THÈSE

pour obtenir le grade de

Docteur de l'Institut Polytechnique de Grenoble

Spécialité : **Informatique**

préparée à
l'**INRIA Rhône-Alpes**

dans le cadre de l'École Doctorale
École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique

présentée et soutenue publiquement par

MATE SOOS

le 6 octobre 2009

Titre
Privacy-preserving Security Protocols for RFIDs

Composition du Jury

ANDRZEJ DUDA *Grenoble INP — Ensimag*	Président
CLAUDE CASTELLUCCIA *INRIA Rhône-Alpes*	Directeur de Thèse
LEVENTE BUTTYÁN *Budapest's University of Technology and Economics*	Rapporteur
REFIK MOLVA *EURECOM*	Rapporteur
OLIVIER BILLET *Orange Labs*	Examinateur
FRANCOIS VACHERAND *CEA-LETI*	Examinateur

Abstract

RFID tags are small electronic devices using radio-frequency to receive power and transmit information. They are predicted to be present on almost every item sold to facilitate scanning and inventorying. Since RFIDs are so small, they can only use extremely lightweight security protocols, offering a unique challenge to researchers. For this reason, security protocols for RFIDs have been in the forefront of academic research in the past years. This thesis outlines what RFIDs are and describes and analyses multiple RFID security solutions. We present the state of the art in the field of RFIDs, including RFID systems and hardware and describe some of the most important RFID security protocols. Next, we show through the analysis of three recently proposed ad-hoc protocols the difficulty of designing secure RFID authentication protocols. We then analyse the Di Pietro-Molva private RFID authentication protocol, detailing its shortcomings and insecurities. Then, we describe the privacy-preserving RFID identification protocol ProbIP and analyse its security. An improved version of ProbIP, EProbIP, is also presented, which counters the insecurities found in the original protocol. Finally, we describe how low hardware-complexity stream ciphers could be used in RFIDs and analyse such ciphers using a SAT solver which we improved for this purpose.

Résumé

Les puces RFID sont de petits appareils électroniques qui seront insérées avec une grande probabilité dans chaque produit commercial. Les RFID étant de petite taille ne peuvent utiliser que des protocoles de sécurité qui nécessitent de faibles capacitées matérielles, c'est la raison pour laquelle elles présentent un grand défi pour les chercheurs. Il en résulte que les protocoles RFID se trouvent au premier-plan de la recherche académique ces dernières années. Cette thèse parcourt les RFID, présente et analyse de multiples solutions RFID. Dans la première partie, nous présentons l'architecture RFID et les protocoles de sécurité RFID les plus importants. Par la suite, nous étudions le protocole d'authentification et d'identification Di Pietro-Molva en nous focalisant sur la présentation, en détail, de ses défauts de sécurité. Ensuite, nous présentons le protocole ProbIP que nous analysons également du point de vue de la sécurité. Nous proposons une version améliorée du protocole ProbIP, le protocole EProbIP qui corrige les problèmes de sécurité du ProbIP. Nous présentons finalement la façon dont il est possible d'utiliser les chiffrements par flot de faible demande matérielle dans les RFID, et nous analysons ces algorithmes de chiffrement à l'aide de l'algorithme SAT élaboré spécifiquement dans ce but.

Contents

Introduction

Radio Frequency Identification (RFID) tags are small electronic devices that are envisioned to be used in many applications to identify and authenticate themselves or their users in a variety of scenarios. In this thesis we are focusing on low-end RFID tags such as EPC tags, that are very restricted in both hardware complexity and energy consumption. These devices can be used in most cases where bar-codes are not adequate, such as identifying items without opening their packaging, or identifying potentially very large number of items (such as a whole truck of items) in very short time. Naturally, if well-implemented, these technical advancements could bring potentially large advantages to their users.

Currently, RFIDs are used mostly to trace goods throughout the supply chain to counter counterfeiting, inventory shrinkage, and unreliable information. In the future, RFIDs could be used for receipt-less guarantee repairs, automated selective waste collection, or even by intelligent washing machines to warn if colours have been mixed in the drum. To realise these possibilities, RFIDs must be cheap, functional, fast, and be accepted by the general public. All of these requirements must be satisfied, a very delicate act, as some (such as functionality and cheapness) conflict which each other, requiring a fine balance.

Problem Statement

The problem treated in this thesis is the privacy and authentication need of RFIDs. There are many ways of achieving these goals, but currently, authentication is not generally implemented in RFIDs, and privacy is assured by the destruction of the tag's communication ability before it can reach the general public. This effectively means that tags are used only internally by organisations, mostly for tracing goods in the supply chain, thereby eliminating many possible use-cases of the tags. In this thesis we wish to analyse the whys and hows of this problem: why is it so difficult to create a good RFID protocol, and how the proposed RFID protocols have failed until now.

Contributions

In the past ten years, a community of researchers have been exploring the possible RFID protocols both by creating new protocols and by developing theoretical frameworks in which protocols could be evaluated. The first part of this thesis explores these practical and theoretical aspects giving an overview of what has been done, how have previous protocols been broken, and what are are their shortcomings. As with almost all aspects of RFIDs, communication protocols affect all other aspects:

price, functionality, speed, and acceptance by the users. A communication protocol that does not provide anonymity, for example, is too risky for the general public, as no one would like to be scanned every time he or she enters into a store, a bank, or a workplace. On the other hand, a communication protocol that is anonymous but makes the RFID too expensive also makes the tagged items too expensive thereby invalidating the use-case for RFIDs.

In the second part of this thesis, we analyse the protocol by Di Pietro and Molva and we propose and analyse two new protocols: the ProbIP and the EProbIP protocols. All three of these protocols offer privacy-preserving identification, and the Di Pietro-Molva protocol also offers authentication. During our analysis of the Di Pietro-Molva scheme, we demonstrate that it fails in some of its objectives, notably in providing secure private identification. The ProbIP protocol is also shown to be insecure, hence an improved version of it, the EProbIP protocol, is proposed. This improved version relies on the more difficult randomly generated NP-hard problem to provide its secrecy, thereby strengthening it against attacks.

Finally, in the third part of this thesis, we elaborate on how to use and analyse low hardware-footprint stream ciphers that have been created specifically for such use-cases as RFIDs. For our analysis of these ciphers, we use and adapt the MiniSat SAT solver to the environment of stream ciphers and give proof that using the demonstrated methodology, it is possible to break two widely deployed types of RFID tags' security. The methodology used is general, capable of analysing or even breaking other ciphers as well: a potential application of its abilities could be applied to, for instance, the KeeLoq cipher, used in car immobilisers.

Organisation

The thesis is organised into three parts. The first part deals with the state of the art in the area of RFIDs: the RFID systems currently deployed (Chapter 1) and the RFID security protocols that have been developed in previous research (Chapter 2). The second part of the thesis deals with the difficulty of designing RFID protocols. In this part, we first demonstrate some insecurities of the Di Pietro-Molva protocol (Chapter 3), then present two of our protocols, ProbIP (Chapter 4) and EProbIP (Chapter 5). The third and final part of the thesis deals with standard stream ciphers in RFIDs, where we present how to use low hardware-complexity stream ciphers in RFIDs (Chapter 6) and then analyse these stream ciphers with advanced SAT solvers (Chapter 7). Finally, we conclude this thesis.

Part I
State of the art

In this part of the thesis, we concentrate on the state of the art of RFIDs. Since RFIDs are such a large field of research, there is ample literature on their hardware and software layers. There are many different types of RFIDs, with varying use-cases and successes. Some, such as the first RFIDs used in Wold War II to identify incoming aircraft, has long been forgotten. Others, such as the Mifare access control system, first deployed in 1996, is still in use in many large cities' public transportation systems. In this part of the thesis we wish to first describe these systems, and then give a survey of all the research proposals on how certain security aspects could be implemented on these RFIDs. Academic security research on RFIDs is about 20 years old, with plenty of different protocols and schemes. We wish to elaborate on the most important schemes, giving a history and overview of their evolution.

Organisation

This first part of the thesis is made up of two chapters. In Chapter 1 we give an overview of RFID hardware and systems. Then, in Chapter 2 we give an overview of RFID security protocols: we describe the security goals and then go through all significant protocols that were designed to achieve these goals.

Chapter 1

RFID systems

In this chapter we explore what the literature calls an RFID system. We will see that there are, in fact, multiple definitions two of which we will explore in this thesis. There are many ways in which to categorise RFID systems: the standards they use, the radio frequency they use, their complexity and their purpose. In this chapter we explore these different aspects.

Organisation

In Sect. 1.1 we explore the different kinds of complexity measures for RFIDs. Then, in Sect. 1.2 we describe the two most influential RFID standards: the ISO14443 and the EPC Class 1 Gen 2 standard. In Sect. 1.3 we elaborate on the Media Access Control (MAC) protocols used by these two standards. Finally, in Sect. 1.4 we conclude this chapter.

1.1 Complexity measures for RFIDs

There are essentially two different complexity measures for RFIDs: their implementation complexity and their battery use. Implementation complexity is usually expressed in Gate Equivalents (or simply GEs), while battery use affects the tag's lifetime and its capabilities. In this section we describe both of these measures and their associated use cases.

1.1.1 Gate Equivalents

The Gate Equivalent (GE) is a manufacturing technology independent complexity measure for digital circuits. The silicon area of a NAND gate on today's CMOS technology constitutes one GE and all other circuit parts can be calculated in terms of such GEs. The GE count is a very important measure of RFIDs: if it can be reduced without sacrificing functionality, more can fit on a given silicon plate and so the tag becomes cheaper to manufacture.

Though the GE count makes it easier to evaluate the price of an extra circuit on an RFID, it is sometimes counter-intuitive to use, and so care must be taken when using it. For example, flip-flops (i.e. digital memory bits) are relatively expensive in terms of GEs, while digital circuit parts (i.e. and-, or-, etc. gates) are relatively cheap. The direct consequence of this is that thinking in terms of computational complexity instead of implementation complexity is counter-productive when evaluating the monetary cost of RFID security protocols.

1.1.2 Battery use

RFIDs can be grouped into the three categories according to their battery requirements: active, semi-passive and passive tags, according to their battery use.

Active Active RFIDs use their on-board battery for both communication and processing. These RFIDs can have very large ranges as they employ transmitters, though their lifetime can be very limited, as they use up their battery at a very high rate. These tags are used, for instance, on shipping containers, where the battery can be very large, and can supply enough energy for the lifetime of such a container.

Semi-Passive Semi-passive RFIDs' communication takes place using the reader-provided signal (through passive back-scatter), but processing uses the tag's on-board battery. These tags can typically operate at a longer distance than those that are fully passive, however, they have a limited lifetime: as soon as the battery empties, the tag becomes non-functional. These tags are used, for instance, in automatic toll collection on highways.

