can reverse engineer and obtain the key material of one of his own tags.

4.4.1 The EC-RAC IV protocol

In this section, we show an insider attack on EC-RAC IV based on the flaw described in Section 4.3. The flaw allows the adversary to modify the messages exchanged in a protocol without either of the agents noticing the change. In particular, the homomorphic properties of the messages allow the adversary to combine protocol messages into meaningful new messages.

The man-in-the-middle attack requires the adversary to eavesdrop on two protocol executions between a tag and a reader. Alternatively, the attacker could query a tag twice with random s and s'. These protocol executions produce the messages T_2, s, T_3 and T_2', s', T_3', where $T_3 = (b + h(s)x)Y$ and $T_3' = (b' + h(s')x')Y$. In order to decide whether $x = x'$ we follow Equation (4.2) and set $\alpha = (h(s)b' + h(s')b)P$ and $\beta = (h(s)b' + h(s')b)Y + h(s)h(s')(x' - x)Y$.

The man-in-the-middle attack described in Section 4.3 modifies the messages of a protocol execution between a legitimate tag and a reader as an oracle. We stress that such an attack is hard to perform in practice since it requires the adversary to intercept messages and modify them before they are forwarded to the communicating partner.

We now assume that the adversary has the key material of one legitimate tag in the system. We call this tag an *insider tag*. Using the key material of an insider tag, the adversary can run the protocol with a legitimate reader. Recall that terms α and β satisfy the following equation if and only if $x = x'$.

$$y\alpha = \beta \tag{4.4}$$

Let x'' be the secret of the insider tag, i.e. the adversary knows x''. To test whether Equation (4.4) holds, the adversary sends $T_2'' = \alpha$ to the reader. The reader responds with a challenge s''. The adversary responds with $T_3'' = \beta + (h(s'')x'')Y$. If the reader accepts the tag, the adversary knows that $x = x'$, otherwise $x \neq x'$. Figure 4.6 depicts the protocol flow between the adversary and the reader.

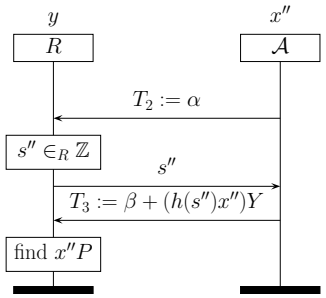


Figure 4.6: Insider attack on EC-RAC IV.

The insider attack is easier to execute than the man-in-the-middle attack in a setting where the adversary has insider tags. It does not require legitimate protocol messages to be captured and modified, but merely requires the attacker to interact with a tag twice, and later once with a legitimate reader.

The man-in-the-middle attack of the previous section can be easily prevented by introducing a message authentication code (MAC) based on a shared secret. Such a MAC would take as input the three messages of the protocol. Since the attacker does not know the shared secret of a legitimate tag, he would not be able to construct a valid MAC. Therefore, the man-in-the-middle attack would no longer be successful. The insider attack presented in this section would, however, not be prevented with this modification.

The messages of the other EC-RAC protocols [LBSV10a, LBV09, LBSV10b], the BCI protocol [BCI08], and the recently proposed hierarchical ECC-based proto-

col [BSSV11] possess homomorphic properties similar to the EC-RAC IV protocol. Therefore, an insider attacker can perform an attack along the lines of the attack described above. The latter two protocols were, however, proven untraceable in a model that does not assume insider attackers.

4.4.2 Protocols with IND-CCA1 encryption

In this section, we show that IND-CCA1 encryption is not sufficient to prevent insider attacks. Consider Vaudenay's protocol [Vau07, HPVP11] depicted in Figure 4.7. The protocol assumes that every pair of reader and tag share a secret key[1] k. The reader starts the protocol by sending a random challenge c to the tag. The tag combines the challenge with k and responds with the encryption of c and k under the public key of the reader. The reader decrypts this message with its public key and identifies and authenticates the tag based on the plaintext of the encryption.

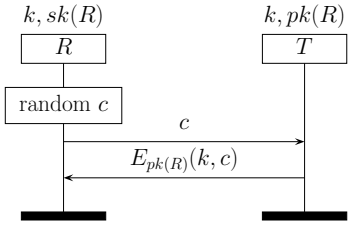


Figure 4.7: RFID protocol with IND-CCA1 encryption

We show that if a homomorphic IND-CCA1 encryption scheme is used, then the protocol is vulnerable to insider attacks. An encryption scheme is said to be homomorphic if the elements of the plaintext set and the ciphertext set form a group with operators $\otimes$ and $\oplus$, respectively, so that for any encryption key k and for any messages m_1 and m_2 the encryption function $\{\cdot\}_k$ satisfies $\{m_1\}_k \oplus \{m_2\}_k = \{m_1 \otimes m_2\}_k$. Examples of homomorphic encryption schemes are ElGamal [Gam85], DEG [Dam91], "lite" version of the Cramer-Shoup encryption scheme [CS98, Section 5.4].

Let $\{m\}_{pk}$ denote a homomorphic IND-CCA1 encryption of message m under key pk. By homomorphy of the encryption scheme, we have

$$\{k, c\}_{pk(R)} \oplus \{k', c'\}_{pk(R)} = \{(k, c) \otimes (k', c')\}_{pk(R)}.$$

To attack the protocol depicted in Figure 4.7, the adversary performs the following insider attack. Suppose tags T_1 and T_2 share secret keys k_1 and k_2 with the reader. Clearly, T_1 and T_2 are the same tag if $k_1 = k_2$. The attacker queries the two tags with the same challenge c. The tags return the ciphertexts $E_{pk(R)}(k_1, c)$ and $E_{pk(R)}(k_2, c)$, respectively.

By correctness of the protocol, the two observations concern the same tag if and only if $k_1 = k_2$. The adversary can test this by using his insider tag with key k_I

[1] This key represents the identity (ID) and the key (K) of the original proposal [Vau07, HPVP11].

and executing one protocol run with an RFID reader. Say, the reader's challenge is c''. The adversary encrypts $E_{pk(R)}(k_I, c'')$ and computes

$$\{k_1, c\}_{pk(R)} \oplus \{k_2, c\}^{-1}_{pk(R)} \oplus \{k_I, c''\}_{pk(R)} \\ = \{(k_1, c) \otimes (k_2, c)^{-1} \otimes (k_I, c'')\}_{pk(R)}.$$

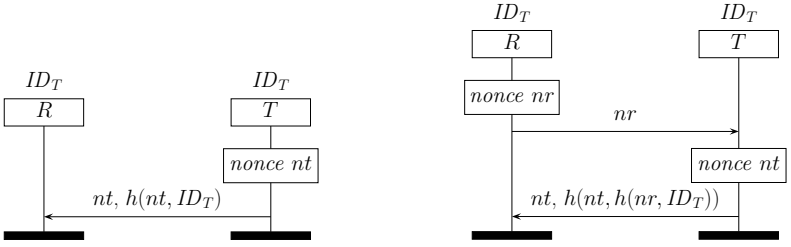
The reader accepts the adversary's response if $k_1 = k_2$ and rejects it otherwise. If the reader accepts the response, the adversary knows that $T_1 = T_2$, otherwise he knows that $T_1 \neq T_2$. Thus, the insider attack breaks the untraceability of the protocol. In the next chapter, we design a protocol based on IND-CCA2 encryption that resists insider attacks.

4.5 Compositionality attacks

A simple strategy to decrease the design and verification complexity is to construct protocols from smaller and simpler building blocks. It is then essential, however, to prove that these building blocks do not break each others' security properties. In fact, it is well known [HT96, KSW97, Can00, Can01, ACG+08] that protocols satisfying a security property when executed in isolation do not necessarily satisfy the same security property when they are executed in an environment containing other protocols. In particular, it has been shown that composition of secrecy-preserving protocols may introduce attacks [Cre06a]. Similar results have been obtained for the composition of authentication protocols [TH99].

We first demonstrate that the composition of two untraceable protocols is not necessarily untraceable. We then discuss the significance of this fact for RFID systems by showing two scenarios in which untraceability would be violated.

Consider the two protocols shown in Figure 4.8. The first protocol (A) is a tag identification protocol and the second protocol (B) is a tag authentication protocol. In both protocols we assume that a reader R and a tag T share a secret ID_T, not known to the adversary. The reader initiates the protocol by querying the tag. Then the tag generates a random number nt and sends its response to the reader.



(a) Protocol A (b) Protocol B

Figure 4.8: Protocols untraceable in isolation, not in a common environment.

If h is a cryptographic hash function and ID_T is not known to the adversary, each of the protocols can be shown to be untraceable in isolation. Let x, y, and z be any terms. For protocol A the following mapping π is a valid reinterpretation.

$$\begin{array}{lll} \pi(x, h(x, y)) = x, h(x, f) & & \textit{for fresh } f \in \textit{Agent} \\ \pi(x) = x & & \textit{elsewhere} \end{array}$$

For protocol B the following mapping π' is a valid reinterpretation.

$$\begin{array}{lll} \pi'(x, h(x, h(y, z)) = x, h(x, h(y, f)) & \textit{for fresh } f \in \textit{Agent} \\ \pi'(x) = x & \textit{elsewhere} \end{array}$$

For any trace (s, t) of protocol A, the trace $\pi((s, t))$ is indistinguishable from (s, t). Similarly, for any trace (s, t) of protocol B, the trace $\pi'((s, t))$ is indistinguishable from (s, t). Furthermore, all runs in the traces $\pi((s, t))$ and $\pi'((s, t))$ are executed by different agents. Thus, both protocols are untraceable. In the next section, we show that in a common environment the protocols do not satisfy untraceability.

Compositionality attack on protocols A and B.

We assume that tags can run both protocol A and protocol B and use the same identifier ID_T for both protocols. An attacker uses protocol A to build a database of tags he is interested in tracing. By querying a tag T, he obtains $nt, h(nt, ID_T)$ which he stores in the database. In order to test whether a random tag T' is equal to a particular tag T in his database, the attacker uses protocol B. He sends the challenge nt to the tag. In protocol B the tag answers with $nt', h(nt', h(nt, ID_{T'}))$. The attacker can then obviously determine whether $ID_T = ID_{T'}$ by computing $h(nt', h(nt, ID_T))$ and comparing it with $h(nt', h(nt, ID_{T'}))$.

There are at least two scenarios in which this type of attack can become a significant problem.

Chosen protocol attack. It is not uncommon for smart cards to implement a suite of protocols in order to host several applications. Therefore it is plausible that in the future RFID tags will host an implementation of several protocols or even protocol versions. Additionally, in the RFID setting, ownership transfer systems (Chapter 7) are frequently constructed by implementing several protocols on the RFID tag. In view of the compositionality attack, however, it is obvious that a tag which implements protocols A and B does not provide untraceability, in spite of the fact that both protocol A and B are untraceable in isolation.

Protocol revision attack. Consider an RFID-based system where a large number of RFID tags implementing protocol A have been deployed. Suppose the RFID tag's ID_T value is linked to a particular customer in any of several participating companies' databases. Since protocol A is untraceable, the RFID tag identifies the customer to an authorized entity, such as a retailer, a transportation company, or the local post office, but not to any entity the customer has not signed up with.

At a certain point in time it is decided that for future applications the identification protocol's security does not suffice, since its messages can be replayed. Protocol B is thus developed for applications which need to authenticate an RFID tag. To avoid the chosen protocol attack, customers will be provided with new RFID tags implementing protocol B, but not protocol A, and their old tags will be destroyed. For convenience and in order not having to update all the customer entries in all distributed databases, the new tags will use

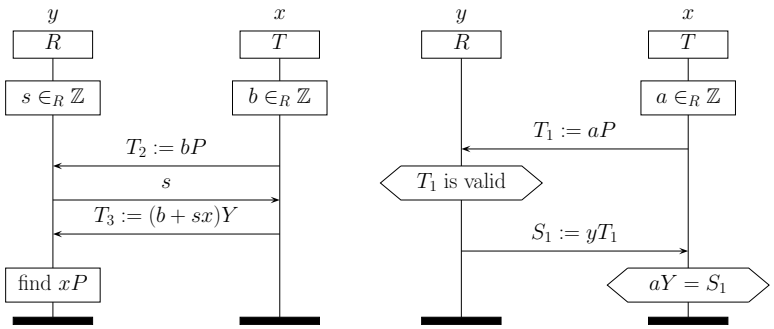
the same credentials as the old tags. In particular, the tag identity communicated by a customer's RFID tag remains the same for each customer. This way, each retailer merely needs to update the firmware of its RFID readers to communicate using protocol B.

The compositionality attack described above, however, still applies. Anybody interested in tracing customers merely needs to be near a customer's tag once before the customer's RFID tag is replaced. This suffices to record the tag's protocol A message. Long after the transition to new tags has been completed and all protocol A tags are destroyed, the message recorded from protocol A can still be used to test whether a tag implementing protocol B belongs to the previously observed customer.

4.5.1 The EC-RAC II protocols

The protocols in Figure 4.8 are specially crafted protocols, designed to show that untraceability is not a safely composable property and to illustrate the principle of using one protocol as an oracle to attack another protocol. In this section, we consider the protocols comprising EC-RAC II, proposed by Lee, Batina, and Verbauwhede [LBV09]. We use the same principle to show that some of the protocol compositions do not satisfy untraceability.

EC-RAC II consists of six protocols which are built from smaller building blocks. We only consider the first and fourth protocol of this set. The first building block, which we call protocol π_1, is identical to the first protocol in the set (see Figure 4.9a). A second building block, which we call protocol π_2, is the server authentication protocol shown in Figure 4.9b. We restrict our analysis to Protocol 4 of EC-RAC II, which is a composition of π_1 and π_2.



(a) Protocol π_1 (b) Protocol π_2

Figure 4.9: Building blocks of EC-RAC II

The protocols in EC-RAC II assume a group of the points on an elliptic curve. The publicly known point P is chosen from a subgroup of prime order of the points on the curve. The point yP can be considered as the RFID reader's public key, y being a scalar only known to the RFID reader. RFID tags store a secret scalar x. The corresponding public key is xP and is used by the reader to identify a tag.

Protocol π_1 follows a commitment-challenge-response structure. The tag begins by sending out a random point on the elliptic curve $T_2 = bP$ which serves as

a commitment. The RFID reader challenges the tag with a random integer s upon which the tag answers with a point $T_3 = (b + sx)Y$. The reader computes $(y^{-1}T_3 - T_2)s^{-1}$ which equals xP. The idea of such schemes is that anybody able to produce the correct response can also compute a particular secret. Thus successful completion of the protocol constitutes a proof of knowledge for the secret. Knowledge of the points xY allows an agent to authenticate itself as the tag whose public key is xP.

Protocol π_2 is designed to provide the tag with assurance of the authenticity of the reader. The tag initiates the protocol by challenging the reader with a random point $T_1 = aP$. The reader verifies whether the point is indeed a valid point on the elliptic curve of the same order as P. In order to prove knowledge of y, the reader responds with the scalar multiplication of y and T_1. The tag can verify the authenticity of the reader by verifying whether the received value is equal to aY.

Protocol 4 (Figure 4.10) is a composition of protocol π_1 and π_2. The first (and second) message is a concatenation of the first (and second) messages of π_1 and π_2. The third message is identical to the third message of protocol π_1. The protocol is claimed to provide tag authentication, reader authentication, and untraceability.

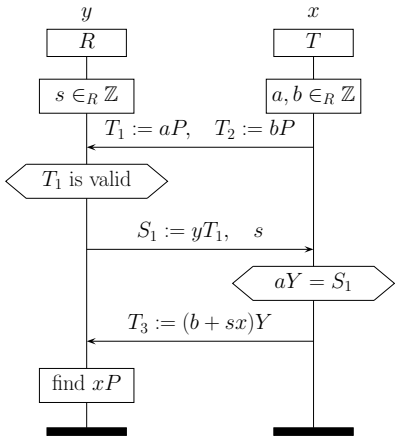


Figure 4.10: Protocol 4 of EC-RAC II

Untraceability attack

We show that there is a flaw in protocol 4 for which both building blocks of the protocol are used. That is, we use the challenge-response loop of protocol π_2 as an oracle for attacking the untraceability of protocol π_1.

The particular computation the oracle performs for the adversary is the multiplication of any nonzero point X by the reader's secret y. In the following we write this loop oracle as the function $O(X) = yX$. To use the loop O in protocol 4 for this purpose, the adversary sends a nonzero point $T_1 = X$, along with a random point T_2 to the reader. The reader replies with a random s and $yT_1 = yX$, the scalar multiplication of the reader's secret key y and the point X. The adversary then simply drops the connection to the reader, aborting the protocol.

Consider the messages bP, s, $(b + sx)Y$ an attacker learns from protocol 4 by eavesdropping on a communication between an RFID reader and a tag. In order to trace the tag, the attacker needs to be able to decide whether a tag presented to him is the same as the one he eavesdropped on earlier. By eavesdropping on another communication of a tag and reader (or by querying a tag himself) the attacker learns $b'P$, s', $(b' + s'x')Y$. He then computes

$$s(b' + s'x')Y - s'(b + sx)Y,$$

which can be rearranged as

$$(sb' - s'b)Y + ss'(x' - x)Y. \tag{4.5}$$

For $s, s' \neq 0$, the term in (4.5) is equal to $(sb' - s'b)Y$ if and only if $x = x'$, that is, if the tag being queried by the attacker later is the same tag as the one that was observed earlier. The attacker uses the oracle to decide whether this is the case or not by querying it with $sb'P - s'bP$: $O(sb'P - s'bP) = O((sb' - s'b)P) = (sb' - s'b)Y$. This equals the term in (4.5) if and only if the tag has been observed before. Thus, protocol 4 of EC-RAC II is not untraceable. It is worth noting that the attack only requires two eavesdropped sessions between a reader and a tag and one interaction with the reader. Alternatively, an attacker could twice query a tag and then have one interaction with a genuine reader.

The attack is a typical compositionality attack. It cannot be executed on either of the protocols π or π'. However, if π and π' are combined into protocol 4, the attacker has an efficient method for tracing tags.

4.6 Pseudonym-based attacks

One of the necessary conditions for untraceability is the ability for a tag to generate randomness. Without randomness in the tag's messages, the adversary can deduce whether messages were sent by the same tag. Since it is not always possible to implement a (pseudo) random number generator on a tag, protocol designers have tried to design protocols based on *pseudonyms*. Rather than using their identifier, tags use a pseudonym derived from their identifier in the protocol execution. After the successful execution of the protocol, both tag and reader update the pseudonym to a new value.

Pseudonym-based protocols are stateful since tags have a temporary identifier that is repeatedly updated. This opens the possibility for several types of untraceability attacks:

- **State-discovery attacks:** In stateful protocols, tags can be in different states. For instance, a tag can keep track of how many times it has been queried by a reader since the last pseudonym update. Depending on which state it is in, tags can respond differently to challenges. If this is the case, then the protocol must ensure that the adversary cannot discover in which state the tag is based on the tag response. If an adversary can discover tag

states, then he can attack the untraceability of one tag as follows. He first forces all tags to be in the same state. Then he chooses one tag T that he is interested in tracing and puts that tag in a different state. In a later stage, the adversary can recognize the one tag that is in a different state as tag T. The HMNB protocol covered in the previous chapter is a protocol that is vulnerable to this type of state-discovery attacks.

- **Pseudonym-update attacks:** Sometimes, pseudonym-based protocols are designed to satisfy a property that is weaker than untraceability. The adversary is not allowed to recognize a previously observed tag if between the two observations, the tag has communicated with a legitimate reader and has updated its pseudonym. We call this security property *serial untraceability*. Serial untraceability is strictly weaker than untraceability: any untraceable protocol is also serially untraceable, but not vice versa.

  Frequently, pseudonyms are updated by applying a function to the previous pseudonym and some terms shared between the reader and the tag. In some cases, the pseudonym update procedure is weak in the sense that the adversary can attribute messages from before and after the update to the same tag. In the next section, we show an example of a protocol in which the attacker can adapt his challenges across pseudonym-updates to let the tag output the same unique attribute.

- **Desynchronization-based attacks:** If both tag and reader carry out an update procedure for the pseudonym, then the protocol must ensure that both agents always carry out the same update. Tags and readers executing a protocol that does not satisfy this property can be desynchronized from one another, rendering future successful protocol execution impossible. We call protocols that do not suffer from this type of flaw *desynchronization resistant* and we formally define this property in Chapter 7. A side-effect of a desynchronization flaw is that an attacker can trace a tag by desynchronizing it from the reader. A desynchronized tag is always rejected by the reader. Thus, if the adversary can observe whether tags are accepted or rejected he can trace desynchronized tags. A simple strategy to trace a single tag T is to desynchronize it from the reader and to later eavesdrop on all communication between readers and tags. Assuming that the attacker did not desynchronize any other tag, a protocol execution that fails indicates that tag T was present. We show a desynchronization attack in Section 7.4.

4.6.1 Pseudonym-update attacks: The Li and Ding protocol

Li and Ding [LD07] proposed a pseudonym-based RFID protocol that is meant to provide serial untraceability. We call the protocol the LD protocol. The LD protocol was designed to be a mutual authentication protocol for re-writable RFID tags for supply chains. In the context of the LD protocol, a supply chain is assumed to consist of a predetermined and ordered sequence of partners, each of which is represented by a reader. Every such reader R_i contains its own secret k_i as well as the secret k_{i+1} of the next reader. Additionally, every reader stores the identity c of every tag it may authenticate. Every tag T contains a pseudonym α representing

its current temporary identity. The value of α is equal to $c \oplus k_i$ where k_i is the secret of some reader R_i currently allowed to identify and authenticate the tag.

The LD protocol is a challenge-response based protocol. The reader R_i challenges tag T with a nonce r. The tag calculates the *xor* of its current secret α and challenge r and responds with the hash of this value. The reader considers the tag authentic if it finds a secret c for which $h(c \oplus r \oplus k_i)$ is equal to the received response. The reader may then stop the protocol execution giving it the possibility to authenticate the tag again in a future communication session. Alternatively, the reader may send the update $a = k_i \oplus k_{i+1}$, accompanied by $b = h(a \oplus c \oplus k_i)$ to the tag. The tag then verifies that $b = h(a \oplus \alpha)$ and updates its secret α by replacing it with the *xor* of a and α. In doing this, ownership of tag T is transferred from reader R_i to reader R_{i+1}. The secret c can be understood as the identifier of the tag, while $c \oplus k_i$ is the i-th pseudonym of the tag. The protocol is depicted as a message sequence chart in Figure 4.11.

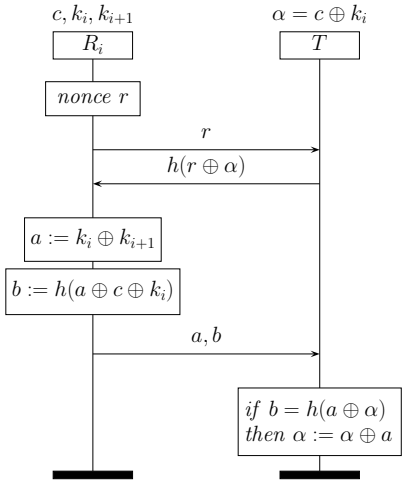


Figure 4.11: The Li and Ding protocol.

The LD protocol has not been designed to be untraceable, but merely serially untraceable[2]. Indeed, the fact that a tag does not introduce any randomness in its response to a reader's query implies that the tag is traceable between key updates. In the following we show, however, that the protocol does not provide serial untraceability either by exhibiting an attack on this property.

To show that the protocol does not satisfy serial untraceability, it suffices to exhibit a scenario in which the adversary recognizes a previously observed tag after the tag has updated its secret α.

By eavesdropping on a valid authentication session between a tag and a reader, the adversary learns $r, h(r \oplus \alpha), a$, and b. At the end of its run, the tag updates its secret α by replacing it with $\alpha \oplus a$. The adversary can now challenge the tag with $r' = r \oplus a$, to which the tag responds with $h(r' \oplus \alpha')$. By a simple algebraic property of *xor* given in Equation (4.1), the response is equal to the previously observed one:

$$h(r' \oplus \alpha') = h(r \oplus a \oplus \alpha \oplus a) = h(r \oplus \alpha). \tag{4.6}$$

[2]Serial untraceability is called unlinkability by Li and Ding [LD07].

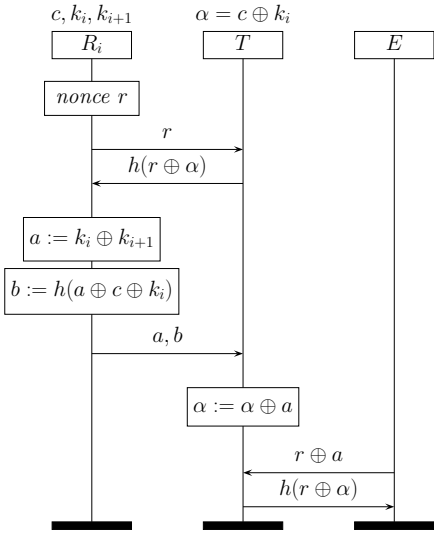


Figure 4.12: Attack on the LD protocol

The attack is depicted in Figure 4.12.

4.7 Related work

4.7.1 Practical attacks

There exist several practical attacks on RFID protocols. Most attacks focus on retrieving cryptographic keys from an RFID tag. These keys can then be used to break authentication.

One of the most widely used RFID tags is the MIFARE classic designed by NXP. The tags are mainly used in payment for public transportation and access control for buildings. The MIFARE protocol relies on the CRYPTO1 cipher of which the design was kept secret. Nohl and Plotz partially reverse engineered CRYPTO1 through hardware analysis. Garcia, De Koning Gans, Muijrers, Van Rossum, Verdult, Whichers Schreur, and Jacobs [GdKGM+08] showed an attack on the protocol that requires an attacker to eavesdrop on two protocol executions between a tag and a reader. Garcia, Van Rossum, Verdult, and Wichers Schreur [GvRVS09] later showed that the cryptographic keys can be recovered if an attacker is only given access to the tag for a short period of time.

SecureMemory, CryptoMemory, and CryptoRF are chips that belongs to the Atmel product family. They are used in smart cards (e.g. access cards and loyalty cards) and embedded in hardware (e.g. print cartridges and set top boxes). SecureMemory and CryptoMemory have a contact interface and CryptoRF has a wireless interface. Their design is based on a proprietary cipher. Garcia, Van Rossum, Verdult, and Wichers Schreur [GvRVS10] reverse engineered the cipher. They have also shown practical attacks that can be executed by an eavesdropping adversary.

Since 2004, many nations have been issuing e-passports with RFID tags embedded in them. Chothia and Smirnov [CS10] demonstrated that the RFID protocol implemented for basic access control is traceable. The protocol is initiated by a tag sending a nonce to the reader. The reader then responds with an encryption and

a MAC of that encryption. The MAC uses a shared secret between the e-passport and the reader as key. The tag first verifies the MAC and then the encryption. If one of these verifications fails, the tag immediately halts and outputs an error message. The attacker can distinguish the two failures by measuring the time it takes the tag to output the error message: a quick response means that the MAC verification failed and a slow response means that the MAC verification succeeded and the verification of the encryption failed. The attack by Chothia and Smirnov works as follows. An attacker eavesdrops and records a valid reader-to-tag message. It later replays this message to the tag. If a quick response follows, the MAC verification succeeded and, therefore, the tag is the same as the one of the eavesdropped message. If a slow response follows the tag is different.

KeeLoq IFF (identify friend or foe) is an RFID protocol that is used in wireless car keys. The protocol is initiated by the reader by sending a random challenge to the tag. The tag encrypts the message with the KeeLoq cipher under a 64-bit key shared with the reader. Bogdanov [Bog07] and Indesteege, Keller, Dunkelman, Biham, and Preneel [IKD+08] have shown practical attacks on the KeeLoq cipher if the adversary has access to an RFID tag for an extended period of time.

4.7.2 Impossibility results

Damgård and Pedersen [DP08] prove an interesting result concerning untraceability and authentication of symmetric-key RFID protocols. Their adversary model essentially allows corruption of all tags except the one that the adversary wants to trace[3]. They prove that if a protocol is sound and complete[4], there cannot be an efficient reader lookup procedure. In particular, the reader has to access the keys of all tags to find out with which tag he is communicating. The main idea behind the proof is the intuition that there can only be an efficient lookup procedure if the reader knows in advance which tag he is communicating with. If that were the case then the adversary would also know this leading to an attack on untraceability.

The model used to prove the result defines untraceability as the ability to distinguish two non-corrupted tags. One of those tags is chosen at random and called the target tag. The adversary breaks untraceability if he can guess with non-negligible probability which of the tags was chosen. The proof starts by showing that if keys are independently assigned to tags, the reader cannot determine if it is communicating with a tag unless it actually has to access the key for that tag. The adversary uses the following strategy to break untraceability. He queries the target tag and performs the reader lookup procedure himself. If he needs access to the key material of a tag he corrupts that tag. If one of the two non-corrupted tags' keys is required, he guesses that that tag is the target tag.

[3]It thus corresponds to our notion of insider attackers (see next chapter).

[4]Intuitively, a protocol is complete if legitimate tags are always accepted by the reader and sound if illegitimate tags are never accepted by the reader. Formal definitions for both properties will be treated in the next chapter.

4.8 Conclusion

There are various reasons why RFID protocols fail to satisfy their untraceability requirement. We have identified five categories of untraceability flaws. In an attribute acquisition attack the attacker interacts with tags and reader to derive a unique attribute for that tag. Man-in-the-middle attacks take advantage of the fact that in RFID protocols the attacker may have the possibility to observe whether a reader accepts a session with the tag as valid or not. Insider attacks requires a laboratory setting so that the attacker can obtain the key material of one tag. He then uses this key material to trace other tags. Compositionality attacks take advantage of the fact that several protocols use the same key material. The messages of one protocol are used to break the security property of another protocol. Finally, pseudonym-based attacks exploit the attacker's knowledge of the state of a tag to break untraceability. For each of the five categories, we have shown one or more new attacks on RFID protocols proposed in literature.

5

Untraceability proof models

In the previous chapter we have seen that for various reasons, designing untraceable protocols is not trivial. Among these reasons are the resource limitation of RFID tags and theoretical results of impossibility showing trade-offs between efficiency, soundness, and untraceability of an RFID protocol. Another reason for the abundance of flawed protocols is the absence of a common definition of untraceability. Intuitively, untraceability is defined as the inability of an attacker to recognize an RFID tag with which he has previously communicated. However, the formalization of this intuition is tricky and has been carried out in different ways. A protocol may thus be considered untraceable by one definition, yet traceable by another.

To verify whether a communication protocol satisfies a security property, such as untraceability, one creates a model which specifies what powers an adversary is given, how the adversary interacts with his environment, and what the definition of the security property within the model is. Proving a protocol correct in such a model should guarantee that a real-world attacker with equal powers is not able to invalidate the modeled security property.

There are two types of models to prove cryptographic protocols secure. The *symbolic protocol analysis* approach considers protocol messages on a high abstraction level and misses implementation details, but is therefore amenable to automation. The model designed in Chapter 3 is an example of the symbolic protocol analysis approach. The *computational approach* defines the security of a protocol by means of a game played by an adversary. A protocol is secure if the adversary cannot win the game. The standard way of proving security of a protocol is by reducing the problem of winning the game to solving a mathematically hard problem. That is, one proves that any adversary that can win the game can be used as a subroutine of an algorithm that can solve a problem that is known to be hard. Compared to the symbolic protocol analysis approach, the computational approach is more difficult and prone to error due to the necessity of manual proofs. However, it considers more implementation details and is therefore more widely applicable.

In this chapter we investigate computational proof models for untraceability. In particular, the chapter is separated into the following four parts.

- Computational proof models for untraceability separate into two categories. *Indistinguishability*-based definitions define untraceability as the inability of the attacker to distinguish two tags. *Unpredictability*-based definitions require the adversary to be unable to distinguish the behavior from something random. We show that two unpredictability-based proof models [HMZH08, NSMSN08] do not coincide with the intuitive notion of untraceability.

- Vaudenay [Vau07] proposed an indistinguishability-based model with 8 different classes of adversarial powers. We show that insider attackers cannot be naturally represented in the model and extend the model with the class of insider attackers. We then show that this is a non-empty class of adversaries by proving separation with the other classes in the model.

- Ng, Susilo, Mu, and Safavi-Naini [NSMSN08] suggested simplifications to the model by Vaudenay. They claimed that under certain assumptions, some of the adversary classes collapse and become identical. We show that these simplifications are incorrect. To build a formally sound argument, we first show a construction that turns a three-message ping-pong protocol π into a protocol ρ that satisfies authentication while maintaining untraceability. We then apply this construction to a protocol from literature to show that the claims by Ng et al. are incorrect.

- Finally, we design a provably untraceable and authenticating RFID protocol exclusively based on elliptic-curve operations.

5.1 Computational proof models

The first formal proof model for location privacy in the RFID setting is due to Avoine [Avo05]. In Avoine's model the adversary is a probabilistic polynomially-bounded Turing machine (PPTM), which interacts with RFID tags and readers through oracles. There is an oracle for communication with the reader, an oracle for communication with tags, and an oracle that gives the adversary access only to the messages sent from a reader to the tag. The last oracle is motivated by the fact that in practice eavesdropping on reader-to-tag messages is much easier than on tag-to-reader messages.

Finally, there is an oracle modeling the physical compromise of a tag by giving the attacker access to the internal state of the tag. After it is queried, the adversary is not allowed to further query the other oracles. The strength of the adversary is modeled by selecting a subset of the available oracles. Untraceability is defined through an experiment in which the adversary is first given access to a target tag. Then the adversary is given access to two tags, of which one is the target tag. The adversary wins if he correctly guesses which of the two tags is the target tag. The protocol is said to be untraceable if the adversary has no non-negligible advantage of correctly guessing the target tag compared to random guesses. Avoine separates untraceability into *existential* and *universal* untraceability. An existentially untraceable protocol allows the adversary to trace a tag for a restricted period of time, while a universally untraceable protocol does not.

Juels and Weis [JW09] extend Avoine's model by providing a slightly stronger definition of untraceability. In their proposal, tags and readers are probabilistic Turing machines modeled as ideal functionalities resembling the equally named interactive Turing machines in the universal composability paradigm by Canetti [Can01]. The tag and reader functionalities each have several interfaces which can be addressed by sending a particular message to the functionality. The adversary has access to these interfaces and controls the channel between all the functionalities. Untrace-

ability (called RFID privacy in [JW09]) is defined through a privacy experiment in which the adversary may interact with all tag functionalities and may compromise all but two tag functionalities. Two of the uncompromised tag functionalities are selected by the adversary. One of them, say T, is chosen at random and the adversary's advantage in guessing which functionality was chosen decides whether the protocol satisfies the privacy property. In order to make a guess about T, the adversary is permitted to interact with T. Additionally, the adversary is permitted to interact with and compromise all but the two selected tag functionalities in the system's environment. It is due to this last fact that the Juels–Weis adversary is stronger than Avoine's adversary.