Passive Passive RFIDs are powered by the reader at all times. These devices have no on-board power, and take all their energy (including energy to communicate and process data) from the signal received from the reader. These tag types are the cheapest to manufacture and the easiest to maintain, but they can only be used in the proximity of the reader, since they must use either passive back-scatter or inductive coupling for communication with the reader. In this thesis we mainly focus on these, passive RFIDs,

1.2 Standards

There are two main standards for contactless Integrated Chips (ICs). The ISO14443 for proximity contactless cards and the EPC Class 1 Generation 2. In this section we go through these two standards, highlighting where they meet and differ.

1.2.1 ISO/CEI 14443

The ISO/CEI 14443 standard [37] defines the tag as the proximity integrated chip card (PICC) and the reader as a proximity coupling device (PCD), both of which work using the carrier signal at 13.56MHz. The reader and the tag can communicate up to 10cm from each other, though the maximal nominal distance for a specific implementation can be shorter. The PICC usually takes the 7810 ID-1 card form factor (i.e. credit card-size) though sometimes it is smaller, such as the Visa micro tag key fob. Since the PICC is so close to the reader and since the PICC is relatively large, there is plenty of available processing power on the PICC: some cards can even carry out complex, public-key operations.

The ISO14443 standard is used for micropayment in transportation systems (e.g. the NXP's MIFARE system), building access control (e.g. the HiTag2 system), and bank cards (e.g. Visa Paypass, Chase Blink). For these uses, it is not required to identify large amounts of tags in a very short time, though speed is a factor in public transportation. The ISO14443-based bank card systems integrate the ISO7816-* (i.e. smart card) standards, giving rise to advanced functionalities, but staying compatible with the already established infrastructure of contact-based smart cards.

There are two ISO14443 communication standards: Type A and Type B. Type A is essentially used only by Philips (now NXP) cards, while Type B is used by all other manufacturers. Both types use the 13.56MHz carrier frequency, and use the subcarrier frequency of $f_c/16$ to $f_c/128$ (where f_c is the carrier frequency) depending on which bandwidth (106kbit/sec to 847kbit/s) the tag and the reader agreed on. The only difference between the two standards is in the coding and modulation type. The used codings and modulations are amplitude shift keying (ASK), Miller coding, non-return to zero (NRZ) and binary pulse shift keying (BPSK).

1.2.2 EPC Class 1 Gen 2

The Electronic Product Code (EPC) is a set of standards of which the most interesting is Class 1 [21], called "Identity Tags". This class has had two generations, the second of which is the currently used one. The EPC Class 1 Gen 2 defines a tag and a reader (referred to as *interrogator* in the EPC literature) working in the UHF (Ultra High Frequency) range. The actual range used depends on the continent and sometimes even the country, according to local frequency regulations. In the European Union the range used is between 865.5MHz and 867.6MHz, and in the US it is between 902MHz and 928MHz. The nominal maximal working range of the standard is several meters, which means that the transmitted power to the tag from the reader is very low, leading to very simple tags. The combination of long working range and simple tags is a major difference between the ISO14443 and the EPC standards.

The use-case for the EPC Class 1 Gen 2 standard is for tagging goods: this standard was specifically designed to be put on almost every product sold. EPC tags are therefore very cheap, in the range of $0.10 per tag and can be read extremely fast, theoretically up to 1000 tags/sec by large, industrial readers. Tagging all goods would allow for very fast inventorying, precise stock counts and real-time monitoring of the supply chain. For these reasons, at least in the near future, retailers are expected to experience the most profit from using EPC tags.

The EPC Class 1 Gen 2 is a very broad standard, specifying not only the tags and readers, but also the very large backend used to process the large amount of information provided by the readers. This backend is crucial for the success of the standard, as the goods come from many different manufacturers, typically go through multiple distribution points, and are of vast quantity, needing very large, distributed systems that can manage non-trivial authorisation boundaries.

The standard uses many different modulation techniques to encompass different implementations. The reader-to-tag radio interface for example allows for three different modulation types (all subtypes of amplitude shift keying) while the tag-to-reader radio interface allows for both amplitude and phase shift keying. Compatibility is maintained by requiring the reader to understand all tag modulation types and vice-versa. Data encoding in the reader-to-tag direction is pulse interval encoding (PIE), and in the reverse it can be FM0 or Miller-based. The data rate in the reader-to-tag direction is between 26.7 to 128kbit/s and in the other direction, it varies between 5-320kbit/s for Miller-based and 40-640kbit/s for FM0 based data encodings.

1.3 Media Access Control protocols

When multiple entities need to communicate at the same time they can cause interference prohibiting communication. Such an interference between two tags is called a *collision* in the RFID literature. To circumvent this problem, there must be a Media Access Control (MAC) protocol used to control who, when and where can talk on which channel(s). This problem is common to all wireless communication protocols such as Wi-Fi, Bluetooth, and RFIDs. In this section we first outline the two MAC methods used by RFID systems, and then we describe in detail how the EPC and the ISO14443A RFID protocols handle MAC.

1.3.1 MAC protocols used in RFIDs

There are two MAC protocols used in RFIDs, one used by the readers to avoid collisions between each other, and the other by the tags to avoid collisions between themselves. These three MAC protocols use different aspects of the communication to distinguish between entities that need to communicate at approximately the same time: the first uses the difference in physical location and angle of orientation, the second the difference in the communication frequency, and the third the difference in the timing of the communication.

Space Division Multiple Access

In Space Division Multiple Access (SDMA), the space is set up such that entities occupy different physical fields. Since the entities can communicate only in different areas around themselves, this solves the problem of multiple access. In RFIDs, it is relatively easy to make a reader that can only communicate in a very selective part of space around itself, thus easing the problem of interference between readers. However, RFID tags are normally made such that they occupy an evenly distributed space around themselves, thus this solution does not solve the problem of interference cased by two (or more) tags near each other.

Frequency Division Multiple Access

In Frequency Division Multiple Access (FDMA), entities can talk in multiple frequencies, and they choose the frequency to talk on in a manner such as to reduce interference with other communicating entities. FDMA is used in for instance by the EPC Class 1 Gen 2 standard to reduce interference and speed up identification.

Time Division Multiple Access

In Time Division Multiple Access (TDMA), the time is divided such that only one entity may talk at any given time. The negotiation of time slices is the most challenging part in this system. Both EPC Class 1 Gen 2 and ISO14443 standards use TDMA to ultimately resolve media access control conflicts as the combination of SDMA and FDMA are usually not enough. The TDMA-based solution is both cheap to implement and suits the typical system configuration of a higher-powered and more intelligent entity (the reader), and multiple, less sophisticated entities (the tags). The only drawback is the reduced speed of identification: if many tags are in the field of the reader, they must divide the time among themselves, slowing down the overall process.

1.3.2 The ISO14443A standard MAC protocol

The ISO14443A standard uses TDMA as the MAC protocol, a protocol which it calls *singulation*, and is mandatory to use to avoid collisions. The singulation protocol is deterministic, the reader explores the Unique Identifier (UID) of the tag in a bit-by-bit tree-walking fashion. This protocol is simple to implement on the tag and is relatively time efficient if the UID is short. However, since the reader calls the different tags using their unique UIDs, the tags cannot remain anonymous during singulation. Furthermore, since the tag UID is repeated during the protocol by the reader, and the reader has much greater range than the response emitted by the tag, the UID can be eavesdropped from a safe distance by a malicious reader.

An example protocol run of the tree-walking algorithm is in Fig. 1.1. Bit-collisions occur when at least one tag sends a binary “0” and the other(s) a binary “1”. These collisions are detected using channel coding. When a collision occurs, the reader must choose which branch to follow. The reader thus explores a chosen part of the binary collision-tree during the protocol.

1.3.3 The EPC Class 1 Gen 2 MAC protocol

The TDMA collision-avoidance protocol used by EPC Class 1 Gen. 2 standard [21] is also called *singulation*, and is mandatory to use to avoid collisions. The course of this protocol is important since it may leak some information that may threaten security or privacy. EPC uses a probabilistic algorithm which does not ensure a fixed time for the singulation of all the tags in the field. It implements the principle of the management of time slots known as the ALOHA protocol.

An example run of the EPC singulation protocol is present in Fig. 1.2. Inventorying of the tags around the reader begins with a *Select* command from the reader. This command determines the tag population that will take part in the process. This selection is done via the “SL” or “inventoried” flag to separate tags in two populations A and B. The selection can also be done on a specified part of the tags memory (EPC, TID or User memory) with a mask which can be fully defined in the command.

When the reader has selected the subset of tags, it can launch the singulation with the *Query* command. The command contains a parameter Q which defines the $2^Q - 1$ time slots. When the tags have received the *Query*, they pick a random

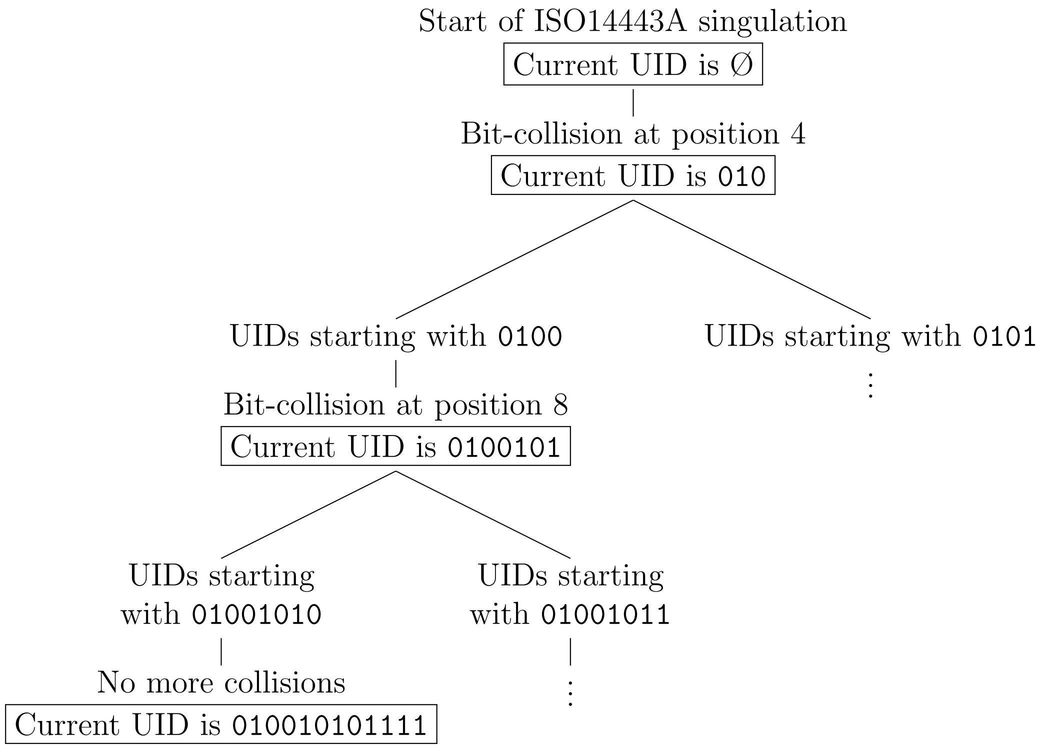

Start of ISO14443A singulation
Current UID is Ø
Bit-collision at position 4
Current UID is 010
UIDs starting with 0100
UIDs starting with 0101
Bit-collision at position 8
Current UID is 0100101
UIDs starting with 01001010
UIDs starting with 01001011
No more collisions
Current UID is 010010101111

Figure 1.1: Example ISO14443A singulation protocol, where the UID length is 12. During singulation, there are two collisions, one at the 4th bit of the UID, and one the 8th. The found tag's UID is 010010101111. There could be many tags in the vicinity of the reader, as there might have been many more collisions in the other (unvisited) parts of the tree.

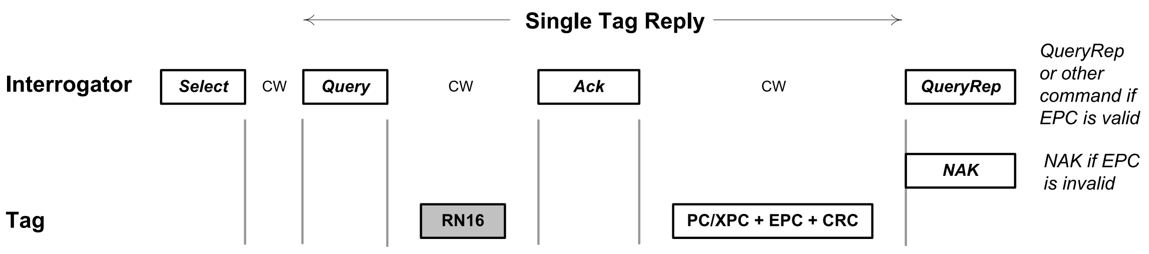


Figure 1.2: An example run of the EPC Class 1 Gen. 2 standard's singulation protocol. In this run, a tag randomly selects 0 as its slot-counter, and so it immediately replies to the *Select* command of the reader with a random number RN16 generated by the on-board Pseudo Random Number Generator (PRNG). There are no collisions, so the reader replies with an *ACK* command, the tag sends its EPC code, and some other data (PC/XPC), plus the error-checking Cyclic Redundancy Check (CRC) code. If the EPC code is valid, the reader can further issue the *QueryRep* command. If the EPC is invalid, the reader issues a *NAK* (Not Acknowledge) command instead.

value in the range $[0, 2^Q - 1]$ which determines the time-slot where they will reply. Tags that pick zero reply immediately with a 16-bit random number RN16, then if there is no collision, the reader acknowledges the tag reply with an ACK command containing the received RN16. After the ACK command, the tag answers with its EPC code and inverts its “inventoried” or “SL” (selected) flag.

To change the time-slot after an EPC reply from a single tag, or after a collision from several tags, or even after an empty slot, the reader sends a *QueryRep* that has the consequence of decreasing all tags’ time-slot counters. The reader can also send a NAK command after the answer of the tag if it did not understand the EPC.

The *Select* command

The *Select* command is mandatory to define the subset of tags in the field which will take part in the inventory. This subset can be determined by applying a mask in any bank of the memory of the tag (e.g. EPC, user memory, etc.) with a fixed position (“pointer” parameter) and/or length (“length” parameter). Only the tags which have the value of the mask at the defined part of the memory will be activated for the inventory. The “truncate” parameter enables to limit the length of the EPC code which later will be replied by the tag.

The *Query* command

The *Query* command initiates and specifies an inventory round. This command lets the reader define miscellaneous parameters of the physical layer such as the data rate of the response. The main parameter is the number $Q \in [0, 15]$, which determines the $2^Q - 1$ time slots. As soon as the *Query* command is received by the tag, it initialises its slot-counter with a random number from its on-board Pseudo-Random Number Generator (PRNG), called RN16 in the EPC standard. At each new *QueryRep* command received from the reader, the tag decrements its slot-counter. If the slot-counter has reached zero, it immediately replies with another random number, again drawn from RN16, and a parity-check code CRC-5.

The properties to which the random number generator RN16 must adhered to is defined in the EPC standard:

- *Probability of a single RN16*: The probability that any RN16 drawn from the RNG has value RN16= j, for any j, is bounded by $0.8/216 < P(\text{RN16} = j) < 1.25/216$.

- *Probability of simultaneously identical sequences*: For a tag population of up to ten thousand tags, the probability that any two or more tags simultaneously generate the same sequence of RN16s shall be less than 0.1%, regardless of when the tags are energised (i.e. put in the field of the reader).

- *Probability of predicting an RN16*: An RN16 drawn from a tag 10 ms after power-down shall not be predictable with a probability greater than 0.025% if the outcomes of prior draws from the tag RNG, performed under identical conditions, are known.

The *ACK* command

A reader sends an *ACK* to acknowledge a single tag. This command echoes the RN16 of the tag reply. If the tag receives an incorrect RN16 then it remains in its inventory mode.

After an *ACK*, the tag answers with the concatenation of its PC/XPC, EPC, and finally a CRC-16 to check the transmission. The PC/XPC parameter defines the length of the EPC, the user memory (enabled or not), and if the application referred to as the "EPC application". The EPC code is the unique identifier of the tag which is used to track items and as a consequence it is the basis of the privacy issue in the EPC standard. The CRC-16 is used to reduce the possibility of faulty transmissions. After the transmission, the tags invert their "SL" (SeLected) or "inventoried" flags.

The *QueryRep* and *QueryAdjust* command

The *QueryRep* command, as shortly described previously, instructs the tags to decrement their slot counters and if the slot reached zero, to modulate an RN16 and the CRC-5 to the reader. *QueryRep* is the command used to move to the next time slot after an EPC reply, a collision or an empty time slot.

The *QueryAdjust* command adjusts the parameter Q without changing any other round parameters. Tags then have to again pick a random value between 0 and $2^Q - 1$ and load it in their slot counter. Without this command, if the parameter Q was chosen by the reader to be too low (underestimating the tag population), the reader would not be able to change Q, thus rendering the identification of tags infeasible. For instance, if Q was chosen as 1, there would be exactly 1 slot, and if there were multiple tags in the vicinity of the reader, they would always collide, thus rendering the reader unable to identify either of them.

The *NAK* command

The *NAK* command shows to the tags a non acknowledgement of the reader. Consequently, tags have to ignore the previous command of the reader.

The *Access* commands

To access to the memory of a specific tag that has just been singulated or to send it a specific command such as the *Kill* command, the reader asks the tag via its RN16 to pick and reply with a new 16-bit random number (command *ReqRN*) called "handle". This new handle becomes the new pointer to the tag and then the argument of the *Access* command to access the tag's memory.

1.4 Conclusions

RFID hardware is complex and rather unnatural for those who have no experience with hardware design. For instance, it was a common belief in the early stages of RFID security research that hash functions were ideal for RFIDs, as they are considered fast and efficient (i.e. having a low computational complexity) on commodity computer hardware. This, however, was shown to be a false hypotheses by Feldhofer and

Rechberger [22]. Similarly, it is rare to see RFID security proposals that take into account the anticollision mechanisms of the standards, thereby ignoring the privacy problem that these cause. A proper investigation of the RFID hardware is therefore essential for any researcher working in the field of RFID security.

Chapter 2

RFID security protocols

RFID security protocols are used to achieve two main goals: private identification and authentication. Private identification is needed to preserve the privacy of the tag owner, and authentication is useful for access control, or authenticity test (e.g. for guarantee repairs). There are many security aspects regarding these two goals: traceability, secure and repeatable authentication, authentication without shared common secrets, the list is long. In this chapter we define the terms and problems, then focus on the most prominent protocols and their security aspects.

There are many ways to categorise RFID protocols. A possible way to do so would be to categorise them according to their hardware complexity on the tag side, or on the reader side. It would also be possible to categorise them according to the kind of cryptographic algorithm they employ, or the kind of services they provide. However, whichever categorisation we take, some protocols will not fit into any, and some will fit into into multiple categories. In this chapter we categorised protocols according to the main the services provided, and then we sub-categorised them according to the employed algorithms. Some protocols provided multiple services — these protocols were listed according to their main service.

Organisation

Firstly, in Sect. 2.1, we describe the definitions used throughout this chapter. Next, in Sect. 2.2 we elaborate on why privacy is so important for RFIDs and how it could be achieved. We then describe some of the most influential RFID authentication protocols in Sect. 2.3. Finally, in Sect. 2.4 we compare the aforementioned protocols using a feature-comparison table and in Sect. 2.5 we conclude this chapter.

2.1 RFID protocol features

In this section we give a short description of the protocol features relevant for RFIDs. Although these features are not unique to RFIDs, their implementation is particular to the setting, given the range of features and restrictions of RFIDs.

2.1.1 Identification

Identification is the act of associating a communicating entity with an identifier in the system. For instance, when a reader communicates with an EPC tag, the tag tells its EPC code, an identifier that can be looked up in the EPC code database. Identification does not make sure that the identifier sent is not maliciously sent — indeed, the EPC code sent by EPC tags can be easily spoofed by anyone with the proper equipment.

Identification can be Untraceable, Unlinkable, Forward-secure, and Anonymous. We now shortly describe these features according to Pfitzmann and Köhntopp [57], put in the context of RFID tags.

Anonymity Anonymity, in the context of RFID tags is the state of being not identifiable within a set of tags, the *anonymity set*. The anonymity set is the set of all possible tags. Therefore, a tag may be anonymous only within a set of potential tags, its anonymity set, which itself may be a subset of all tags worldwide. Anonymity is the stronger, the larger the anonymity set is and the more evenly distributed the tags are within the set.

Untraceability Suppose that an adversary is able to accumulate logs of tag-reader interactions, i.e., a (partially) successful protocol executions. Further, suppose that an adversary has accumulated a set L_1 of execution logs of a protocol between various tags and readers, and the adversary has access to the set L_2 of execution logs of the same identification protocol between one specific tag $\mathcal{T}_j$ and various readers. The identification protocol is said to provide untraceability, if the adversary cannot decide whether some of the logs in L_1 relate to $\mathcal{T}_j$ with a higher probability than a random guess.

Unlinkability Unlinkability of two or more items (e.g. tags, tag messages, tag actions, ...) means that within the RFID system, these items are no more and no less related than they are related concerning the a-priori knowledge. This means that the probability of those items being related stays the same before (a-priori knowledge) and after the run within the system (a-posteriori knowledge of the attacker). E.g., two messages are unlinkable if the probability that they are sent by the same tag is the same as those imposed by the a-priori knowledge.