Vaudenay [Vau07] proposes a more flexible, hierarchical model for location privacy. His model captures eight classes of adversary capabilities ranging over four different types of tag corruption and two modes of observation. An adversary is a PPTM whose strength is defined by the set of oracles it is allowed to query. A *weak* adversary is never allowed to corrupt a tag, that is, he may never query the *corrupt* oracle. A *forward* private adversary may corrupt a tag at the end of the attack, a *destructive* adversary may corrupt a tag at any time, which leads to the destruction of the tag, that is, the adversary may no longer interact with the tag. A *strong* adversary may corrupt a tag at any time without destroying it. Corresponding to the two modes of observation, an adversary is called *wide* if he may observe whether the protocol ended successfully, and *narrow* else. Since the four types of corruption are orthogonal to the narrow/wide separation, eight different adversarial classes are considered. Privacy is defined by comparing the adversary to a special adversary which makes no use of protocol messages, as follows. An adversary is called *blinded* if he is not allowed to communicate with tags and reader. An adversary is *trivial* if there exists a blinded adversary which essentially performs equally well at guessing a tag's identity. More precisely, the trivial adversary has an at most negligible advantage of guessing a tag's ID over the blinded adversary. A protocol is *P-private*, where P is one of the eight adversary classes, if all adversaries that belong to that class are trivial. We give a more elaborate description of the model in Section 5.3.1. Since it may corrupt tags, the Juels-Weis adversary is stronger than the wide-weak adversary of Vaudenay. The Juels-Weis adversary is weaker than the wide-strong adversary, since the wide-strong adversary may even corrupt the target tag. Paise and Vaudenay [PV08] extended the model with the notion of reader authentication.

Ng, Susilo, Mu, and Savavi-Naini [NSMSN08] extend Vaudenay's work by showing that under certain assumptions, the eight adversary classes can be reduced to three. Their first assumption is that authentication protocols do not produce "false-negatives". The second assumption is that tags do not share key material. In Section 5.4 we invalidate the results by Ng et al. by showing that both assumptions are unrealistic and do not hold in the general case.

Ng, Susilo, Mu and Safavi-Naini [NSMSN09] extended their earlier work by categorizing different types of symmetric key protocols. They argue that none of the categories can achieve narrow-forward and wide-weak privacy simultaneously. As a result no symmetric key protocol can be wide-forward private.

5.2 Unpredictability-based proof models

Computational proof models typically define untraceability as the ability of the attacker to distinguish a *target* tag from some other functionality. Ma, Li, Deng, and Li [MLDL09] suggest a classification of computational proof models for untraceability in two classes. Their separation is based on the adversary's goal in the experiment of the proof model.

- In *Indistinguishability*-based proof models the adversary wins the experiment if he can distinguish the target tag from some *other* tag. The main idea is that if the attacker cannot observe any difference between the messages of the target tag and some other tag, then he cannot trace the target tag. The proof models discussed in the previous section are all considered to belong to this class. The formal-languages based model of Chapter 3 follows the same approach.
- In *Unpredictability*-based proof models the adversary wins the experiment if he can distinguish the target tag from something random. The main idea is that if the attacker cannot predict the messages of the target tag, then he cannot trace it. The proof models by Ha, Moon, Zhou, and Ha [HMZH08], Ma, Li, Deng, and Li [MLDL09], and Lai, Deng, and Li [LDL10] belong to this class.

In this section, we consider the unpredictability-based proof models by Ha et al. and Ma et al. In particular, we construct protocols that are intuitively (and by all indistinguishability-based models) considered untraceable, but are traceable in these unpredictability-based models. We also construct protocols that are intuitively (and by all indistinguishability-based models) considered traceable, but are untraceable in these unpredictability-based models.

Untraceability is called differently depending on the authors of papers. Since our goal is to compare different proof models, we use the terminology originally used by the authors, whenever possible.

5.2.1 The model by Ha, Moon, Zhou, and Ha

We briefly outline the unpredictability-based proof model of Ha, Moon, Zhou, and Ha. The reader is referred to the original paper [HMZH08] for full details. The HMZH model defines two attack games: one for indistinguishability and one for forward secrecy. We restrict our analysis to the authors' definition of *weak location privacy*, allowing us to only focus on the indistinguishability game. However, similar results can be obtained when considering the authors' notion of strong location privacy.

The indistinguishability attack game consists of three phases. In the initialization phase, tags are created and the RFID system's database is populated. In the learning phase, the adversary may, depending on his capabilities, query a set of oracles allowing him to interact with tags and database. In the challenge phase, the adversary chooses a target tag T and may again query a set of oracles. Additionally,

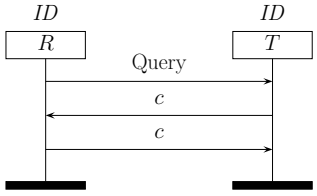


Figure 5.1: Constant-response protocol

he may query the *reveal*-oracle that reveals the contents of the tag for every tag except T. At the end of the challenge phase, the adversary calls the *test*-oracle. The test oracle tosses a fair coin b.

- If $b = 1$: the message that T would send after being queried is given to the adversary.
- If $b = 0$: a random value of the same bit length as T's messages is given to the adversary.

It is then the adversary's task to guess the value of b. The adversary wins the game if his guess is correct.

Note that the test oracle does not provide a full transcript of the protocol execution between the reader and the tag, but only a random message or the message the tag would send to the reader.

The protocol is defined to be *weakly location private* if the adversary does not have a non-negligible advantage of winning the indistinguishability game.

Constant-response protocol

Our first example is a protocol that is intuitively and by the notions of [Avo05, Vau07, JW09] and our definition in Section 3.5 untraceable, but which can be proven not to be weak location private in the HMZH model.

The protocol description is as follows. The reader R and tag T share a secret ID. The term c is a public, system-wide constant. The protocol (Figure 5.1) starts by R querying T for a response. The tag T responds with the constant c, after which R sends c back to the tag. We emphasize that every tag responds with the same constant c. For simplicity, we omit the communication between reader and database, since it is assumed to be secure.

The protocol is intuitively untraceable since every tag responds with the same message. In fact, the tags in this protocol could even be identically built and do not need to have an ID. Thus, regardless of his behavior, it is not possible for the adversary to recognize a tag he previously observed.

It is easy to give a proof for untraceability in any of the proof models cited above.

We now use the HMZH model to prove that the protocol is not location private.

Lemma 5.1. *The constant-response protocol (Figure 5.1) does not satisfy weak location privacy for an active adversary.*

Proof. The adversary's strategy is as follows. He does not query any oracle during the learning phase. In the challenge phase, he selects one of the tags at random, and he only queries the *Test*-oracle, in order to obtain an answer x. The adversary guesses $b = 1$ if $x = c$, and $b = 0$ otherwise.

The adversary wins this game with probability $1-2^{-k-1}$, where k is the bit length of the constant. He thus has a non-negligible advantage to win the game. Therefore, the constant-response protocol does not satisfy weak location privacy in the sense of [HMZH08]. □

This example shows a weakness in the indistinguishability game of [HMZH08]. At the end of the challenge phase, the adversary must be able to distinguish a tag's response from a random value. The set of possible tag answers is not considered in the game. Intuitively, what matters for untraceability is that the adversary must not be able to distinguish one tag's response from other tags' responses, but not necessarily that the adversary cannot distinguish the tag's response from any arbitrary value.

The flaw in the model extends to all protocols in which the tag responds with a cryptographic hash or encryption that is not indistinguishable from a random bit string of equal length. Since in general, pseudorandom output has not been an explicit requirement for hash functions, it is not safe to assume that the outputs *are* indistinguishable from random bitstrings. For instance, it is obvious that the output of hash functions whose range are points on an elliptic curve [Ica09] can be distinguished from random bit strings. Furthermore, there are proposals for computationally "light-weight" RFID protocols emulating public-key encryption, such as the EC-RAC protocol described in the previous chapter, where messages communicated between reader and tag are constructed from points on an elliptic curve. This class of protocols can also be expected to become more numerous in the near future. The HMZH model, however, would not be adequate to prove untraceability for these protocols, since their messages can be distinguished from random bit strings.

Plaintext-ID protocol

Our second example concerns a protocol that is intuitively and by the notions of [Avo05, Vau07, JW09] and our definition (Section 3.5) not untraceable, but can be proven weak location private with respect to the HMZH model.

We assume that a legitimate reader R knows the *ID*s of all tags in the system. Aside from the reader, nobody except for a tag itself knows the tag's *ID*. Let $pk(R)$ be a reader's public key with corresponding private key $sk(R)$ and let $\{m\}_{pk(R)}$ denote an IND-CCA2 public-key encryption of m with the public key $pk(R)$. We further assume that the encryption scheme has the *ciphertext pseudo-randomness* property, such as the scheme proposed by Möller [Möl04] which makes ciphertexts indistinguishable from pseudorandom strings of equal length. Figure 5.2 depicts the protocol.

It is easy to see that the protocol is not untraceable, since the identity *ID* of the tag is transmitted in the clear in every execution of the protocol. Thus even a passive adversary, namely one which merely observes messages, can trace tags.

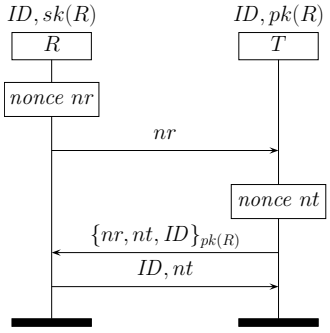


Figure 5.2: Plaintext-ID protocol

Since all the adversaries considered in the models cited above are at least as strong as a passive adversary, this protocol is not untraceable in any of the corresponding models.

We now use the proposed model to prove that the protocol *is* weak location private.

Lemma 5.2. *Protocol 2 satisfies weak location privacy for a passive adversary.*

Proof. In the learning phase, the adversary may query the *execute*-oracle to build a list of tuples $(nr, \{nr, nt, ID\}_{pk(R)}, ID, nt)$, corresponding to observed communications.

In the challenge phase, the adversary selects a challenge tag T and queries the *reveal*-oracle for all tags except T. He further extends his list of tuples $(nr, \{nr, nt, ID\}_{pk(R)}, ID, nt)$ by querying the *execute*-oracle.

Finally, the adversary queries the *Test*-oracle on T. The oracle tosses a fair coin b and

- for $b = 1$ it outputs the message $\{nr, nt, ID\}_{pk(R)}$,
- for $b = 0$ it outputs a random value of the same length as the second protocol message.

The adversary has to guess the bit b. If the adversary were able to guess this bit with a non-negligible advantage, then he could distinguish $\{nr, nt, ID\}_{pk(R)}$ from a random value with a non-negligible advantage. But this would contradict the ciphertext pseudo-randomness assumption on the encryption scheme. □

This example shows a weakness in the challenge phase of the indistinguishability game. Before calling the *test* oracle, the adversary has full access to all messages sent by the tag and reader. But once he calls the *test* oracle, his capabilities are limited, in that the messages sent from the reader to the tag are not given to him and he must make a decision based on a single message of the tag. Thus he is not allowed to use information that in standard models would be available to a passive adversary.

Note that if the reader would, in the third message, additionally transmit sufficient information for the adversary to be able to *verify* whether the encryption in message two is indeed an encryption of ID, then the proof would still go through, while untraceability would seem even less plausible.

5.2.2 The model by Ma, Li, Deng, and Li

The unpredictability-based proof model by Ma, Li, Deng, and Li [MLDL09] was designed to overcome the weaknesses of the HMZH model. We call the model MLDL after the last names of the authors. The adversary model consists of four oracles modeling capabilities of the intruder. Central to the model are the *experiments* in which the adversary interacts with the oracles. An RFID protocol satisfies a security property if the adversary has at most a negligible chance of performing the experiment successfully.

We give a simplified, yet for our purposes sufficient, description of the model. For more details on the model, refer to [MLDL09].

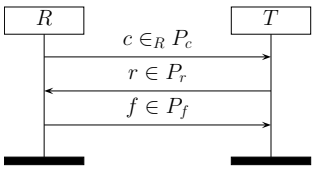


Figure 5.3: Canonical RFID protocol of the MLDL model

Ma et al. observe that the majority of proposed RFID protocols follow a three-message ping-pong structure initiated by the reader. They restrict their proof model and analysis to the type of protocols captured by their *canonical RFID protocol* shown in Figure 5.3. The canonical RFID protocol is initiated by the reader R choosing a challenge c at random from the set of challenges P_c. A tag T, chosen from the set $\mathcal{T}$, computes the message $r \in P_r$ and sends it to the reader. The reader computes $f \in P_f$ and sends it to the tag. This last message is optional. We set S to be the set of session identifiers and M to be the set of all messages.

An adversary is modeled by a probabilistic polynomial-time interactive Turing machine. The capabilities of adversaries are modeled by the oracles they are allowed to query. There are oracles for starting a session with a reader, starting a session with a tag, and an oracle to send the tag response to the reader. The physical compromise of a tag is modeled by an oracle that returns the cryptographic key(s) of a tag. The oracles are formally defined as follows.

- InitReader: $S \times P_c$ invokes the reader R to start a session of the protocol. It returns a fresh session identifier taken from S and a challenge message $c \in_R P_c$.
- InitTag: $\mathcal{T} \times S \times P_c \to S \times P_r$ invokes the tag $T \in \mathcal{T}$ to start a session of the protocol with session identifier $sid \in S$. The message $c \in P_c$ to be read by the tag is given as input. The tag responds with a message $r \in P_r$ and the session identifier sid.
- SetTag: $M \times \mathcal{T} \to M$ updates the key and state information of tag $T \in \mathcal{T}$ to a new value $m \in M$ and returns the tag's current key and state information $m' \in M$.
- SendRes: $S \times P_c \times P_r \to P_f$ returns the challenge and the response messages $c \in P_c$ and $r \in P_r$ to the reader's session sid and returns the reader's final message $f \in P_f$.

In the remainder of this section, the oracles O_1, O_2, O_3, and O_4 denote the InitReader, InitTag, SetTag, and SendRes respectively.

The model specifies four (security) properties: completeness, soundness, indistinguishability, and unpredictability. *Completeness* is a functional requirement that ensures that the protocol is well-specified. It requires a legitimate tag to always be accepted by the reader if the adversary merely forwards the messages between reader and tag.

Definition 5.3 (Completeness [MLDL09])**.** *Let (c, r, f) be the protocol messages as observed by the reader in session sid with tag T and let r be generated by T. A protocol satisfies completeness if for any such session, the reader outputs "accept".*

Experiment $\mathbf{Exp}^{\text{Sound}}_{\mathcal{A}}$
1. setup the reader R and a set of tags $\mathcal{T}$
2. $\{(c_{sid}, r_{sid}, f_{sid}), T\} \leftarrow \mathcal{A}^{O_1,O_2,O_4}(R, \mathcal{T})$

Figure 5.4: Soundness experiment

A protocol satisfies *soundness* if illegitimate tags are always rejected by the legitimate reader R. More specifically, soundness requires that if at the end of a protocol execution reader R accepts tag T, then T is a legitimate tag and generated the message r received by R in that session. This property is formalized by the experiment in Figure 5.4. Let $x \leftarrow \mathcal{A}^{O_1,O_2,O_4}(R, \mathcal{T})$ denote the algorithm $\mathcal{A}$ with access to oracles O_1, O_2, O_4, which upon input $(R, \mathcal{T})$, returns x. The adversarial algorithm interacts with oracles O_1, O_2, and O_4 polynomially many times. At the end of the experiment, the algorithm outputs a session between a reader and a tag. The adversary wins the experiment if at the end of this session, the reader accepted the tag while it was not legitimate or the tag's message was modified.

Definition 5.4 (Soundness [MLDL09])**.** *Let E denote the event that in session sid with protocol messages $(c_{sid}, r_{sid}, f_{sid})$, T was accepted by the reader, but T did not send r_{sid}. A protocol satisfies soundness if for any adversary $\mathcal{A}$ the probability of E occurring is negligible.*

Experiment $\mathbf{Exp}^{\text{ind}}_{\mathcal{A}}$
1. setup the reader R and a set of tags $\mathcal{T}$
2. $\{T_i, T_j, st\} \leftarrow \mathcal{A}_1^{O_1,O_2,O_4}(R, \mathcal{T})$
3. set $\mathcal{T}' = \mathcal{T} \backslash \{T_i, T_j\}$
4. $b \in_R \{0, 1\}$
5. if $b = 0$ then $T_c = T_i$ else $T_c = T_j$
6. $b' \leftarrow \mathcal{A}_2^{O_1,O_2,O_4}(R, \mathcal{T}', st, T_c)$
7. if $b = b'$ output 1 else output 0

Figure 5.5: Indistinguishability experiment

Ma et al. specify two definitions of untraceability: an indistinguishability-based definition and an unpredictability-based definition. Their first definition, called

indistinguishability, defines untraceability as the ability of the adversary to distinguish two tags. The corresponding experiment (Figure 5.5) consists of two phases. In the *learning phase* the adversary interacts with tags and reader polynomially many times by querying the oracles. At the end of this phase, he selects two target tags T_i and T_j. The experiment randomly chooses one of these tags as the challenge tag. In the *challenge phase*, the adversary may again interact with all oracles. He is given access to the challenge tag, but not to the other tag. At the end of the experiment, the adversary guesses which of T_i and T_j was chosen as challenge tag. A protocol is *indistinguishable* if the adversary cannot perform better at the experiment than random guessing.

Definition 5.5 (Indistinguishability [MLDL09])**.** *An adversary $\mathcal{A}$ wins if the indistinguishability experiment $\boldsymbol{Exp}_{\mathcal{A}}^{ind}$ outputs* 1*. An RFID protocol satisfies* indistinguishability *if no adversary can win the indistinguishability experiment with probability $\frac{1}{2} + \epsilon$ for non-negligible ϵ.*

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{unp}}$
1. setup the reader R and a set of tags $\mathcal{T}$
2. $\{T_c, c, st\} \leftarrow \mathcal{A}_1^{O_1,O_2,O_4}(R, \mathcal{T})$
3. set $\mathcal{T}' = \mathcal{T} \backslash \{T_c\}$
4. $b \in_R \{0, 1\}$
5. if $b = 0$ then $(r^*, f^*) \in_R P_r \times P_f$
else $(c, r, f) \leftarrow \pi(R, T_c, sid)$ and $(r^*, f^*) = (r, f)$
6. $b' \leftarrow \mathcal{A}_2^{O_1,O_2,O_4}(R, \mathcal{T}', st, r^*, f^*)$
7. if $b = b'$ output 1 else output 0

Figure 5.6: Unpredictability experiment

The second definition of untraceability is called *unpredictability* and specifies that the adversary cannot distinguish the protocol messages of one tag from random messages. The *learning phase* is similar to that of the indistinguishability game. At the end of the phase, the adversary outputs one target tag T_c and a challenge c. The experiment then tosses a fair coin b. If $b = 0$ it gives the adversary random messages $r \in_R P_r$ and $f \in_R P_f$. If $b = 1$, the experiment runs a protocol execution between R and T_c with challenge c (denoted by $(c, r, f) \leftarrow \pi(R, T_c, sid)$). The messages r and f from that session are given to the adversary. In the *challenge phase* the adversary may again interact with all oracles. He is not given access to T_c. At the end of the experiment, the adversary guesses the value of b. The experiment is depicted in Figure 5.6. A protocol is *unpredictable* if the adversary cannot perform better at the experiment than random guessing.

Definition 5.6 (Unpredictability [MLDL09])**.** *An adversary $\mathcal{A}$ wins if the unpredictability experiment $\boldsymbol{Exp}_{\mathcal{A}}^{unp}$ outputs* 1*. An RFID protocol satisfies* unpredictability *if no adversary can win the unpredictability experiment with probability $\frac{1}{2} + \epsilon$ for non-negligible ϵ.*

Unpredictability and indistinguishability are two related, but different definitions of untraceability. In the original paper, it was proven [MLDL09, Theorem 3] that for

complete and sound protocols, unpredictability implies indistinguishability. Furthermore, a proof is given [MLDL09, Theorem 4] that there exist protocols that are indistinguishable, but not unpredictable. Thus, unpredictability is claimed to be strictly stronger than indistinguishability. Finally, the authors show that as a minimal requirement for unpredictability, it is necessary that the tag's computational capabilities must be equal or stronger than those necessary to compute a pseudo-random function. The main result of the paper is that instead of proving indistinguishability, one can prove the stronger (but possibly easier to prove) property of unpredictability.

We now show that the proof of [MLDL09, Theorem 4] is incorrect. To show that the theorem still holds we provide an alternative proof. We also invalidate [MLDL09, Theorem 3] by showing that there exist protocols that are unpredictable, but not indistinguishable. By combining both proofs, we show that the definitions of unpredictability and indistinguishability are incomparable, that is, neither of them is stronger than the other.

Echo protocol

The first claim by Ma et al. is that indistinguishability does not imply unpredictability [MLDL09, Theorem 4]. We restate their theorem below (Theorem 5.7). The proof consists of the following counter-example. Let π be an indistinguishable protocol with messages $c \in P_c$, $r \in P_r$, and $f \in P_f$. Let π' be like π, but the second message is concatenated with itself: $r' = (r, r) \in P'_r$. The proof states that π' is not unpredictable since the adversary can easily distinguish a protocol message of the form $(c, (r, r), f) \in P_c \times P_r^2 \times P_f$ from a random tuple $(c, (r_1, r_2), f) \in P_c \times P_r^2 \times P_f$ by checking whether $r_1 = r_2$.

Unfortunately, the last observation is not sufficient for the proof. By changing the protocol from π to π' the authors assume the response set P_r changes to $P'_r = P_r \times P_r$. This is not true, since the only valid responses are the ones where a value from P_r is concatenated with itself. The response set P'_r is thus defined by $P'_r = \{(x, x) \mid x \in P_r\}$. As a result, the adversary does not have a way to distinguish a random element from a tag response. In fact, π' is unpredictable if and only if π is.

Although the proof in the paper is incorrect, the theorem can still be shown to be correct. We construct a protocol which we call *echo protocol* (Figure 5.7) by slightly modifying the constant-response protocol of the Section 5.2.1. The reader generates a random value c and sends it to the tag. The tag echoes the value to the reader and the reader sends it to the tag again. The next Theorem shows that the echo protocol is indistinguishable but not unpredictable.

Theorem 5.7. *Indistinguishability (Definition 5.5) does not imply unpredictability (Definition 5.6).*

Proof. The echo protocol is indistinguishable since every tag responds with the message it was queried with. The identifier *ID* of the tag is never used in the protocol and, therefore, the tags in this protocol could even be identically built and do not need to have an *ID*. Thus, regardless of their behavior, it is not possible to distinguish two tags.

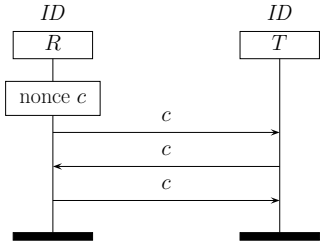


Figure 5.7: Echo protocol.

The adversary can attack unpredictability as follows. He does not query any oracle during the learning phase. He submits a random tag T_c and a random challenge c. The experiment returns either a random pair $(r, f) \in_R P_r \times P_f$ or the tag's response (c, c). In the challenge phase, the adversary guesses 1 if the tag response is (c, c) and 0 otherwise.

The adversary wins with probability $1 - 2^{-2k-2}$ where $|c| = k$. Thus, the echo protocol is not unpredictable. □

The echo protocol shows that there are protocols that are intuitively (and by the definition of indistinguishability) untraceable, but which are not untraceable according to the proposed definition of unpredictability.

Backdoor protocol

Ma et al. show that unpredictability implies indistinguishability [MLDL09, Theorem 3] for any protocol that satisfies completeness and soundness. As a consequence, unpredictability would be stronger than indistinguishability, allowing protocol designers to only prove unpredictability of their protocol. The proof relies on the fact that all queries made by the adversary in the indistinguishability experiment, can be simulated in the unpredictability experiment. There is, however, one fundamental mistake in the simulation of the SendRes oracle: in the unpredictability experiment the adversary is not allowed to modify the message sent from the tag to the reader, but in the indistinguishability game he is. This allows one to construct a protocol that is unpredictable as long as the attacker does not modify messages, but which is not indistinguishable if messages are modified. Lai, Deng, and Li [LDL10] have designed such a protocol. However, their protocol is not sound. Since, the theorem only holds for complete and sound protocols, the protocol does not invalidate the theorem.

We now construct a complete, sound, and unpredictable protocol that is not indistinguishable. The protocol is based on the plaintext-ID protocol (Section 5.2.1). Let ℓ be a security parameter. We assume that all tags are equipped with identifiers ID and IDS both known to the legitimate reader R. The identifiers ID and IDS are chosen independently at random from the set $\{0,1\}^\ell$. Aside from the reader, nobody except for a tag itself knows the tag's ID and IDS. Let $pk(R)$ be a reader's public key with corresponding private key $sk(R)$ and let $\{m\}_{pk(R)}$ denote an IND-CCA2 public-key encryption of m with the public key $pk(R)$.

The reader R initiates the protocol (Figure 5.8) by sending a random value c to the tag. The tag generates a random value r and sets $V := \{c, r, ID\}_{pk(R)}$ and $s := r$.

It sends the response V, s to the reader. The reader verifies $V = \{c, r, ID\}_{pk(R)}$, learns r and ID, and checks whether $r = s$. If both checks succeed, he responds with a random value and accepts the tag. If only the first check succeeds, he responds with IDS and rejects the tag. If both checks fail, R halts and rejects the tag. We call the protocol *backdoor* protocol since the protocol has a backdoor that allows the attacker to easily recover IDS, a unique attribute that allows tracing a tag. It is important to note that knowledge of IDS does not allow an attacker to impersonate a tag.

A key observation is that if the adversary merely forwards messages, the third message is random. If the adversary modifies s in the second message, then the reader always responds with the IDS of the tag with which it is communicating. Finally, the reader only accepts the tag if the adversary does not modify the first and second message. We first show that the protocol is not indistinguishable and then proceed by proving completeness, soundness, and unpredictability of it.

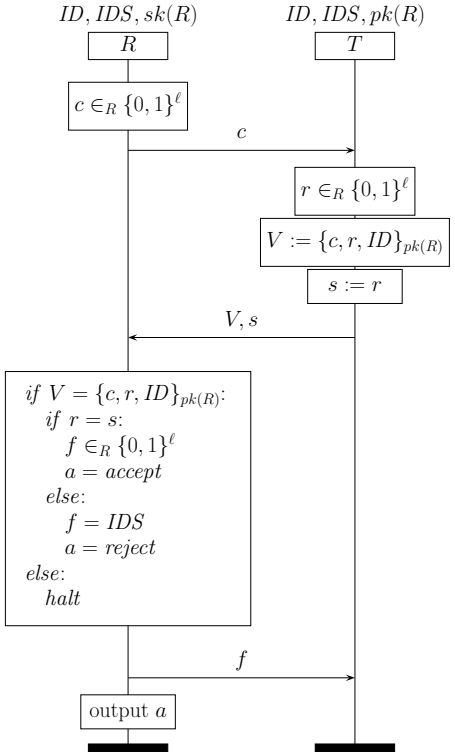


Figure 5.8: Backdoor protocol.

The protocol is not indistinguishable since the attacker can obtain a unique attribute for any tag T. By modifying s in the second message, the reader responds with IDS. The adversary can, therefore, perform an attribute acquisition attack on the protocol.

Lemma 5.8. *The backdoor protocol is not indistinguishable (Definition 5.5).*

Proof. We construct an adversary $\mathcal{A}$ that wins the indistinguishability experiment. In the learning phase, $\mathcal{A}$ randomly selects two tags T_1 and T_2 with identifiers (ID_1, IDS_1) and (ID_2, IDS_2). The adversary starts a protocol execution by querying the InitReader oracle. He then calls the InitTag oracle for tag T_1 with the challenge obtained from the reader. The tag responds with V, s. The adversary

sets s to 0 and forwards the messages to the SendRes oracle. After verification this oracle returns some value IDS.

The adversary initiates a second session with the reader and obtains a challenge c. He then submits T_1, T_2, and c to the experiment. The experiment chooses one of T_1 and T_2 at random and calls it T_c. In the challenge phase, $\mathcal{A}$ performs the same steps as in the learning phase obtaining some IDS'. The adversary guesses $b' = 0$, if $IDS = IDS'$ and $b = 1$ otherwise.

Let l be the bitlength of r. Then $\mathcal{A}$ wins the experiment with probability $1 - 2^{-l-1}$. Therefore, the backdoor protocol is not indistinguishable. □

Completeness of the protocol follows directly from the protocol specification.

Lemma 5.9. *The backdoor protocol satisfies completeness.*

Proof. Let (c, r, f) be the protocol messages observed by the reader in session sid with tag T and T generated response r. Since r was generated by a legitimate tag, it must be of the form $r = \{c, r, ID\}_{pk(R)}, s$ with $r = s$. Following the reader lookup procedure, the reader responds with a random value f and output accept with probability 1. □

Soundness of the backdoor protocol follows from the IND-CCA2 property of the ciphertext. Thus, the adversary must know the identifier ID to generate a valid response.

Lemma 5.10. *The backdoor protocol satisfies soundness.*

Proof. Suppose towards a contradiction that there exists an adversary $\mathcal{A}_s$ that can win the soundness experiment with non-negligible adversary. We show that $\mathcal{A}_s$ can be turned into an adversary $\mathcal{A}_e$ that can break the IND-CCA2 property of the encryption.

We first show that $\mathcal{A}_e$ can win the IND-CCA-MU experiment (Section 2.1.4). The experiment generates a public key pk with corresponding secret key sk. The public key is given to $\mathcal{A}_e$. At the start of the experiment, a challenge bit b is selected at random. To predict the value of b, $\mathcal{A}_e$ uses the advantage of the soundness adversary $\mathcal{A}_s$ in winning the soundness experiment.

The main idea behind the proof is to construct two sets of IDs: ID_0 and ID_1. We let the LR-oracle of the IND-CCA-MU generate a ciphertext for an ID selected from ID_b. If $\mathcal{A}_s$ can win the soundness experiment, he must be able to construct a valid V different from all V's generated by the IND-CCA-MU experiment. Adversary $\mathcal{A}_e$ is allowed to decrypt this V. The decryption reveals whether the ID was chosen from ID_0 or ID_1 and thus the value of b.

Adversary $\mathcal{A}_e$ simulates responses to queries of the adversary A_s as follows:

- InitReader: $\mathcal{A}_e$ returns a randomly generated challenge $c \in_R P_c$ and a fresh session identifier sid. He stores the tuple (c, sid).
- InitTag(T_i, sid, c): $\mathcal{A}_e$ generates a random $s \in_R \{0,1\}^\ell$ and queries the LR oracle for the encryption $V = LR((c, s, ID_{0,i}), (c, s, ID_{1,i}), b)$. He stores the tuple $(c, i, (V, s))$ and returns (V, s).

- SendRes($sid, c, (V, s)$): $\mathcal{A}_e$ searches its list for a tuple $(sid', c', (V', s'))$ with matching session identifier and challenge, i.e. $sid = sid'$ and $c' = c$.
  - If $(V, s) = (V', s')$ then $\mathcal{A}_s$ simply forwarded the messages obtained from the InitTag oracle. Adversary $\mathcal{A}_e$ returns a random $f \in_R \{0,1\}^\ell$ and accepts the tag.
  - If $V = V'$ and $s \neq s'$ then the encryption was forwarded, but the randomness was changed. Following the protocol specification, adversary $\mathcal{A}_e$ returns IDS_i and rejects the tag.
  - If $V \neq V'$ then $\mathcal{A}_s$ forged or replayed the response. If V was replayed, then $\mathcal{A}_e$ halts the protocol execution and rejects. Otherwise, $\mathcal{A}_e$ submits V' to the decryption oracle of the IND-CCA-MU experiment obtaining c', r', ID'.
    - If $c = c'$, $r' = s'$, and $(ID' = ID_{0,i} \vee ID' = ID_{1,i})$ then he returns a random $f \in_R \{0,1\}^\ell$ and accepts the tag.
    - If $c = c'$, $r' \neq s'$, and $(ID' = ID_{0,i} \vee ID' = ID_{1,i})$ then he returns IDS_i and rejects.
  - Otherwise $\mathcal{A}_e$ halts the protocol execution and rejects.

At the end of the soundness experiment the adversary $\mathcal{A}_s$ outputs a conversation $(c, (V, s), f, T_i)$ that was accepted by the reader while T_i did not send (V, s). Thus, $\mathcal{A}_e$ has previously submitted V to the decryption oracle to obtain ID'. If $ID' = ID_{0,i}$ then $\mathcal{A}_e$ guesses $b = 0$, otherwise $b = 1$.

If $\mathcal{A}_s$ can win the soundness experiment with non-negligible probability, then $\mathcal{A}_e$ can win the IND-CCA-MU experiment with non-negligible probability. An adversary that can win the IND-CCA-MU experiment with non-negligible probability can do so for the IND-CCA2 experiment as well [BBM00, Theorem 1]. This contradicts the IND-CCA2 property of the encryption. Thus, the protocol must satisfy soundness. □

Finally, we show unpredictability of the backdoor protocol. The main weakness in the model that allows us to show unpredictability is the way in which the experiment generates the challenge to the adversary. The experiment merely forwards the message from tag to reader. Man-in-the-middle attacks on untraceability, however, rely on the assumption that attackers can modify messages sent from reader to tag. The backdoor protocol is unpredictable and is only traceable if the adversary modifies the second message.

Lemma 5.11. *The backdoor protocol is unpredictable (Definition 5.6).*

Proof. We show unpredictability by contradiction. Let $\mathcal{A}$ be the adversary that can break unpredictability. Then there exists an adversary $\mathcal{B}$ that breaks the IND-CCA2 property of the encryption by using $\mathcal{A}$.

Adversary $\mathcal{B}$ can be constructed as follows. He queries a key generation oracle that generates a key pair $sk(R)$ and $pk(R)$. The adversary $\mathcal{B}$ obtains the public key $pk(R)$ and generates a set of tags $\mathcal{T}$. Adversary $\mathcal{B}$ can answer queries from $\mathcal{A}$ in the learning phase by using the decryption oracle. In between phases $\mathcal{A}$ chooses a

target tag T_c with secret ID and a challenge c. At this point $\mathcal{B}$ chooses r, r', f, and ID' at random and submits the following two plaintexts to the encryption oracle of the IND-CCA2 game: r, c, ID and r', c', ID'. The oracle chooses one at random, encrypts it under $pk(R)$ and returns it to B. Let E be this encryption returned by the oracle. Adversary $\mathcal{B}$ continues by sending E, r and f to $\mathcal{A}$. In the challenge phase $\mathcal{B}$ again simulates all queries of $\mathcal{A}$. At the end of the experiment $\mathcal{A}$ returns a guess b'. If $b' = 1$, then $\mathcal{B}$ guesses 1, otherwise 0.

If $\mathcal{A}$ has a non-negligible advantage in winning the unpredictability experiment, then $\mathcal{B}$ has a non-negligible advantage in winning the IND-CCA2 game. This contradicts the assumed IND-CCA2 property of the encryption. Thus the backdoor protocol must be unpredictable. □

The main Theorem of this section combines the previous four Lemmas. It shows that indistinguishability and unpredictability are incomparable. As a result, proving unpredictability is not sufficient for proving indistinguishability. Furthermore, it shows that there are protocols that are not untraceable, but can be proven to be unpredictable. Hence, we believe that unpredictability does not properly model the intuitive notion of untraceability.