Forward privacy Suppose that an adversary has access to the inner state of a tag $\mathcal{T}_j$, e.g. by physically compromising $\mathcal{T}_j$, reading out all its memory. Also, let us suppose that the adversary has collected a set L_1 of past execution logs of the protocol between various tags and various readers. Then, an authentication protocol is said to provide forward-secrecy, if an adversary is unable to tell which execution of the authentication protocol relates to $\mathcal{T}$ other than with negligible advantage over a random guess.

Among these notions of privacy, the following implications can be identified [68]:

$$\text{Forward privacy} \Rightarrow \text{Unlinkability} \Rightarrow \text{Untraceability} \Rightarrow \text{Anonymity}$$

2.1.2 Authentication

To make sure that the entity sending the identifier A is indeed entity A, authentication is required. Authentication is the act of making sure that the other partner is indeed the partner it claims to be. For authentication to take place, the entity wishing to authenticate to the other party must contain a secret whose knowledge can be verified by the other party. We say that the authentication is unidirectional when either the reader authenticates the tag, or the tag authenticates the reader. Authentication is *mutual* when both the tag authenticates the reader and the reader authenticates the tag.

Since most RFIDs have no way of notifying the owner whether the authentication with the reader was successful or not, unidirectional authentication between the tag and the reader is often meant in the sense that the reader authenticates the tag. Nevertheless, mutual authentication has its benefits, for example, if the tag carries some secret data such as single-use discount coupons.

Authentication requires either randomisation or tight time synchronisation to prevent so-called "replay attacks". In a replay attack an adversary eavesdrops on the communication between reader and tag, stores the communication, and later replays the communication to impersonate a tag or a reader. Time synchronisation in RFIDs is rarely used as it would require expensive hardware on the tag, such as a battery. Therefore, randomising the protocol (through random challenges) is the generally accepted way to ensure *freshness* in the RFID setting. The challenge is usually sent from the reader to the tag (for tag authentication) before further communication, and it not only ensures the freshness but can also be used to make subsequent data exchanged dependent on the challenge. For mutual authentication, a challenge must be sent by both parties.

2.2 Privacy-preserving RFID identification

People dislike to be identified, especially on a large scale. However, when need be (e.g. when using a pay-per-use service), a fast and easy way to identify and authenticate oneself is very important. RFIDs could help fill this role as they are small, resilient to tear and wear, and operate at a distance, without line of sight. These advantages, however, can also be used against RFIDs: since they are small, it is very difficult to use well-established cryptographic algorithms, and since they operate without line of sight, it is very hard to hide or deactivate them. People are used to the notion that if something cannot be seen, heard, or physically felt, it is private. RFIDs, on the other hand, can be as small as a grain of sand and use radio frequencies that are imperceptible to humans.

Since RFID readers are aimed to be used in every store, and RFID tags are aimed to be put on every produced item, tags and readers are easily obtainable, small, and cheap. This means that anyone can obtain or even make one. Some have even experimented making very sensitive readers, that could capture tags' IDs as far as 4 meters away when the nominal reading range for the tag was only 10cm [42]. Readers can also be made to be totally passive if another (e.g. legitimate) reader is nearby. This means that misuse can be extremely hard to detect.

The imperceptibility and technical simplicity of RFIDs makes them uniquely dangerous. The potential for RFID misuse is very real: for example, in 2003 the garment maker Benetton equipped all its clothes with RFIDs and then used the customer's currently worn clothes as identification to offer other clothes [15]. This misuse plainly ignored the privacy of customers and lead to large negative publicity for the firm.

There are multiple definitions of RFID privacy. A well-known one is by Ohkubo et al. [43]. They define *Indistinguishably* and *Forward security* and then show that their scheme is indistinguishable and forward secure according to their definitions. Juels et al. define privacy in another way [41]. In this model, an RFID system is secure if an adversary has a non-negligible chance of distinguishing between two tags in a clearly defined *Privacy Experiment*. There are multiple refinements of the Juels model, e.g. one by Ouafi et al. [53], who for instance put different constraints on the adversary. A more complete but also more complex definition is by Vaudenay [68] who considers different types of attackers, namely STRONG, DESTRUCTIVE, FORWARD, WEAK and NARROW and then demonstrates protocols that resist attacks by certain types of attackers.

The problem of privacy in RFIDs could be countered either on a physical or on a communication level. In this section we first examine the physical-layer solutions, then describe the four main types of security protocols that aim to protect the tag's privacy: information-theoretic protocols, hash-based protocols, key-tree protocols, and experimental protocols.

2.2.1 Physical layer solutions

There are very few physical layer solutions for preserving RFID privacy. These solutions all operate on the hardware, mostly radio-frequency level, that is, they either block, selectively allow, or deactivate the communication between the tag and the reader.

Faraday cage A simple solution for protecting the privacy of RFID owners is to put the item containing the RFID into a Faraday cage, i.e. a wire mesh that blocks all radio communication. This solution is very effective, however, it can be very inconvenient, as the tag would need to be taken out of the container every time it needs to be used. Furthermore, some tags cannot be put inside containers, for example, RFIDs hidden inside jackets or trousers. This method of protection is therefore infeasible for many items to be equipped with RFIDs, but it could be effective for certain objects, such as RFID-equipped passports.

Blocker tag The blocker tag by Juels et al. [39] is an advanced RFID tag that interferes with the anti-collision mechanism (see Sect. 1.3) of the RFID standards, thereby selectively allowing some communication between tags and readers. For the selection of tags and readers that can communicate, privacy zones can be defined, and searching for tags outside these pre-entered limits is prohibited. The advantages of the blocker tag solution is cheapness, effectiveness, and convenience for the end users. The disadvantages are to do with privacy management: if too few tags are allowed to communicate with too few readers, then the potential that RFIDs could bring cannot be materialised. On the other hand, if the privacy controls are too lax, the privacy of the owner could be endangered. Therefore, if used as an everyday item, managing the blocker tag's privacy rules would be difficult and most people would find it too complex to use.

Noisy tag The noisy tag by Castelluccia and Avoine [16] is a tag that is synchronised with the reader and continuously emits radio signals that add noise to the channel. Since the reader is synchronised with the noisy tag, it can substract the noise from the signal, recovering what was on the channel. This solution effectively bars anyone from listening on the channel, thus it is very useful to hide secrets such as the access PIN exchanged between the tag and the reader. However, for protecting the privacy of the owner, the noisy tag must be worn all the time, and it must manage the privacy settings of the user. Therefore, noisy tags could fill a similar role as blocker tags, but would also inherit similar problems: they would need to be worn all the time, and managing their privacy rules would be non-trivial.

Noisy reader The noisy reader by Savry et al. [62] is a reader that emits noise on the radio channel, much like the noisy tag does, and substracts the noise internally to retrieve the information that was originally on the channel. The noisy reader is better than the noisy tag in that it does not need synchronisation with the reader, however, it cannot be transported as easily as the noisy tag. A further advantage of the noisy reader is that it has a prototype implementation, whereas for the noisy tag, implementation is not as straightforward [33].

Kill PIN The Kill PIN approach is currently used by the EPC Class 1 Gen 2 standard: if the tag is given a tag-specific 32-bit PIN code, the communication capabilities of the tag are damaged (through the overloading of a part of the circuit) and the tag cannot be used any more. This solution is easy to implement and is secure as long as every tag is deactivated before the customer starts to use it (i.e. at the checkout counter or when it is put on the shelves). The problem with this solution is that every benefit that the tags could bring other than supply-chain management (e.g. automatic waste recycling, receipt-less warranty repairs, etc.) becomes impossible.

2.2.2 Information-theoretic protocols

Information-theoretic protocols simply withhold information from the attacker such that only the legitimate reader can identify the tag. The most simple way of implementing such a protocol is by Juels [38], in which tags keep a set of randomly chosen pseudonyms $\alpha_1, \alpha_2, \ldots, \alpha_n$, and use them one after the other when identifying themselves to a reader. Since the reader is aware that pseudonyms are used to hide the real identity of the tag, it causes no problems for the reader to identify the tag. An eavesdropper cannot distinguish that the observed α's belong to the same or to different tags. However, since the tag's memory is limited, it can only store a limited n number of pseudonyms. After these have been exhausted, the tag must re-use an already used pseudonym, thereby becoming traceable.

The advantage of information-theoretic protocols is that they are guaranteed to be secure until a certain point. The problem with them, however, is that they all

Reader $\mathcal{R}$		**Tag $\mathcal{T}_{ID}$**
Database L: $\{\ldots, (ID, s_i), \ldots\}$		Secret state: s_i
	Identify $\longrightarrow$	
		$a_i = G(s_i)$
		$s_{i+1} = H(s_i)$
	$\longleftarrow a_i$	
find $(ID, s_i) \in L$ s.t. $G(s_i) = a_i$		

Figure 2.1: The OSK protocol. The tag generates its pseudonym a_i, sends it to the reader, and updates its current state s_i. The reader, using its internal database containing all the tags, finds the entry that matches the received a_i.

loose their security after a limited number of identification attempts. To alleviate this problem, the pseudonym-rotation protocol updates its internal pseudonyms after every authentication with a legitimate reader. This unlinks the current state of the tag from its old state, but the tag can again be traced by simply demanding identification n number of times.

2.2.3 Hash-based protocols

The most well-known privacy-preserving RFID protocol using hashing is the OSK protocol [43]. The protocol can be characterised as present in Fig. 2.1.

The advantages of the OSK protocol are twofold. First, it provides privacy for an unlimited amount of identification attempts, unlike the information-theoretic protocols. Second, it is secure if the functions H and G are preimage-resistant. There are many disadvantages associated with the protocol, however. First, and foremost, the tag can simply be queried by a malicious reader for a long period, after which the state has evolved, let's say, m times. To find which tag that sent a_{i+m}, the backend (which has n tags) needs to iterate function H and execute G for each tag in the system exactly m times. Therefore, it takes $2mn$ operations to find the tag, which can quickly take too much time with large enough populations and large enough m.

There have been many iterations of the OSK scheme to alleviate these problems. One such iteration is by Avoine et al. [1], which proposes a time-memory trade-off to shorten the time it takes to find the tag. The trade-off works using Hellman's idea [35], later improved by Oechslin [50] that reduces the amount of work T needed to invert any value in a set of N outputs of a one-way function given enough memory. The trade-off can reduce the amount of work from N to $N^{2/3}$ using $N^{2/3}$ units of memory. The authors find appropriate trade-offs and present multiple examples. One such example uses $1GB$ of memory, and given $m = 2^{10}$ and $n = 2^{20}$, the time to find the tag is approx 40'000× faster than without memory usage. Though this improvement partially solves one of the principle problems of OSK, a tag can still be made unreachable by the system by simply querying it $m >> 2^{10}$ times by a malicious entity — a so-called denial of service (DoS) attack.

The YA-TRAP protocol, proposed by Tsudik [66] is another generation of hash-based private identification schemes, which, as an added feature, also provides

Reader $\mathcal{R}$		**Tag** $\mathcal{T}_i$
Database L:		Shared secrets:
$\{\ldots,\ (t_j,\ \mathrm{HMAC}_{K_i}(t_j)),\ldots\}$		K_i, t_0, t_i, t_{max}
	$\longrightarrow t_j$	
		if $(t_j < t_i)$ or $(t_j > t_{max})$
		$h_j = \mathrm{PRNG}_i^j$
		else
		$h_j = \mathrm{HMAC}_{K_i}(t_j)$ and
	$\longleftarrow h_j$	update $t_i \leftarrow t_j$
check $\exists t_j$ s.t. $(t_j, h_j) \in L$		

Figure 2.2: The YA-TRAP protocol. On the top, the shared secrets are indicated. The protocol starts with the reader interrogating the tag with current time t_j, to which the tag responds either PRNG_i^j or $\mathrm{HMAC}_{K_i}(t_j)$. The reader then checks using its internal database L, whether the returned value is correct or not.

authentication. The YA-TRAP protocol is described in Fig. 2.2. The main idea behind YA-TRAP is to store a timestamp on the tag that can only increase to a given t_{max} time, and which is updated after every query by the reader. Essentially, the reader sends a timestamp t_j which must be larger than the current timestamp in the tag t_i, and the tag replies with the HMAC_{K_i} of the timestamp t_j, and the tag updates its timestamp to t_j. The reader simply checks if the pair (timestamp, HMAC) corresponds to a K_i in its database. This scheme is interesting, as it allows for batch operation: if many tags need to be identified and authenticated, such as envisioned by the EPC, e.g. when filling up the stock of a supermarket, the batch of tags can be identified in $\mathrm{O}(n)$ operations. This batch-processing is done as follows: the tags' replies, the h_j-s are collected in a hash table data structure, and when the reader goes through each element in the database, it searches the hash table for matches. Since it takes $\mathrm{O}(1)$ to search in a hash table, it takes $\mathrm{O}(n)$ to go through all elements of the database, finding the match for every collected h_j on the way.

The YA-TRAP scheme has multiple drawbacks. First of all, a denial-of-service (DoS) attack is trivial to carry out, as it suffices to send a very high t_j to the tag, e.g. $t_{max} - 1$. Also, as shown by Ouafi et al. [53], the tag can be made traceable by first making the tag think it is in the future then observing a validation check of the tag by the reader, thus winning the Privacy Experiment of Juels and Weis [41]. Subsequently, YA-TRAP was revisited and updated by Burmester et al. into YA-TRAP+ and O-TRAP [12], both of which have later been shown to be traceable by Ouafi et al. [53].

RIPP-FS by Conti et al. [18] is another hash-based protocol, which distinguishes itself by employing hash chains, originally proposed by Lamport in [44]. Each tag $\mathcal{T}_i$ is initialised with a tag key K^0_{Tag}, shared with the reader. The tag also stores the initial pair $(K_0\ ,\ t_0)$ generated by the reader, where K_0 is the last value in the tag-specific hash chain:

$$\begin{aligned}
K_l &= w \\
K_i &= H(K_{i+1}) = H^{l-1}(w), i = 0, \ldots, l-1 \\
K_0 &= H^l(w)
\end{aligned}$$

where w is the seed, and t_j $(j = 0, \ldots, l)$ is a time interval counter. One of the goals of RIPP-FS's design was to achieve untraceability, and offer more security guarantees than YA-TRAP and its variants. However, the untraceability properties of RIPP-FS were broken by Ouafi et al. [53] in the Juels and Weis model, the Privacy Experiment.

Although hash-based protocols have very useful properties, they are still unpractical for two reasons. Firstly, hash functions have been shown to be much more resource-intensive to implement on RFID tags than previously thought [22]. Secondly, with hash-based protocols it is often the case that either the number of queries allowed to the tag is limited by the protocol, or if the tag is queried too many times, it can be lost from the system: in YA-TRAP and RIPP-FS if the time-stamp given is too large the tag is rendered inoperative, while with OSK if the tag is queried too many times, the resulting evolution of the tag's internal state renders the tag unidentifiable and thus dysfunctional.

2.2.4 Key-tree based protocols

Key-tree based protocols are a family of protocols based on the original protocol by Molnar and Wagner [47]. In this protocol, each tag is a leaf of a balanced tree of depth d, with a branching factor b, thus the total number of tags in the system is b^d. At every level, each branch of the tree has an associated key. The tags know all keys on their paths from the root to the leaf. An example tree is shown in Fig. 2.3. When a tag wants to identify itself to the reader, it executes the protocol as described in Fig. 2.4 at each level of the tree from the root to its corresponding leaf.

As there are only b branches of the tree at each level, the function f only needs to be calculated on average $b/2$ times at each level, for a total of $b \cdot d/2$ executions of f on average. Since there are b^d tags in the system, this is logarithmic in the number of tags. The authors describe a trade-off between the branching factor and the depth, which is further elaborated upon by Buttyan et al. [13], where the authors calculate the depth and branching factor needed for different tag populations and timing constraints.

The key-tree approach has several features that make it exceptional in many respects. First of all, it is adaptable to any protocol: the PRF function simply needs to be replaced by the protocol in question. Secondly, if the underlying protocol has the appropriate security features (a feature that PRFs provide), it also provides authentication of the tag. Finally, if the last step is carried out, the protocol provides mutual authentication.

The drawback of using key-trees is that tags must share secrets. Since tags are usually not in a controlled environment, and they are rarely tamper-resistant, the risk of key compromise by malicious parties is not negligible. Once a tag is tampered with and its stored keys are recovered, the adversary can use the keys to undermine

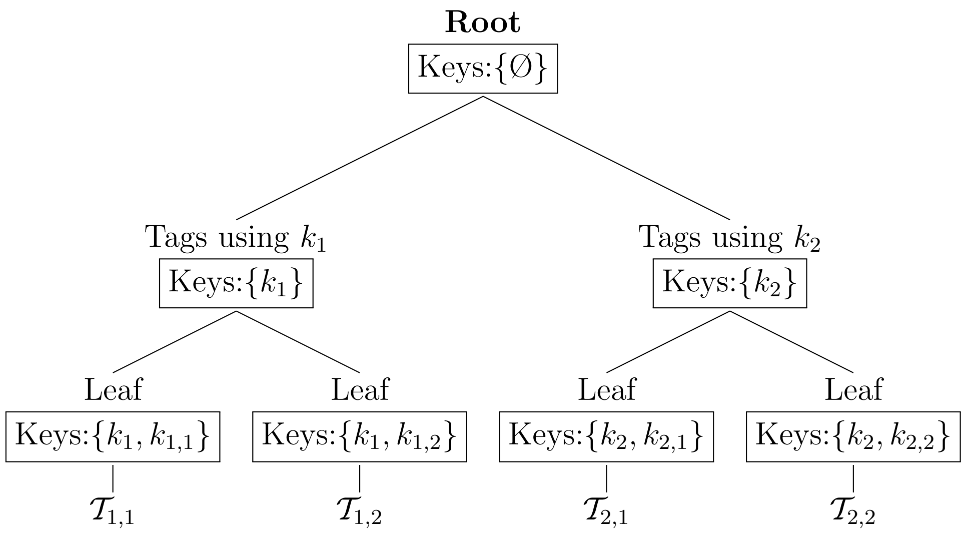


Figure 2.3: An example Molnar-Wagner key-tree. The branching factor b is 2 and the depth d is also 2, for a total of four tags in the system. The weakness of the system is the following: by tampering with and compromising the keys stored in tag $\mathcal{T}_{2,2}$, the privacy of tag $\mathcal{T}_{2,1}$ is compromised as there are no other tags on the branch with key k_2. Also, the remaining two tags' anonymity set is halved, as the original anonymity set was $\{\mathcal{T}_{1,1}, \dots, \mathcal{T}_{2,2}\}$, but only $\{\mathcal{T}_{1,1}, \mathcal{T}_{1,2}\}$ remains — a drop from a set of 4 to a set of 2.

Reader $\mathcal{R}$		**Tag $\mathcal{T}_i$**
Generate nonce r_1		
	$\longrightarrow r_1$	
		Generate nonce r_2 and calculate $\sigma = ID \oplus f_k(0, r_1, r_2)$
	$\longleftarrow r_2, \sigma$	
find $(k, ID) \in L$ s.t. $ID = \sigma \oplus f_k(0, r_1, r_2)$		
	optional — only for mutual authentication	
calculate $\tau = ID \oplus f_k(1, r_1, r_2)$		
	$\longrightarrow \tau$	
		check $\tau \overset{?}{=} ID \oplus f_k(1, r_1, r_2)$

Figure 2.4: The Molnar-Wagner protocol, as executed at each level of the tree, starting from the root of the tree. L is the database of tags and their respective keys for the current level, and f is a Pseudo-Random Function (PRF) implemented in all tags and readers.

the privacy of other tags. For instance, if the branching factor was two, then by recovering the keys in only one tag, the anonymity group of every tag in the system is at least halved. The paper by Karsten and Evans [49] characterises this privacy loss for key-trees with different parameters, and arrives at the conclusion that two-depth trees are the most appropriate for many RFID scenarios.

2.2.5 Protocols based on experimental crypto-primitives

Complexity limits of RFIDs often require protocol designers to invent new primitives, as standard primitives take up too many gate counts, or are unpractical for the RFID setting. It often happens that these new crypto-primitives are found to have weaknesses that were not anticipated by their designers. To overcome the newly discovered shortcomings, the protocols are updated by their original designers or others, and the challenge of finding weaknesses starts again. This cycle is often repeated until the protocol is sufficiently robust to withstand serious attacks.

There are multiple examples in the literature where RFID protocols were designed, published, shown to be weak against certain attacks, and then re-designed. One such example is LMAP [55], shown to be susceptible to an active attack by Barasz et al. [3], then re-designed as M2AP [56], and again shown to be weak against certain attacks by Barasz et al. [4]. Another such protocol is ProbIP by Castelluccia and Soos [17], present in Chapter 4, shown to be insecure by Ouafi et al. [53], and re-designed as presented in Chapter 5.

The DPM protocol by Di Pietro and Molva [58] is also part of the family of experimental protocols and it too has been shown to have some insecure features both by Deursen et al. [67] and by Soos [65]. A complete description of the attack by Soos is present in Chapter 3. Continuing the improvement cycle of (re-)design-and-attack, the DPM protocol has been updated by its designers and others to the Ff family of protocols [6].