Theorem 5.12. *Unpredictability (Definition 5.6) does not imply indistinguishability (Definition 5.5) for sound (Definition 5.4) and complete (Definition 5.3) protocols.*

Proof. The backdoor protocol is sound and complete as shown by Lemmas 5.9 and 5.10. Lemmas 5.11 and 5.8 show that the backdoor protocol is unpredictable, but not indistinguishable. Hence, the backdoor protocol is an example protocol showing that unpredictability does not imply indistinguishability for sound and complete protocols. □

5.3 Insider attacks in Vaudenay's model

In the previous chapter we have shown a man-in-the-middle attack on EC-RAC IV. The man-in-the-middle attack was a result of the attacker being able to modify messages without the reader noticing. By adding a message authentication code (MAC) based on an independent shared secret k, the protocol can be turned into a protocol satisfying security. However, an attacker that has one insider tag in the system can trace other tags.

In this section, we consider the computational proof model for untraceability by Vaudenay [Vau07]. The model can be considered to be the most comprehensive computational proof model, but has also been in need of refinements and clarifications in several ways [PV08, NSMSN08, NSMSN09, BCI09, HPVP11]. We show that an adversary with insider capabilities cannot be naturally represented in the model. To this end, we first present Vaudenay's adversary model and security definitions. We provide more precise definitions wherever the original paper was underspecified. Subsequently, we extend the model with an adversary class that contains attackers with insider capabilities.

5.3.1 Vaudenay's model

Adversary model

The powers of the adversaries are modeled by a set of oracles the adversary is allowed to query. Tags may either be in the vicinity of the adversary (we call the tag *drawn*) or out of the vicinity of the adversary (*free*). If a tag is drawn, it gets a temporary identity *vtag* through which it can be addressed by the adversary. Initially, all tags are free tags.

The following eight oracles define the capabilities of the adversary.

- CreateTag^b(*ID*): Creates a new tag with fresh identifier *ID*. The adversary may choose to create a legitimate tag ($b = 1$) or an illegitimate tag ($b = 0$). In case of a legitimate tag, the tag is added to the database.
- DrawTag(*distr*) $\rightarrow$ *vtag*: Moves a tag from the set of free tags to the set of drawn tags creating a fresh identifier *vtag* by which the tag can be addressed. The tag is drawn at random following the distribution *distr*.
- Free(*vtag*): Moves *vtag* to the set of free tags. Oracle access to *vtag* is no longer allowed after calling the Free oracle.
- Launch $\rightarrow \pi$: makes the reader launch a new protocol instance π.
- SendReader$(m, \pi) \rightarrow m'$: Sends the message m to the reader's protocol instance π. If the message m corresponds to the message that the reader expected, he responds with message m'.
- SendTag$(m, vtag) \rightarrow m'$: Sends the message m to the tag with identifier *vtag*. If the message m corresponds to the message that the tag expected, he may respond with a message m'.
- Result$(\pi) \rightarrow x$: when the protocol instance with identifier π completed successfully, the oracle returns 1, otherwise 0.
- Corrupt(*vtag*) $\rightarrow S$: Returns the state S of the tag. The state contains the current values of the variables mentioned as initial knowledge of the tag.

Vaudenay defines eight adversary classes by restricting access to the Corrupt and Result oracles. There are four different types of adversaries related to restrictions on the Corrupt oracle:

- *Weak*: The adversary controls all communication channels and may forward, inject, block, or modify messages. He does not have access to the Corrupt oracle. This means that the attacker cannot recover the keys of a tag by any other means than communicating with the tag and reader.

  This adversary corresponds to a Dolev–Yao [DY83] attacker.

- *Forward*: In addition to the capabilities of a weak adversary, a forward adversary may make queries to the Corrupt oracle. However, the adversary may no longer query the other oracles after the first Corrupt query.

A forward adversary models the possibility of an attacker breaking a tag's untraceability with respect to the tag's past interactions as soon as the tag's keys are revealed to the attacker.

- *Destructive*: A destructive adversary has more flexible access to the CORRUPT oracle than a forward adversary in that he may query all oracles after querying the corrupt oracle. The adversary is limited with respect to the tags he has corrupted, in that corrupting a tag makes the tag inaccessible. Destructive adversaries therefore model the real-world attacker that can obtain the tag's keys, but destroys the tag in the process.
- *Strong*: A strong adversary is not restricted with respect to the usage of the CORRUPT oracle. This models the case in which an adversary can obtain the secrets of a tag without having to physically destroy the tag. The secrets obtained from the tag can then be used to mount a traceability attack on the corrupted tag in a later stage.

A second separation concerns the ability of the attacker to recognize whether a protocol execution between a reader and a tag was successful. In many practical situations this is a reasonable assumption. For instance, in an RFID system for electronic transport tickets, a reader flashing a green light indicates that authentication was successful while a red light indicates it failed. In the model, we call an adversary with access to the RESULT oracle *wide*, while an adversary with no access to the RESULT oracle is called *narrow*.

The relations between the 8 adversary classes are depicted in the diagram below (N stands for "narrow", W stands for "wide"), where $\mathcal{A} \Rightarrow \mathcal{A}'$ means that $\mathcal{A}'$ is more restricted with respect to oracle access than $\mathcal{A}$.

$$\begin{array}{ccccccc} \text{W-Strong} & \Rightarrow & \text{W-Destructive} & \Rightarrow & \text{W-Forward} & \Rightarrow & \text{W-Weak} \\ \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow \\ \text{N-Strong} & \Rightarrow & \text{N-Destructive} & \Rightarrow & \text{N-Forward} & \Rightarrow & \text{N-Weak} \end{array}$$

Security definitions

Untraceability, called privacy in the model, is defined by comparing the adversary to a special adversary which makes no use of protocol messages, as follows. An adversary is called *blinded* if he is not allowed to communicate with tags and reader. An adversary is *trivial* if there exists a blinded adversary which essentially performs equally well at guessing a tag's identifier.

More precisely, let $\mathcal{A}$ be an adversarial algorithm that outputs *True* or *False*. A blinder B for an adversary $\mathcal{A}$ is an algorithm that sees the same messages as $\mathcal{A}$, and simulates the LAUNCH, READERTAG, SENDTAG, and RESULT oracles to $\mathcal{A}$. A blinded adversary $\mathcal{A}^B$ is an adversary that does not access the LAUNCH, READERTAG, SENDTAG, and RESULT oracles. Let p_A be the probability that $\mathcal{A}$ outputs *True* and p_B be the probability that $\mathcal{A}^B$ outputs true. An adversary $\mathcal{A}$ is trivial if there exists a B such that $|p_A - p_B|$ is negligible.

Definition 5.13 (Privacy [Vau07])**.** *A protocol is* P-private*, where* P *is one of the eight adversary classes, if all adversaries that belong to that class are trivial.*

One can show privacy of a protocol by proving that all oracle access to the LAUNCH, READERTAG, SENDTAG, and RESULT oracles can be perfectly simulated. Conversely, one shows that a protocol is not private by showing that there exists an adversary $\mathcal{A}$ for which these oracle accesses cannot be simulated.

Vaudenay defines authentication as the impossibility of a reader accepting a tag with which it did not have a matching conversation. Matching conversation, in turn, is defined as a combination of well-interleaved and faithful exchange of messages. We first formalize the notions of faithfulness, well-interleaving, and matching conversation and then give the definition of authentication.

We state our definitions in terms of protocol runs. A protocol specifies the actions to be carried out by a reader or a tag. A protocol *run* is a sequence of events which occur as a consequence of the actions specified by a protocol. The possible events of a run are *send*, *receive*, *accept*, and *reject*. The send and receive events are associated with messages, the accept event is associated with a particular RFID tag. We say that a run ends successfully if all its events have occurred and no reject event has occurred.

Well-interleaving specifies that messages between a reader run and a tag run are never received before they were sent. It prevents replay and pre-play attacks.

Definition 5.14 (Well-interleaving)**.** *We say that a target run r_1 and another run r_2 are* well-interleaved*, if for all $i, j \in \mathbb{N}$ the i-th receive event of r_1 is preceded by the i-th send event of r_2 and j-th receive event of r_2 is preceded by the j-th send event of r_1.*

Faithfulness defines that messages exchanged between a reader run and a tag run are not modified by the adversary.

Definition 5.15 (Faithfulness)**.** *We say that a target run r_1 and another run r_2 have exchanged messages* faithfully*, if for all $i, j \in \mathbb{N}$ the message of the i-th receive event of r_1 is equal to the message of the i-th send event of r_2 and the message of the j-th receive event of r_2 is equal to the message of the j-th send event of r_1.*

A tag and reader have a *matching conversation* if the pair of reader run and tag run satisfies well-interleaving and faithfulness.

Definition 5.16 (Matching conversation)**.** *A reader R and a tag T have a matching conversation for a target run r of the reader, if there is a run t of T such that r and t are well-interleaved and exchange messages faithfully.*

Authentication, called *security* in the model, is defined by requiring matching conversations on accepted runs. Whenever the reader accepts a tag in a target run, then there must exist a run of that tag that has a matching conversation with the target run.

Definition 5.17 (Security)**.** *A protocol is said to be* secure *if for each target run r for which the reader accepts an uncorrupted legitimate tag T, there is a run t of T such that r and t have a matching conversation.*

Finally, *correctness* of an RFID protocol is defined as the property that a legitimate tag is always accepted in a protocol execution in which the reader merely forwards the messages.

Definition 5.18 (Correctness)**.** *Let S be an RFID system with a reader R and a set of tags $\mathcal{T}$. Draw a tag $T \in \mathcal{T}$ and execute a protocol between R and T with runs r and t. A protocol is* correct *if R accepts T if and only if T is legitimate.*

5.3.2 Modeling insider attacks

Vaudenay's model for security and privacy of RFID protocols does not discuss the possibility of insiders. In the model, there are initially no legitimate tags under the control of the adversary. Adversaries can obtain the contents of legitimate tags through tag corruption. Insider attacks are a very weak form of corruption: the tag whose contents are obtained is *never* the subject of the traceability attack. In practice, the adversary can purchase one tag and spend an extended amount of time to break the tag's security. The adversary does not even need physical access to the tag he wants to trace.

We extend the adversary model by introducing an InsiderTag oracle defined as follows:

- InsiderTag(ID) $\rightarrow S$: creates a legitimate tag with identifier ID and runs the SetupTag algorithm to update the system database. The new tag is immediately corrupted and destroyed and the state S of the tag is returned.

The InsiderTag oracle call are not blinded. Also, calling the oracle gives the blinder access to the tag state S. This is to prevent the adversary from having an advantage by tracing the insider tag, which would lead to a false attack. Note that the same restrictions hold for the Corrupt oracle [Vau07].

We define a ninth class of adversaries, called *wide-insider*, to model adversaries that have access to to the InsiderTag oracle and all other oracles except for the corrupt oracle. Figure 5.9 shows how the class *wide-insider* fits in the hierarchy of privacy models. By construction, wide-destructive is stronger than wide-insider which in turn is stronger than wide-weak. In the following sections we show protocols that separate the class of wide-insider attackers from all other classes of adversaries. This shows that the Vaudenay adversary model does not capture insider attackers and that the extension with the class of wide-insider adversaries is useful.

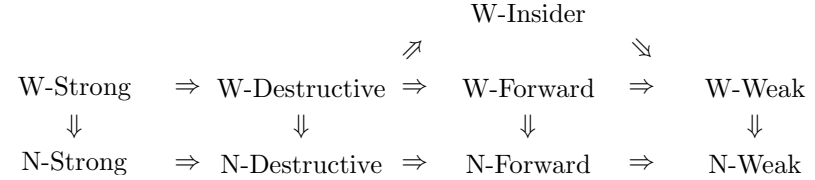


Figure 5.9: Extension of the adversary classes in Vaudenay's model.

Separation with wide-destructive and wide-weak

To show separations between the class weak-insider and the other 8 classes, we take the *wide-weak* private protocol from [Vau07]. The protocol is depicted in

Figure 5.10. The protocol is initiated by the reader by sending a random value a to the tag. The tag selects a random value b and computes $F_{K(R,T)}(a, b)$ where F is a pseudo-random function (PRF) and $K(R, T)$ is a shared secret between reader and tag. The reader accepts the tag if it has a pair $(ID, K(R, T))$ in its database.

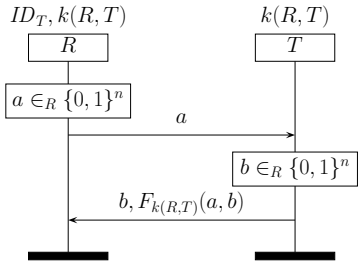


Figure 5.10: Protocol P: Wide-weak private RFID protocol based on a PRF.

Theorem 5.19. *Protocol P is wide-insider private.*

Proof. The protocol is wide-weak private by [Vau07, Theorem 13]. Therefore, there exists a blinder $\mathcal{B}$ for adversary $\mathcal{A}$ that simulates the all oracles perfectly. Since tag keys are chosen independently at random, the InsiderTag oracle does not give useful information to break privacy. We construct a blinder $\mathcal{B}'$ for $\mathcal{A}'$ from $\mathcal{B}$ as follows:

- The simulation for SendTag queries of blinder $\mathcal{B}'$ remains the same as for $\mathcal{B}$. Since tag keys are chosen independently at random, this simulation is perfect. If it were not, then $\mathcal{A}$ could have generated his own set of random keys to obtain an advantage over $\mathcal{A}^B$ for the wide-weak privacy game.

- In protocol P, there are only two types of SendReader events. The first one takes no argument and returns a random value. The second does not return a value. These can be trivially simulated.

- Launch queries can be trivially simulated.

- For the Result oracle there are two cases. The oracle can be queried on a session concerning the insider tag or on a session with another legitimate tag. The blinder can distinguish these two cases as follows. If there is a well-interleaved and faithful sequence of SendReader and SendTag queries for π then the blinder outputs 0. Otherwise, the query relates to a session with the insider tag.

  Since the blinder knows a, b and k, he can verify whether $F_k(a, b)$ submitted to π with the SendReader query for π is correct. If it is, the blinder outputs 1, otherwise 0. By definition of a PRF, $F_k(a, b) = F_{k'}(a', b')$ iff $(k, a, b) = (k', a', b')$. Thus, the simulation of the Result oracle is perfect.

Since the simulation of all oracles is perfect, $\mathcal{A}'$ does not have a non-negligible advantage over $\mathcal{A}'^{B'}$. Therefore, protocol P is wide-insider private. □

Since protocol P is wide-insider private, but not wide-forward [Vau07, Note 14] private, we have that wide-insider privacy does not imply wide-forward privacy. As a consequence, wide-insider privacy also does not imply wide-destructive privacy.

To show separation of the adversary classes of wide-insider and wide-forward, we first show how a three-message ping-pong protocol can be turned into a protocol that is secure (according to Definition 5.17).

Three-message ping-pong protocols

Let π be a three-message ping-pong protocol with messages m_1, m_2, and m_3 initiated by the tag T. A standard way of providing security is to add a message authentication code (MAC) based on a shared secret. Let R and T share a key k generated independently at random. To ensure security, we append a MAC over the three messages m_1, m_2, m_3 to the last message. For privacy, we encrypt the MAC under the public key of the reader $pk(R)$. Protocol ρ is thus constructed from π by appending the IND-CCA2 encryption of $MAC_k(m_1, m_2, m_3)$ under the public key of the reader $pk(R)$ to the third message. Protocol ρ is depicted in Figure 5.11. This construction allows us to turn any three-message ping-pong protocol into a protocol satisfying security.

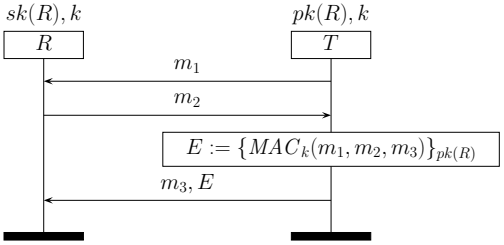


Figure 5.11: Protocol ρ: 3-message ping-pong protocol with IND-CCA2 encrypted MAC.

In the following two lemmas, we show that protocol ρ satisfies security and that it satisfies the same privacy property as π. We denote the entropy of m_i (for $i \in \{1, 2\}$) by $e_i(l)$, where l is a security parameter, and we denote the length of k by $|k|$. We assume that $e_i(l)$ and $|k|$ are such that there exists a constant $c > 0$ with $e_1(l), e_2(l), |k| > c \cdot l$.

Lemma 5.20. *Protocol ρ (Figure 5.11) satisfies security (Definition 5.17).*

Proof. If security is not satisfied then there must be a run r in which the reader accepts an uncorrupted legitimate tag T, while there is no run t of T for which r and t have a matching conversation. Therefore, either there exists no corresponding run of T or there exists a run t of T, but well-interleaving (Definition 5.14) or faithfulness (Definition 5.15) between r and t is violated. We show that in all three cases, the adversary breaks the IND-CCA2 property of the encryption, or is able to generate existential MAC forgeries.

Let $poly(x)$ be a polynomial. The adversary is allowed up to $poly(l)$ observations and computations. We have the following 3 cases:

(1) No corresponding run of T:

The adversary wins the game if the reader R successfully completes a target run r without a corresponding run of T. The adversary must choose an m_1 after which the reader generates a fresh m_2. The adversary wins if he can find a valid pair m_3, E.

The adversary must either reuse an old E or generate a new, valid one.

The probability that the adversary already observed a run with messages m_1 and m_2 together with a valid m_3, E is bounded above by $poly(l)/2^{e_2(l)}$, thus negligible.

Otherwise, the adversary needs to forge m_3, E. Such an adversary can be turned into an adaptive algorithm which constructs existential MAC forgeries as follows. Let $(m_3, E) = \mathcal{A}(m_1, m_2, transcript)$ be the adversary's output, where $transcript$ consists of a $poly(l)$ list of quadruples (m_1, m_2, m_3, E).

Let k be the key in the MAC existential forgery game. Let tr be a polynomial list of $((m_1, m_2, m_3), MAC_k(m_1, m_2, m_3))$ pairs obtained from the existential MAC forgery game upon submission of m_1, m_2, m_3 queries. We create the list $transcript$ from tr by replacing $MAC_k(m_1, m_2, m_3)$ in tr by $E_{pk(R)}(MAC_k(m_1, m_2, m_3))$.

Let $(m_3, E) = \mathcal{A}(m_1, m_2, transcript)$ be the adversary's output. We submit (m_1, m_2, m_3) and the decryption of E to the forgery game. If the adversary $\mathcal{A}$ wins with non-negligible probability, then our algorithm wins the forgery game with non-negligible probability. Thus by the existential forgery property of the MAC, the adversary's advantage to produce a correct pair (m_3, E) is negligible for sufficiently large $|k|$.

(2) Faithfulness:

The adversary wins the game if there is a protocol instance between a tag and a reader where the reader's view of the protocol messages differs from the tag's view. Thus at least one of the terms m_1, m_2, m_3, E has been modified by the adversary, yet the reader accepts the received pair (m_3, E).

If one or more of m_1, m_2, m_3 are modified by the adversary, we can turn the adversary into an algorithm which creates existential MAC forgeries as in case (1) above.

If m_1, m_2, m_3 are not modified, then except with negligible probability, the MAC is the same. Thus, the adversary must produce a second ciphertext E' from E, violating non-malleability of the IND-CCA2 encryption.

(3) Well-interleaving:

The adversary wins the game if a target run of the reader ends successfully without a well-interleaved run of a tag. Thus a receive event must have occurred for which no corresponding send event had occurred according to Definition 5.14. By faithfulness, it follows that the adversary has successfully predicted one of the three messages m_1, m_2, m_3', where $m_3' = (m_3, E)$.

Since for $i = 1, 2$, the messages m_i have entropy $e_i(l)$, the adversary has only a negligible probability of $poly(l)/2^{e_i(l)}$ to predict m_i.

Else, the adversary must have predicted m'_3. The probability of successfully predicting m'_3 is bounded by the probability of the adversary winning the ciphertext indistinguishability game of the IND-CCA2 encryption scheme, hence, negligible.

Thus, protocol ρ satisfies security. □

Lemma 5.21. *If a protocol π satisfies narrow-{strong, destructive, forward, weak} privacy, then so does ρ.*

Proof. Let $c \in$ {strong, destructive, forward, weak} and let P be a narrow-c private protocol. Then, there must exist a blinder B that simulates messages m_1, m_2 and m_3. It remains to construct a blinder B' from B that simulates E.

To simulate E, B generates a random value r, of length equal to the output size of the MAC function, and returns $\{r\}_{pk(R)}$. By the ciphertext indistinguishability property of the IND-CCA2 encryption, the simulation is perfect.

Thus, P' must be narrow-c private. □

We can now show our theorem that allows to turn any 3-message ping-pong protocol initiated by the tag into a protocol that satisfies security. The privacy property of the protocol is maintained across the transformation.

Theorem 5.22. *Let l be a security parameter, $c > 0$ a constant, $e_1(l), e_2(l)$ be greater than $c \cdot l$.*

Let π be a three-message ping-pong protocol between a reader R and a tag T, initiated by the tag. For $i \in 1, 2, 3$, let m_i be the i-th message of π and $e_i(l)$ be the entropy of the i-th message. Let, furthermore, $pk(R)$ be R's public key and k a unique, randomly generated, shared secret key between R and T with $|k| > c \cdot l$.

Then the protocol ρ obtained from π by concatenating m_3 with the IND-CCA2 encryption $\{MAC_k(m_1, m_2, m_3)\}_{pk(R)}$ satisfies security.

If π satisfies narrow-{strong, destructive, forward, weak} privacy, then so does ρ.

Proof. Security follows from Lemma 5.20 and privacy follows from Lemma 5.21. Correctness follows from the correctness of the IND-CCA2 encryption and the fact that the reader can identify the tag based on the key k used in the MAC. □

Separation with wide-forward

To show separation between wide-forward and wide-insider privacy we modify the randomized Schnorr protocol [BCI08] using the construction in Section 5.3.2. The randomized Schnorr protocol is based on a cyclic group of points on an elliptic curve. The group is of order q and P is a generator element. The group is chosen so that the decisional Diffie-Hellman problem is hard. The protocol is initiated by the tag generating two random values a and b. The tag sends aP and byP to the reader upon which it responds with a random challenge c. The tag computes $z = a + b + x \cdot c$ and sends it to the reader for verification. We modify the protocol

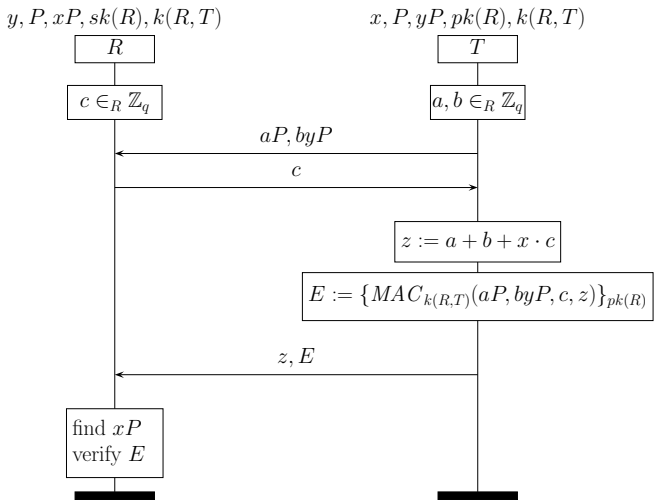


Figure 5.12: Protocol Q: BCI with encrypted MAC

by appending an IND-CCA2 encrypted MAC as in Theorem 5.22. The protocol is depicted in Figure 5.12.

The protocol is narrow-strong and secure by [BCI08, Theorem 2] and Theorem 5.22. We now show that the protocol is vulnerable to insider attacks.

Theorem 5.23. *The randomized Schnorr protocol [BCI08] with IND-CCA2-encrypted MAC (Figure 5.12) is not wide-insider private.*

Proof. The attack strategy is as follows. The attacker eavesdrops on two valid protocol executions of tags T and T'. He combines the tag responses and uses a protocol execution between an insider tag and the reader to verify whether $T = T'$.

We construct an adversary that eavesdrops on two protocol executions between tags T and T' and a legitimate reader. The protocol transcripts for these executions are (aP, byP, c, z, E) and $(a'P, b'yP, c', z', E')$, respectively. By the protocol specification, z and z' are defined by $z = (a + b + x \cdot c)$ and $z' = (a' + b' + x' \cdot c')$. It is the attacker's goal to decide whether $T = T'$ which amounts to deciding whether $x = x'$.

The adversary computes α, β, and γ as follows:

$$\begin{aligned} \alpha &= c' \cdot aP - c \cdot a'P \\ \beta &= c' \cdot byP - c \cdot b'yP \\ \gamma &= c' \cdot z - c \cdot z' \end{aligned} \tag{5.1}$$

Terms α, β, and γ satisfy the following equation if and only if $x = x'$.

$$\alpha + \beta \cdot y^{-1} = \gamma P \tag{5.2}$$

The adversary now calls the InsiderTag oracle and receives yP, P, and x'' of a legitimate tag T''. To test whether Equation (5.2) holds, the adversary initiates a protocol execution with a reader. He sends α and β, upon which the reader challenges with c''. The adversary computes $z'' = \gamma + x'' \cdot c''$ and the corresponding encrypted MAC. The reader accepts the adversary's insider tag if and only if

$x = x'$. Therefore, if the reader accepts the insider tag, we know that $T = T'$, otherwise $T \neq T'$. The protocol flow between the reader and adversary is depicted in Figure 5.13.

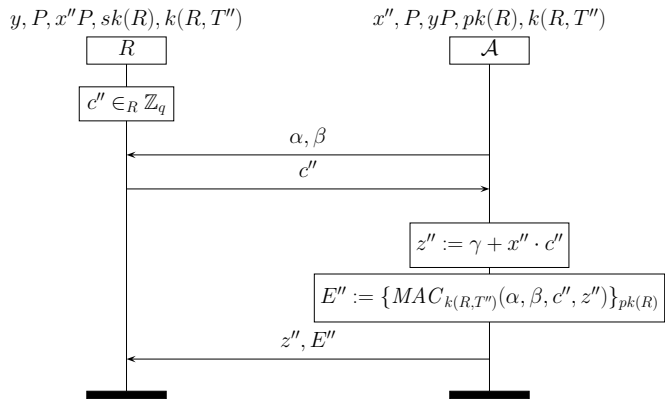


Figure 5.13: Insider attack on BCI with encrypted MAC.

Thus, the protocol is not wide-insider private. □

Bringer et al. [BCI08, Theorem 2] showed that BCI is narrow-forward private and Theorem 5.22 shows that BCI with encrypted MAC is narrow-forward private and satisfies security. Vaudenay has shown [Vau07, Lemma 8] that any narrow-forward private protocol that satisfies security is also wide-forward private. BCI with encrypted MAC is, therefore, wide-forward private. By Theorem 5.23 it is not wide-insider private. Therefore, wide-forward privacy does not imply wide-insider privacy and wide-weak privacy does not imply wide-insider privacy.

5.4 Results by Ng, Susilo, Mu, and Safavi-Naini

Vaudenay showed that any narrow-weak (or narrow-forward) private protocol is also wide-weak (or wide-forward) private if it satisfies security and always accepts a legitimate tag with which it has a matching conversation [Vau07, Lemma 8]. The only difference between a narrow and a wide adversary is that the wide adversary can access the Result oracle. To show that any narrow-private protocol is also wide-private, the blinder needs to simulate answers to Result queries. By security, the oracle always returns 0 if there no matching conversation with a legitimate tag. By the hypothesis, the oracle always returns 1 if there is a matching conversation with a legitimate tag. Forward and weak adversaries can not query the Result oracle after a Corrupt oracle access. Thus, to simulate answers to Result queries, the blinder returns 1 if there is a matching conversation and 0 otherwise. This simulation is perfect, proving the lemma.

The above result was extended for destructive and strong adversaries by Ng, Susilo, Mu, and Safavi-Naini [NSMSN08, Proposition 2]. They claim that a narrow-private protocol can be turned into a wide-private protocol under the assumption that it is secure and legitimate tags are always accepted. Their proof relies on the fact that whenever a reader runs a protocol execution, there is always some tag with which it is communicating (be it legitimate or not). However, irrespective of the protocol, a

destructive or strong adversary can corrupt a tag and simulate a protocol execution with the adversary. In such a case, there is no tag present in the communication with the reader. Building on our notion of insider attacks, we can now show that Proposition 2 by Ng et al. [NSMSN08] is invalid.

Lemma 5.24. *A narrow-destructive (narrow-strong) private protocol that satisfies security is not necessarily wide-destructive (wide-strong) private.*

Proof. We show the narrow-destructive case by exhibiting a counter-example. Take the narrow-destructive private protocol BCI by Bringer et al. [BCI08]. By applying Theorem 5.22, it can be turned into a narrow-destructive protocol satisfying security (Figure 5.12). Theorem 5.23 shows that this protocol is not wide-insider private. Since wide-destructive implies wide-insider privacy, we know that the protocol can also not be wide-destructive private.

Since the BCI protocol is also narrow-strong, the narrow-strong case can be shown analogously. □

Lemma 5.24 shows that Proposition 2 by Ng et al. [NSMSN08] does not apply to all narrow-destructive protocols. In particular, it shows that the separation between narrow and wide classes of protocols achieving *security* is indeed necessary for the classes *destructive* and *strong*, as opposed to what was claimed.

A second result [NSMSN08, Proposition 3] states that under the assumption that tags do not share correlated keys, the destructive class is unnecessary. The following lemma contradicts this statement by showing that there exist protocols without correlated keys that are wide-forward private, but not wide-destructive private.

Lemma 5.25. *For the class of RFID protocols employing uncorrelated keys among tags, forward privacy does not imply destructive privacy.*

Proof. We take the narrow-destructive private BCI protocol by Bringer, Chabanne, and Icart [BCI08]. In the BCI protocol, tag keys are not shared. By applying Theorem 5.22, it can be turned into a narrow-destructive protocol satisfying security (Figure 5.12). The resulting protocol is wide-forward private by Lemma 8 of Vaudenay [Vau07]. However, Theorem 5.23 shows that the protocol is not wide-insider private and as a consequence not wide-destructive private. The example shows that wide-forward privacy does not imply wide-destructive privacy for protocols employing uncorrelated keys. □

5.5 Provable wide-strong privacy

We present the first provably wide-strong and authenticating RFID protocol exclusively based on elliptic-curve and scalar operations. Such a scheme is interesting for two reasons. A public-key-based scheme permits more efficient tag identification: As shown by Damgård and Pedersen [DP08], for symmetric schemes, RFID privacy can only be obtained at the cost of an inefficient tag lookup procedure for the RFID reader. The implementation of a typical IND-CCA2 public-key cryptosystem on an RFID tag is, however, quite expensive. To achieve IND-CCA2 security, most

cryptosystems rely on three components. An intractable number-theoretic problem, a symmetric block cipher, and a cryptographic hash function. The main cost in such a scheme is incurred by the large number of gates required to implement the number-theoretic operations on one side and an even larger number of gates to implement the cryptographic hash function on the other. Thus, there is interest in attempting to do away with the hash function by reusing the number-theoretic circuits to implement the same functionality provided by the hash function.

Our protocol is an implementation[1] of Vaudenay's public-key RFID protocol which has been proved to satisfy authentication in [Vau07] and wide-strong privacy in [HPVP11]. The privacy proof requires the protocol to employ an IND-CCA2 public-key encryption scheme. The encryption scheme we use in the protocol is the hash-free variant of the Cramer-Shoup scheme [CS98]. It provides IND-CCA2 security assuming only the decisional Diffie-Hellman assumption.

5.5.1 Preliminaries and notation

In the following we first briefly recall the Cramer-Shoup public-key encryption scheme. The RFID protocol requires RFID tags to encrypt messages for the system's RFID readers, thus one private key and public key pair needs to be generated. The RFID readers store the private key, the RFID tags are equipped with the public key.

Let $\mathbb{F}_{2^n}$ be a finite field with 2^n elements. Let $\mathcal{E}$ be the group of $\mathbb{F}_{2^n}$-rational points of an ordinary elliptic curve over $\mathbb{F}_{2^n}$. That is, $\mathcal{E}$ denotes the set of points which satisfy the equation $y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_{2^n}$ being fixed parameters, together with $\mathcal{O}$, the "point at infinity", which serves as the group's neutral element. We will assume that the group $\mathcal{E}$ contains a subgroup $\mathcal{G}$ of large prime order p and small index in $\mathcal{E}$.

Let $P_1 \in \mathcal{G}$, $P_1 \neq \mathcal{O}$ and $1 < w, c, d, z < p$ be randomly chosen, system-wide parameters and let h be a collision-resistant hash function. Set $P_2 = wP_1$, $C = cP_1$, $D = dP_1$, $H = zP_1$. The tuple (P_1, P_2, C, D, H) is the RFID reader's public key and (w, c, d, z) its secret key.

To encrypt a message $M \in \mathcal{G}$, we choose a random integer $1 < r < p$ and compute $U_1 = rP_1$, $U_2 = rP_2$, $E = rH + M$, $\alpha = h(U_1, U_2, E)$, and $V = rC + r\alpha D$. The ciphertext is (U_1, U_2, E, V).

To decrypt, correctness of V and U_2 needs to be verified first. For this, $\alpha = h(U_1, U_2, E)$ is computed, then V is compared to $cU_1 + \alpha dU_1$ and U_2 is compared to wU_1. If the terms are equal, then the plaintext is recovered via $M = E - zU_1$.

5.5.2 Mapping into the elliptic curve

During the course of the protocol, a tag needs to encrypt a message M. This message needs to be a concatenation of *ID* and challenge to avoid algebraic attacks as well as an element of the group $\mathcal{G}$. Thus we represent *ID* and challenge as bit strings and map their concatenation into the group $\mathcal{G}$.

[1] The implementation is due to Saša Radomirović.

To map the reader's challenge and the tag's identity into the elliptic curve, we use a simple try-and-increment method [Ica09]. In the following, we identify elements in finite extensions of $\mathbb{F}_2$ with bit strings. Let k be a security parameter. The map $\phi : \mathbb{F}_{2^{n-k}} \to \mathcal{G} \cup \{\mathsf{fail}\}$ is defined as follows. It assigns to $x \in \mathbb{F}_{2^{n-k}}$ an element $(x', y) \in \mathcal{G}$, where the $n-k$ most significant bits of $x' \in \mathbb{F}_{2^n}$ are equal to x and the remaining k bits are such that $(x', y) \in \mathcal{G}$. To find such a pair (x', y) we simply step through all 2^k possible bit strings. If no such bit string is found the map returns fail. Since the expected number of try-and-increment steps is 2 [Ica09], the probability of failure is $1/2^{2^k}$. Thus the security parameter k can be fairly small. We refer to [Ica09] for a discussion on how to implement the try-and-increment algorithm securely, that is, resistant to timing attacks.