Protocols that are widely different from the kind usually accepted by the cryptographic establishment could bring a big leap forward. However, from the designer's perspective, a good cryptographic background is indispensable for the creation of such protocols, otherwise well-established cryptographic techniques will be used with high success rates. From the attacker's perspective, flexibility is required to use such cryptographic techniques in an unfamiliar environment.

2.3 RFID Authentication protocols

Authentication for RFIDs, though is a secondary objective, has received much attention due to the many advantages it could bring. For instance, if RFIDs could be equipped with authentication mechanism, they could be used for securing building access, or for micropayment in public transport. If EPC tags could be cheaply equipped with a means of authentication, they could be used to authenticate warranty repairs, bringing paper-less warranties for customers and less fraudulent repairs for shops.

We have identified five different types of authentication protocols for RFIDs. These are: symmetric cipher-based protocols, protocols based on the Rabin cryptosystem, public-key protocols, PUF-based protocols, and LPN (Learning Parity with Noise)-based protocols.

2.3.1 Symmetric cipher and hash-based protocols

Given a symmetric cipher with a low hardware footprint, it is relatively easy to make a challenge-response authentication protocol. Therefore, there has been a large research effort on implementing standard crypto-primitives on hardware-constrained devices.

In the area of block ciphers, a well-known result is that by Feldhofer et al. [24], implementing AES on 3500 gate equivalents. To reduce the hardware footprint to such an extent, the authors use an 8-bit architecture, calculate the round-key on-the-fly, and store the state only once (unlike FPGA-implementations). This implementation of AES takes 1032 clock cycles to encrypt a cleartext, and 1165 clock cycles to decrypt a ciphertext. Another block-cipher implementation is that of DES on 2300 gate equivalents by Poschmann et al. [59], which takes only 144 clock cycles to encrypt a ciphertext, but cannot decrypt (which the authors argue is not of primary importance).

Low hardware-footprint implementation of the SHA-1 hash function has also been attempted. However, the RFID-optimised implementation by Feldhofer and Wolkerstorfer [23] requires 10868 gate equivalents, much more than an RFID could handle. In the same paper, the authors implement MD5 and other hash functions, all of which require more than 8000 gate equivalents. From the paper it follows that most of the implementation challenge consists of reducing the number of registers needed and the number of flip-flops clocked. However, hash functions typically have a message expansion phase which requires many registers to store the intermediate values, and they usually also operate on multi-byte words, requiring many flip-flops to be clocked at the same time. In contrast, AES has only a storage need of 256bits and operates on single-byte words. Therefore, as Feldhofer et al. [22] have previously pointed out, it is debatable whether currently used hash function designs such as MD5 and SHA-1 are suitable for RFIDs at all.

Another way of solving the problem of hardware constraints is to tweak the cipher itself instead of tweaking its implementation. For stream ciphers, the eStream project's low hardware footprint portfolio [2] is such an attempt. Two of its most well-studied candidates, Trivium [14] and Grain [34] could be promising for applications in RFIDs. Of the two, Trivium operates with a 287-bit state while Grain needs only 160 bits of state to operate. Though Grain uses more complex filter and feedback functions than Trivium, it still seems a better candidate for RFIDs since its extra functional complexity is more than overset by its much lower register usage, and consequently its much smaller hardware footprint. Grain also needs far less initialisation steps: it uses only $2\cot 80 = 160$, clocking both of its registers fully twice, while Trivium clocks its register set four times for $4 \cdot 288 = 1152$ initialisation steps in total. As for bock ciphers specifically designed for hardware-constrained devices, DESL [59] is variation of DES that uses a serialised S-box, thereby reducing the implementation footprint to 1848 GEs from 2300 GEs for the DES implementation [59], but keeping the same speed of operation. Finally, a block cipher designed specifically for RFIDs is PRESENT by Bogdanov et al. [8]. It has a candidate implementation of only 1570 gate equivalents that takes only 32 clock cycles to encrypt a cleartext and, similarly to the DESL implementation, does not have decryption functionality.

2.3.2 Rabin cryptosystem-based protocols

The Rabin cryptosystem-based MAC for RFIDs was first introduced by Shamir [63]. This cryptosystem relies on a variation of the Rabin cryptosystem [60], but it replaces the squaring of the plaintext with an addition of the random multiple of the divisor to create a function, which is ideally suited to RFID-based challenge-response authentication. This cryptosystem is used in SQUASH [64] by Shamir and in WIPR [51] by Oren and Feldhofer.

SQUASH uses multiple techniques to reduce the size of the resulting RFID implementation. For the modulus n it uses a composite Mersenne number of the form $n = 2^k - 1$ to reduce the storage cost. To reduce on-the-tag computation and the communication overhead, the tag does not send the whole encrypted ciphertext: a limited number of bits, say t suffice to make the scheme 2^{-t}-secure. SQUASH suggests to use a t-long window in the middle, but instead of computing all previous bits, suggests to compute only *u guard bits* before this t-long window, further reducing computational costs. After a detailed description of these techniques, the author specifies SQUASH-128 as an example proposal. SQUASH-128 uses about half the number of gates in GRAIN-128 [34] and claims a protection against an adversary with less than 2^{64} of time and space.

Although Rabin cryptosystem-based MACs rely on a proven cryptosystem for security, its specific instantiations can be insecure. This happened with SQUASH-0, the first iteration of SQUASH, presented by Shamir during an invited lecture to RFIDSec 2007 [63]. SQUASH-0 has been shown to be susceptible to attack by Ouafi and Vaudenay in [54].

2.3.3 Protocols based on public-key cryptography

Since public-key cryptography is very hardware-intensive to implement, the only viable way it has been proposed to be implemented on RFID tags is by using a token-based approach where pre-computed tokens, *coupons* are stored on the tag and are used as an aid. The tag, when queried, uses up a coupon to authenticate itself to the reader. The coupons are such that the tag only needs to do a limited number of operations to use them.

The coupon-type scheme is implemented by McLoone and Robshaw [46] in their RFID-optimised implementation of the already ISO-standardised (ISO/IEC FDIS 9798-5) GPS protocol [30]. McLoone and Robshaw propose to use an elliptic curve variant of GPS due to Girault [31] and also require the reader to use Low Hamming Weight challenges, an improvement by Girault and Lefranc [32], to reduce parameter sizes. In their scheme, McLoone and Robshaw replace the modular exponentiation with a coupon and a simple integer (non-modular) calculation. They propose multiple implementations of their scheme, notably with and without a PRNG to help re-generate the random number inside the coupon. The PRNG takes about 1000 gate equivalents on the tag, but drastically reduces the coupon sizes. With the PRNG, the implementation fits on no more than an estimated 1500 gate equivalents, and 10 such reduced-sized coupons take up approximately 500 GEs, for a total of 2000 GEs.

The advantage of the coupon-type scheme in comparison with using one-time passwords is that untrusted readers can authenticate the tag, as the knowledge of the public key does not allow an untrusted reader to fake authentication. The disadvantage of the coupon-type approach in comparison with other (shared-key) cryptographic approaches is that the tag's ability to authenticate can be easily disabled by a malicious reader through the simple exhaustion of coupons — a type of denial of service (DoS) attack. Coupons then need to be re-charged by a non-malicious reader for the tag to be able to authenticate again.

2.3.4 HB$^+$ and its variants

The HB$^+$ protocol by Juels and Weis [40] was a great leap forward by the RFID community towards low hardware footprint authentication protocols. The protocol uses the idea by Hopper and Blum [36], which in turn uses the Learning Parity with Noise (LPN) problem, also known as the minimal disagreement parity problem [19], to base its security on. The LPN problem is known to be NP-hard [5], though finding the solution to it has been consequently improved with ever newer algorithms. The original BKW algorithm [7] was superseded by that of Fossorier et al. [25] and then by that of Levieil and Fouque [69]. Using the newest algorithm, HB$^+$'s claimed 2^{80}-security drops to around 2^{52}. On top of algorithmic advances on LPN, the protocol itself has also been shown to be susceptible to an active attack by Gilbert et al. [27].

To overcome both the active attack and the low parameter sizes offered by HB$^+$, many variants emerged. The literature counts HB^{++} [10], HB* [20], HB-MP [48] and HB$^\#$ [29], all but last of which was broken by Gilbert et al. [28]. HB$^+$'s newest iteration, HB$^\#$ by Gilbert et al. is still resistant to attacks, though one of the research paper's proposed protocol variation and some of the parameter sizes have been shown to be insecure by Ouafi et al. [52].

Most incarnations of the HB$^+$-type protocols have been broken, which indicates that making a correct variation is very difficult. Furthermore, advancements in solving the LPN problem could prove fatal to not only to a specific iteration, but also to the whole concept. The parameter needs increase for every new LPN-solving algorithm and every attack method, thereby increasing the computation, communication, and storage needs of tags implementing HB-based protocols. This continuous increase could eventually make the concept so resource-intensive to implement that other protocol families would become more suitable.

2.3.5 Physically Uncloneable Functions

A Physically Uncloneable Function (PUF) is a function that depends on the differences introduced during manufacture to map inputs to outputs. Therefore, for the same input, different physical incarnations of the same circuit yield different outputs. For electronic circuits, the differences exploited are the wire and gate-delays which are difficult to predict, measure and, most importantly, to accurately model [45]. Since the wire gate-delays cannot be modelled, the circuit cannot be reproduced with accuracy and so the outputs of the circuit for untested inputs cannot be predicted. Therefore, by using a suitable function to eliminate the differences introduced by environmental (temperature, pressure, etc.) variations, the output of the PUF can be used as a means to uniquely identify an electronic circuit.

PUFs were originally invented for optical media by Ravinkanth [61], but later they were ported to silicon by Gassend et al. [26]. They have been proposed to be used in RFIDs by Bolotnyy and Gabriel [9] as a means to authenticate the tag. The protocol proposed the following: the tag is queried at manufacture for random inputs, the obtained (input,output) pairs are stored, and later used to authenticate the tag. Since the attacker can neither build a sufficiently large (input, output) database due to the bitsize of the input, nor can she model the PUF inside the tag given a number of (input, output) pairs, PUFs seem to be ideal to authenticate a tag. Although PUFs also promise the low hardware need of only a couple of hundred gate equivalents, we know of no real-world implementation of them on RFIDs, which undermines their verifiability.

2.4 Protocol properties overview

In this section we provide a list of the previously mentioned protocols and their offered features in a matrix-like fashion.

The matrix of features provided by the previously mentioned protocols is in Table 2.1. Most of the protocols in the table have a star next to their security features, meaning that their offered security feature(s) have been shown to be broken. This is expected in the field of computer security, as given enough time, most cryptographic and security protocols are found to have weaknesses: attacks continually get better, they never get worse. It is interesting to see that almost all RFID security protocols that relied on a new primitive, such as ProbIP, SQUASH-0, DPM, etc. have had at least some of their offered security features broken. It is also evident from the feature-matrix that the protocol that offers the most features and has stood the test of time is the Molnar-Wagner key-tree protocol.

2.5 Conclusions

From our recap of the most influential RFID protocols and their derivatives, it is apparent that RFID protocols follow the pattern prevalent in the more general field of computer security: protocols are conceived, refined, attacked, only to be re-born again with improved design to counter the newly found security vulnerabilities. This never-ending cycle of refinement brings to light the fundamental advantages and limits of RFIDs. Based on this refined view of RFID systems, new protocols are conceived that fit the domain ever more perfectly.

There are many different goals in RFID systems: identification speed, authentication, privacy, system-wide resistance to attacks, tag cheapness, etc. Some protocols try to achieve all of these, some only a particular subset. Some of these goals have even been shown to be conflicting: Burmester et al. have shown [11] that less than linear-time (in the number of tags) private identification is not possible without either shared secrets (possibly compromising system-wide resistance) or public-key cryptography (possibly compromising tag cheapness). Even if some protocols will prove to be more useful than others over the course of time, there surely will remain

Table 2.1: Overview of the previously mentioned RFID protocols' claimed offered features. Features that have been shown to be broken are clearly marked with a star. Note that HB$^{\#}$ can be made resilient to attacks using higher parameters.

Protocol	Unlinkable ident.	Untraceable ident.	Tag auth.	Reader auth.
ISO14443A coll.-avoidance [37]	No	No	No	No
EPC coll.-avoidance [21]	No	No	No	No
Pseudonym-rotation [38]	**Yes***	**Yes***	No	No
ProbIP [17]	**Yes***	**Yes***	No	No
OSK [43]	**Yes***	**Yes***	**Yes**	No
YA-TRAP [66]	**Yes***	**Yes***	**Yes**	No
YA-TRAP+ [12]	**Yes***	**Yes***	**Yes**	No
O-TRAP [12]	**Yes***	**Yes***	**Yes**	No
RIPP-FS [18]	**Yes***	**Yes***	**Yes**	**Yes**
Molnar-Wagner [47]	**Yes**	**Yes**	**Yes**	**Yes**
DPM [58]	**Yes***	**Yes***	**Yes**	**Yes**
SQUASH-0 [63]	No	No	**Yes***	No
SQUASH-128 [64]	No	No	**Yes**	No
WIPR [51]	No	No	**Yes**	No
HB$^+$ [40]	No	No	**Yes**	No
HB$^{\#}$ [29]	No	No	**Yes**	No
PUF [9]	No	No	**Yes**	No

many to be made use of, due to the different trade-offs for the different RFID settings.

Bibliography

[1] Avoine, G., and Oechslin, P. A Scalable and Provably Secure Hash-Based RFID Protocol. In *The 2nd IEEE International Workshop on Pervasive Computing and Communication Security - PerSec 2005* (2005), pp. 110–114.

[2] Babbage, S., Canniere, C. D., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Paar, C., Parker, M., Preneel, B., Rijmen, V., Robshaw, M., and Wu, H. The eSTREAM portfolio. Tech. rep., eStream Project, September 2008.

[3] Bárasz, M., Boros, B., Ligeti, P., Lója, K., and Nagy, D. Breaking LMAP. In *Conference on RFID Security — RFIDSec'07* (Malaga, Spain, July 2007), pp. 69–78.

[4] Báráśz, M., Boros, B., Ligeti, P., Lója, K., and Nagy, D. A. Passive attack against the M2AP mutual authentication protocol for RFID tags. In *RFID 2007 — The First International EURASIP Workshop on RFID Technology* (September 2007).

[5] Berlekamp, E., McEliece, R., and van Tilborg, H. On the inherent intractability of certain coding problems (corresp.). *Information Theory, IEEE Transactions on 24*, 3 (May 1978), 384–386.

[6] Blass, E.-O., Kurmus, A., Molva, R., Noubir, G., and Shikfa, A. The Ff-Family of Protocols for RFID-Privacy and Authentication. In *Workshop on RFID Security — RFIDSec'09* (Leuven, Belgium, July 2009).

[7] Blum, A., Kalai, A., and Wasserman, H. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM 50*, 4 (2003), 506–519.

[8] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., Seurin, Y., and Vikkelsoe, C. PRESENT: An ultra-lightweight block cipher. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2007* (Vienna, Austria, September 2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 450–466.

[9] Bolotnyy, L., and Robins, G. Physically unclonable function-based security and privacy in RFID systems. In *PerCom 2007* (March 2007), IEEE, pp. 211–220.

[10] Bringer, J., Chabanne, H., and Dottax, E. HB^{++}: a lightweight authentication protocol secure against some attacks. In *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006 — SecPerU 2006* (June 2006), pp. 28–33.

[11] Burmester, M., de Medeiros, B., and Motta, R. Robust, anonymous RFID authentication with constant key-lookup. In *ASIACCS* (2008), M. Abe and V. D. Gligor, Eds., ACM, pp. 283–291.

[12] Burmester, M., Le, T. v., and Medeiros, B. d. Provably secure ubiquitous systems: Universally composable RFID authentication protocols. In *Conference on Security and Privacy for Emerging Areas in Communication Networks — SecureComm '06* (Baltimore, Maryland, USA, August-September 2006), IEEE.

[13] Buttyán, L., Holczer, T., and Vajda, I. Optimal key-trees for tree-based private authentication. In *Workshop on Privacy Enhancing Technologies — PET 2006* (Cambridge, United Kingdom, June 2007), pp. 332–350.

[14] Cannière, C. D. Trivium: A stream cipher construction inspired by block cipher design principles. In *ISC* (2006), S. K. Katsikas and et al, Eds., vol. 4176 of *LNCS*, Springer, pp. 171–186.

[15] CASPIAN - Consumers Against Supermarket Privacy Invasion and Numbering. Boycott Benetton. Press release, March 2003. http://www.boycottbenetton.com.

[16] Castelluccia, C., and Avoine, G. Noisy tags: A pretty good key exchange protocol for RFID tags. In *CARDIS* (2006), J. Domingo-Ferrer, J. Posegga, and D. Schreckling, Eds., vol. 3928 of *Lecture Notes in Computer Science*, Springer, pp. 289–299.

[17] Castelluccia, C., and Soos, M. Secret shuffling: A novel approach to RFID private identification. In *RFIDSec'07* (July 2007), pp. 169–180.

[18] Conti, M., Pietro, R. D., Mancini, L. V., and Spognardi, A. RIPP-FS: an RFID identification, privacy preserving protocol with forward secrecy. In *International Workshop on Pervasive Computing and Communication Security — PerSec '07* (New York City, New York, USA, March 2007), IEEE, IEEE Computer Society Press, pp. 229–234.

[19] Crawford, J. M., Kearns, M. J., and Shapire, R. E. The minimal disagreement parity problem as a hard satisfiability problem. Tech. rep., Computational Intelligence Research Laboratory and AT&T Bell Labs, February 1994.

[20] Duc, D., and Kim, K. Securing HB^{+} against GRS man-in-the-middle attack. *Institute of Electronics, Information and Communication Engineers, Symposium on Cryptography and Information, Security* (2007).

[21] EPCglobal. 13.56 MHz ISM band class 1 radio frequency identification tag interface specification (2003). Tech. rep., Auto-ID cetner, MIT, February 2003.

[22] Feldhofer, M., and Rechberger, C. A case against currently used hash functions in RFID protocols. In *OTM Workshops (1)* (2006), R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4277 of *Lecture Notes in Computer Science*, Springer, pp. 372–381.

[23] Feldhofer, M., and Wolkerstorfer, J. Strong crypto for RFID tags - a comparison of low-power hardware implementations. *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on* (May 2007), 1839–1842.

[24] Feldhofer, M., Wolkerstorfer, J., and Rijmen, V. AES implementation on a grain of sand. In *Information Security* (2005), IEEE, pp. 13–20.

[25] Fossorier, M. P. C., Mihaljević, M. J., Imai, H., Cui, Y., and Matsuura, K. A novel algorithm for solving the LPN problem and its applicatio to security evaluation of the HB protocol for RFID authentication. In *INDOCRYPT* (2006), R. Barua and T. Lange, Eds., vol. 4329 of *Lecture Notes in Computer Science*, Springer, pp. 48–62.

[26] Gassend, B., Clarke, D., van Dijk, M., and Devadas, S. Controlled physical random functions. In *Proceedings of the 18th Annual Computer Security Applications Conference — ACSAC '02* (2002), IEEE.

[27] Gilbert, H., Robshaw, M., and Sibert, H. An active attack against HB$^+$ - a provably secure lightweight authentication protocol. In *IEE Electronic Letters 41, 21* (2005), pp. 1169–1170.

[28] Gilbert, H., Robshaw, M. J., and Seurin, Y. Good variants of HB+ are hard to find. In *Financial Cryptography* (January 2008), Springer.

[29] Gilbert, H., Robshaw, M. J. B., and Seurin, Y. HB$^\#$: Increasing the security and efficiency of HB$^+$. In *Advances in Cryptology — EUROCRYPT '08* (2008), N. P. Smart, Ed., vol. 4965 of *Lecture Notes in Computer Science*, Springer, pp. 361–378.

[30] Girault, M. Self-certified public keys. In *Advances in Cryptology — EUROCRYPT '91* (1991), pp. 490–497.

[31] Girault, M. Low-size coupons for low-cost IC cards. In *CARDIS* (2000), J. Domingo-Ferrer, D. Chan, and A. Watson, Eds., vol. 180 of *IFIP Conference Proceedings*, Kluwer, pp. 39–50.

[32] Girault, M., and Lefranc, D. Public key authentication with one (online) single addition. In *Cryptographic Hardware and Embedded Systems - CHES 2004* (2004), vol. 3156/2004 of *Lecture Notes in Computer Science*, pp. 967–984.

[33] Hancke, G. Modulating a noisy carrier signal for eavesdropping-resistant HF RFID. *e&i — Elektrotechnik und Informationstechnik 124*, 11 (November 2007), 404–408.

[34] Hell, M., Johansson, T., and Meier, W. Grain — a stream cipher for constrained environments. In *Proceeding of the Workshop on RFID and Lightweight Crypto* (July 2005), pp. 114–125.

[35] Hellman, M. E. A cryptanalytic time-memory trade off. In *IEEE Transactions on Information Theory* (1980), vol. IT-26/4, pp. 401–406.

[36] Hopper, N. J., and Blum, M. Secure human identification protocols. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security* (London, UK, 2001), Springer-Verlag, pp. 52–66.

[37] ISO/IEC. 14443-3 — Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 3: Initialization and anticollision, 2001, Stage: 90.92 — 2007-12-11.