Lemma 5.26. *If $\mathcal{E}$ has cardinality $2p$, p prime, then the map ϕ can be implemented with 2^{k+1} computations of the trace function of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, and one square root computation over $\mathbb{F}_{2^n}$.*

Remark 5.27. There are several more sophisticated algorithms to map bit strings to points on an elliptic curve than the try-and-increment method we employ above. The most efficient, deterministic maps are Icart's $f_{a,b}$ function [Ica09] and the SWU map [CI09, SvdW06, Ula07]. However, special care needs to be taken in order to implement them securely. The $f_{a,b}$ function, for instance, can have up to four elements in the preimage of a point (x, y). For the case of characteristic 2, where the point satisfies the equation $y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_{2^n}$ being fixed parameters, these four elements are the solutions of the quartic polynomial $u^4 + u^2 + xu + (y + a)$ over $\mathbb{F}_{2^n}$. If Icart's function is used in the way our ϕ function is used above, an adversary might be able to launch a man-in-the-middle insider attack. The attacker's goal would be that the victim's answer is accepted by the reader if and only if the quartic polynomial contains the adversary's solution as well as the victim's which would identify the victim to the adversary.

5.5.3 The basic protocol

For simplicity, we first demonstrate how the regular Cramer-Shoup scheme can be used to implement the protocol. In the next section we replace the cryptographic hash function by elliptic-curve point operations to obtain a purely elliptic-curve-based protocol.

Let ID_T be a tag T's identity, encoded as a randomly chosen bit string of length $\frac{1}{2}(n-k)$, where k is the security parameter associated with the ϕ function. The basic protocol now runs as follows. The reader challenges the tag with a randomly generated bit string N of length $m = \frac{1}{2}(n-k)$. The tag concatenates its identity ID_T with the challenge string N and applies the ϕ function to obtain the point $M = \phi(ID_T, N)$ on the elliptic curve. Thus, the tag sends $rP_1, rP_2, rH + \phi(ID_T, N), rC + r\alpha D$ to the reader. The reader accepts the tag if the response verifies correctly. The protocol is depicted in Figure 5.14 (left).

5.5.4 A purely elliptic-curve-based solution

We now use the hash-free variant of the Cramer-Shoup scheme [CS98, Section 5.3] to implement a purely elliptic-curve-based protocol. Recall that $\mathcal{E}$ is an elliptic

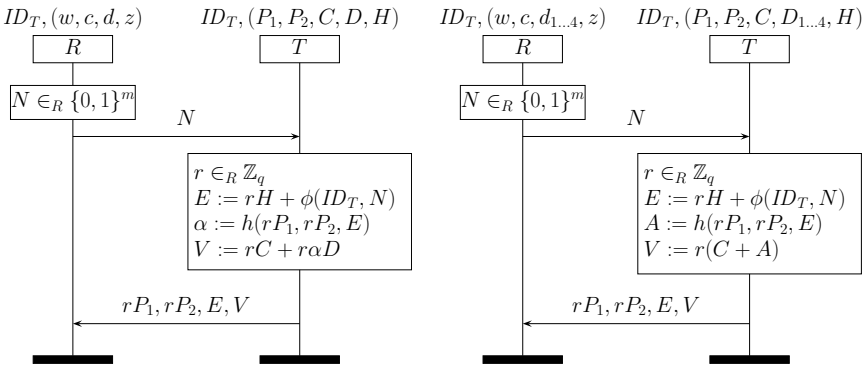


Figure 5.14: Elliptic-curve-based protocol with a cryptographic hash function (left) and with an elliptic-curve-based hash function (right).

curve over a finite field $\mathbb{F}_{2^n}$ such that it contains a subgroup $\mathcal{G}$ of order p, where p is a large prime.

Let $P \in \mathcal{G}$ and $c, d_1, \ldots, d_4, w, z \in \mathbb{Z}_p$ be randomly chosen, system-wide parameters. Set $C = cP$, $D_i = d_i P$ for $1 \leq i \leq 4$, and $H = zP_1$. Then the reader's public key is $(P_1, P_2, C, D_1, \ldots, D_4, H)$ and its secret key is $(c, d_1, \ldots, d_4, w, z)$. Encryption and decryption are as in the regular scheme, but the value αD is replaced by $A = h_{\mathcal{E}}(U_1, U_2, E)$, where $h_{\mathcal{E}}$ is a function whose range is a subset of the elliptic curve $\mathcal{E}$. It remains to define the function $h_{\mathcal{E}}$.

The hash function $h_{\mathcal{E}}$. The hash function in the encryption scheme needs to hash three points on the elliptic curve onto a single point in a collision-resistant manner. Let $x(Q)$ be the x-coordinate of a point Q and $y(Q)$ be its y-coordinate. In characteristic 2, a pair $(x(P), y(P))$ is a point on the elliptic curve if and only if $(x(P), x(P) + y(P))$ is [Ser98]. Thus for each point on the curve, given its x-coordinate, there are only two possible y-coordinates. Therefore, only one bit is needed to encode the y-coordinate.

Define $h_{\mathcal{E}}(X_1, X_2, X_3)$ as follows. For $i = 1, 2, 3$, let s_i be the $n + 1$-bit strings obtained from the $n + 1$-bit encoding of the points $X_1, \ldots, X_3$. We then split the string s_1, s_2, s_3 into the four $\lfloor \log_2 p \rfloor$-bit strings $a_1, \ldots, a_4$. Then $h(X_1, X_2, X_3) = \sum_{i=1}^{4} a_i D_i$.

Let ID_T be a bit string of length $\frac{1}{2}(n - k)$. The hash-free variant of the protocol is depicted in Figure 5.14 (right). The protocol can be implemented with 8 point multiplications, 5 point additions, 2^{k+1} computations of the trace function of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, and one square root computation over $\mathbb{F}_{2^n}$.

Correctness, security, and privacy Correctness of the hash-free scheme follows immediately from correctness of the Cramer-Shoup encryption scheme [CS98]. Privacy and security follow Vaudenay [Vau07] and Hermans et al. [HPVP11] and the IND-CCA2 security of the hash-free Cramer-Shoup encryption scheme.

5.5.5 Practicality

The protocol presented in the preceding section cannot be considered practical for most applications. There are several aspects to our approach that could be attempted in a different manner. Our current solution employs four elliptic curve point multiplications to implement the "hash-free" collision resistant function suggested by Cramer and Shoup [CS98]. The main reason for using a purely elliptic-curve based function is that existing circuits can be reused.

If we allow for hybrid encryption approaches, then a particularly efficient solution would be the OTP-PSEC-3 encryption scheme [OP00]. This scheme uses two elliptic curve point multiplications and two hash function applications. The scheme has been shown to be IND-CCA2 secure in the random oracle model and based on the elliptic curve gap Diffie-Hellman assumption. One of the two hash functions takes as input a random bitstring, the other takes two bitstrings and two points on the elliptic curve. Using the methods above to produce a purely elliptic-curve based solution, it can be easily seen that the number of point multiplications is at least as large as in our solution.

5.6 Conclusion

There are two types of computational proof models for untraceability of RFID protocols: indistinguishability-based proof models and unpredictability-based proof models. We have shown that the unpredictability based definitions by Ha, Moon, Zhou, and Ha [HMZH08] and by Ma, Li, Deng, and Li [MLDL09] do not coincide with the intuitive notion of untraceability. Specifically, for both models we have given protocols that should be untraceable but can be proven to be traceable. Conversely, we have also given protocols that are traceable but can be proven to be untraceable in both models.

We have argued that insider attacks are a plausible and important class of attacks, relevant for wide adversaries. In Vaudenay's model, insider attacks are not naturally represented, but can be modeled by assuming a destructive adversary. This is, however, an unreasonable over-approximation of the powers of an attacker who cannot corrupt tags, but who does have means to introduce insiders in the system. We have constructed a new class of adversaries modeling insider attackers. We have shown its relevance by showing separations with existing adversary classes in Vaudenay's model. Using the intuition behind insider attacks, we have shown that the eight privacy classes introduced by Vaudenay do not collapse into three, as was suggested by Ng, Susilo, Mu, and Safavi-Naini [NSMSN08].

As a final contribution, we have constructed the first provably secure and wide-forward private RFID protocol based only on elliptic-curve operations.

Part II

RFID protocols and authentication

6

Authentication attacks

Authentication is a security property that is not specific to RFID protocols. In fact, various frameworks for proving authentication have been developed, e.g. BAN logic [BAN90], strand spaces [THG99], and the applied pi calculus [Bla01]. Furthermore, several automatic tools for verifying authentication have been developed, e.g. AVISPA [ABB+05], Scyther [Cre06b], ProVerif [Bla01], and Casper/FDR [Low98]. For this reason, in this chapter, we focus on authentication problems that are specific to RFID protocols.

The main difficulty in designing RFID protocols that satisfy authentication is that the protocols run in a resource-constrained environment. RFID protocol designers often assume that it is not possible to implement cryptographic hash functions or public key cryptography on a tag. Instead, they design protocols using operators such as exclusive or, bitwise shifting, modular addition, and logical *and* and *or*. The advantage of using these operators is that the footprint of the chip implementing the RFID protocol is much smaller. The disadvantage is that it becomes much harder (if not impossible) to formally prove security requirements of the protocol. Furthermore, attack-finding using existing tools becomes harder since they cannot deal with these operators.

The purpose of this chapter is to show the different types of authentication flaws from which current RFID protocols suffer. To this end we first discuss different definitions of authentication and argue when they should be used. We then describe three types of flaws and for each of these types illustrate attacks on protocols from literature. The main theme in these attacks is that the attacker abuses some algebraic property of the messages being exchanged between reader and tag.

This chapter is of a very similar nature as the chapter on untraceability attacks (Chapter 4). It also serves as a reference to protocol designers showing common mistakes in designing authentication protocols. Like in Chapter 4 we do not give explicit protocol traces of the attack. We thus give a description of the attack that is sufficient to reconstruct attack traces.

6.1 Forms of authentication

One of the most common requirements on security protocols is *authentication*. The goal of an authentication protocol is to convince one agent of the identity of another. Authentication is a *local* property: a protocol between a reader and a tag may, for instance, provide tag-to-reader authentication, but not reader-to-tag authentication. In most cases, we are mainly concerned with tag-to-reader authentication, but some applications also require reader-to-tag authentication. If

a protocol satisfies both, we say the protocol satisfies *mutual authentication*.

For most applications, assurance of the identity of the communicating party is too weak as an authentication requirement. Consider an RFID system used to unlock garage doors. An authentication protocol would ensure that at the end of the protocol execution, the RFID reader is convinced that he communicated with the valid tag. There is no guarantee, however, that the RFID tag executed his run of the protocol at the same time. An attacker could have captured messages generated by the tag and replayed them to the reader. For this reason, we consider *recentness* of the tag's protocol execution to be essential. Recentness can be formalized by requiring that between the start and the end of the run, the communicating party has generated a message. We consider *recent aliveness* to be a minimal requirement for tag-to-reader authentication.

Definition 6.1 (Recent aliveness [Low97])**.** *A protocol guarantees to an agent a* recent aliveness *of an agent b if, whenever a completes a run of the protocol, apparently with b, then b has been running the protocol during a's run.*

This definition has been formalized in different ways. For instance, Lowe [Low97] defined the property in CSP and Cremers [CM05, Cre06b] gave a definition in a trace model similar to the one developed in Chapter 3.

If an RFID tag implements more functionality than mere identification and authentication of objects, the RFID protocol may have to provide stronger guarantees of authenticity. These RFID tags can be found in applications such as electronic transportation cards, electronic payment cards, and e-passports. In all these applications data is transferred between reader and tag. The integrity of this data must be protected during transmission. This property is formalized by requiring *agreement* on the data: at the end of their protocol executions the agents agree on the values of the data exchanged in the protocol.

Definition 6.2 (Agreement [Low97])**.** *A protocol guarantees to an agent a in role A* agreement *of an agent b in role B on a set of data items ds if, whenever a completes a run of role A, apparently with b, then b has been running the protocol with a. Furthermore, a and b agree on the values of ds and each run of a corresponds to a unique run of b.*

The requirement of unique correspondence of runs is necessary to prevent replay attacks. It ensures that the messages of one run of an agent cannot be reused to complete another run. Most RFID applications allow agents to execute only one role: a tag can only execute the tag role and a reader can only execute the reader role. However, some applications, such as near-field communication (NFC) allow agents to act both as initiator and as responder in the protocol.

The formalization of the above definition has been carried out in different ways. Lowe defined the requirement in CSP [Low97], Cremers in the aforementioned trace model [CM05, Cre06b], and Blanchet in the applied-pi calculus [Bla07]. In the computational setting, it is often formalized by the concept of matching conversations [War03]. The definition of *soundness* by Ma, Li, Deng, and Li (Definition 5.4, Section 5.2.2) and the definition of *security* by Vaudenay (Definition 5.17 , Section 5.3.1) are both computational definitions of agreement.

6.2 Algebraic replay attacks

A common way to authenticate RFID tags is by means of the challenge-response mechanism depicted in Figure 6.1. The RFID reader challenges the tag with a nonce c. The tag generates a nonce r and derives a response from c, r, and some shared information s that identifies the tag. The nonce r is not necessary for authentication, but frequently ensures indistinguishability of tag responses. In case of a mutual authentication protocol, it may serve as a challenge to the reader. We can thus represent the tag's response to the reader's challenge as the term $r, g(c, r, s)$ with the understanding that r may be constant or empty. The reader verifies the authenticity by applying the inverse of the function g to the term and checking whether the response contains c and a valid s. If g is a one-way function then the reader verifies the authenticity of the tag by computing the function $g(c, r, s)$ and comparing it to the received value. The reader can compute this function, since it generated the value c itself, the value r is supplied by the tag, and the reader has a database with values of s for every tag it may authenticate.

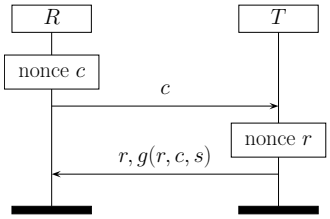


Figure 6.1: Basic challenge-response structure

The following two properties are necessary in order for the challenge-response mechanism to guarantee authentication of the tag.

Freshness For fixed r and s the range of the function $c \to g(c, r, s)$ must be large. More precisely, for r and s, the adversary's advantage in guessing $g(c, r, s)$ correctly for a randomly chosen c must be negligible.

Algebraic replay resistance (ARR) Let $O_s(x)$ be an oracle which upon input x randomly chooses y and returns y and $g(x, y, s)$. If s is unknown, then given access to a polynomial number of oracle outputs $O_s(x_1), \ldots, O_s(x_l)$, it is infeasible to compute $g(c, r, s)$ for a given $c \notin \{x_1, \ldots, x_l\}$ and any choice of r.

If freshness is satisfied, then the probability of the adversary guessing $g(c, r, s)$ is negligible. Thus with overwhelming probability, a response $r, g(c, r, s)$, to the reader's challenge c must have been generated *after* the challenge was sent. This property is necessary for recentness and in particular excludes classical replay attacks.

Algebraic replay resistance (ARR) guarantees that there is no efficient algorithm to compute a response $r, g(c, r, s)$ to the challenge c even after having observed polynomially many challenge-response pairs. An attacker's ability to compute such a response violates authentication. Such an attack generalizes replay attacks in that instead of merely replaying previously observed messages, the attacker

combines previously obtained challenge-response pairs to compute the response to a fresh challenge. Hence, we refer to attacks on challenge-response authentication protocols exploiting the lack of the ARR property as *algebraic replay attacks*.

For a function $g(c, r, s)$ to have the ARR property, it must preserve the secrecy of s. Indeed, cryptographic hash functions are frequently used for the type of challenge-response mechanism considered here. Since the collision resistance property of cryptographic hash functions does not seem necessary for the challenge-response mechanism, the question arises whether all one-way functions satisfy the ARR property. In general, this is not true for homomorphic one-way functions. The following example shows the case of the Rabin function.

Example 6.3 (Rabin function does not satisfy ARR)**.** *Let N be the product of two large primes. The Rabin function $f(x)$ is defined by $f(x) = x^2 \bmod N$. Let the function g be defined by $g(c, r, s) = (c \cdot r \cdot s)^2 \bmod N$. Then given a single challenge-response pair $(c, g(c, r, s))$ one can compute g for any challenge c': $g(c', r, s) = g(c, r, s) \cdot (c'/c)^2$. Therefore, the protocol in Figure 6.1 instantiated with the Rabin function is not resistant to algebraic replay attacks.*

Even non-homomorphic one-way functions in general do not have the ARR property if their *argument* has algebraic properties. We show that several protocols fail to achieve authentication for this reason. In these protocols the challenge-response construction can typically be represented as $g(c, r, s) = f(c \circ r, s)$, where f is a (non-homomorphic) cryptographic hash function and $\circ$ denotes an operator with the following algebraic property. Given a, b, and c, it is easy to find d with $a \circ b = c \circ d$. This construction does not have the ARR property, regardless of the properties of f. The algebraic replay attack on such a protocol works as follows. An adversary observes one execution of the protocol and learns c, r, and $f(c \circ r, s)$. When challenged with c', the adversary finds r' such that $c \circ r = c' \circ r'$ and responds with $r', f(c \circ r, s)$. The attack succeeds because $f(c \circ r, s) = f(c' \circ r', s)$.

Examples of operators $\circ$ for which this type of attack succeeds are bitwise exclusive or, bitwise disjunction/conjunction, and modular addition/multiplication.

The EC-RAC I protocol discussed in Chapter 4, is vulnerable to an algebraic replay attack in which the adversary needs to observe three protocol executions. The algebraic replay attack can then be executed by solving a small system of equations yielding a constant particular to the tag. While this constant does not reveal the tag's secret information, it can still be used to compute the correct response to a reader's challenge. This attack has been first described by Bringer, Chabanne, and Icart [BCI08].

6.2.1 The Chien and Huang protocol

The protocols by Chien and Huang [CH07], Kim, Choi, Lee, and Lee [KCLL06], Lee, Asano, and Kim [LAK06], Cai, Zhan, and Wang [CZW08] and Song and Mitchell [SM08] all suffer from algebraic replay attacks. These attacks abuse the fact that a hash-like function or a cryptographic hash function is composed with *xor* and fit into the challenge-response construction with the function $f(c \circ r, s)$ shown above.

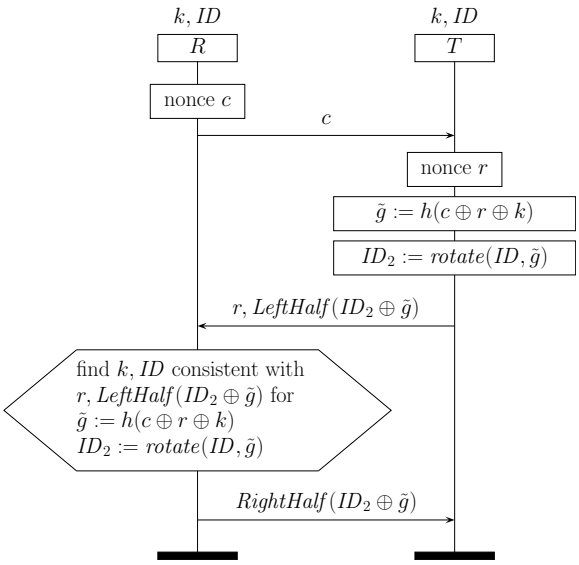


Figure 6.2: The Chien and Huang protocol

We illustrate a complete attack on the protocol proposed by Chien and Huang [CH07], depicted in Figure 6.2. The reader R and tag T share secrets k and ID. The reader starts by sending a random bit string c. The tag generates a random string r and hashes the *xor* of c, r, and the secret k. This hash and ID are used as input for a function in which the ID is rotated by a value depending on the hash. The tag computes the *xor* of the rotated ID and the hash, before sending the left half of the resulting bits and r to the reader. The reader performs the same operations on every pair of ID and k until it finds the corresponding tag. It then sends the right half of the corresponding bits to the tag.

To impersonate a tag, it suffices to notice that the tag's response to the reader's challenge only depends on $c \oplus r$ and a shared secret. The composition of functions applied to the *xor* and shared secret can be represented by the function f, defined above. Thus, the adversary can challenge a tag with any c to obtain a valid combination of c, r, and $Left(ID_2 \oplus \tilde{g})$. This information suffices for the adversary to be able to respond to any future challenge c' received from a reader. When challenged, the adversary sets $r' = c' \oplus c \oplus r$ and sends $r', Left(ID_2 \oplus \tilde{g})$.

6.3 Compositionality attacks

We revisit protocol 4 of EC-RAC II of Section 4.5 (see Figure 6.3). The attack described in Section 4.5 shows that there exists a compositionality attack on untraceability of the protocol. We here show that there also exists a compositionality attack on authentication of the protocol.

In order to break tag authentication in the protocol, an adversary needs to know xY. Recall that an adversary can obtain the multiplication of any nonzero point X by the reader's secret y. In the following we write this loop oracle as the function $O(X) = yX$. To use the loop O in protocol 4 for this purpose, the adversary sends a nonzero point $T_1 = X$, along with a random point T_2 to the reader. The reader replies with a random s and $yT_1 = yX$, the scalar multiplication of the reader's

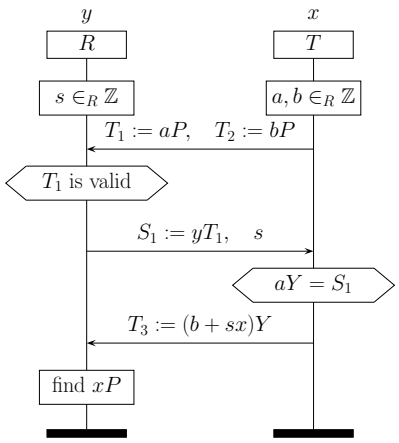


Figure 6.3: Protocol 4 of EC-RAC II

secret key y and the point X.

By eavesdropping on one communication between a tag and a reader, an attacker obtains bP, the challenge s, and $(b + sx)Y$. He then computes $s^{-1}bP$ and $s^{-1}(b + sx)Y = (s^{-1}b + x)Y$. Using the oracle, the attacker obtains $O(s^{-1}bP) = s^{-1}bY$ and computes the difference $(s^{-1}b + x)Y - s^{-1}bY = xY$.

After learning xY the adversary can impersonate a tag as follows. The attacker chooses a random integer r and submits rP, rP to the reader. The reader responds with rY and a challenge s. To answer this challenge, the attacker responds with $sxY + rY = (r + sx)Y$.

6.4 Leakage attacks

The authentication and untraceability properties of RFID protocols often rely on the secrecy of shared keys. In some cases, revealing parts of a secret key may already be enough to trace the tag. If sufficiently many bits of a key can be revealed, brute-forcing the remaining bits may become feasible. This allows an attacker to impersonate the tag to a reader and break authentication.

In an effort to keep RFID tags cheap, a number of lightweight protocols have been proposed. These protocols do not use standard cryptographic primitives, but instead rely on simple operators such as modular addition and rotation. A natural point of attack is to set up equations involving the terms on whose secrecy the protocol depends. Such equations may be obtained by observing several protocol runs. Alternatively, one could modify parts of messages and observe the responses generated by reader or tag. One may then attempt to apply any known cryptanalytic method to recover (parts of) the shared secret. Recovering the shared secret immediately breaks untraceability and authentication.

We show two examples of protocols that can be broken with simple cryptanalytic methods.

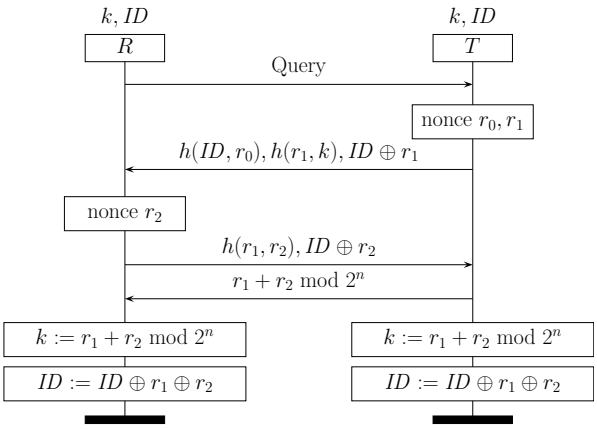


Figure 6.4: The Kang and Nyang protocol.

6.4.1 The Kang and Nyang protocol

We consider the protocol by Kang and Nyang [KN05] that uses modular addition, exclusive or, and hash functions. The tag initiates the protocol by generating a random value r_0 from a small domain and a random value r_1 of length n. The tag sends the two hashes $h(ID, r_0)$, $h(r_1, k)$ and $ID \oplus r_1$ to the reader. Using $h(ID, r_0)$, the reader finds ID by trying out all combinations of values for ID stored in its database and of all possible values for r_0. This is possible for the reader because r_0 is chosen from a small domain and the number of IDs stored in its database is very small compared to the number of possible IDs. Using ID the reader retrieves k from its database, and using $ID \oplus r_1$ and ID, the reader finds r_1 and may then verify the correctness of the value of $h(r_1, k)$. The reader then generates a random value r_2 of length n and sends $ID \oplus r_2$ and $h(r_1, r_2)$ to the tag. The tag verifies these and sends $r_1 + r_2 \mod 2^n$ back to the reader. Both tag and reader update the ID by *xor*-ing it with $r_1 \oplus r_2$. The protocol is depicted in Figure 6.4.

Since hash functions are assumed to be perfect, we consider the terms $ID \oplus r_1$, $ID \oplus r_2$, and $r_1 + r_2 \mod 2^n$, setting up a system of equations involving the variables ID, r_1, r_2, and the values observed during runs of the protocol. A moment's thought shows that we may combine the first two equations to obtain $r_1 \oplus r_2$.

For convenience, we set $V = r_1 + r_2 \mod 2^n$ and $W = r_1 \oplus r_2$. Let $V[i]$ be the i-th bit of V, and similarly for W, r_1, and r_2. Furthermore, let $V[1]$ be the least significant bit of V. By comparing addition modulo 2^n with *xor* it is easy to see that $V[i+1] \neq W[i+1]$ only if there is a carry bit in the computation of $V[i]$. If this is the case, then $r_1[i] \neq r_2[i] \Leftrightarrow W[i] = 1$ and $r_1[i] = r_2[i] = 1 \Leftrightarrow W[i] = 0$.

Since the latter case determines $r_1[i]$ and $r_2[i]$ uniquely, it follows that it can be used to find the i-th bit of ID. More bits from ID can be obtained by noticing that a carry bit in $V[i]$ followed by no carry bit in $V[i+1]$ implies $r_1[i+1] = r_2[i+1] = 0$.

Since r_1 and r_2 are chosen at random, on average, every communication session leaks roughly $\frac{n-1}{4}$ bits of ID. Revealing all bits of ID, once sufficiently many bits are known, can be achieved with a brute-force search over possible values for ID and r_0 and comparing their hash to $h(ID, r_0)$. Revealing all bits of ID is made a little more complicated by the fact that reader and tag update ID at the end of every protocol execution by setting it to $ID \oplus r_1 \oplus r_2$. The adversary may therefore

need to keep track of two or three consecutive protocol executions between the tag and reader before performing the exhaustive search in order to completely reveal the tag's ID. Knowing the ID, the adversary can impersonate both tag and reader and trace the tag.

6.4.2 The Di Pietro and Molva protocol

In this section, we show a passive attack[1] on the Di Pietro and Molva protocol discussed in Section 4.3.1. The protocol is depicted in Figure 6.5.

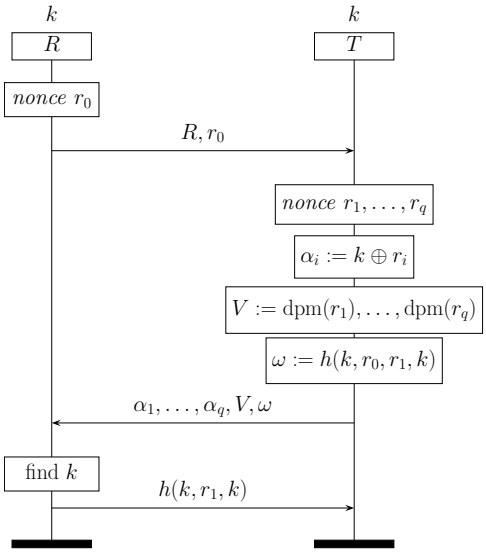


Figure 6.5: The Di Pietro and Molva protocol

For ease of notation, we represent bitstrings as vectors over the finite field with two elements $\mathbb{F}_2$. The majority function $M : \mathbb{F}_2^3 \to \mathbb{F}_2$ and the function $\mathrm{dpm} : \mathbb{F}_2^\ell \to \mathbb{F}_2$ are then defined as follows:

$$M(x, y, z) = xy + yz + xz$$
$$\mathrm{dpm}(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell/3} M(x_{3i-2}, x_{3i-1}, x_{3i})$$

Our goal is to recover the key k from a collection of α_i's and corresponding $V[i]$'s generated by the same tag. To this end, we set up a system of linear equations involving α_i's and $V[i]$'s that when solved yields the key k.

We first observe that for $(a, b, c), (x, y, z) \in \mathbb{F}_2^3$,

$$M(a + x, b, c) = ac + bc + ab + cx + bx$$

with analogous equations for $M(a, b + y, c)$ and $M(a, b, c + z)$. Furthermore, we have

$$M(a+x, b+y, c+z) = M(a+x, b, c) + M(a, b+y, c) + M(a, b, c+z) + M(x, y, z).$$

[1]The attack and analysis are due to Saša Radomirović.

It follows that

$$M(a+x,b+y,c+z) = M(a,b,c) + M(x,y,z) + a(y+z) + b(x+z) + c(x+y)$$

which after reordering we write as

$$(y+z, x+z, x+y) \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = M(a+x,b+y,c+z) + M(a,b,c) + M(x,y,z). \quad (6.1)$$

For convenience, we define the function cross : $\mathbb{F}_2^\ell \to \mathbb{F}_2^\ell$:

$$\begin{aligned}\text{cross}(x_1, y_1, z_1, \ldots, x_{\ell/3}, y_{\ell/3}, z_{\ell/3}) = \\ (y_1+z_1, x_1+z_1, x_1+y_1, \ldots, y_{\ell/3}+z_{\ell/3}, x_{\ell/3}+z_{\ell/3}, x_{\ell/3}+y_{\ell/3}).\end{aligned}$$

The cross function is linear and it follows that $\text{cross}(r_1)+\text{cross}(r_2) = \text{cross}(r_1+r_2) = \text{cross}(\alpha_1+\alpha_2)$. We now represent $\text{cross}(r_i)$ as a row vector and transpose the key vector k. From Equation (6.1) and the definition of dpm we get

$$\text{cross}(r_1) \cdot k^T = \text{dpm}(k+r_1) + \text{dpm}(k) + \text{dpm}(r_1) \quad (6.2)$$
$$\text{cross}(r_2) \cdot k^T = \text{dpm}(k+r_2) + \text{dpm}(k) + \text{dpm}(r_2) \quad (6.3)$$

If we add Equations (6.2) and (6.3) we obtain

$$\text{cross}(r_1+r_2) \cdot k^T = \text{dpm}(\alpha_1) + \text{dpm}(\alpha_2) + \text{dpm}(r_1) + \text{dpm}(r_2). \quad (6.4)$$

For $i = 2, \ldots, \ell+1$, let the $\ell \times \ell$ matrix A be given by the row vectors $\text{cross}(\alpha_1+\alpha_i)$ and let the column vector v be given by the entries $\text{dpm}(\alpha_1)+\text{dpm}(\alpha_i)+\text{dpm}(r_1)+\text{dpm}(r_i)$. Consider then the linear equation $Ax = v$, *viz.*

$$\begin{pmatrix} \text{cross}(\alpha_1+\alpha_2) \\ \text{cross}(\alpha_1+\alpha_3) \\ \vdots \\ \text{cross}(\alpha_1+\alpha_{\ell+1}) \end{pmatrix} x = \begin{pmatrix} \text{dpm}(\alpha_1)+\text{dpm}(\alpha_2)+\text{dpm}(r_1)+\text{dpm}(r_2) \\ \text{dpm}(\alpha_1)+\text{dpm}(\alpha_3)+\text{dpm}(r_1)+\text{dpm}(r_3) \\ \vdots \\ \text{dpm}(\alpha_1)+\text{dpm}(\alpha_{\ell+1})+\text{dpm}(r_1)+\text{dpm}(r_{\ell+1}) \end{pmatrix}$$

By construction, the vector $x = k^T$ is a solution of the linear equation. Any y such that $Ay = 0$ (i.e. any y in the null space

of A) yields a solution $k^T + y$ to the equation. Thus, the null space of A can be considered the adversary's uncertainty about k. From the definition of the cross function, one can see that the null space of A contains the $\ell/3$ vectors

$$(1,1,1,0,\ldots,0)^T, (0,0,0,1,1,1,0,\ldots,0)^T, \ldots, (0,\ldots,0,1,1,1)^T. \quad (6.5)$$

We now show that the null space of A is actually spanned by these vectors whenever A is constructed from linearly independent vectors $\alpha_1, \ldots, \alpha_{\ell+1}$. Therefore, if one solution x is found, *all* others can be constructed by adding (a subset) of the vectors in (6.5). No other solutions exist. In other words, the adversary can learn all bits of k up to complements of $\ell/3$ consecutive bit-triplets.

Theorem 6.4. *If $\alpha_1, \ldots, \alpha_{\ell+1}$ are linearly independent, then the null space of A is spanned by the vectors in* (6.5)*. Equivalently, the number of linearly independent rows (i.e. the rank) of A is $\frac{2}{3}\ell$.*

Proof. We know that the $\ell/3$ vectors listed in (6.5) are in the null space of A. Since they are linearly independent, the rank of A is at most $\frac{2}{3}\ell$.

Conversely, consider the matrix $\tilde{A}$ obtained from A by deleting every third column of A. We construct a matrix B consisting of the rows $\alpha_1 + \alpha_2, \ldots, \alpha_1 + \alpha_{\ell+1}$. By construction, the vectors in B are linearly independent and, therefore, B has rank ℓ. We now construct $\tilde{B}$ from B by adding every third column to the preceding two columns and swapping those preceding two columns. The rank of $\tilde{B}$ is equal to the rank of B. Now, $\tilde{A}$ can be obtained from $\tilde{B}$ by deleting every third column of $\tilde{B}$. Since deleting a column of a matrix decreases its rank by at most 1, and $\tilde{B}$ had rank ℓ, the rank of $\tilde{A}$ is at least $\frac{2}{3}\ell$. Thus, the rank of A is at least $\frac{2}{3}\ell$. □

The theorem assumes that the matrix A contains ℓ linearly independent rows. A legitimate tag generates r_i's at random, thus if the r_i's are linearly independent, then so are the α_i's.

To estimate the number of r_i's needed to obtain a matrix of rank n, we use a theorem about the rank of a random $(n + s) \times n$ matrix M by Cooper [Coo00, Theorem 1]. The theorem states that the probability p_n that the rank of M equals n converges to

$$\lim_{n\to\infty} p_n(s) = \prod_{j=s+1}^{\infty} \left(1 - \left(\frac{1}{2}\right)^j\right). \tag{6.6}$$

Equation (6.6) gives a lower bound on the probability that a random $(n + s) \times n$ matrix is of rank n. For instance, for $s = 1$ the probability is greater than 0.58 and for $s = 7$, the probability is greater than 0.99.

It thus follows that we need at most $\ell + 7$ random vectors to obtain ℓ linearly independent vectors with probability 0.99. Hence, after roughly $(\ell + 7)/q$ protocol executions, the adversary is able to compute the secret key of the tag up to complements of consecutive bit-triplets with high probability. As discussed, the protocol designers suggest for a system with 2^{16} tags, the value ℓ to be 117 and q to be $2 \log n$. For these values, the attacker needs to observe (or execute) $\lceil (117 + 7)/32 \rceil = 4$ protocol executions with the tag. The attacker can then with high probability efficiently recover the key up to complements of consecutive bit-triplets.

The procedure outlined above reduces the complexity of computing the secret key of a tag to a brute force search of a space with $2^{\ell/3}$ elements. Each of the candidate keys k in this space can be verified by computing the hash $h(k, \alpha_1 \oplus k, k)$ and comparing the value against the third message of the protocol. For the suggested parameter $\ell = 117$ this brute-force search is feasible in practice. If the key is recovered, the adversary can successfully complete a protocol execution with the reader violating authentication.

6.5 Conclusion

Authentication is a reasonably well-studied and well-understood security property in security protocol design. In RFID authentication protocols, the majority of the flaws are of an algebraic nature. That is, the attacker can abuse algebraic properties of the operators in the protocol to construct new valid messages. In standard protocol analysis, these algebraic properties often do not play a role since cryptography is assumed to be perfect (see Chapter 3).

We have identified three classes of algebraic attacks on authentication of RFID protocols. In an algebraic replay attack, an attacker combines a number of challenge-response pairs to compute a response to a fresh challenge. Compositionality attacks combine messages from different protocols to break authentication of one of them. Finally, leakage attacks aim at recovering the secret key used in a protocol by setting up equations involving messages from the protocol. For each of these three categories, we have pointed out flaws in RFID protocols from literature.

In a case study [ICFEM], we have been able to automate the process of finding algebraic replay attacks on protocols with exclusive or (XOR). The approach we have used extends the verification method by Küsters and Truderung [KT08]. Their method reduces protocols with XOR to their XOR-free equivalents, enabling automatic verification by the ProVerif [Bla01] tool. By introducing bounded verification we have been able to analyze the protocols by Lee, Asano, and Kim [LAK06] and Cai, Zhan, and Wang [CZW08] and we have confirmed the presence of their algebraic replay attacks. We expect that the process of finding algebraic replay attacks based on the algebraic properties of other operators can be automated along the lines of this work.

Part III

RFID protocols and ownership transfer

7

Formalization of ownership transfer

Because of its large potential to save costs, RFID is becoming a key technology in supply chain management. RFID-tagged products can be scanned automatically without requiring line-of-sight. This ability significantly reduces labor-intensive tasks such as checking and scanning incoming and outgoing inventory [MM05]. Moreover, item-level tracking allows organizations to have an accurate view on their current stock levels. This enables organizations to reduce the size of their stock as well as preventing out-of-stock occurrences. Finally, since RFID tags can store and process information and execute simple communication protocols, they can be used to detect counterfeit products [STF05].

As products flow through a supply chain, they are processed and stored by multiple supply chain partners. Therefore, ownership of products changes during the life cycle of a product. This transfer of ownership extends to the RFID tags attached to these products. This means that at some point in time a supply chain partner owns the products and RFID tags legally, by means of a title, and physically by the fact that the goods are at his premises.

Ownership of an RFID tag allows one to interact with it and use its functionality. This functionality could, for instance, be to obtain the identifier of a product for tracking purposes. Other examples include access to the data stored on the tag or the ability to transfer ownership of the tag to another party. Note that we do not require a tag to have exactly one owner at all times: tags could have multiple owners or even zero owners.

Since ownership of a tag implies access to tag functionality, ownership changes must be sufficiently protected. For instance, since owning a tag means being able to trace the tag, previous owners must be properly "disowned". The purchaser of a product may not want previous owners (of the tag) such as the manufacturer, retailer, and post office to be able to trace him. A second reason to protect ownership of tags is that they may provide access to valuable data. This data could be stored on the tag, but could also be obtained from other sources such as sensors attached to the tag [Wan04]. A third reason to protect the process of ownership transfer is to prevent denial-of-service attacks. If an attacker can steal ownership of tags, he could critically disrupt business processes relying on a functioning RFID system. In extreme cases, an attacker could extort a supply chain partner by grabbing ownership of a batch of tags and only releasing them after his demands are fulfilled.

The common way of transferring ownership of a tag from one party to another is by means of an *ownership transfer protocol*. One of the most basic requirements in an RFID system with ownership transfer protocols is that of *secure ownership*. Secure ownership ensures that an agent's ownership can only be transferred to

another agent, but can never be "stolen" from him. As a second requirement, *Secure ownership transfer* states that if an agent becomes owner of a tag, it must be as a result of the execution of an ownership transfer protocol.

In this chapter, we formalize the concept of ownership as well as its corresponding security requirements using the formal model introduced in Chapter 3. We find several attacks on ownership-related requirements by applying our definitions to existing RFID protocols. Finally, we discuss the connection between ownership and desynchronization resistance and give a formal definition of the latter.

7.1 Preliminaries

Our formalization is based on our model designed in Chapter 3. To be able to give a formal definition of ownership transfer protocols, we first introduce and define the notion of sequential composition of protocols.

We define the sequential composition $P \cdot Q$ of two protocols P and Q by concatenating their event sequences for every role R. Let N and N' be sets of nonces, T and T' be sets of temporary variables, P and P' be sets of permanent variables, and Ev and Ev' be lists of events. Without loss of generality, we may assume that the nonce sets and temporary variable sets are pairwise distinct. Let role R of protocol P be defined by $P[R] = (N, T, P, Ev \cdot \bot)$ and let role R of protocol Q be defined by $Q[R] = (N', T', P', Ev' \cdot \bot)$. We define the sequential composition of P and Q for role R by $(P \cdot Q)[R] = (N \cup N', T \cup T', P \cup P', Ev \cdot Ev' \cdot \bot)$. If role R is not defined for protocol P (or Q) then we define $(P \cdot Q)[R] = Q[R]$ (or $(P \cdot Q)[R] = P[R]$).

7.2 Ownership

We consider two views on tag ownership. The first view, which we call the *system view*, defines ownership of a tag as the *ability* to interact with the tag in a predefined manner. Ownership of a tag can, for instance, be defined as an agent's ability to inspect the tag's identifier. The second view is called the *agent view*. It is based on the fact that each agent records in a local data structure the tags it *believes* to be the owner of. It is important to note that the agent view can only change due to an action of that particular agent, while the system view can change due to an action of *any* agent in the system. We define consistency between the two views as a security requirement.

7.2.1 System view of ownership

Ownership test protocol

Central to our definitions of ownership and ownership transfer is the notion of an *(ownership) test protocol*. A test protocol defines a sequence of actions that an owner of a tag must be able to carry out successfully. If an agent can execute the test protocol, then he has access to the tag and can thus be considered to be owner

of the tag. This leads to a natural definition of tag ownership as the ability to successfully execute the test protocol.

In a typical setting, the test protocol involves a reader and a tag. However, we do not exclude the possibility of having (trusted) third parties in the test protocol. We stress that the test protocol is a protocol that is not necessarily implemented on the tag. It merely represents the protocol designer's idea of what constitutes an owner of a tag. In this sense it is used by the protocol verifier as a tool to provide an objective statement about whether an agent owns a tag. As a consequence, in every state of the system, the ownership relation between tags and other agents is precisely defined.

Since we define ownership properties relative to a test protocol, the choice of a proper test protocol is an important step in all verification efforts. In some contexts the knowledge of a tag key may be the defining notion of ownership. In this case, a simple proof-of-knowledge protocol would be a suitable test protocol. If a tag implements multiple protocols including an ownership transfer protocol, then frequently only an owner is allowed to transfer ownership to another agent. In this setting, the ownership transfer protocol can be used as test protocol.

Choosing an insufficient test protocol may lead to ownership-related vulnerabilities being overlooked. A trivial example is the test protocol that can be successfully executed by any agent and thus declares everyone as the owner of a tag. This problem is, however, mitigated by the fact that an intuitive notion of ownership frequently coincides with the ability to complete an authentication protocol with a tag. In such cases, the authentication protocol can simply be taken to be the test protocol.

Micro traces: Testing for ownership

We define ownership as the *ability* to execute the test protocol. This means that the test protocol is not actually executed by the agents in the system. To test whether an agent owns a tag in state s, one can simply generate all traces starting from s. If the agent is owner of the tag then one of these traces must contain the successful execution of the test protocol.

To simplify the analysis we only consider a subset of the traces. In particular, we exclude all agents except for the testing agent and the other agents involved in the test protocol. Furthermore, we allow only one run to be created for every role. Finally, since ownership is a functional property of the system rather than a security requirement, we disallow the adversary from injecting, modifying, or blocking any messages. We call the resulting set of traces *micro traces*.

Let $P(R_1, R_2, \ldots, R_n)$ be a test protocol that tests whether an agent a_1 in role R_1 owns an agent a_2 in role R_2, involving a finite number of other agents $a_3 \ldots a_n$ in roles $R_3 \ldots R_n$. Let $s = \langle A, \sigma, I \rangle$ be a system state for which we want to test ownership of a_1 with respect to a_2. It is rather straightforward to adapt the framework introduced in Chapter 3 to define micro traces:

- The initial state s_0 is set to $\langle \emptyset, \sigma, \emptyset \rangle$ to maintain the persistent variable assignment of all agents.

- To enforce agents $a_1 \dots a_n$ to execute the correct role we set $Agent(R_i) = \{a_i\}$ for all $1 \leq i \leq n$.
- The create rule should prevent any role from being executed more than once. To this end we extend the temporary variable assignment with a variable rn that stores the role name. That is, $\{\theta \mapsto f\}$ is replaced by $\{\theta \mapsto f, rn \mapsto R\}$. We can now restrict the *create* rule by adding the conjunct $R \notin \{v(rn) \mid (f, n, e, v) \in A\}$ as a precondition, requiring that no role with the same name has been created yet.
- Since roles can be executed at most once, no run needs to be prematurely ended. Therefore, we remove the *end* rule from the semantics.
- The original *read* rule allows any term that can be deduced from the adversary knowledge to be readable. To enforce that the adversary merely forwards messages, we set the initial adversary knowledge to $\emptyset$ and replace $I \vdash m$ by $m \in I$.

Given these conditions, micro traces are defined in the same way as traces in Chapter 3:

$$\begin{aligned}\mu Traces_{P(a_1,\dots,a_n)} = \{(s_0, \dots, s_n, t_1, \dots, t_n) \mid\ & s_0 \dots s_n \in State,\\ & t_1, \dots t_n \in Label,\\ & \forall_{1 \leq i < n}\ s_i \xrightarrow{t_i} s_{i+1},\\ & s_0 = \langle \emptyset, \sigma, \emptyset \rangle\}.\end{aligned}$$

It now follows from the semantics that a run of a protocol ended successfully if its list of events contains only the symbol $\perp$. Formally, for a run $r = (f, n, e, v)$ we define $\mathsf{success}(r) \equiv e = \perp$.

Tag owner

We can now define tag ownership as a property on the micro traces of the test protocol. Informally, an agent r *owns* a tag t with respect to a test protocol P, if r and t have a way of successfully completing the protocol P. In this context, R is called the *owner* of T and T is called R's *property*. In case the test protocol P involves other agents than r and t, any agents that allow successful execution of the test protocol may be involved.

Definition 7.1 (Tag owner)**.** *For any state $s = \langle A, \sigma, I \rangle$, let $Active(s) = A$ denote the set of active runs. Ownership of agent r of tag t with respect to test protocol $P(R, T, O_1, \dots, O_n)$ is defined by*

$$\begin{aligned}\mathsf{owns}_P(r, t, s) \equiv \exists_{o_1 \dots o_n \in Agent}\ & \\ & \exists_{(s', t') \in \mu\mathsf{traces}_{P(r,t,o_1,\dots o_n)}(s)}\\ & \quad \forall_{a \in Active(s'_{|(s',t')|})}\ \mathsf{success}(a).\end{aligned}$$

We stress that our definition of ownership is not the definition of a security requirement. Instead, we use our notion of ownership as a basis to define security requirements such as secure ownership and secure ownership transfer.

7.2.2 Agent view of ownership

The tag owner definition precisely describes for each state of the system whether an agent owns a tag. It misses, however, the fact that the owner of a tag only has a partial view of the system. This partial view is entirely based on the messages sent and received in the agent's protocol executions. The agent's view is important when discussing the intention of an agent to engage in a transfer of ownership, i.e. the fact that the owner executes an ownership transfer protocol. Thus we introduce the agent's view regarding ownership of a tag by defining *tag holders*.

We model tag holding explicitly by requiring that agents store in their memory of which tags they are the owner. More specifically, we model whether an agent r holds a tag T with respect to test protocol P by a variable $holds(P, T)$. The value (*True* or *False*) is stored as a persistent variable and is thus accessible across protocol executions. We call an agent a *tag holder* if based on its protocol executions and local data structure, it believes to be the owner of a tag.

Definition 7.2 (Tag holder)**.** *Let $s = \langle A, \sigma, I \rangle$ be a system state. We define agent r to be the tag holder of a tag t with respect to test protocol P in system state s by*

$$\mathsf{holds}_P(r, t, s) \equiv \sigma_r(holds(P, T)) = True.$$

Our decision to require that agents keep track of which tags they believe to own has two advantages. First, we can let the protocol execution depend on the value of the *holds* variable. This allows us, for instance, to specify that an agent shall not transfer ownership of a tag, unless it actually holds the tag. Second, it simplifies modeling ownership-related properties that involve the belief of an agent's ownership.

7.2.3 Secure ownership

In an ideal world, the notions of tag owner and tag holder coincide. In general, tag ownership changes when a tag updates its knowledge, while a change in tag holder is carried out by a different agent. Since we assume communication to be asynchronous, it is impossible to guarantee that changes of tag owner and tag holder occur simultaneously.

We define *secure ownership* as a consistency requirement between the system view and the agent view of ownership. Since the tag may implement multiple protocols, we define our security requirements as properties of the set of protocols in the system. Formally, a set of protocols π can be modeled by one protocol P that contains the roles of all protocols in π. We say that a set of protocols provides secure ownership, if whenever an agent is holder of a tag it is also an owner of that tag. Note that we do not require the converse, i.e. owning a tag implies holding it, to be true. As a consequence, an agent may own a tag, yet not be aware about it. Secure ownership enforces that if an agent believes to be the owner of a tag, then this is indeed true. An agent can thus never falsely believe to be the owner of a tag and can thus never lose ownership unintentionally.

Definition 7.3 (Secure ownership)**.** *A set of protocols* Π *provides secure ownership with respect to test protocol* P *if and only if*

$$\forall_{(s,t)\in Traces(\Pi)}\ \forall_{0\leq i\leq|(s,t)|}\ \forall_{R,T\in Agent}\ \mathsf{holds}_P(R,T,s_i) \Rightarrow \mathsf{owns}_P(R,T,s_i).$$

Secure ownership guarantees that an owner cannot be "disowned" of a tag as long as he holds it. It does, however, *not* prevent other agents from owning the same tag simultaneously. We define *exclusive ownership* as the requirement that if an agent holds a tag, no other agent is owner of the tag. It is clear that in an environment where owners can trace tags, exclusive ownership is a necessary condition to satisfy untraceability against previous and future owners of tags.

Definition 7.4 (Exclusive ownership)**.** *A set of protocols* Π *provides exclusive ownership with respect to test protocol* P *if and only if*

$$\forall_{(s,t)\in Traces(\Pi)}\ \forall_{0\leq i\leq|(s,t)|}\ \forall_{R,T\in Agent} \mathsf{holds}_P(R,T,s_i) \Rightarrow \neg\exists_{R'\in Agent\setminus\{R\}}\ \mathsf{owns}_P(R',T,s_i).$$

It follows that secure ownership and exclusive ownership are incomparable security properties: one does not imply the other.

7.3 Ownership transfer

In this section we define the notion of an ownership transfer protocol and the natural security requirement for such a protocol. We call a protocol Q an *ownership transfer protocol* if it can be used to make somebody owner of a tag. That is, by executing Q an agent can become the owner of a tag that he currently does not own.

Definition 7.5 (Ownership transfer protocol)**.** *Let* P *be an ownership test protocol. We say that* $Q \in \Pi$ *is an ownership transfer protocol with respect to* P *if and only if*

$$\exists_{(s,t)\in Traces(\Pi)}\ \exists_{0\leq i\leq|(s,t)|}\ \exists_{R,T\in Agent}\ \neg\mathsf{owns}_P(R,T,s_i) \wedge \mathsf{owns}_{Q\cdot P}(R,T,s_i).$$

Informally, the definition states that Q is an ownership transfer protocol, if there exists an agent R for whom the following two conditions are met. First, R is not an owner of T and hence cannot successfully complete the protocol P with T. Second, R is able to successfully complete the sequential composition of Q followed by P with a tag T.

7.3.1 Ownership transfer protocols

Signals

In order to reason about secure ownership transfer we need to capture the intention of the participants when running an ownership transfer protocol. An agent engaging in an ownership transfer protocol may have the intention to *release* a tag

to a new owner or to *obtain* the tag from a previous owner. To keep track of these intentions we decorate protocols and protocol executions with obtain and release *signals*. A release signal indicates that at that point in the protocol (execution), the agent releases the tag for transfer to the new owner. Similarly, an obtain signal indicates that at that point in the protocol (execution), the agent obtains the tag from a previous owner.

We formalize signals as predicates on traces. To this end, we assume that there exists a temporary variable *NO* in the releasing role representing the new owner of the tag. The signal $\mathsf{obtain}_P(A,T)$ now indicates that agent A assigned *True* to its $holds(P,T)$ variable. Similarly, the signal $\mathsf{release}_P(A,T,B)$ denotes that agent A assigned *False* to $holds(P,T)$. Furthermore, the agent name B is assigned to the A's temporary variable *NO* of the run in which the tag was released.

Definition 7.6 (Signals)**.** *Let (s,t) be a trace and let $runidof : Label \to \mathbb{N}$ return the run identifier of a label in the trace. Signals are then defined by*

$$signal_{(s,t)}(i) = \begin{cases} \mathsf{obtain}_P(A,T) & \text{if } \neg\mathsf{holds}_P(R,T,s_i) \wedge \mathsf{holds}_P(R,T,s_{i+1}) \\ \mathsf{release}_P(A,T,B) & \text{if } \mathsf{holds}_P(R,T,s_i) \wedge \neg\mathsf{holds}_P(R,T,s_{i+1}) \wedge \\ & \quad s_{i+1} = \langle A,\sigma,I\rangle \wedge a = (f,n,e,v) \in A \wedge \\ & \quad runidof(t_i) = f \wedge v(NO) = B \\ \bot & otherwise \end{cases}$$

Remark 7.7. For secure ownership it is important to release and obtain tags in the correct position in the ownership transfer protocol. A tag must be released at a point causally preceding a tag's ownership update, typically at the start of the role for the current owner of the tag. The tag can be obtained at a point causally following a tag's confirmed ownership update, thus typically at the end of the role for the new owner. It is easy to see that if a tag is released too late or obtained too early, an agent may be holder of a tag while not owning the tag, thus violating secure ownership.

7.3.2 Secure ownership transfer

We say that a set of protocols provides *secure ownership transfer* if every ownership change is carried out as intended. That is, whenever an agent becomes owner of a tag, it must be as a result of an execution of an ownership transfer protocol. For changes in ownership, making an agent R owner of a tag T, we require the following conditions to be met.

- The ownership change must be preceded by a release signal. This captures the fact that the previous owner intended to hand over ownership of the tag.
- Ownership changes must be in one-to-one correspondence with release signals. If this were not the case then a tag can be obtained more than once as a result of one release.

- No corresponding release and ownership changes related to T may interleave with other corresponding release and ownership-change events of T. That is, the one-to-one map must be such that the ownership change for T is mapped to the latest preceding release signal for T.

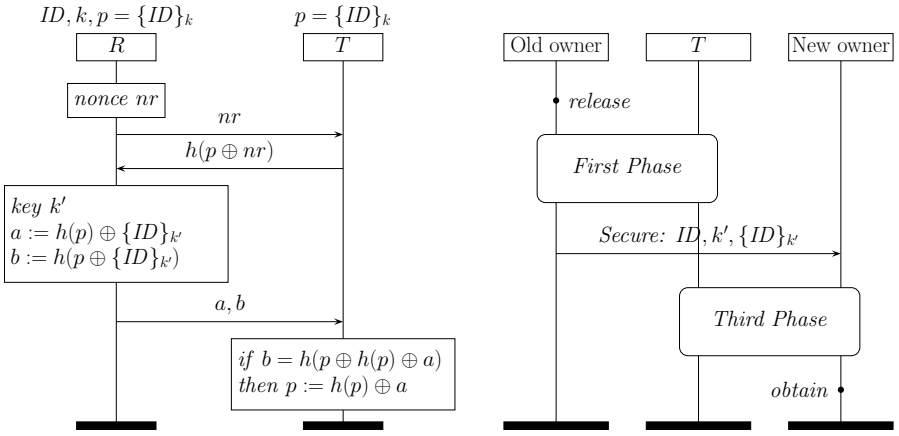
The formalization of secure ownership transfer is complicated by the fact that we do not disallow the adversary to own tags. The adversary does not have to follow the protocol specification and, therefore, does not have to update his *holds* variable. As a consequence, no signals need to be issued if the adversary releases or obtains a tag. Therefore, the above requirements cannot be enforced for tags owned by or released to the adversary. As a final condition we allow an ownership change of any tag to happen immediately after the tag was released to an agent E controlled by the adversary.

Definition 7.8 (Secure ownership transfer)**.** *Let $E \in Agent$ be any agent controlled by the adversary. Let $R' \in Agent$ be any honest agent and let $R'' \in Agent$ be any agent. A set of protocols Π provides secure ownership transfer with respect to P if and only if*

$$\forall_{t \in Traces(\Pi)}\ \exists_{f:\mathbb{N}\to\mathbb{N},injective}\ \forall_{0\le k<|t|}\ \forall_{R,T\in Agent}$$
$$\neg \mathsf{owns}_P(R,T,k) \wedge \mathsf{owns}_P(R,T,k+1) \Rightarrow$$
$$\exists_{0\le i\le k}\ f(k) = i \wedge \neg\exists_{i<j\le k}\ signal_{(s,t)}(j) = \mathsf{release}_P(R',T,R'') \wedge$$
$$(signal_{(s,t)}(i) = \mathsf{release}_P(R',T,R) \vee signal_{(s,t)}(i) = \mathsf{release}_P(R',T,E)).$$

7.3.3 The Yoon and Yoo protocol

We demonstrate our definitions on the ownership transfer protocol by Yoon and Yoo [YY08].



(a) Authentication protocol (b) Ownership transfer protocol

Figure 7.1: The Yoon and Yoo protocol

The Yoon and Yoo protocol is an authentication protocol (Figure 7.1a) that relies on a shared secret $p = \{ID\}_k$ between owner and tag. The shared secret is called a *pseudonym* and can be changed by the owner of a tag by executing the authentication protocol. The protocol is initiated by the reader by sending a nonce nr to the

tag. The tag responds with $h(p \oplus nr)$. The reader can identify a tag by exhaustively searching its database for a pseudonym p' such that $h(p' \oplus nr)$ equals the received message. The reader can then update the pseudonym of the tag by generating a fresh key k' and responding with $a = h(p) \oplus \{ID\}_{k'}$ and $b = h(p \oplus \{ID\}_{k'})$. The tag then updates p to $\{ID\}_{k'}$.

The ownership transfer protocol (Figure 7.1b) consists of three phases involving the old owner, new owner, and the tag. The first and third phase are instantiations of the authentication protocol. In the first phase, the old owner updates the pseudonym p, using a fresh key k'. This key together with the real identity and the pseudonym are sent over a secure channel to the new owner in the second phase. The use of a secure channel ensures that no attacker can eavesdrop, modify, block, or inject messages in the second phase. The final phase consists of another pseudonym update executed by the new owner and the tag using a fresh key.

Since the pseudonym p of the tag is all that is used in communication with the tag, we take as ownership test protocol a proof-of-knowledge protocol of p. Following Remark 7.7 the tag is released by the old owner at the start of the first phase and obtained by the new owner at the end of the third phase. We can now analyze the protocol with respect to exclusive ownership, secure ownership, and secure ownership transfer.

Lemma 7.9. *The Yoon and Yoo protocol does not satisfy exclusive ownership.*

Proof. Let s be a state of the system such that reader R is holder of tag T, i.e. $\mathsf{holds}_P(R, T, s)$. We further assume that there is no $R' \neq R$ that owns T. That is, R is the only agent that knows the pseudonym p.

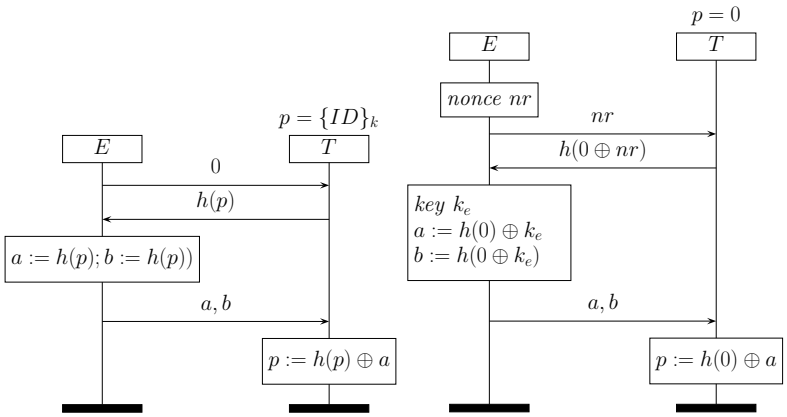
Let 0 be the neutral element with respect to $\oplus$, known to all agents in the system. The attacker sends 0 to T. The tag T responds with $h(p)$ after which E returns $a = h(p)$ and $b = h(p)$. Tag T verifies B and updates p to $h(p) \oplus h(p) = 0$. Let s' be the state in which T has updated its secret. The attack is depicted in Figure 7.2a.

The ownership test protocol declares any agent that knows the pseudonym to be owner of the tag. Since the tag's pseudonym p equals 0, all agents are owners of T. However, R has not released tag T and is still tag holder. This violates exclusive ownership. □

Lemma 7.10. *The Yoon and Yoo protocol does not satisfy secure ownership.*

Proof. Lemma 7.9 shows that there exists a state s' in which R is holder of the tag T and all agents own the tag. In particular, the adversary owning the tag allows him to execute the authentication protocol. In state s', tag T's pseudonym is 0. This allows the adversary to execute the authentication protocol with T and to update the pseudonym to a new freshly generated value k_e. Since R does not own the tag while still being tag holder, secure ownership is not satisfied. The attack is depicted in Figure 7.2b. □

Lemma 7.11. *The Yoon and Yoo protocol does not satisfy secure ownership transfer.*



(a) Exclusive ownership (b) Secure ownership

Figure 7.2: Ownership attacks on the Yoon and Yoo protocol.

Proof. Consider an execution of the ownership transfer protocol by R, T, and R', where initially R is the tag holder (and owner) of the tag T and intends R' to become the new tag holder (and owner). We show that the adversary E can obtain ownership of the tag without being the intended new owner. To achieve this, he queries the target tag T with the constant 0 to which the tag responds with $h(p)$. By eavesdropping on the first phase of the ownership transfer protocol, the adversary obtains $a = h(p) \oplus \{ID\}_{k'}$. As soon as the tag updates its pseudonym to $\{ID\}_{k'}$ the adversary becomes owner of the tag. Since T was not released to E, secure ownership transfer is not satisfied. □

7.4 Desynchronization

The execution of a stateful RFID protocol frequently ends with reader and tag updating shared information. An attacker may attempt to disrupt the communication between reader and tag so that the updates of the two agents are not related. A flawed protocol does not allow the agents to recover from this disruption and the reader and tag will be in a state of *desynchronization*: they will no longer be able to successfully communicate with each other. We call a protocol that is not vulnerable to this type of attack *desynchronization resistant*.

For a given protocol, one can characterize desynchronization as a relation on the persistent variables of reader and tags. Unfortunately, it is not straightforward to transform this into a generic definition of desynchronization resistance. We argue, however, that our notion of ownership is closely related to desynchronization resistance.

In general, RFID authentication protocols do not need to satisfy secure ownership requirements, since the owner of a tag never changes. However, observe that if there does not exist a reader that can successfully communicate with a tag using its protocol P, then the tag has no owners with respect to P. We thus define desynchronization resistance as the property that a protocol P never loses all its owners with respect to P. This guarantees that there always exists an agent R that can execute P. Note that we require R to be honest to prevent the case that

only the adversary can communicate with the tag, but nobody else.

Definition 7.12 (Desynchronization resistance)**.** *Let $HA \subset Agent$ be the set of honest agents. A protocol $P \in \Pi$ is desynchronization resistant if and only if*

$$\forall_{(s,t)\in Traces(\Pi)}\ \forall_{0\leq i<|(s,t)|}\ \forall_{T\in Agent} \\ \exists_{R\in HA}\ \mathsf{owns}_P(R,T,s_i) \Rightarrow \exists_{R'\in HA}\ \mathsf{owns}_P(R',T,s_{i+1}).$$

It is interesting to note that desynchronization resistance together with exclusive ownership can imply secure ownership. Therefore in order to prove secure ownership with respect to a test protocol P it is sufficient, under the conditions stated in the following theorem, to prove desynchronization resistance of P and exclusive ownership with respect to P. Note that the second condition in the theorem corresponds to placing obtain signals in protocols at a point in which an agent is sure to have become owner of a tag, as described in Remark 7.7

Theorem 7.13. *Let Π be a set of protocols containing the test protocol P. Suppose that Π provides exclusive ownership with respect to P and that P is desynchronization resistant. Then Π provides secure ownership for every trace which satisfies the following two conditions.*

(1) *In the initial state every holder of a tag is owner of the tag.*

(2) *An agent only becomes holder of a tag if it owns the tag.*

Proof. Suppose towards a contradiction that there is a trace $(s,t) \in Traces(\Pi)$ such that in a state s_i an agent R holds a tag T, but does not own the tag. By exclusive ownership, if R holds tag T, then no other agent R' owns the tag in state s_i. Thus, tag T is not owned by anybody in state s_i. Desynchronization resistance implies that if no agent owned tag T in state s_i, then no agent owned the tag in s_{i-1}. This argument can be repeated to conclude that no agent owned T in states $s_0 \dots s_i$. Since R did not own tag T in states $s_0 \dots s_i$, by condition (2) he cannot have become the holder in states $s_0 \dots s_i$. Thus, R must have been the holder of tag T in the initial state contradicting condition (1). Therefore, secure ownership must be satisfied. □

7.4.1 The Song and Mitchell protocol

Song and Mitchell [SM08] proposed a stateful RFID protocol, called the SM protocol, that relies on a shared secret for authentication. Their protocol is claimed to achieve identification and authentication of the tag which are necessary in scenarios like supply chain management or access control. They notice that in many proposed protocols tags and readers can be desynchronized by blocking certain messages from reader to tag. They attempt to prevent desynchronization attacks by storing additional information, allowing the reader to re-synchronize with a tag in case messages are blocked.

Figure 7.3 shows the SM protocol. Tag and reader share a pseudonym k that is updated at the end of a successful protocol execution. The value of k is equal to

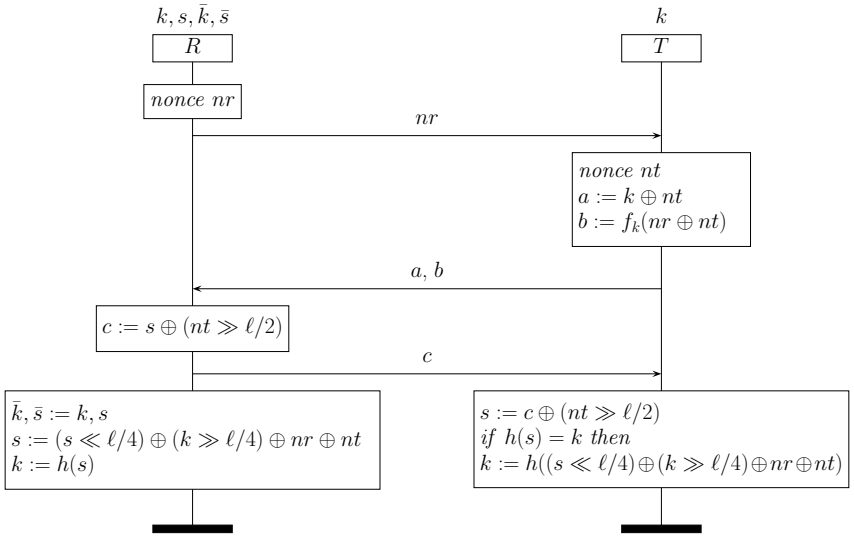


Figure 7.3: The Song and Mitchell protocol.

the hash of another secret s. The reader stores s and a backup of k and s in $\bar{k}$ and $\bar{s}$.

We use $f_k(\cdot)$ to denote a keyed hash function with key k. Bit rotations are denoted by $\gg$ and $\ll$ where $a \ll b$ means a shifted cyclically to the right by b bits. All secrets and random values are of length ℓ.

The protocol is initiated by the reader sending a random value nr to the tag. The tag responds by generating a random value nt and sending $a := k \oplus nt$ and $b := f_k(nr \oplus nt)$ to the reader. The reader verifies the validity of a and b and sends $c := s \oplus (nt \gg \ell/2)$ to the tag. At the end of the protocol execution, the reader stores the values of k and s in $\bar{k}$ and $\bar{s}$. It also updates s to $(s \ll \ell/4) \oplus (k \gg \ell/4) \oplus nr \oplus nt$ and k to $h(s)$. The tag updates its k to $h((s \ll \ell/4) \oplus (k \gg \ell/4) \oplus nr \oplus nt)$, which under normal circumstances is the same value as the reader's value of k.

We now apply our definition of desynchronization resistance to demonstrate that by modifying and blocking certain messages an attacker can force a tag and reader to carry out different updates of their shared secret.

Theorem 7.14. *The Song and Mitchell protocol does not satisfy desynchronization resistance.*

Proof. To show that a protocol does not satisfy desynchronization resistance, we need to show that starting from a state where a tag has at least one owner, a state is reachable in which the tag does not have any owners. An agent owns a tag if he can successfully execute the SM protocol with it.

Consider an RFID system with a reader R, a tag T, and the adversary's agent E. Suppose R owns T in state $s = \langle A, \sigma, I \rangle$. That is, there exists a value k_0 such that $\sigma_R(k) = \sigma_T(k) = k_0$ and a value s_0 such that $\sigma_R(s) = s_0$ and $h(s_0) = k_0$. Furthermore, suppose R is the only owner of tag T. Finally, let the adversary knowledge $I = \emptyset$.

To attack the protocol, the attacker can force tag and reader to carry out different pseudonym updates. The attack is depicted in Figure 7.4 and works as follows.

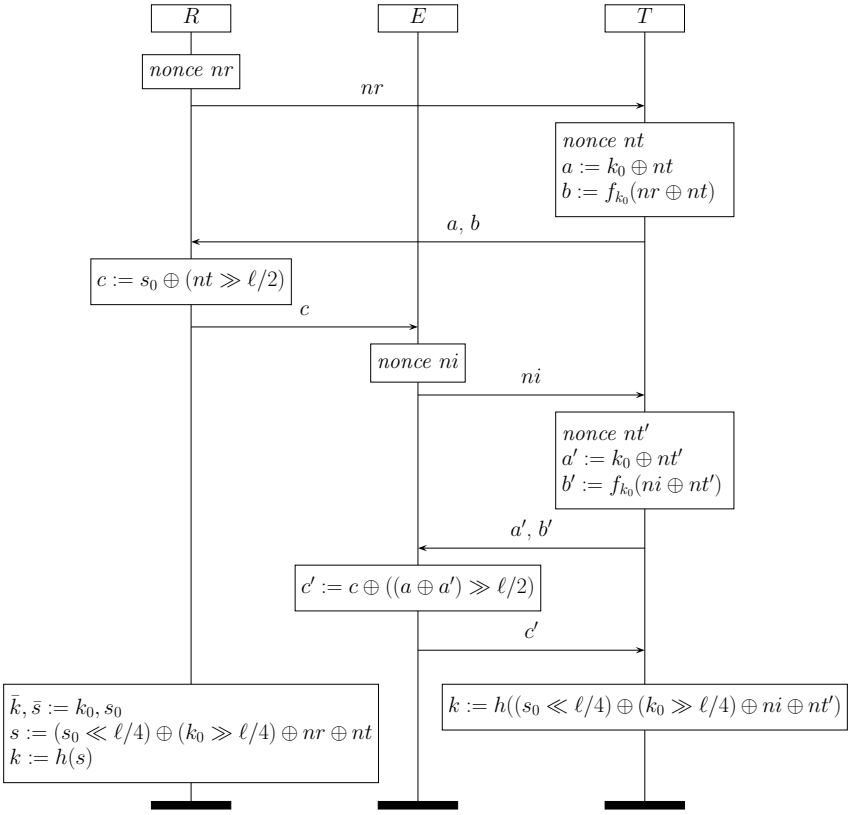


Figure 7.4: Desynchronization attack on the Song and Mitchell protocol.

The attacker eavesdrops on the first two messages nr and a, b exchanged between a reader R and tag T and then aborts the protocol execution after the reader sends the third message c. The reader updates $\bar{k}$, $\bar{s}$, k, and s. In particular, k is set to $h((s_0 \ll \ell/4) \oplus (k_0 \gg \ell/4) \oplus nr \oplus nt)$. The tag has not successfully completed its run and therefore does not carry out its update. In this state, R still owns T since $\bar{k}$ contains k_0 and $\bar{s}$ contains s_0 and the tag has not updated his value for k.

The attacker then challenges the same tag with his own nonce ni. The tag responds with $a' = k_0 \oplus nt'$ and $b' = f_{k_0}(ni \oplus nt')$. Using distributivity of $\oplus$ over $\gg$, the attacker can now construct a valid reader response $c' = c \oplus ((a \oplus a') \gg \ell/2) = s \oplus (nt' \gg \ell/2)$. The tag accepts the message and updates its value for k to $h((s_0 \ll \ell/4) \oplus (k_0 \gg \ell/4) \oplus ni \oplus nt')$.

Let $s' = \langle A', \sigma', I' \rangle$ be the state at the end of the attack. It follows that $\sigma'_T(k) = h((s_0 \ll \ell/4) \oplus (k_0 \gg \ell/4) \oplus ni \oplus nt')$. Since R does not know $(s_0 \ll \ell/4) \oplus (k_0 \gg \ell/4) \oplus ni \oplus nt'$, he can not successfully execute the test protocol. Therefore, he no longer owns T. Furthermore, the adversary also does not know $(s_0 \ll \ell/4) \oplus (k_0 \gg \ell/4) \oplus ni \oplus nt'$. Hence, nobody can complete the protocol with the tag in state s'. Therefore, the Song and Mitchell protocol is not desynchronization resistant. □

7.5 Related work

Work on ownership transfer in RFID systems has thus far mostly focused on designing ownership transfer protocols, but not on their security requirements. An exception is the work by Song [Son08]. It provides a first informal discussion of security

requirements related to ownership transfer. Song also proposes a set of protocols for secure ownership transfer based on earlier work by Song and Mitchell [SM08]. The SM protocol is not desynchronization resistant (Section 7.4) and does not satisfy tag authentication (Section 6.2). Since the ownership transfer protocol by Song is an extension of the SM protocol it suffers from the same flaws.

The first treatment of ownership transfer in RFID systems is due to Molnar, Soppera, and Wagner [MSW05]. They describe a protocol that relies on a trusted center. Readers send tag pseudonyms to the center requesting the real identity of a tag. If the reader is the owner of the tag it receives the identity. Owners of tags can ask the trusted center to transfer the ownership of a tag to a new owner. The trusted center subsequently refuses identity requests from the old owner, and accepts them from the new owner. A trusted party is also used by the protocol of Saito, Imamoto, and Sakurai [SIS05]. Here, the trusted party shares a key with the tag which is used to update the owner's key. Hence an ownership transfer consists of a request to the trusted party to encrypt the new owner's key for the tag.

Lim and Kwon [LK06] propose a stateful authentication protocol based on key chains. Upon successful execution of the protocol, the tag and reader update the tag pseudonym. They argue that the protocol satisfies forward untraceability under the assumption that the adversary cannot eavesdrop on all future protocol executions of the tag. That is, the adversary cannot trace the tag, even if it knows previous pseudonyms and corresponding keys of the tag. Lim and Kwon reason that due to forward untraceability, their protocol can be used as an ownership transfer protocol.

Osaka, Takagi, Yamazaki, and Takahashi [OTYT06] are among the first to propose an ownership transfer protocol. Unfortunately, the protocol does not satisfy reader authentication allowing an adversary to corrupt the key of the tag. As a result, the protocol does not satisfy desynchronization resistance and secure ownership.

The flaw in the OTYT protocol triggered several others to propose improved protocols. Lei and Cao [LC07] propose an ownership transfer protocol that mitigates the flaw in the OTYT protocol. Vullers [Vul09] analyzed the protocol for secure ownership and secure ownership transfer and showed that the protocol does not satisfy exclusive ownership. Jäppinen and Hämäläinen [JH08] proposed a protocol designed to improve the OTYT protocol. Unfortunately, it does not fix the flaw in the protocol, allowing an adversary to break desynchronization resistance and secure ownership.

Koralalage, Reza, Miura, Goto, and Cheng [KSM+07] and Fouladgar and Afifi [FA07] propose protocols based on symmetric cryptography. Both protocols carry out updates of the key material at the end of the protocol and are thus pseudonym-based protocols. Ownership can be transferred by sending the key to the new owner after which the new owner updates it to "disown" the previous owner.

Finally, one of the most recent protocols in this area is due to Dimitriou [Dim08]. Its distinguishing feature is that it enables the owner of a tag to revert the tag to its original state. This is useful for after-sales services, since it makes it possible for the tag's new owner to let a retailer recognize a sold tag without losing ownership of the tag.

7.6 Conclusion

Ownership of RFID tags is not only concerned with physical possession of the tags, but more importantly with the ability to interact with the RFID tag. We have identified the latter as a property of RFID protocols and formalized it using the notion of test protocols. In the course of its lifetime, RFID tags may change hands a multitude of times, in particular when they are used in supply chains. To facilitate this process, a number of ownership transfer protocols have been proposed in literature. These protocols are often accompanied by informal security requirements and informal security analyses.

In the absence of precise definitions of any security property it is hard to verify whether a given protocol is secure. For this reason, we have given formal definitions of ownership and ownership transfer, as well as their secure variants. Secure ownership guarantees that tags cannot be stolen from an agent. Secure ownership transfer states that an agent can only become owner by running an ownership transfer protocol. Finally, if exclusive ownership is satisfied only the agent believing to be the owner of the tag can actually be the owner. We have shown the applicability of our definitions by exhibiting attacks on secure ownership, exclusive ownership, and secure ownership transfer on the Yoon and Yoo [YY08] protocol. Finally, we have used our concept of ownership to formalize desynchronization resistance and shown that the protocol by Song and Mitchell [SM08] does not satisfy the property.

Security properties such as secrecy and authentication can be defined as *local* or *role-specific* security requirements. The main reason behind locality is that agents only have a partial view on the system state, based on the messages they receive. The protocol should ensure that based on this local view, the user can be certain that the security property holds. For instance, an authentication protocol could provide the guarantee that at the end of a protocol execution, the communicating partner has been recently alive. Similarly, an untraceable protocol provides the guarantee to an agent that a run cannot be linked to an earlier run of that agent. Secure ownership, exclusive ownership, secure ownership transfer, and desynchronization resistance are properties of the system rather than of one role. Therefore, we have defined them as global security properties.

Part IV

Reverse engineering RFID systems

8

Carving

In the previous chapters we have studied RFID systems at the level of the cryptographic communication protocols. These protocols aim to achieve security properties such as authentication and untraceability. One of the most important requirements to satisfy these properties is that the attacker must not be able to obtain the cryptographic keys. One quickly sees that this requirement is not specific to the protocol, but must be satisfied by the entire RFID system. If the attacker can obtain the keys of a tag through side-channel analysis or from a central database, the security of many RFID systems cannot be guaranteed. Making RFID tags tamper resistant, however, results in more expensive RFID tags.

At the same time, there is a tendency to implement more functionality on a tag. Some of the more expensive RFID tags are equipped with memory that can contain several kilo bytes of data. For instance, the electronic passport stores a picture of its holder and several public transportation cards can store products purchased by the user or an electronic wallet. Furthermore, in some cases the contact interface of a smart card is replaced with a contactless RFID interface. The applications using RFID tags with memory often regard the RFID tags as a trusted storage for their data.

Of course, one must be careful when storing valuable information on a tag that is not tamper resistant. As discussed in Section 4.7.1, several practical attacks on RFID protocols have been reported. These attacks recover the cryptographic keys used for authentication between a tag and a reader. An attacker with these keys can read and modify the data on the RFID tag. However, having access to the contents of a tag does not immediately allow an attacker to abuse the system in which the tags are used. It merely allows the attacker to read and modify the raw binary data of the card. For a more sophisticated attack, the attacker has to find the relevant data on the card and he has to understand how to interpret and modify this data.

In this chapter, we focus on recovering the structure and information from raw binary data. In digital forensics, this process is called *carving*. The main objective of current file carving approaches is to reconstruct (partially) deleted, damaged, or fragmented files. A typical example is the analysis of memory dumps from cell phones [BH10]. Because a file can be permuted in many possible ways, the process of reassembling files is very labor intensive. Therefore, fully and semi-automatic file carving tools have been developed that aid the human inspection process.

Traditional carving approaches aim to analyze a single memory dump. In some cases, however, one may have access to a series of similarly structured dumps. This may result from observing a system that progresses in time, while making memory

dumps at regular time intervals, or from dumping the memory of a collection of similar systems. If we assume that an attacker has obtained the cryptographic keys, he can collect dumps of several cards after each usage. Since the amount of available memory is limited, we expect these dumps to be similarly structured and to contain little metadata on the structure. Therefore, we aim to take advantage of the fact that we have many similarly-structured small memory dumps rather than one large dump.

We will investigate the problem of carving sets of dumps under two simplifying assumptions. The first assumption is that we can observe certain relevant properties of the system at the moment of dumping its memory. In this way, we can collect the values of a number of attributes that characterize part of the state of the system, and link that information to the memory dump. An example of such an attribute is the number of rides left on a public transportation card, which can be easily observed from the display of the card reader when validating the card. The carving problem for such attributed dump sets is then described as the problem of finding at which location in the memory dump the attributes are stored.

The second assumption is that the memory layout is either static or semi-dynamic. A memory layout is static if the attributes are stored at the same location in every dump and the dumps have the same length. An attribute is stored semi-dynamically if it is stored alternatingly in a number of different locations.

8.1 Carving attributed dump sets

8.1.1 Definition of the carving problem

The central concept to the carving problem is the concept of a *dump*. A dump consists of raw binary data that is captured from a system, for instance, from a computer's memory, a data carrier, or a communication transcript. We assume that the process of creating a dump can be repeated, allowing a number of dumps of the same system to be collected. We call such a collection of dumps a *dump set*.

Example 8.1 (Public transportation card)**.** *Consider an electronic fare collection system for public transportation systems. The users of such a system carry an RFID tag with non-volatile memory. The system stores the number of rides the user is entitled to on the card. Upon entering a bus, the user swipes his card across an RFID reader. The RFID reader deducts one trip and stores the new number of rides on the card. A dump of a card can be created by reading the entire memory of a card after it has been used. In repeating this process, a dump set can be created.*

We assume that different dumps of the same system have the same length. If we denote the bit strings of length $n \in \mathbb{N}$ by $\mathbb{B}^n$ and bit strings of arbitrary, finite length $\mathbb{B}^*$, then a set of dumps of length n is denoted by $S \subseteq \mathbb{B}^n$. The length n of bit string $s \in \mathbb{B}^n$ is denoted by $|s|$ and the number of elements in set S is denoted by $|S|$. We denote by the closed interval $[i, j]$ the set of integers z such that $i \leq z \leq j$. The half-open interval $[i, j)$ denotes the set of integers z such that $i \leq z < j$. For $i \in [0, |s|)$ we denote the i-th bit of s by s_i. For $I \subseteq [0, |s|)$, we denote the subsequence of s that consists of all elements with index in I by $s|_I$.

A dump contains information about the state of the system. We call these state properties *attributes*. For each dump set we consider a set $\mathbb{A}$ of attributes. The function $\text{type}\colon \mathbb{A} \to \mathbb{D}$ assigns to every attribute a finite value domain, where $\mathbb{D}$ denotes the set of all finite value domains. The value of attribute $a \in \mathbb{A}$ expressed in dump s is denoted by $\text{val}_a\colon S \to \text{type}(a)$.

A dump contains the system's attribute values in a binary representation. The mapping from an attribute domain to its binary representation is called an *encoding*. We assume that for a given attribute $a \in \mathbb{A}$ the length of an encoding is fixed, so an encoding of a is a function from $\text{type}(a)$ to $\mathbb{B}^n$ for some $n \in \mathbb{N}$. This function is required to be injective and deterministic. The set of all encodings of $D \in \mathbb{D}$ is denoted by $\mathbb{E}_D$.

Example 8.2 (Type, value, encoding)**.** *An example of an attribute encoded on a public transportation card is the number of rides the user is entitled to. The type of the attribute* rides-left $\in \mathbb{A}$ *is* $[0, 15]$*. A particular dump* $s \in S$ *of a card can have* 5 *rides left, so* $\text{val}_{\textit{rides-left}}(s) = 5$*. The encoding of the attribute rides-left is the 5-bit binary representation. Thus, the value* 5 *is encoded by* 00101*.*

The type of the attribute last-used *is the set of all dates between 1/1/2000 and 1/1/2050, extended with the time of day in hh:mm:ss format. A possible encoding of the attribute is the number of seconds since 1/1/2000, 00:00 hrs expressed in binary format.*

We start with the assumption that an attribute is always stored at the same location in all dumps of the system. In Section 8.1.5 we extend this to semi-dynamic attributes. With this assumption we can identify which bits of the dump are related to a given attribute. This is captured in the notion of an *attribute mapping*.

Definition 8.3 (Attribute mapping)**.** *Let* $S \subseteq \mathbb{B}^n$ *be a dump set with dumps of length* n*. An* attribute mapping *for* S *is a function* $f\colon \mathbb{A} \to \mathcal{P}([0, n))$*, such that*

$$\forall_{a \in \mathbb{A}}\ \exists_{e \in \mathbb{E}_{\text{type}(a)}}\ \forall_{s \in S}\ s|_{f(a)} = e(\text{val}_a(s)).$$

An attribute mapping is non-overlapping *if*

$$\forall_{a_1, a_2 \in \mathbb{A}}\ a_1 \neq a_2 \Rightarrow f(a_1) \cap f(a_2) = \emptyset.$$

An attribute mapping is contiguous *if*

$$\forall_{a \in \mathbb{A}}\ \exists_{0 \leq i, j \leq n}\ f(a) = [i, j).$$

Given a dump set S and all attribute values for each dump in S, the *carving problem for attributed dump sets* is the problem of finding an attribute mapping for S. The existence of an attribute mapping does not imply that the attributes are indeed encoded in the dump, but merely that they could have been encoded at the indicated positions in the dumps. Conversely, if an attribute cannot be mapped in S, it means that this attribute is not present through a deterministic, injective encoding. Of course, this does not rule out the possibility that a non-deterministic encoding is used, such as a probabilistic encryption, or that the attribute is stored dynamically, i.e. not always at the same location. We consider the search for high-entropy information and semi-dynamic attributes later in this chapter.

Example 8.4 (Attribute mapping)**.** *The notion of an attribute mapping is illustrated in Figure 8.1. This example consists of five dumps,* $s_1, \ldots s_5$*, of length* $n = 18$*. We look at the attribute* rides-left *(rl) with the values as given in the figure and we consider two possible encodings* enc_1 *and* enc_2*. The first encoding is the standard binary encoding of natural numbers. It can be found in the dumps at two different (contiguous) positions:* $[5, 8]$ *and* $[12, 15]$*. The second encoding, which is not standard, occurs at positions* $[3, 6]$*. Each of these three cases defines a contiguous attribute mapping for* rides-left*. There might be more candidate encodings.*

	rl	dump	enc_1	enc_2
s_1	4	0101001001110100 00	0100	1001
s_2	4	0011001000010100 10	0100	1001
s_3	5	1011101010110101 00	0101	1101
s_4	6	0010101101110110 11	0110	0101
s_5	6	1110101100110110 01	0110	0101

Figure 8.1: Example of a dump set with three possible attribute mappings.

If we assume that the sizes of the attribute value domains are known, we have an information-theoretic lower bound on the number of bits that must have been used for encoding the attribute. This is expressed in the following lemma, which can be used to further limit the search space. The lemma follows from the pigeonhole principle.

Lemma 8.5. *Let* $\mathbb{A}$ *be an attribute set and let* f *be an attribute mapping for dump set* $S \subseteq \mathbb{B}^n$*, then* $\forall_{a \in \mathbb{A}} |f(a)| \geq \log_2(|\text{type}(a)|)$*.*

Given the values of an attribute for the dumps in a dump set S, we can use the commonalities and dissimilarities of these dumps to derive restrictions on the possible attribute mappings for S. Such restrictions are derived in two steps. In the first step we look at dumps that have the same attribute value. In this case, we can derive those positions in the bit strings that cannot occur in the encoding of the attribute. In the second step we look at dumps of which the attribute values differ, allowing us to determine positions in the bit strings that should occur in the encoding of the attribute.

In the next sections, we first investigate restrictions induced by the commonalities of a dump set, then by the dissimilarities of a dump set, and finally by a combination of commonalities and dissimilarities.

8.1.2 Commonalities

We start by observing that an attribute $a \in \mathbb{A}$ induces a partition

$$\text{bundles}(a, S) = \{\{s \in S \mid \text{val}_a(s) = d\} \mid d \in \text{type}(a)\}$$

on a dump set S. An element of this partition is called a *bundle*. Thus, a bundle is a set of dumps with the same attribute value.

The *common set* determines which bits in the dumps of a dump set are equal if the attribute values are equal.

Definition 8.6 (Common set)**.** *Let $a \in \mathbb{A}$ be an attribute and $S \subseteq \mathbb{B}^n$ be a dump set. The* common set *of S with respect to a, denoted by* $\mathrm{comm}(a, S) \subseteq [0, n)$*, is defined by*

$$\mathrm{comm}(a, S) = \bigcap_{b \in \mathrm{bundles}(a,S)} \{i \in [0, n) \mid \forall_{s,s' \in b}\, s_i = s'_i\}.$$

Example 8.7 (Bundles and commonalities)**.** *Consider the dump set S containing the five dumps shown in Figure 8.1. The attribute* rides-left *induces a partition of the five dumps into three bundles:* $\mathrm{bundles}(\textit{rides-left}, S) = \{\{s_1, s_2\}, \{s_3\}, \{s_4, s_5\}\}$.

The common set for set S with respect to rides-left *is* $\mathrm{comm}(\textit{rides-left}, S) = \{3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 15\}$. *This set includes, among others, the three contiguous subsequences of Example 8.4.*

Given that the encoding of an attribute value is deterministic, this gives an upper bound on the bits used for this attribute. The following lemma states two properties about the relation between attribute mappings and the commonalities set. The first property states that every possible attribute mapping is enclosed in the common set, so one can restrict the search for attribute mappings to the locations in the common set. The second property expresses that every extension of an attribute mapping is also an attribute mapping, provided that it does not extend beyond the common set.

Lemma 8.8. *Let $\mathbb{A}$ be an attribute set and let f be an attribute mapping for dump set $S \subseteq \mathbb{B}^n$, then*

1. $\forall_{a \in \mathbb{A}}\, f(a) \subseteq \mathrm{comm}(a, S)$*,*
2. *if $I_a \subseteq [0, n)$ is a family of sets for $a \in \mathbb{A}$, such that $f(a) \subseteq I_a \subseteq \mathrm{comm}(a, S)$, then the function $f' : \mathbb{A} \to \mathcal{P}([0, n))$, defined by $f'(a) \mapsto I_a$, is an attribute mapping.*

Proof. We show the first property by contradiction. Assume that there exists an attribute a such that $f(a) \not\subseteq \mathrm{comm}(a, S)$. Then there exists an index $i \in f(a)$ such that $i \notin \mathit{common}(a, S)$. It follows from the definition of comm that there is a bundle that contains bit strings s and s' such that $s_i \neq s'_i$. However, since f is an attribute mapping, index $i \in f(a)$, and $val_a(s) = val_a(s')$, we have that $s_i = s'_i$. Thus, $f(a)$ must be a subset of $\mathrm{comm}(a, S)$.

The second property follows from the fact that if we extend an encoding, it remains an encoding. We know that $e(val_s(a)) = s|_{f(a)}$ is an encoding for attribute mapping f. The map $e'(val_s(a)) = s|_{f'(a)}$ is an encoding as long as for all $j \in f'(a)$ we have $s_j = s'_j$ if $val_a(s) = val_a(s')$ and e' is injective. The former follows from the assumption that $j \in \mathrm{comm}(a, S)$. The latter follows from the fact that extending the range of the encoding maintains the injectivity of it. Hence, $f'(a) \mapsto I_a$ is an attribute mapping. □

8.1.3 Dissimilarities

For the second step, we look at dumps with different attribute values. Injectivity of the encoding function implies that the encoding of two different values must differ at least in one bit. This is captured in the notion of a *dissimilarity set*. This set consists of all intervals that, for each pair of dumps with a different attribute value, contain at least one location where the two dumps differ.

Definition 8.9 (Dissimilarity set)**.** *Let $a \in \mathbb{A}$ be an attribute and $S \subseteq \mathbb{B}^n$ be a dump set. The* dissimilarity set *of S with respect to a, denoted by* $\mathrm{diss}(a, S) \subseteq \mathcal{P}([0, n))$*, is defined by*

$$\mathrm{diss}(a, S) = \{I \subseteq [0, n) \mid \forall_{s,s' \in S} \, (\mathrm{val}_a(s) \neq \mathrm{val}_a(s') \Rightarrow \exists_{i \in I} \, s_i \neq s'_i)\}.$$

Example 8.10 (Dissimilarities)**.** *Consider the dump set S containing the five dumps shown in Figure 8.1 and the attribute* rides-left *(rl). The interval $[3, 4]$ satisfies the condition that for any dumps s and s', if the values of rl differ, then $s|_{[3,4]} \neq s'|_{[3,4]}$. Other elements in the dissimilarity set are the intervals $[4, 6]$ and $[13, 16]$ and all supersets of these intervals.*

The next lemma expresses that every attribute mapping is enclosed in the dissimilarity set. Consequently, we can restrict the search for possible attribute mappings to the elements of the dissimilarity set.

Lemma 8.11. *Let $\mathbb{A}$ be an attribute set and let f be an attribute mapping for dump set $S \subseteq \mathbb{B}^n$, then $\forall_{a \in \mathbb{A}} \, f(a) \in \mathrm{diss}(a, S)$.*

Proof. Let $a \in \mathbb{A}$ and let $s, s' \in S$, such that $\mathrm{val}_a(s) \neq \mathrm{val}_a(s')$. Due to the injectivity requirement on encodings we know that $e(\mathrm{val}_a(s)) \neq e(\mathrm{val}_a(s'))$. From the definition of an attribute mapping we derive that $s|_{f(a)} \neq s'|_{f(a)}$. Therefore, there exists an $i \in f(a)$, such that $s_i \neq s'_i$, and thus $f(a)$ satisfies the definition of $\mathrm{diss}(a, S)$. □

An encoding of an attribute value a must at least contain the indexes from one of the sets in $\mathrm{diss}(a, S)$. This implies that we are mainly interested in the smallest sets in $\mathrm{diss}(a, S)$, i.e. those sets of which no proper subset is in $\mathrm{diss}(a, S)$. In order to make this precise, we introduce some notation.

Let F be a set and let $P \subseteq \mathcal{P}(F)$. We define the *superset closure* of P, notation $\overline{P}$, by $\overline{P} = \{p \subseteq F \mid \exists_{p' \in P} \, p' \subseteq p\}$. A set P is *superset closed* if $P = \overline{P}$. We observe from its definition that $\mathrm{diss}(a, S)$ is superset closed.

Given $P \subseteq \mathcal{P}(F)$, we say that P is *subset minimal* if for every $p, p' \in P$, $p' \subseteq p \Rightarrow p' = p$. The following lemma states that for every $P \subseteq \mathcal{P}(F)$ a unique subset-minimal Q exists such that their superset closures are equal.

Lemma 8.12. *Let F be a finite set and let $P \subseteq \mathcal{P}(F)$. Then there exists a unique subset-minimal set Q such that $\overline{Q} = \overline{P}$.*

Proof. We define $Q = \{p \in P \mid \forall_{p' \in P} \, p' \subseteq p \Rightarrow p' = p\}$ and prove that this is the required set. From the definition of Q it follows directly that Q is subset minimal.

The inclusion $\overline{Q} \subseteq \overline{P}$ follows directly from $Q \subseteq P$. For the converse, $\overline{P} \subseteq \overline{Q}$, we use the fact that strict set inclusion on $\mathcal{P}(F)$ is well-founded for finite F.

Let $p \in \overline{P}$, then there exists $p' \in P$, such that $p' \subseteq p$. We consider two cases: $p' \in Q$ and $p' \notin Q$. If $p' \in Q$, then from $p' \subseteq p$ it follows that $p \in \overline{Q}$, as required. In the second case, $p' \notin Q$, we use the definition of Q to find $p'' \in P$ such that $p'' \subsetneq p'$. Again, we can consider two cases: $p'' \in Q$ and $p'' \notin Q$. In the first case, $p'' \in Q$ we have $p'' \subsetneq p' \subseteq p$, so $p \in \overline{Q}$, as required. In the second case we can repeat this construction to find $p''' \subsetneq p'' \subsetneq p' \subseteq p$. Given well-foundedness, it is impossible to create an infinite sequence in this way. Therefore, there is a point where the loop will be broken by finding $p^{(k)} \in P$, such that $p^{(k)} \subsetneq p^{(k-1)} \subsetneq \ldots \subsetneq p' \subseteq p$. We know that there exists no $\hat{p} \in P$ such that $\hat{p} \subsetneq p^{(k)}$. Therefore, from the definition of Q we have that $p^{(k)} \in Q$, which implies that $p \in \overline{Q}$.

Finally, we prove uniqueness. Assume that X and Y are two subset-minimal sets with $X \neq Y$ and $\overline{X} = \overline{P} = \overline{Y}$. Without loss of generality, we may assume that there exists $x \in X$, such that $x \notin Y$. We derive a contradiction and conclude $X = Y$ as follows. If $x \in X$, then $x \in \overline{Y}$. From $x \notin Y$, we find $y \in Y$, such that $y \subsetneq x$. From $y \in \overline{Y}$, it follows that $y \in \overline{X}$, so there exists $x' \in X$ with $x' \subseteq y$. Thus, we have $x' \subseteq y \subsetneq x$ for $x', x \in X$, which contradicts the assumption of subset minimality of X. □

Given P as in Lemma 8.12, we denote the unique subset-minimal set by $\mathrm{smin}(P)$. Then, in order to determine whether an encoding of an attribute contains at least the indexes from one of the sets in $\mathrm{diss}(a, S)$, it suffices to verify that it at least contains one of the sets from $\mathrm{smin}(\mathrm{diss}(a, S))$.

By combining the results of the previous lemmas, we get the following main result.

Theorem 8.13. *Let $\mathbb{A}$ be an attribute set and let f be an attribute mapping for dump set $S \subseteq \mathbb{B}^n$, then*

$$\forall_{a \in \mathbb{A}} \exists_{I \in \mathrm{smin}(\mathrm{diss}(a,S))}\ I \subseteq f(a) \subseteq \mathrm{comm}(a, S).$$

This theorem says that if an attribute is expressed in a dump set, then its encoding position should contain at least one of the minimal dissimilarity sets and must be contained in the common set. Thus, by calculating these common set and dissimilarity sets, we can limit the search space when looking for this attribute in the dumps. In Section 8.2, we will investigate algorithms for determining the two sets $\mathrm{smin}(\mathrm{diss}(a, S))$ and $\mathrm{comm}(a, S)$.

A consequence of the theorem is that by calculating $\mathrm{diss}(a, S)$ and $\mathrm{comm}(a, S)$, we can limit the search space when looking for the attribute mapping $f(a)$ in the dumps. We will now investigate how to further limit the search space.

8.1.4 Reducing the search space

In this section, we investigate how to further limit the search space. The main idea behind our reduction is that we do not compute the commonalities and dissimilarities separately. Instead, we restrict the computation of the dissimilarities by taking only one dump from each bundle into account.

Let $\text{filter}(A, c) = \{a \in A \mid a \subseteq c\}$ denote the filtration of a collection of sets in A with respect to a set c. The sets of interest for an attribute mapping in Theorem 8.13 are characterized by the set

$$\text{smin}(\text{filter}(\text{diss}(a, S), \text{comm}(a, S))). \tag{8.1}$$

Let R be a set of representatives of $\text{bundles}(a, S)$, i.e. $\forall_{b \in \text{bundles}(a,S)} \ \exists!_{s \in R}\ s \in b$. The following theorem states that the set

$$\text{smin}(\text{diss}(a, R|_{\text{comm}(a,S)})) \tag{8.2}$$

contains the same index sets as (8.1). Expression (8.2) suggests, however, a smaller search space than (8.1), since the diss function is computed only over a restricted set of indexes and a subset of the dump set.

Theorem 8.14. *Let $a \in \mathbb{A}$ be an attribute and $S \subseteq \mathbb{B}^n$ a dump set. Let R be a set of representatives of* $\text{bundles}(a, S)$. *Then* $\text{smin}(\text{diss}(a, R|_{\text{comm}(a,S)})) = \text{smin}(\text{filter}(\text{diss}(a, S), \text{comm}(a, S)))$.

To build up our intuition, we first formulate the lemma that by expanding a dump set we might be able to locate an attribute more precisely.

Lemma 8.15. *Let S, $S' \subseteq \mathbb{B}^n$ be dump sets and $a \in \mathbb{A}$ an attribute. Then $S' \subseteq S \Rightarrow \text{diss}(a, S') \supseteq \text{diss}(a, S)$.*

Proof. By Lemma 8.12, let T be the unique subset-minimal set for which $\overline{T} = \text{diss}(a, S)$. We show that $T \subseteq \text{diss}(a, S')$.

Let $I \in T$. Then by definition of diss, $\forall_{s,s' \in S}\ (\text{val}_a(s) \neq \text{val}_a(s') \Rightarrow \exists_{i \in I}\ s_i \neq s'_i)$. But since $S' \subseteq S$, the statement holds in particular for any two dumps in S'. Thus $I \in \text{diss}(a, S')$. □

The preceding lemma indicates in particular that a dump set contains more information about an attribute than its subset of representatives. If we filter the $\text{diss}(a, S)$ sets with respect to the $\text{comm}(a, S)$ set, however, then the representatives are sufficient.

Lemma 8.16. *Let $S \subseteq \mathbb{B}^n$ be a dump set and $a \in \mathbb{A}$ an attribute. Let R be a set of representatives of* $\text{bundles}(a, S)$. *Then* $\text{filter}(\text{diss}(a, S), \text{comm}(a, S)) = \text{filter}(\text{diss}(a, R), \text{comm}(a, S))$.

Proof. We obtain $\text{filter}(\text{diss}(a, S), \text{comm}(a, S)) \subseteq \text{filter}(\text{diss}(a, R), \text{comm}(a, S))$ by Lemma 8.15.

For the reverse inclusion, let $I \in \text{filter}(\text{diss}(a, R), \text{comm}(a, S))$ be an index set in the filtration of $\text{diss}(a, R)$ with respect to the common set of the attribute a of the dumps in S. Then we have that $I \in \text{diss}(a, R)$ and $I \subseteq \text{comm}(a, S)$.

Suppose towards a contradiction that $I \notin \text{filter}(\text{diss}(a, S), \text{comm}(a, S))$. Then by definition of $\text{diss}(a, S)$, there must be dumps $s, s' \in S$ such that $s|_I = s'|_I$, but $\text{val}_a(s) \neq \text{val}_a(s')$.

Consider representatives $r, r' \in R$ of s and s' such that $\mathrm{val}_a(r) = \mathrm{val}_a(s) \neq \mathrm{val}_a(s') = \mathrm{val}_a(r')$. Since $I \subseteq \mathrm{comm}(a, S)$, it follows that $r|_I = s|_I = s'|_I = r'|_I$. However, since $\mathrm{val}_a(r) \neq \mathrm{val}_a(r')$, we have $I \notin \mathrm{diss}(a, R)$. This contradicts $I \in \mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S))$. □

The filter with respect to the $\mathrm{comm}(a, S)$ set in the preceding lemma is indeed necessary. In general, the set $\mathrm{diss}(a, R)$ does not coincide with $\mathrm{diss}(a, S)$.

Consider, for instance, the three two-bit dumps $s_1 = 01$, $s_2 = 00$, and $s_3 = 11$. Suppose the dumps encode the attribute a with $\mathrm{val}_a(s_1) = \mathrm{val}_a(s_2) = A$ and $\mathrm{val}_a(s_3) = B$. Then we have the following bundles and dissimilarity sets.

$$\begin{aligned}
\mathrm{bundles}(a, \{s_1, s_2, s_3\}) &= \{\{s_1, s_2\}, \{s_3\}\} \\
\mathrm{diss}(a, \{s_1, s_2, s_3\}) &= \overline{\{\{0\}, \{0, 1\}\}} \\
&= \{\{0\}\} \\
\mathrm{diss}(a, \{s_2, s_3\}) &= \overline{\{\{0\}, \{1\}, \{0, 1\}\}} \\
&= \{\{0\}, \{1\}\}
\end{aligned}$$

Thus, in spite of the fact that s_1 and s_2 have a common value for the attribute a, considering both in the dissimilarities set provides more information.

Proof of Theorem 8.14. It follows from Lemma 8.16 that it is sufficient to prove that $\mathrm{smin}(\mathrm{diss}(a, R|_{\mathrm{comm}(a,S)})) = \mathrm{smin}(\mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S)))$.

By uniqueness of subset minimal sets (Lemma 8.12), it suffices to show that $\mathrm{diss}(a, R|_{\mathrm{comm}(a,S)}) = \mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S))$.

The inclusion $\mathrm{diss}(a, R|_{\mathrm{comm}(a,S)}) \subseteq \mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S))$ holds as follows. Let $I \in \mathrm{diss}(a, R|_{\mathrm{comm}(a,S)})$. Then by the fact that dissimilarity sets are superset closed, we have that $I \in \mathrm{diss}(a, R)$ and $I \subseteq \mathrm{comm}(a, S)$. Therefore, we have $I \in \mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S))$.

The inclusion $\mathrm{diss}(a, R|_{\mathrm{comm}(a,S)}) \supseteq \mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S))$ holds as follows. Let $I \in \mathrm{filter}(\mathrm{diss}(a, R), \mathrm{comm}(a, S))$. Then by definition of filter, we have that $I \in \mathrm{diss}(a, R)$ and $I \subseteq \mathrm{comm}(a, S)$, thus $I \in \mathrm{diss}(a, R|_{\mathrm{comm}(a,S)})$. □

8.1.5 Cyclic attribute mappings

In this section we extend our results to a class of dynamic mappings, which we call semi-dynamic or cyclic attribute mappings[1]. Whereas a static mapping always encodes an attribute in a single location, cyclic mappings store the attributes cyclically in a fixed set of locations. This data structure resembles a ring buffer. We will show that cyclic mappings can be straightforwardly derived from normal attributes. Therefore, the theory developed in Section 8.1 as well as the algorithms of Section 8.2 can be reused.

Example 8.17 (Cyclic mappings)**.** *Cyclic mappings can be used to store* trip frames *on a public transportation card. Such a trip frame contains all information related to a single ride. Trip frames are stored in one of a fixed number of slots in the card's memory. When validating the card for a new ride, a new trip frame*

[1]The formalization and analysis of cyclic attribute mappings is due to Sjouke Mauw.

will be written to the next available slot. If all slots have been filled, the next trip frame will be written to the first slot again, etc. In this manner, a trip frame with x slots will keep a history of the x most recent rides.

Because cyclic mappings consider the evolution of a given object in time, we first assume additional structure on the dump set corresponding to the history of an object. We assume that for each dump we can determine to which object it belongs through the attribute *ID* (e.g. the unique identifier of a public transportation card). For each object we further assume that its dumps are ordered as expressed by an attribute *seqnr*.

Definition 8.18 (Bundle ordering)**.** *Let $S \subseteq \mathbb{B}^n$ be a dump set and let ID and seqnr be attributes. We say that the pair $(ID, seqnr)$ is a* bundle ordering *if* $\text{type}(seqnr) = \mathbb{N}$ *and*

$$\forall_{b \in \text{bundles}(ID,S)}\ \forall_{s,s' \in b} \quad s \neq s' \Rightarrow \text{val}_{seqnr}(s) \neq \text{val}_{seqnr}(s').$$

Because the combination of a device identifier and a sequence number uniquely determines a dump, we can consider an attribute a as a function on $\text{type}(ID) \times \mathbb{N}$. Given $i \in \text{type}(ID)$ and $n \in \mathbb{N}$ we write $a(i, n)$ for $\text{val}_a(s)$, where $s \in S$ is the dump uniquely determined by $\text{val}_{ID}(s) = i$ and $\text{val}_{seqnr}(s) = n$. Based on the *ID* and *seqnr*, we are able to generate new attributes from a given attribute a. We call these attributes *derived attributes*.

Example 8.19 (Derived attribute)**.** *We consider the history of a device. In particular, suppose we want to verify whether the previous value of an attribute a is stored. We derive the attribute a_{-1}, which is the a-value of the direct predecessor of a dump. This attribute is defined by $a_{-1}(i, n) = a(i, n-1)$. It is defined on a subset of S, viz. $\{s \in S \mid \exists s' \in S\ \text{val}_{ID}(s') = \text{val}_{ID}(s) \wedge \text{val}_{seqnr}(s') = \text{val}_{seqnr}(s) - 1\}$. This generalizes to a_{-r} for $r \in \mathbb{N}$.*

Derived attributes are particularly useful when dealing with cyclic attribute mappings. A cyclic mapping of attribute a considers a number of locations to store the value of a, for instance, $[i_0, j_0)$, $[i_1, j_1)$ and $[i_2, j_2)$. In the first dump of an ordered *ID*-bundle the value of a is stored at $[i_1, j_1)$. In the second dump a is stored at $[i_2, j_2)$, etc. The location for the fourth value of a is again $[i_0, j_0)$. The main observation here is that the attribute a is stored in the dumps as three derived attributes.

In order to locate a cyclic mapping of cycle length c for attribute a we derive c new attributes $a_{cycle(x/c)}$ (for $0 \leq x < c$). Using notation $\lfloor r \rfloor$ for the *floor* of rational number r, we obtain the following definition of these new attributes:

$$a_{cycle(x/c)}(i, n) = a(i, c \cdot \left\lfloor \frac{n - x}{c} \right\rfloor + x).$$

In order to find the cycle length of a cyclically mapped attribute, it suffices to search for attributes $a_{cycle(0/c)}$, where c ranges from 2 to the expected maximum cycle length.

	rl	$rl_{cycle(0/3)}$	$rl_{cycle(1/3)}$	$rl_{cycle(2/3)}$	$seqnr_{mod(3)}$
s_1	8	8	-	-	1
s_2	7	8	7	-	2
s_3	6	8	7	6	3
s_4	5	5	7	6	1
s_5	4	5	4	6	2
s_5	3	5	4	3	3

Figure 8.2: Derived attributes with cycle length 3.

Example 8.20 (Cyclic mapping: rides left)**.** *We consider* 6 *consecutive dumps of a single public transportation card. Furthermore, we assume that the* rides-left *(rl) attribute is a cyclic attribute of cycle length* 3*. Figure 8.2 shows the attribute rl and the three derived attributes* $rl_{cycle(x/3)}$ *(for* $x = 0, 1, 2$*).*

We conclude our observations on cyclic mappings by considering pointers to such attributes. An example is the use of a pointer (at a static location), pointing at the block in memory where the information on the most recent trip is stored. Clearly, if the trip information is stored cyclically at different locations, the pointer has a similar cyclic behavior. We can search for such cyclic pointers by introducing attributes $seqnr_{mod(c)}$, which consider the sequence number of the dump modulo cycle length c. Figure 8.2 contains an example for cycle length $c = 3$.

8.2 Algorithms

8.2.1 Commonalities

The algorithm computing the comm function identifies all positions in which given bitstrings have the same value. We implement it using the function $\mathrm{fc}\colon \mathcal{P}(\mathbb{B}^*) \times \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ which we define recursively as follows, using the symbol $\cup$ for the disjoint union of sets.

$$\begin{aligned} \mathrm{fc}(\emptyset, I) &= I \\ \mathrm{fc}(\{s\}, I) &= I \\ \mathrm{fc}(S \dot\cup \{s, s'\}, I) &= \mathrm{fc}(S \cup \{s\}, \{i \in I \mid s_i = s'_i\}) \end{aligned}$$

For dumps of length n,

$$\mathrm{comm}(a, S) = \bigcap_{b \in \mathrm{bundles}(a,S)} \mathrm{fc}(b, [0, n)).$$

The time complexity of this step is $O(n \cdot |S|)$.

Example 8.21 (Commonalities algorithm)**.** *The function* comm *is explained in Figure 8.3. For each of the three bundles* fc *is calculated as the set of all positions where all dumps from the bundle agree on the bit value (indicated by the asterisk symbols). Finally, the* comm *set cm is the intersection of these* fc *sets.*

	rl	dump	
s_1	4	0101001001110100 00	
s_2	4	0011001000010100 10	
		*..******..*****.*	fc
s_3	5	1011101010110101 00	
		**********	fc
s_4	6	0010101101110110 11	
s_5	6	1110101100110110 01	
		..*******.******.*	fc
		...******..*****.*	*cm*

Figure 8.3: Calculation of the comm set.

8.2.2 Dissimilarities

Given a set of bundles, the algorithm for the diss function identifies intervals in which any two bitstrings from different bundles differ in at least one position.

We implement the diss function in the case where the attribute mapping is assumed to be contiguous using the *dissimilarity interval function* $\mathrm{iv}_{a,S}(i)$. It denotes the shortest interval that a contiguous encoding of attribute a must have if it is to start at position i. Such an interval does not exist if there are dumps in S which do not differ at any position in $[i, n)$. In the remainder of this chapter, we adopt the conventions $\min(\emptyset) = \infty$, $\max(\infty, k) = \infty$, and $\max(-\infty, k) = k$ for all $k \in \mathbb{N} \cup \{-\infty, \infty\}$.

Definition 8.22 (Dissimilarity interval function)**.** *Let $a \in \mathbb{A}$ be an attribute and $S \subseteq \mathbb{B}^n$ be a dump set. The* dissimilarity interval function $\mathrm{iv}_{a,S} : [0, n) \to \mathbb{N} \cup \{\infty\}$ *of S with respect to attribute a is defined by*

$$\mathrm{iv}_{a,S}(i) = \min\{k \in [i, n) \mid \forall_{s,s' \in S} \, (\mathrm{val}_a(s) \neq \mathrm{val}_a(s') \Rightarrow \exists_{i \leq j \leq k} \, s_j \neq s'_j)\}.$$

The following lemma expresses that the dissimilarity set for contiguous attribute maps can be obtained from the dissimilarity interval function. To state the lemma, we first need to define subset minimality and superset closure for sets of intervals.

Let $\mathcal{I}_n = \{[i, j] \subseteq \mathbb{N} \mid i, j < n\}$ be the set of intervals in $[0, n)$. We define the *interval-superset closure* of a set $P \subseteq \mathcal{I}_n$ by $\{p \in \mathcal{I}_n \mid \exists_{p' \in P} \, p' \subseteq p\}$. The interval-superset closure of P is equal to $\overline{P} \cap \mathcal{I}_n$. A set P is said to be *interval-superset closed* if $P \subseteq \mathcal{I}_n$ and $P = \overline{P} \cap \mathcal{I}_n$. We say that P is *interval-subset minimal* if $P \subseteq \mathcal{I}_n$, and for every $p, p' \in P$, $p' \subseteq p \Rightarrow p' = p$. One can see that for every set of intervals $P \subseteq \mathcal{I}_n$, there is a unique interval-subset minimal set $Q \subseteq \mathcal{I}_n$ such that $\overline{Q} \cap \mathcal{I}_n = \overline{P} \cap \mathcal{I}_n$. The proof is analogous to the proof of Lemma 8.12.

Lemma 8.23. *Let $S \subseteq \mathbb{B}^n$ be a dump set and $a \in \mathbb{A}$ an attribute. Let the set T be defined by*

$$T = \{[i, j) \subseteq \mathbb{N} \mid i \in [0, n) \wedge j = \mathrm{iv}_{a,S}(i) \wedge j < \mathrm{iv}_{a,S}(i + 1)\}$$

then T is the interval-subset-minimal set that satisfies $\overline{T} \cap \mathcal{I}_n = \mathrm{diss}(a, S) \cap \mathcal{I}_n$.

Proof. T is interval-subset-minimal by definition.

From the definition of T, we observe that $T \subseteq \mathrm{diss}(a,S) \cap \mathcal{I}_n$. Since $\mathrm{diss}(a,S) \cap \mathcal{I}_n$ is interval-superset closed, it follows that $\overline{T} \cap \mathcal{I}_n \subseteq \mathrm{diss}(a,S) \cap \mathcal{I}_n$.

Suppose towards a contradiction that $\overline{T} \cap \mathcal{I}_n \subsetneq \mathrm{diss}(a,S) \cap \mathcal{I}_n$. Then there exists $I \in \mathrm{diss}(a,S) \cap \mathcal{I}_n$ such that $I \notin \overline{T} \cap \mathcal{I}_n$. Let $I = [i,j)$ and consider $\mathrm{iv}_{a,S}(i)$. By definition of $\mathrm{iv}_{a,S}$, we have $[i, \mathrm{iv}_{a,S}(i)) \subseteq I$ and we know that $[i, \mathrm{iv}_{a,S}(i)) \in \overline{T} \cap \mathcal{I}_n$. This contradicts $I \notin \overline{T} \cap \mathcal{I}_n$. □

To compute $\mathrm{iv}_{a,S}(i)$ for $i \in [0,n)$, we assume that no two dumps in S have the same value for attribute a, that is, we are restricting ourselves to a set of representatives R of $\mathrm{bundles}(a,S)$.

A naive algorithm for $\mathrm{iv}_{a,R}(i)$ is to select a dump $r \in R$ and compare it to all other dumps in R. In each comparison the minimal position k after i in which the two dumps differ is sought for. The output of this step is the maximal value of k and it defines the minimal interval in which r differs from all other dumps in R. This process is repeated for all dumps in R and the maximum k of all iterations is returned. More precisely, let $\mathrm{fiv} : \mathcal{P}(\mathbb{B}^*) \times \mathbb{N} \to \mathbb{N} \cup \{-\infty, \infty\}$ be recursively defined as follows.

$$\begin{aligned}
\mathrm{fiv}(\emptyset, i) &= -\infty \\
\mathrm{fiv}(\{r\}, i) &= -\infty \\
\mathrm{fiv}(\{r, r'\}, i) &= \min\{k \in \mathbb{N} \mid k \geq i \wedge r_k \neq r'_k\}) \\
\mathrm{fiv}(R \cup \{r\}) &= \max(\mathrm{fiv}(R, i), \max_{r' \in R}\{\mathrm{fiv}(\{r, r'\}, i)\})
\end{aligned}$$

The number of comparisons of two dumps, i.e. the number of calls to $\mathrm{fiv}(\{r,r'\},i)$, is easily seen to be quadratic in $|R|$. We can improve the number of comparisons to $O(|R| \log |R|)$ by sorting the set of dumps first. We write $r <_i r'$ if and only if $\exists_{j \in [i,n)} r_j < r'_j \wedge \forall_{i \leq k < j} r_k = r'_k$. We write $r \leq_i r'$ if $r <_i r'$ or $\forall_{j \in [i,n)} r_j = r'_j$.

A more efficient algorithm $\mathcal{A}$ to compute $\mathrm{iv}_{a,S}(i)$ runs as follows.

1. Set $k = -\infty$.
2. Sort the dump set R in ascending order with respect to $\leq_i$. Let $r^{(1)} \leq_i r^{(2)} \leq_i \ldots \leq_i r^{(|R|)}$ be the sorted list of these dumps.
3. For j from 1 to $|R| - 1$, compare $r^{(j)}$ with $r^{(j+1)}$. For the comparison, start with the i-th bit and move towards the $n-1$-st bit. Let k_j be the index of the first bit for which $r^{(j)}_{k_j} \neq r^{(j+1)}_{k_j}$. If no such bit exists, output ∞ and stop. Otherwise, set $k = max(k, k_j)$
4. Output k.

Example 8.24 (Dissimilarity intervals)**.** *The calculation of the* diss *intervals is illustrated in Figure 8.4. We start by taking a representative of each of the bundles. Then, starting from the left, we calculate for each position how far to the right we must go in order to find a distinguishing bit for each pair of dumps. For position* 0

the first two bits already make a distinction between the three dumps, which gives the interval $[0, 1]$ *(indicated by the first line with asterisk symbols). For position* 1 *we need three bits, because* s_3 *and* s_4 *coincide at positions* 1 *and* 2*. This gives the interval* $[2, 4]$*, etc. Those sets belonging to the subset-minimal* diss *set are marked with "minimal".*

	rl	dump	
s_1	4	0101001001110100 00	
s_3	5	1011101010110101 00	
s_4	6	0010101101110110 11	
		**.	minimal
		.***.	
		. .**.	minimal
		. . .**.	minimal
		. . . .****.	minimal
		. . . . .****.	
		. . . . . .***.	
		. . . . . .**.	minimal
		etc.	

Figure 8.4: Calculation of the diss intervals.

Example 8.25 (Commonalities and dissimilarities)**.** *If we combine the* comm *set from Figure 8.3 and the* diss *set from Figure 8.4, under the assumption that the number of rides is encoded with* 4 *bits, we obtain the four remaining possibilities from Figure 8.5. This result includes the three possible attribute mappings from Figure 8.1.*

	rl	dump
s_1	4	0101001001110100 00
s_2	4	0011001000010100 10
s_3	5	1011101010110101 00
s_4	6	0010101101110110 11
s_5	6	1110101100110110 01
		. . .****. . . . .
		. . . .****. . . . .
		. . . . .****. . . . .
		. . . . . .****. .

Figure 8.5: The resulting attribute mappings.

Theorem 8.26. *Let* R *be a set of representatives of the sets in* bundles(a, S')*. Let* n *be the bit length of the dumps in* R*. Then* $\mathrm{iv}_{a,R}(i)$ *is computed for all* $0 \leq i < n$ *by* $\mathcal{A}$ *in time* $O(n^2|R| + n|R| \log |R|)$*.*

Proof. We first prove correctness of the algorithm and then compute its time complexity.

Correctness We need to show that for all $r, r' \in R$ there exists an index $ind \in [i, k]$ such that $r_{ind} \neq r'_{ind}$. To this end we show the following invariant of the algorithm $\mathcal{A}$:

$$inv = \forall_{1 \leq x < y < j+1} \exists_{i \leq ind \leq k} r^{(x)}_{ind} \neq r^{(y)}_{ind}.$$

The invariant *inv* implies that any intermediate result k in step j is correct. For $j = 0$, *inv* trivially holds. Assume *inv* holds for j, we show that it also holds for $j + 1$.

To maintain *inv* we set $k' = \max(k, k_{j+1})$ and show that for all $1 \leq x < y < j + 2$ there exists an index *ind* such that $i \leq ind \leq k'$ and $r^{(x)}_{ind} \neq r^{(y)}_{ind}$. Since we know that *inv* holds for j, we need to show that for all $1 \leq x < j+1$, there exists such an index and $r^{(x)}_{ind} \neq r^{(j+1)}_{ind}$. This amounts to showing that the dump $r^{(j+1)}$ differs from all dumps $r \leq_i r^{(j+1)}$. The case $x = j$ is immediately satisfied since the algorithm computes k_{j+1} such that $r^{(j)}_{k_{j+1}} \neq r^{(j+1)}_{k_{j+1}}$. For any $x < j$, due to the sorting step $r^{(x)} \leq_i r^{(j)} <_i r^{(j+1)}$, thus $r^{(x)}_{k_{j+1}} < r^{(j+1)}_{k_{j+1}}$. Since $k' = max(k, k_{j+1})$, the invariant *inv* holds for $j + 1$. It thus holds for all j, in particular $|R| - 1$, showing correctness of the algorithm.

Complexity The complexity of the algorithm is given by the complexity to sort the dump set and the complexity to compare adjacent dumps in the sorted list. The bit-complexity for comparing the adjacent dumps $r^{(j)}, r^{(j+1)}$ is bounded above by the bit length n of the dumps. Sorting the dumps in R can be done in $|R|log|R|$ comparisons. Thus $\mathrm{iv}_{a,R}(i)$ can be computed in time $O((n-i)|R| + (n-i)|R| \log |R|) = O((n-i)|R| \log |R|)$.

If $\mathrm{iv}_{a,R}(i)$ is computed for all $i \in [0, n)$, the sorting complexity for $i > 0$ can be lowered by taking advantage of the sorted list of dumps with respect to $>_{i-1}$. We merely need to perform a merge-sort for $<_i$ on two sets given by the restrictions $r_{i-1} = 0$ and $r_{i-1} = 1$ and ordered with respect to $<_{i-1}$. This can be performed in time $O((n-i)|R|)$. By summing up the time it takes to compute $\mathrm{iv}_{a,R}(i)$ for $i \in [0, n)$ we obtain the theorem.

□

8.3 The mCarve tool

We have implemented the algorithms of Section 8.2 in a prototype. The prototype, called mCarve[2], allows the forensic analyst to input a collection of dumps and a collection of attributes. Each of the dumps can be accompanied by its attribute values. The prototype was written in Python and consists of approximately 1200 lines of code (excluding graphical user interface).

After entering the dumps and attributes the user can run the *commonalities* algorithm for an attribute. The output of the algorithm is the set of indexes I for which all dumps with the same attribute value are the same. The set I is used as a coloring mask to display any dump d selected by the user: if $i \in I$, then d_i is colored blue, otherwise red. The *dissimilarities* algorithm computes a subset-minimal set of dissimilarity intervals. Since these intervals may be overlapping, the prototype enumerates them rather than showing them as one coloring mask. This allows the user to step through the intervals. The prototype displays the interval iv by applying a yellow coloring mask to all bits d_i for $i \in \mathrm{iv}$. A *combined* procedure consolidates the results from the commonalities and dissimilarities algorithms.

[2]mCarve is available at `http://satoss.uni.lu/mcarve`.

The prototype further allows users to specify two types of special attributes: a *constant* attribute and a *hash*

attribute. The former has a constant value for all dumps and can be used to determine which bits never change. The latter has a different value for all pairwise different dumps and can be used to detect encrypted attributes. The tool allows one to derive new attributes from other attributes. These derived attributes can be used to find cyclic attribute mappings. The tool further allows one to apply an encoding to a selected interval in each dump. A number of standard encodings, such as ASCII and base 10, are implemented. Aside from displaying the output on-screen, the user can choose to export the results to JPEG or to LaTeX (see Figure 8.7 for an example).

8.3.1 Performance

We illustrate the performance of our prototype by running our prototype on a generated test suite. The test suite consists of dumps of sizes 8KB, 16KB, 32KB, 64KB, 128KB, and 256KB. For each file size, 5 dump sets were generated. Each dump embeds one attribute at a random position which is encoded in at most 64 bits. The remaining bits are randomly generated.

The running time of the commonalities procedure is linear in the number of dumps and the dissimilarities procedure is quadratic in the number of bundles. Therefore, the execution time of the combined procedure is mainly dependent on the number of bundles in the dump set. Convergence tests show that (Section 8.3.2), in general, fewer than 10 bundles are needed to find an attribute in a dump set. This allows us to restrict our performance tests to dump sets of 10 bundles.

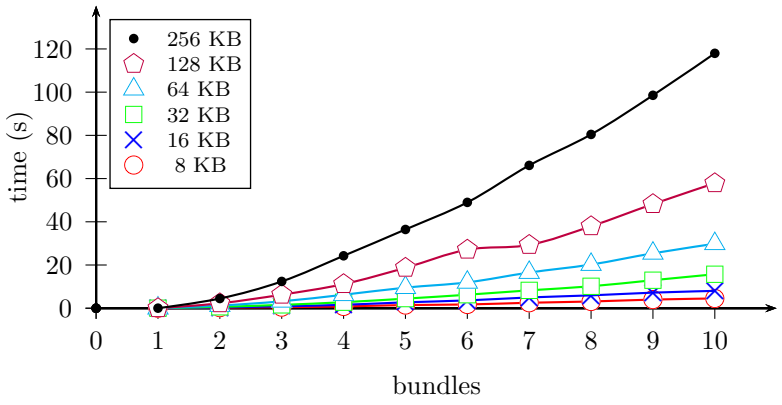


Figure 8.6: Performance

The tests were run on a Linux machine (kernel 2.6.31-22) with Intel Core 2 6400 @ 2.13 GHz processor running Python 2.6.4. Figure 8.6 shows on the horizontal axis the number of bundles included in the dump set. On the vertical axis it shows for each of the file sizes the time in seconds (averaged over the 5 dump sets) needed to perform the combined procedure. The test shows that our prototype is best suited for dumps of size smaller than 32KB, but it can deal reasonably well with size up to 256KB. Initial experiments have shown that performance of the tool can be significantly improved by implementing the core procedures in a lower-level language.

8.3.2 Convergence

Another interesting measure for the mCarve tool is the *rate of convergence* of the carved intervals[3]. We will measure it by computing the number of dumps that are necessary in order to find an attribute in a dump set. For simplicity, we assume that the dumps as well as the attribute values are given by a uniformly random distribution.

Let q denote the bit length of the attribute's encoding in the dump, let N denote the number of dumps and let x be the number of bundles. We first compute the probability of false positives, i.e. the probability of an accidental occurrence of values matching an attribute. The probability that the bit string formed by a particular interval of q bits in all N dumps matches a particular given string of bits is 2^{-qN}. There are $\binom{2^q}{x} \cdot x!$ possible encodings of x different values. The probability that the q bits in all N dumps match one of these representations is therefore $2^{-qN}\binom{2^q}{x} \cdot x!$.

Thus if l denotes the length of the bit strings representing dumps, then the probability p_{nfp} of no false positives is given by

$$p_{nfp} \geq \left(1 - \frac{2^{-qN} \cdot 2^q!}{(2^q - x)!}\right)^{l-q+1}.$$

The inequality is due to the fact that the product on the right does not concern independent trials. We are interested in those values of x and N for which the probability p_{nfp} is large enough that the discovery of an attribute is not coincidental.

Using the inequality $\binom{n}{k} \leq \frac{n^k}{k!}$ we obtain

$$\left(1 - \frac{2^{-qN} \cdot 2^q!}{(2^q - x)!}\right)^{l-q+1} \geq \left(1 - 2^{q(x-N)}\right)^{l-q+1}.$$

Fixing the number of bundles x and a false positive probability ϵ, we obtain the following inequality for the number of dumps N:

$$\left(1 - 2^{-(N-x)q}\right)^{l-q+1} > 1 - \epsilon.$$

Thus

$$N > \frac{-1}{q} \log_2 \left(1 - (1-\epsilon)^{\frac{1}{l-q+1}}\right) + x.$$

This formula can be used in two ways. If we know the length q of the encoding, we fix a number of bundles x and a false positive probability ϵ and compute the number of dumps N needed for convergence. If we do not know the length of q, we set it to $\log_2(x)$ and perform the same computation. For instance, for dumps of length $l = 1024$, false positive probability of $\epsilon = 0.05$, number of bundles $x = 4$, and length $q = \log_2(x) = 2$ we get $N > 11.14$. This means that to have convergence with probability 0.95 we need to analyze 12 dumps comprising 4 different attribute values.

[3]The convergence analysis is due to Saša Radomirović.

8.4 Case study: The e-go system

We illustrate our methodology by reverse engineering part of the memory structure of the Luxembourg public transportation card.

8.4.1 The e-go system

The fare collection system for public transportation in Luxembourg, called e-go, is based on radio frequency identification (RFID) technology. The RFID system consists of credit-card shaped RFID *tags* that communicate wirelessly with RFID *readers*. Readers communicate with a central back-end system to synchronize their data. Travelers can buy e-go cards with, for instance, a book of 10 tickets loaded on it. Upon entering a bus, the user swipes his e-go card across a reader and a ticket is removed from the card.

Since most RFID readers of the e-go system are deployed in buses the e-go is an *off-line* RFID system [GvR10]. Readers do not maintain a permanent connection with the back-end system, but synchronize their data only infrequently. Since readers may have data that is out-of-date and tags may communicate with multiple readers, the e-go system stores information on the card.

The RFID tags used for the e-go system are, in fact, MIFARE classic 1k tags. These tags have 16 sectors that each contain 64 bytes of data, totaling 1 kilo byte of memory. Sector keys are needed to access the data of each sector. Garcia et al. [GdKGM+08, GvRVS09] showed that these keys can be efficiently obtained with off-the-shelf hardware. Therefore, it is easy to create a memory dump of an e-go card.

8.4.2 Data collection

Over a period of 2 months, we collected 68 dumps for 7 different e-go cards of different types. Four cards are of type *10-rides/2nd-class*, two of type *1-ride/2nd-class* and one of type *1-ride/1st-class*. According to information published by the transportation companies, a card can contain up to 6 products of the same type. We considered two classes of events that change the state of a card: (1) charging the card with a new product (including the purchase of a new, charged card), and (2) validating a ride by swiping the card. After each event we dumped the memory of the card as a binary file. This gave a sequence of consecutive events for each card. Because the e-go system is an off-line system, we expected to find several attributes encoded on the card. For each event we therefore collected some contextual information, which we attributed to the dump following the event. For charge events we collected the following attributes: card ID (the decimal number printed on the card); charged product; date, time and location of charging; card charger ID (as printed on the receipt). For validation events we collected: card ID; date and time of swiping; expiration time of the ride; card reader ID (because the card readers have no visible identification we collected the license plate number of the bus and the location of the reader within the bus); rides left; bus number; bus stop.

These are the attributes that one would expect to find on the card and that are easy to observe. Most of these attributes can be obtained by reading the sales slips or the display of the reader. Since cards are purchased anonymously, no personal identifying information, such as name, address, or date-of-birth can be stored on the card.

In addition to our basic set of dumps, we had access to 47 dumps from earlier experiments which were less structured and less documented. We used these dumps to validate the results of the experiments with our main set of dumps.

It is important to note that our analysis is entirely passive: no data on the card needs to be modified and no data needs to be written to the card.

8.4.3 Data analysis

Using our mCarve tool, we verified the presence of three classes of attributes: (1) external attributes (i.e. the observable attributes mentioned above); (2) internal attributes (related to the organization of the data within the card's memory); and (3) attributes with high entropy (such as CRCs and cryptographic checksums). We searched for static as well as cyclic mappings of these attributes.

Memory layout

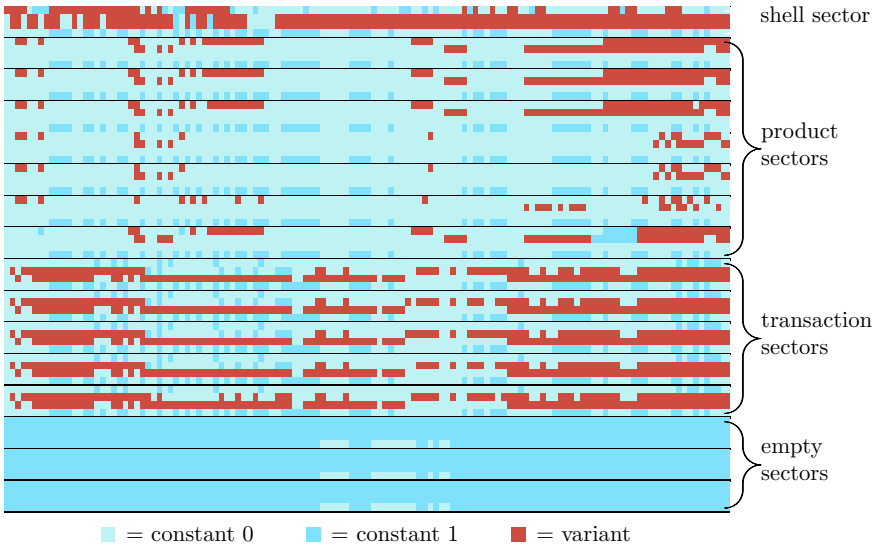


Figure 8.7: E-go memory layout (applying common to a unique attribute)

The first step in our analysis is to determine the general memory layout of an e-go card. For this purpose we define a *constant attribute* with the same value for all dumps. Applying the commonalities algorithm for this attribute results in the high-level memory layout shown in Figure 8.7. The card's memory is displayed in 64 lines of 128 bits, giving a total of 8192 bits (1KB). Bits that have a constant value in all dumps are colored differently from bits that vary in value. The recurring structures immediately suggest a partitioning of the memory into 16 sectors of 4 lines each. There seem to be four different types of sectors. The structure of the first sector is unique. We call this sector the *shell sector*. Lines 2 and 3 of the

shell sector are identical. Next there are seven sectors with a similar appearance (three of these look a bit less dense than the others because they are used less frequently in our dump set). We call these sectors the *product sectors*. The next five sectors are similar. We call them *transaction sectors*. Finally, there are three *empty sectors*, which we ignore for the rest of our analysis.

Further inspection shows that the last line of each sector is constant across all dumps. This line contains the two 48-bit *sector keys* as described by the MIFARE classic standard. Because the last lines of each of the sectors (except the empty sectors) are equal, we can conclude that the same key is used for all sectors.[4]

External attributes

The second step in our analysis is to carve the external attributes. This step only revealed the card ID. We can conclude that the other external attributes are either not represented on the card or not at a static location. Figure 8.8 shows for each sector type which attributes were discovered with our tool. The card ID, which is located in the shell sector in Figure 8.8, is detected as follows. The output of our tool on the card ID attribute consists of a number of intervals between bits 0 to 37 plus the interval 35 to 108. Clearly, the last interval is too large to contain the card ID, so we can consider that interval a false positive. We conclude that bits 0 to 37 are related to the card ID. Indeed, the MIFARE standard describes that identification numbers are hard-coded in the first 32 bits (4 bytes). If we reverse these 4 bytes and interpret them as a decimal number, we obtain the number printed on the card. The fact that bits 32 to 37 relate to the card ID is also consistent with the MIFARE standard because bits 32 to 39 contain the checksum of the card ID.

Internal attributes

The tool can be used to step through a sequence of dumps and observe the changes between consecutive dumps. In this way, one can step through the "history" of a particular card and observe recurring patterns. This process indicates a periodicity in the updates of the transaction sectors of the e-go card. Successive validation events write to successive transaction sectors, thereby cycling back from the fifth transaction sector to the first. One would expect a similar periodicity in the product sectors, but that is not the case. Writing to the product sectors occurs in an alternating way between two selected sectors.

Based on the hypothesis that there is a notion of a "current" sector, we carve for pointers with cycle lengths 2 to 7. By making a selection of those sequences of dumps that showed the cyclic behavior, we can locate a pointer to the currently active transaction sector (see tsec-ptr in the shell sector of Figure 8.8). This 3-bit pointer has a cycle of length 5 from 000 to 100. In a similar way one obtains a pointer with cycle 2, located at bit 169. Inspection of dumps reveals that this concerns a 3-bit pointer to the next active product sector (next-psec-ptr, bits 168-170). Two other pointers with cycle length 2 are only revealed when carving

[4] In order to not reveal sensitive data, we display keys that are different from those used in the e-go system.

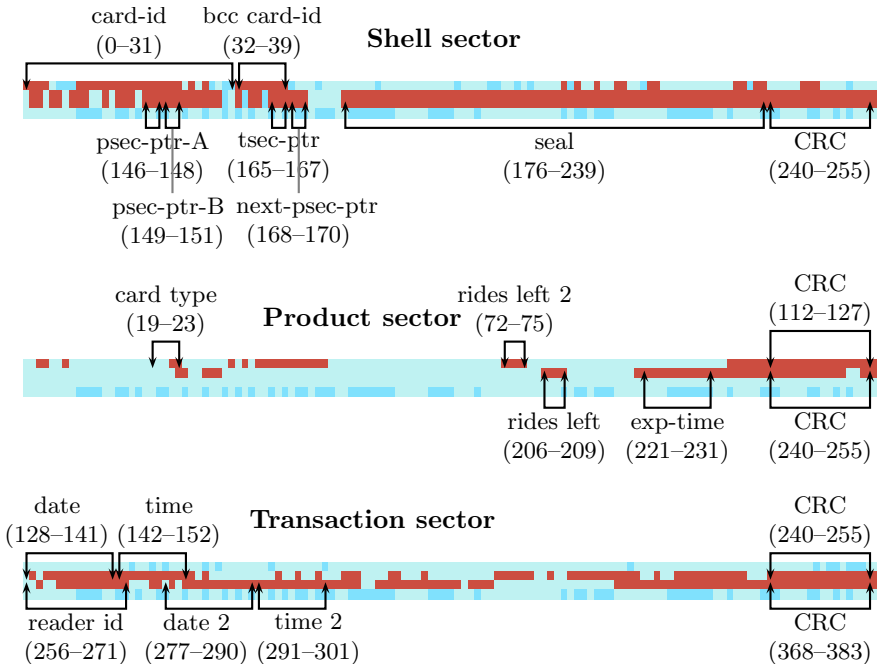


Figure 8.8: Attributes located in the three sector types

well-chosen subsets of the collection of dumps. In the figure they are labeled with psec-ptr-A and psec-ptr-B. When stepping through the dumps, it becomes clear that after each validation event the values of next-psec-ptr and one of psec-ptr-A or psec-ptr-B are swapped. When charging the card, psec-ptr-A and psec-ptr-B change roles.

Cyclic external attributes

After having been able to locate only a single static external attribute, we continue by searching for dynamically stored external attributes. By using cycle length 5, we can find two locations in each of the transaction sectors related to the date of the most recent validation In Figure 8.8 these locations are labeled with “date” and “date 2”. By stepping through a sequence of dumps swiped on consecutive days, it becomes clear that the date field is a counter. It counts the number of days since 1/1/1997. In our dump set the two dates are always identical. In a similar way we can find two fields related to the time of the most recent validation event. They count the number of minutes since midnight. The first and second time are different, but, surprisingly, their difference is not constant, which would have indicated a relation to the expiration time. The last attribute that can be located in the transaction sector is the reader ID. As explained, we use the license plate of the bus and the location of the reader within the bus to identify each reader. By combining these two attributes we obtain a new attribute that relates to the reader ID. Surprisingly, this new attribute does not occur in the dumps, but the license plate attribute does. This means that all readers in a given bus have the same ID. When interpreting the reader as a decimal number, one typically obtains numbers in the range from 1 to 150 for readers in a bus and from 10150 to 10200 for readers in a train station. This is consistent with carving for the attribute “bus-or-train”, which points at the most significant bits of the reader ID.

These attributes were found by reducing cyclic attributes to static attributes as described in Section 8.1.5. With this approach an attribute of cycle length 5 will change its value only every 5 dumps. As a consequence, this attribute has a rather slow convergence. Convergence can be improved, however, by focusing on the *active* transaction sector. In order to do this we created a new set of dumps, each of which only contained the active transaction sector of the old dump. Carving for the static external attributes in this new set of dumps results in the same findings, but the attributes can be located with significantly fewer dumps.

Using this approach we can easily locate three more attributes in the product sectors: the card type, the number of rides left on the card and the expiration time of the current product. A second field related to the number of rides left was also located (rides left 2 in the figure), which equals 12 minus rides left for 10-rides cards and 3 minus rides left for 1-ride cards.

High-entropy attributes

While using the mCarve tool, one quickly observes that the diss function returns intervals of varying widths sliding through the index set of the dumps. Heuristically, one expects the width of these sliding windows to be shorter over intervals corresponding to high-entropy attributes than over indexes corresponding to low-entropy attributes. Furthermore, the step size or distance between two such windows is expected to be smaller for high-entropy intervals. The observation of short-step narrow sliding windows led to the conjecture that the cards contain cryptographic data.

To confirm the existence of high-entropy attributes we carved for the *hash* attribute. This serves as an indicator for equality or inequality of two dumps and is a more robust approach to labeling distinct dumps with different attribute values than simply enumerating all dumps in a set. Carving for this artificial attribute amounts to looking for attribute values which change whenever the contents of the dump change. The tool revealed an 80-bit string in the shell sector. The same method applied to dumps of the product and transaction sectors revealed 16-bit strings which only change when the data in the corresponding sector changes.

Whereas an 80-bit string was expected to be a cryptographic hash, the 16-bit strings were suspected to be checksums such as CRCs. By trying out a list of commonly used CRCs to the data in the product and transaction sectors, the CRC-16-ANSI with polynomial $x^{16} + x^{15} + x^2 + 1$ was found to produce the observed values. This step led to the suspicion that a CRC might also be part of the 80 bit string in the shell sector, which was indeed found to be the case. The remaining 64 bits are expected to be a cryptographic seal protecting the integrity of the card's data.

Evaluation

Our tool performed quite well in this case study. We located the attributes as displayed in Figure 8.8 and have been able to infer the encoding scheme for most of them. On the other hand, we have not been able to locate all collected attributes.

We did not find the date, time and location of charging, the card charger ID, the bus number and the bus stop. Our experiments prove that they are not stored in a static or cyclic way on the card. We may assume that if the date and time of charging and the card charger ID were represented in the card's memory, they would have been encoded in the same way as the other dates, times and IDs. A search for these encoded values in the binary dumps did not give a hit. Therefore, we conjecture that these attributes are not stored on the card, not even at a dynamically determined location. Given that a validated ride allows for unrestricted travel through the whole country for two hours, there is also no need to store the bus number and bus stop on the card.

As a consequence of carving for internal attributes we have not only located four pointers, but we have also reverse engineered part of the dynamics of updating e-go cards. The transaction sectors are written to cyclically. They contain data related to the history of the card. The current state of each of the products on the card is stored in the product sectors. Every product is assigned to one sector, except the currently active product. This product is updated alternatingly in two sectors. This redundancy is probably built in to keep a consistent product state even if a transaction does not finish successfully. More safeguards against update errors are found in the frequent checksums that we have been able to locate. A protection against intentional modification of the stored data is the cryptographic seal in the shell sector.

Even though we found the majority of observable attributes, there are still locations in the card's memory that we have not been able to assign a meaning to. Of course, the current dump set provides no information on the meaning of the constant (blue) bits in Figure 8.8. The variant (red) bits either have to do with the internal organization of the card or with attributes that we did not or could not observe.

With respect to convergence, we see that the dumps in this case study behave slightly worse than the dumps in the idealized set from Section 8.3.2. Finding an attribute requires roughly 12 dumps covering 5 bundles. The time performance of the tool is sufficient to quickly perform experiments on the dump set. A single experiment with the tool typically costs one to a few seconds.

Occasionally, we incorrectly entered an attribute value. The algorithms that we developed are not robust against such mistakes, since a single modification in the input can drastically change the output. In practice, however, such mistakes were quickly identified by regularly performing experiments on a subset of the dump set, such as all dumps belonging to a single card.

A very useful feature of our methodology is that in the search for an attribute we do not presuppose a particular encoding of that attribute. This allowed us to search for the combination of license plate number and reader location in order to find the reader ID. Similarly, we found a rides-left counter counting down and one that counts up while searching for one attribute.

8.5 Related work

Closest to our work are file carving approaches for recovering files from raw data. These approaches try to recover the data of a single dump whereas we focus on recovering data (and data structures) of a set of dumps. Garfinkel [Gar07] describes several carving algorithms that recover files by searching for headers of known file formats. These algorithms reconstruct files based on their raw data, rather than using the metadata that points to the content. Cohen [Coh07] formalizes file carving as a construction of a mapping function between raw data bytes and image bytes. Based on this formalization, he derives a carving algorithm and applies it to PDF and ZIP file carving. Sencar and Memon [SM09] describe an approach to identify and recover JPEG files with missing fragments. Common to these file carving approaches is that they are designed for one (or a small set of) known file format(s). In contrast to our approach, they cannot be applied if the encoding of the attributes is not known. However, if the encoding is known, the algorithms are more efficient than ours.

More general, but perhaps less powerful than the previously mentioned approaches is visual inspection of binary data. Conti, Dean, Sinda, and Sangster [CDSS08] describe a tool that allows analysts to visually reverse engineer binary data and files. Their tool supports simple techniques such as displaying bytes as pixels, but also more complicated techniques that visualize self-similarity in binary data. Helfman [Hel96] first visualized self-similarity in binary data using dotplot patterns. Using dotplot patterns he revealed redundancy in various encodings of information. Although these visualization techniques require more manual labor than our approach, our mCarve tool might benefit from these research efforts.

Some information in a memory dump may be constructed using CRCs, cryptographic hashes, or encryption. Since the entropy of these pieces of data is higher than of structured data, they can be detected using entropy analysis. Several methods to efficiently find cryptographic keys are described by Shamir and Van Someren [SvS99]. Some of these techniques are based on trial-and-error, while others identify possible keys by measuring entropy. Testing whether a given string is random has been studied extensively. An overview and implementation of the most important algorithms is given by Rukhin, Soto, Nechvatal, Smid, and Banks [RSN+00]. It remains to be explored how these approaches relate to the analysis of high-entropy attributes as discussed in Section 8.4.3.

8.6 Conclusion

If an attacker obtains the cryptographic keys of an RFID tag he gets access to the raw binary data on the card. Understanding the structure of this raw data is essential for executing sophisticated attacks. In this chapter, we have developed a methodology for reverse engineering such data structures. This methodology takes advantage of the fact that an attacker can *repeatedly* dump the memory of a device and record attributes associated with each dump.

In terms of contributions, we have defined the carving problem for attributed dump sets as the problem of recovering the attribute mapping and encoding of attributes

in a dump. We have proposed algorithms for recovering the attribute mapping and proven their correctness. The first algorithm computes the commonalities to determine the positions in a dump that cannot be contained in the mapping. The second algorithm computes subset-minimal dissimilarity intervals to give a lower-bound on the bits that need to be contained in the attribute mapping. By combining these two algorithms, a set of possible mappings is derived.

We have validated our approach by implementing a prototype application, called mCarve, with commonality and dissimilarity algorithms. A case study performed on data from the electronic fare collection system in Luxembourg showed that mCarve is valuable in analyzing real-world systems. Using mCarve, we have located more than a dozen attributes on the e-go card as well as their encoding. We have also partly reverse engineered the dynamics of updating e-go cards.

Part V

Concluding remarks

9

Conclusion and future work

9.1 Conclusion

We restate the main research question of this thesis.

Research question. *How can we apply formal verification to analyze the security of RFID systems?*

Throughout this thesis we have applied formal verification to analyze the security of RFID systems. As a first step, we have created a model for analyzing RFID protocols (see Chapter 3). The model includes a language for describing RFID protocols and allows one to systematically derive the execution traces of the protocol. As such, it provides a rigorous way of describing attacks and proofs.

Within the model we have expressed the security requirement of untraceability. Our definition very closely resembles the intuitive definition of the property, i.e. a protocol is untraceable if an attacker cannot recognize a previously observed tag. We have shown how to apply our model to analyze the untraceability of existing RFID protocols.

We have analyzed a large number of existing RFID protocols with untraceability claims. Many of these protocols do not satisfy untraceability. It turns out that techniques to attack one protocol can often be used to attack another protocol. We have, thus, been able to classify attacks on untraceability into a number of categories (see Chapter 4). This classification can be used as a reference for common attacks, so that new RFID protocol proposals do not suffer from the same flaws.

There exist a number of different computational proof models for proving untraceability. They can be separated into unpredictability-based and indistinguishability-based proof models. We have studied two unpredictability-based proof models and have shown that they are neither sound, nor complete with respect to the intuitive notions of untraceability and to indistinguishability-based models (see Chapter 5). That is, in the unpredictability-based models, one can prove traceable protocols untraceable and vice versa. This shows that current unpredictability-based proof models should not be used to prove or disprove untraceability.

We have shown that insider attacks are a realistic threat to RFID protocols when implementing public-key operations on RFID tags becomes feasible (see Chapter 5). Insider attackers are attackers that have corrupted one or more legitimate tags in the system. They use the cryptographic key material of those tags to trace other tags in the same RFID system. We have extended an existing computational proof model with the notion of insider attackers. With this extension, it can be used to prove protocols (un)traceable against insider attackers.

We have proposed an RFID protocol that uses solely elliptic-curve operations (see Section 5.5). As a basis for the protocol, we have used an RFID protocol proposed by Vaudenay and the Cramer-Shoup encryption scheme. The construction of an elliptic-curve based protocol is interesting for several reasons. First, it has been shown that one cannot design an untraceable protocol without public-key cryptography if the attacker can corrupt tags. Second, it has been shown that it is feasible to implement public-key operations on a low-cost passive tag.

The majority of authentication flaws in RFID protocols is of an algebraic nature. That is, the attacker can abuse algebraic properties of the operators in the protocol to construct new valid messages. We have identified three classes of algebraic attacks on authentication of RFID protocols (see Chapter 6). For each of these classes, we have shown protocols that suffer from the attack.

We have argued that ownership of an RFID tag is not only concerned with physical possession of the tag, but also with the ability to interact with it in a meaningful manner. Since RFID tags may change hands during their lifetime, their "virtual" ownership must change as well. This is achieved by using ownership transfer protocols. We have extended our formal model with definitions of ownership and ownership transfer, as well as their secure variants (see Chapter 7). Our definitions can be used by protocol designers to formally prove secure ownership and secure ownership transfer of their protocols. We have also been able to use our notion of ownership to formalize desynchronization resistance. We have used this definition to show an attack on desynchronization resistance of a published protocol.

As a final topic, we have analyzed the problem of reverse engineering memory structures from a set of similarly structured dumps. This problem is relevant to RFID systems if the attacker can repeatedly dump the memory of an RFID tag. We have designed algorithms that recover the location of attributes in dumps based on the commonalities and dissimilarities between dumps (See Chapter 8). We have implemented these algorithms in a tool called mCarve and we have applied this tool to reverse engineer the memory structure of the public transportation card used in Luxembourg.

These contributions show how formal verification can be used to analyze RFID systems, answering our main research question. They show that formal verification is a useful approach to analyze the security of RFID systems. In particular, formal verification provides ways to define relevant security properties, identify pitfalls in designing secure RFID protocols, show the absence of flaws in RFID protocols, and ultimately design secure RFID systems.

9.2 Future work

There are several research directions that remain to be explored. We discuss research directions related to automated verification, ownership transfer, and carving.

9.2.1 Automated verification

We have only briefly touched upon automatic verification of RFID protocols. In Chapter 6 we have reported on a case study on an automated verification method for protocols employing the exclusive-or operator. The method developed in this case study was able to find an algebraic replay attack on a published RFID protocol.

A first direction is the automated verification of RFID protocols based on the protocol syntax and semantics developed in Chapter 3. The model was deliberately developed along the lines of the operational semantics for security protocols by Cremers and Mauw. The latter model is supported by a tool, called Scyther, that implements automated ways of checking secrecy and authentication of stateless security protocols. A possible next step is to extend the algorithms to support stateless security protocols. Of interest is also the design of algorithms to verify or falsify untraceability. We expect to be able to build on results by Arapinis, Chothia, Ritter, and Ryan [ACRR10] and Brusò, Chatzikokolakis, and Den Hartog [BCdH10].

A second direction concerns automating the process of finding attacks on RFID protocols. We believe that automated attack finding is possible for algebraic replay attacks and attribute acquisition attacks. Since exploring possible attacks requires repeated application of functions, we expect to be able to build on existing tools for state-space exploration.

9.2.2 Ownership transfer

Ownership transfer has not been studied in the same depth as untraceability and authentication. Our formalization of ownership and ownership-related properties is the first effort in formally analyzing ownership transfer.

We concern the construction and analysis of ownership transfer protocols as the most important next step. Most protocols in literature do not satisfy secure ownership and secure ownership transfer. A further direction concerns automatic ways of verifying their security properties. The shape of the secure ownership transfer definition suggests that correspondences as defined by Blanchet [Bla09] as well as the tool ProVerif [Bla01] can be used for automatic verification.

While we consider a formal definition of ownership to be of independent interest, it will clearly become much more valuable when combined with existing security and privacy properties. For instance, in a parcel delivery system, where RFID tags are attached to parcels, *non-repudiation* for obtaining ownership of RFID tags and *untraceability* of these tags by unauthorized entities become important. We have only briefly indicated the connections between untraceability and exclusive ownership. A useful next step is to study conditions under which untraceable protocols can be safely composed with ownership transfer protocols. This requires in particular an investigation into the interplay between two or more untraceable protocols out of a set of protocols.

9.2.3 Carving

Our algorithms and our tool mCarve have been designed to recover the location of attributes. To be able to understand the attribute values themselves, the encoding has to be recovered as well. In our case study, we have recovered the encoding of attributes manually, while automatic approaches should in some cases be feasible. Heuristic approaches seem most viable, possibly approaches based on file carving techniques. Secondly, the robustness of our algorithms can be improved. Currently, a small error in the data, due to, for instance, a transmission error or a mistake in inputting the attribute value will make the results unreliable. Although these mistakes can be found by hand, an automatic way would be preferable.

We would like to apply mCarve to other case studies. An interesting application would be the memory of a cell phone. Our performance results show that we have to optimize the implementation of our algorithms to analyze cell phone dumps. Initial experiments have shown that performance of the tool can be significantly improved by implementing the core procedures in a lower-level language. Another use of our mCarve will be to analyze proprietary communication protocols. By recording the data and applying our algorithms, we could reconstruct their specification. For all these applications, the user can create multiple dumps and observe values for possible attributes. Therefore, mCarve will be helpful in these reverse engineering case studies.

Finally, our procedure for combining commonalities and dissimilarities simply merges the outputs of the two algorithms. We would like to design a more efficient algorithm that exploits knowledge of commonalities to find dissimilarity intervals.

Bibliography

[ABB+05] A. Armando, D.A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P.H. Drielsma, P-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[AC06] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.*, 367(1-2):2–32, 2006.

[ACG+08] S. Andova, C.J.F. Cremers, K. Gjøsteen, S. Mauw, S.F. Mjølsnes, and S. Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206:425–459, February-April 2008.

[ACRR10] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *CSF*, pages 107–121. IEEE Computer Society, 2010.

[Avo05] G. Avoine. Adversary model for radio frequency identification. Technical report, Swiss Federal Institute of Technology (EPFL), Security and Cryptography Laboratory (LASEC), 2005.

[BAN89] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *SIGOPS Oper. Syst. Rev.*, 23(5):1–13, 1989.

[BAN90] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *Trans. Comput. Syst.*, 8(1):18–36, 1990.

[BBM00] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000.

[BCdH10] M. Brusò, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *CSF*, pages 75–88. IEEE Computer Society, 2010.

[BCI08] J. Bringer, H. Chabanne, and T. Icart. Cryptanalysis of EC-RAC, a RFID identification protocol. In *CANS*, volume 5339 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 2008.

[BCI09] J. Bringer, H. Chabanne, and T. Icart. Efficient zero-knowledge identification schemes which respect privacy. In *ASIACCS*, pages 195–205. ACM, 2009.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[BH10] D. Billard and R. Hauri. Making sense of unstructured flash-memory dumps. In *SAC*, pages 1579–1583. ACM, 2010.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.

[Bla07] B. Blanchet. Computationally sound mechanized proofs of correspondence assertions. In *CSF*, pages 97–111. IEEE Computer Society, 2007.

[Bla09] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[Bog07] Andrey Bogdanov. Linear slide attacks on the keeloq block cipher. In *Inscrypt*, volume 4990 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2007.

[BSSV11] L. Batina, S. Seys, D. Singelée, and I. Verbauwhede. Hierarchical ECC-based RFID authentication protocol. In *RFIDSec*, 2011. To appear.

[Can00] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[CDSS08] G.J. Conti, E. Dean, M. Sinda, and B. Sangster. Visual reverse engineering of binary and data files. In *VizSEC*, volume 5210 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.

[CH07] H.-Y. Chien and C.-W. Huang. A lightweight RFID protocol using substring. In *EUC*, volume 4808 of *Lecture Notes in Computer Science*, pages 422–431. Springer, 2007.

[CI09] J.-S. Coron and T. Icart. An indifferentiable hash function into elliptic curves. Cryptology ePrint Archive, Report 2009/340, 2009.

[CJ97] John A. Clark and Jeremy L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997.

[CM05] C.J.F. Cremers and S. Mauw. Operational semantics of security protocols. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *Lecture Notes in Computer Science*, pages 66–89. Springer, 2005.

[Coh07] M. I. Cohen. Advanced carving techniques. *Digital Investigation*, 4(3-4):119–128, 2007.

[Coo00] C. Cooper. On the distribution of rank of a random matrix over a finite field. *Random Struct. Algorithms*, 17(3-4):197–212, 2000.

[Cre06a] C.J.F. Cremers. Feasibility of multi-protocol attacks. In *ARES*, pages 287–294. IEEE Computer Society, 2006.

[Cre06b] C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Ph.D. dissertation, Eindhoven University of Technology, 2006.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[CS10] T. Chothia and V. Smirnov. A traceability attack against e-passports. In *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2010.

[CZW08] Q. Cai, Y. Zhan, and Y. Wang. A minimalist mutual authentication protocol for RFID system and BAN logic analysis. In *CCCM*, pages 449–453. IEEE Computer Society, 2008.

[Dam91] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 1991.

[Dim08] T. Dimitriou. Proxy framework for enhanced RFID security and privacy. In *CCNC*, pages 843–847. IEEE, 2008.

[DM07] R. Di Pietro and R. Molva. Information confinement, privacy, and security in RFID systems. In *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2007.

[DP08] I. Damgård and M.Ø. Pedersen. RFID security: Tradeoffs between security and efficiency. In *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2008.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[EA11] I. Erguler and E. Anarim. Scalability and security conflict for RFID authentication protocols. *Wireless Personal Communications*, 59, 2011.

[FA07] S. Fouladgar and H. Afifi. A simple privacy protecting scheme enabling delegation and ownership transfer for RFID tags. volume 2, pages 6–13. Academy Publisher, 2007.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004.

[FHV10] J. Fan, J. Hermans, and F. Vercauteren. On the claimed privacy of ec-rac iii, 2010.

[Gam85] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Gar07] S.L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 4s:2–12, 2007.

[GdKGM+08] F.D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs. Dismantling MIFARE classic. In *ESORICS*, volume 5189 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2008.

[GHPvR05] F.D. Garcia, I. Hasuo, W. Pieters, and P. van Rossum. Provable anonymity. In *FMSE*, pages 63–72. ACM, 2005.

[Gol98] D. Gollmann. Insider fraud (position paper). In *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 213–219. Springer, 1998.

[GRS05] Henri Gilbert, Matt Robshaw, and Herve Sibert. An active attack against HB^+ - A provably secure lightweight authentication protocol. Cryptology ePrint Archive, Report 2005/237, 2005.

[GvR10] F.D. Garcia and P. van Rossum. Modeling privacy for off-line rfid systems. In *CARDIS*, volume 6035 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2010.

[GvRVS09] F.D. Garcia, P. van Rossum, R. Verdult, and R. Wichers Schreur. Wirelessly pickpocketing a MIFARE classic card. In *IEEE Symposium on Security and Privacy*, pages 3–15. IEEE Computer Society, 2009.

[GvRVS10] F.D. Garcia, P. van Rossum, R. Verdult, and R. Wichers Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In *ACM Conference on Computer and Communications Security*, pages 250–259, 2010.

[Hel96] J. Helfman. Dotplot patterns: A literal look at pattern languages. *TAPOS*, 2(1):31–41, 1996.

[HMNB07] J. Ha, S.-J. Moon, J.M. González Nieto, and C. Boyd. Low-cost and strong-security RFID authentication protocol. In *EUC Workshops*, volume 4809 of *Lecture Notes in Computer Science*, pages 795–807. Springer, 2007.

[HMZH08] J. Ha, S.-J. Moon, J. Zhou, and J. Ha. A new formal proof model for RFID location privacy. In *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2008.

[HPVP11] J. Hermans, A. Pashalidis, F. Vercauteren, and B. Preneel. A new RFID privacy model. In *ESORICS*, Lecture Notes in Computer Science. Springer, 2011. To appear.

[HT96] N. Heintze and J.D. Tygar. A model for secure protocols and their compositions. *IEEE Trans. Software Eng.*, 22(1):16–30, 1996.

[Ica09] T. Icart. How to hash into elliptic curves. In *CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316. Springer, 2009.

[IKD+08] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A practical attack on keeloq. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[ISO08] ISO. ISO/IEC 9798-2,Information technology - security techniques entity authentication - Part 2: Mechanisms using symmetric encipherment algorithms, 2008.

[JH08] P. Jäppinen and H. Hämäläinen. Enhanced RFID security method with ownership transfer. In *CIS (2)*, pages 382–385. IEEE Computer Society, 2008.

[JW05] A. Juels and S.A. Weis. Authenticating pervasive devices with human protocols. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[JW09] A. Juels and S.A. Weis. Defining strong privacy for rfid. volume 13, 2009.

[KCL07] I.J. Kim, E.Y. Choi, and D.H. Lee. Secure mobile RFID system against privacy and security problems. In *SecPerU 2007*, pages 67–72. IEEE Computer Society, 2007.

[KCLL06] K.H. Kim, E.Y. Choi, S.-M. Lee, and D.H. Lee. Secure EPCglobal class-1 gen-2 RFID system against security and privacy problems. In *OTM Workshops (1)*, volume 4277 of *Lecture Notes in Computer Science*, pages 362–371. Springer, 2006.

[KN05] J. Kang and D. Nyang. RFID authentication protocol with strong resistance against traceability and denial of service attacks. In *ESAS*, volume 3813 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2005.

[KSM+07] K.H.S.S. Koralalage, M.R. Selim, J. Miura, Y. Goto, and J. Cheng. POP method: an approach to enhance the security and privacy of RFID systems used in product lifecycle with an anonymous ownership transferring mechanism. In *SAC*, pages 270–275. ACM, 2007.

[KSW97] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1997.

[KT08] R. Küsters and T. Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. In *ACM Conference on Computer and Communications Security*, pages 129–138, 2008.

[LAK06] S. Lee, T. Asano, and K. Kim. RFID mutual authentication scheme based on synchronized secret information. In *Symposium on Cryptography and Information Security*, 2006.

[LBSV10a] Y.K. Lee, L. Batina, D. Singelée, and I. Verbauwhede. Low-cost untraceable authentication protocols for RFID. In *WISEC*, pages 55–64. ACM, 2010.

[LBSV10b] Y.K. Lee, L. Batina, D. Singelée, and I. Verbauwhede. Wide-weak privacy-preserving rfid authentication protocols. In *MOBILIGHT*, volume 45 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 254–267. Springer, 2010.

[LBV08] Y.K. Lee, L. Batina, and I. Verbauwhede. EC-RAC (ECDLP based randomized access control): Provably secure RFID authentication protocol. In *IEEE RFID*, pages 97 –104. IEEE, 2008.

[LBV09] Y.K. Lee, L. Batina, and I. Verbauwhede. Untraceable RFID authentication protocols: Revision of EC-RAC. In *IEEE RFID*, pages 178–185. IEEE, 2009.

[LC07] H. Lei and T. Cao. RFID protocol enabling ownership transfer to protect against traceability and DoS attacks. In *ISDPE*, pages 508–510. IEEE Computer Society, 2007.

[LD07] Y. Li and X. Ding. Protecting RFID communications in supply chains. In *ASIACCS*, pages 234–241. ACM, 2007.

[LDL10] J. Lai, R.H. Deng, and Y. Li. Revisiting unpredictability-based RFID privacy models. In *ACNS*, volume 6123 of *Lecture Notes in Computer Science*, pages 475–492. Springer, 2010.

[LK06] C.H. Lim and T. Kwon. Strong and robust RFID authentication enabling perfect ownership transfer. In *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[Low97] G. Lowe. A hierarchy of authentication specification. In *CSFW*, pages 31–44. IEEE Computer Society, 1997.

[Low98] G. Lowe. Casper: a compiler for the analysis of security protocols. *J. Comput. Secur.*, 6(1-2):53–84, 1998.

[MLDL09] C. Ma, Y. Li, R.H. Deng, and T. Li. RFID privacy: relation between two notions, minimal condition, and efficient construction. In *ACM Conference on Computer and Communications Security*, pages 54–65. ACM, 2009.

[MM05] K. Michael and L. McCathie. The pros and cons of RFID in supply chain management. In *ICMB*, pages 623–629. IEEE Computer Society, 2005.

[Möl04] B. Möller. A public-key encryption scheme with pseudo-random ciphertexts. In *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 335–351. Springer, 2004.

[MSW05] David Molnar, Andrea Soppera, and David Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2005.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[NSMSN08] C.Y. Ng, W. Susilo, Y. Mu, and R. Safavi-Naini. RFID privacy models revisited. In *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 251–266, 2008.

[NSMSN09] C.Y. Ng, W. Susilo, Y. Mu, and R. Safavi-Naini. New privacy results on synchronized RFID authentication protocols against tag tracing. In *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2009.

[OP00] T. Okamoto and D. Pointcheval. PSEC-3: Provably secure elliptic curve encryption scheme - V3 (Submission to P1363a). In *IEEE P1363a*, 2000.

[OTYT06] K. Osaka, T. Takagi, K. Yamazaki, and O. Takahashi. An efficient and secure RFID security method with ownership transfer. In *CIS*, volume 4456 of *Lecture Notes in Computer Science*, pages 778–787. Springer, 2006.

[PV08] R.-I. Paise and S. Vaudenay. Mutual authentication in rfid: security and privacy. In *ASIACCS*, pages 292–299. ACM, 2008.

[RSN+00] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and D.L. Banks. A statistical test suite for random and pseudorandom number generators for statistical applications. *NIST Special Publication in Computer Security*, pages 800–22, 2000.

[Ser98] G. Seroussi. Compact representation of elliptic curve points over $\mathbb{F}_{2^n}$. Technical report, Research Contribution to IEEE P1363, 1998.

[SIS05] J. Saito, K. Imamoto, and K. Sakurai. Reassignment scheme of an RFID tag's key for owner transfer. In *EUC Workshops*, volume 3823 of *Lecture Notes in Computer Science*, pages 1303–1312. Springer, 2005.

[SM08] B. Song and C.J. Mitchell. RFID authentication protocol for low-cost tags. In *WISEC*, pages 140–147. ACM, 2008.

[SM09] H.T. Sencar and N. Memon. Identification and recovery of JPEG files with missing fragments. *Digital Investigation*, 6:88–98, 2009.

[Son08] B. Song. RFID Tag Ownership Transfer. In *Workshop on RFID Security - RFIDSec'08*, Budapest, Hungary, July 2008.

[STF05] T. Staake, F. Thiesse, and E. Fleisch. Extending the EPC network: the potential of RFID in anti-counterfeiting. In *SAC*, pages 1607–1612. ACM, 2005.

[SvdW06] A. Shallue and C. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANTS*, volume 4076 of *Lecture Notes in Computer Science*. Springer, 2006.

[SvS99] A. Shamir and N. van Someren. Playing "hide and seek" with stored keys. 1648:118–124, 1999.

[TH99] W.-G. Tzeng and C.-M. Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocols. *Inf. Process. Lett.*, 69(6):297–302, 1999.

[THG99] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.

[Ula07] M. Ulas. Rational points on certain hyperelliptic curves over finite fields. *Bull. Pol. Acad. Sci. Math.*, 55(2):97–104, 2007.

[Vau07] S. Vaudenay. On privacy models for rfid. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2007.

[Vul09] P. Vullers. Secure ownership and ownership transfer RFID systems. Master's thesis, Technische Universiteit Eindhoven, 2009.

[Wan04] R. Want. Enabling ubiquitous sensing with RFID. *IEEE Computer*, 37(4):84–86, 2004.

[War03] B. Warinschi. A computational analysis of the Needham-Schröeder-(Lowe) protocol. In *CSFW*, pages 248–262. IEEE Computer Society, 2003.

[YY08] E-J. Yoon and K. Yoo. Two security problems of RFID security method with ownership transfer. In *NPC Workshops*, pages 68–73. IEEE Computer Society, 2008.

Publications

[ICFEM] X. Chen, T. van Deursen, and J. Pang. Improving automatic verification of security protocols with XOR. In *ICFEM*, volume 5885 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 2009.

[Prime] T. van Deursen. 50 ways to break RFID privacy. In *PrimeLife*, volume 352 of *IFIP AICT*, pages 192–205. Springer, 2011.

[Wistp08] T. van Deursen, S. Mauw, and S. Radomirović. Untraceability of RFID protocols. In *WISTP*, volume 5019 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[Esorics] T. van Deursen, S. Mauw, S. Radomirović, and P. Vullers. Secure ownership and ownership transfer in RFID systems. In *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 637–654. Springer, 2009.

[AIR] T. van Deursen and S. Radomirović. Security of an RFID protocol for supply chains. In *AIR*, pages 568–573. IEEE Computer Society, 2008.

[Wistp09] T. van Deursen and S. Radomirović. Algebraic attacks on RFID protocols. In *WISTP*, volume 5746 of *Lecture Notes in Computer Science*, pages 38–51. Springer, 2009.

[ePrint] T. van Deursen and S. Radomirović. Attacks on RFID protocols (version 1.1). Cryptology ePrint Archive, Report 2008/310, 2009.

[IPL] T. van Deursen and S. Radomirović. On a new formal proof model for RFID location privacy. *Information Processing Letters*, 110(2):57–61, 2009.

[STM] T. van Deursen and S. Radomirović. Security of RFID protocols – A case study. In *STM*, volume 244 of *ENTCS*, pages 41–52. Elsevier, 2009.

[RFIDSec] T. van Deursen and S. Radomirović. EC-RAC: Enriching a capacious RFID attack collection. In *RFIDSec*, volume 6370 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2010.

[Usenix] T. van Deursen, S. Mauw, and S. Radomirović. mCarve: Carving attributed dump sets. In *USENIX Security Symposium*, pages 107–121. USENIX Association, 2011.

[EuroPKI] T. van Deursen and S. Radomirović. Insider attacks and privacy of RFID protocols. In *EuroPKI*, Lecture Notes in Computer Science. Springer, 2011. To appear.

Index of subjects

Curriculum Vitae

2011 – present	Security consultant at Madison Gurkha, Eindhoven, The Netherlands.
2007 – 2011	Ph.D. student in the Security and Trust of Software Systems group, University of Luxembourg, Luxembourg.
2009	Traineeship, ING Direct Head Office, Hoofddorp, The Netherlands.
2005 – 2007	Master of Science in Computer Science and Engineering, Eindhoven University of Technology, The Netherlands.
2002 – 2005	Bachelor of Science in Computer Science and Engineering, Eindhoven University of Technology, The Netherlands.
1996 – 2002	Secondary education, Scholengemeenschap Were Di, Valkenswaard, The Netherlands.

Born on June 8, 1984, Eindhoven, The Netherlands.