[38] Juels, A. Minimalist cryptography for low-cost RFID tags. In *International Conference on Security in Communication Networks — SCN 2004* (Amalfi, Italia, September 2004), C. Blundo and S. Cimato, Eds., vol. 3352 of *LNCS*, Springer-Verlag, pp. 149–164.

[39] Juels, A., Rivest, R., and Szydlo, M. The blocker tag: Selective blocking of RFID tags for consumer privacy. In *ACM CCS 2003* (October 2003), V. Atluri, Ed., ACM Press, pp. 103–111.

[40] Juels, A., and Weis, S. Authenticating pervasive devices with human protocols. In *Advances in Cryptology — CRYPTO'05* (Santa Barbara, California, USA, August 2005), V. Shoup, Ed., vol. 3126 of *LNCS*, IACR, Springer-Verlag, pp. 293–308.

[41] Juels, A., and Weis, S. Defining Strong Privacy for RFID. In *International Conference on Pervasive Computing and Communications — PerCom 2007* (New York City, New York, USA, March 2007), IEEE, IEEE Computer Society Press, pp. 342–347.

[42] Kirschenbaum, I., and Wool, A. How to build a low-cost, extended-range RFID skimmer. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2006), USENIX Association.

[43] Koutarou, M. O., Suzuki, K., and Kinoshita, S. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop* (MIT, Massachusetts, USA, November 2003).

[44] Lamport, L. Password authentication with insecure communication. *Commun. ACM 24*, 11 (1981), 770–772.

[45] Lim, D., Lee, J. W., Gassend, B., Suh, G. E., van Dijk, M., and Devadas, S. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2005), 1200–1205.

[46] McLoone, M., and Robshaw, M. J. B. Public key cryptography and RFID tags. In *CT-RSA* (2007), M. Abe, Ed., vol. 4377 of *Lecture Notes in Computer Science*, Springer, pp. 372–384.

[47] Molnar, D., and Wagner, D. Privacy and security in library RFID: issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), ACM Press, pp. 210–219.

[48] Munilla, J., and Peinado, A. HB-MP: A further step in the hb-family of lightweight authentication protocols. *Comput. Netw. 51*, 9 (2007), 2262–2267.

[49] Nohl, K., and Evans, D. Hiding in Groups: On the Expressiveness of Privacy Distributions. In *Proceedings of The Ifip Tc 11 23rd International Information Security Conference — SEC 2008* (Milan, Italia, September 2008), vol. 278 of *Lecture Notes in Computer Science*, Springer, pp. 1–15.

[50] Oechslin, P. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology — CRYPTO 2003* (2003), vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 617–630.

[51] Oren, Y., and Feldhofer, M. WIPR — a public key implementation on two grains of sand. In *Workshop on RFID Security 2008* (2008), S. Dominikus, Ed., pp. 15 – 27.

[52] Ouafi, K., Overbeck, R., and Vaudenay, S. On the security of HB# against a man-in-the-middle attack. In *Advances in Cryptology — Asiacrypt 2008* (Melbourne, Australia, December 2008), vol. 5350 of *Lecture Notes in Computer Science*, Springer, pp. 108–124.

[53] Ouafi, K., and Phan, R. C.-W. Privacy of Recent RFID Authentication Protocols. In *Information Security Practice and Experience, 4th International Conference, ISPEC 2008* (Berlin, 2008), Lecture Notes in Computer Science, Springer, pp. 263–277.

[54] Ouafi, K., and Vaudenay, S. Smashing SQUASH-0. In *EUROCRYPT* (2009), A. Joux, Ed., vol. 5479 of *Lecture Notes in Computer Science*, Springer.

[55] Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J., and Ribagorda, A. LMAP: A real lightweight mutual authentication protocol for low-cost RFID tags. In *Proceedings of RFIDSec'06* (Graz, Austria, July 2006), Ecrypt.

[56] Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J., and Ribagorda, A. M2AP: A minimalist mutual-authentication protocol for low-cost RFID tags. In *International Conference on Ubiquitous Intelligence and Computing — UIC'06* (September 2006), vol. 4159 of *LNCS*, Springer-Verlag, pp. 912–923.

[57] Pfitzmann, A., and Köhntopp, M. Anonymity, unobservability, and pseudonymity — A proposal for terminology. In *Designing Privacy Enhancing Technologies* (2001), vol. 2009 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–9.

[58] PIETRO, R. D., AND MOLVA, R. Information confinement, privacy, and security in RFID systems. In *Proceedings of the 12th European Symposium On Research In Computer Security* (September 2007), pp. 187–202.

[59] POSCHMANN, A., LE, G., SCHRAMM, K., AND PAAR, C. A family of light-weight block ciphers based on DES suited for RFID applications. In *Proceedings of FSE 2007, LNCS* (2006), Springer-Verlag.

[60] RABIN, M. O. Digitalized signatures and public-key functions as intractable as factorization. Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[61] RAVINKANTH, P. Physical one-way functions. Tech. rep., MIT, 2001. Ph.D. Thesis.

[62] SAVRY, O., PEBAY-PEYROULA, F., DEHMAS, F., ROBERT, G., AND REVERDY, J. RFID noisy reader — How to prevent from eavesdropping on the communication? In *CHES* (2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 334–345.

[63] SHAMIR, A. SQUASH: A new one-way hash function with provable security properties for higley contrained devices such as RFID tags. In *Invited lecture to the RFID Securty 2007 Workshop* (2007).

[64] SHAMIR, A. SQUASH — a new MAC with provable security properties for highly constrained devices such as RFID tags. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, pp. 144–157.

[65] SOOS, M. Analysing the Molva and Di Pietro Private RFID Authentication Scheme. In *Workshop on RFID Security — RFIDSec'08* (Budapest, Hungary, July 2008).

[66] TSUDIK, G. YA-TRAP: Yet another trivial RFID authentication protocol. In *International Conference on Pervasive Computing and Communications — PerCom 2006* (Pisa, Italy, March 2006), IEEE, IEEE Computer Society Press, pp. 640–643.

[67] VAN DEURSEN, T., MAUW, S., AND RADOMIROVIC, S. Untraceability of RFID protocols. In *WISTP* (2008), J. A. Onieva, D. Sauveron, S. Chaumette, D. Gollmann, and C. Markantonakis, Eds., vol. 5019 of *Lecture Notes in Computer Science*, Springer, pp. 1–15.

[68] VAUDENAY, S. On privacy models for RFID. In *Advances in Cryptology — Asiacrypt 2007* (Kuching, Malaysia, December 2007), vol. 4833 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 68–87.

[69] ÉRIC LEVIEIL, AND FOUQUE, P.-A. An improved LPN algorithm. In *Security and Cryptography for Networks — SCN* (2006), R. D. Prisco and M. Yung, Eds., vol. 4116 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 348–359.

Part II

On the difficulty of designing ad-hoc RFID security protocols

In this part of the thesis we elaborate on the difficulties arising when designing ad-hoc RFID security protocols. Ad-hoc or in other words experimental protocols (see Sect. 2.2.5) rely on an innovative approach to replace a function that is normally provided by a standard cryptographic algorithm. In this class of protocols is for instance LMAP by Peris et al. [20] and the protocol by Di Pietro and Molva [22]. Since these non-cryptographic protocols are highly innovative, the cryptanalysis/re-design cycle that has refined standard techniques do not apply, and so these protocols are often found to have failures, which are then corrected and the analysis/re-design cycle starts again. This happened with LMAP [1] which has been redesigned as M2AP [21] which again was cryptanalysed by Barasz et al. [2]. For these reasons, ad-hoc protocols are notoriously difficult to design such that their security stands the test of time.

Organisation

This second part of the thesis is made up of three chapters. In Chapter 3 we present the Di Pietro-Molva protocol and describe its analysis to uncover some shortcomings of the protocol. In Chapter 4 we present the Probabilistic Identification Protocol (ProbIP), and describe the published attack against it, which breaks its security. Finally, in Chapter 5 we present the updated version of ProbIP, EProbIP, which significantly improves on the security of the original scheme.

Chapter 3

Analysing the Molva and Di Pietro Private RFID Authentication Scheme

In this chapter we examine a protocol by Refik Molva and Roberto Di Pietro [22] that tries to solve the authentication and privacy problems of RFIDs with novel design ideas. This protocol is part of a class of protocols we call experimental (see Sect. 2.2.5), that is, it relies on an innovative approach to replace a function that is normally provided by a standard cryptographic algorithm. The Di Pietro-Molva protocol uses this non-standard algorithm to achieve private identification. To achieve authentication, the protocol uses a standard hash function.

Organisation

In Sect. 3.1 we present the Di Pietro-Molva protocol and in Sect. 3.2 we analyse one of its function's unintended behaviour. In Sect. 3.3 we analyse the private identification part of the protocol, then in Sect. 3.4 we show a practical active attack against it, requiring only a few identification attempts to break privacy, and additionally $2^{K/3}$ operations to break authentication. Finally, in Sect. 3.5 we present a list of found design flaws and their remedies and in Sect. 3.6 we draw some conclusions.

3.1 A short summary of the Molva - Di Pietro scheme

The scheme of Molva and Di Pietro [22] aims to be a private RFID mutual authentication scheme. As such it aims to solve three problems at the same time for RFID tags: it privately indicates the tag ID to the reader, it authenticates the tag to the reader and authenticates the reader to the tag. For clarity of explanation, we make a clear distinction between these three parts throughout this analysis.

There are n tags $\mathcal{T}_i$ in the system, each of which is configured with a unique key k_i which serves as the tag ID and the key at the same time. This key is used as a bitvector with $k_i[x]$ representing the x^{th} bit of k_i. Each reader also has a unique ID_j. At system initialisation, each reader $\mathcal{R}_j$ is configured with the reader-specific key of each tag, $k_{i,j} = h(k_i||ID_j||k_i)$, where $h(\cdot)$ is a secure hash function available both on the tag and the reader.

3.1.1 Private identification

For private identification the protocol relies on the function DPM. The input to DPM is l bits where l is divisible by 3, and the output is one bit. DPM is defined as:

$$DPM(x) = \bigoplus_{i=0}^{l/3} M(x[3i], x[3i+1], x[3i+2]) \tag{3.1}$$

where M is the majority function: its input is 3 bits, and its output is one bit. M decides whether there are more `1`-s in its input is than `0`-s and returns the value `1` or `0` accordingly.

The private identification part of the protocol is as follows:

1. $\mathcal{R}_j$ sends ID_j to the tag
2. $\mathcal{T}_i$ computes $k_{i,j} = h(k_i||ID_j||k_i)$. It then generates q l-bit random nonces, r_p ($p = 1 \dots q$). It then sends q α_p-s where $\alpha_p = r_p \oplus k_{i,j}$ and it sends a q-bit long vector V that is set up as $V[p] = DPM(r_p)$ to the reader
3. $\mathcal{R}_j$ computes $DPM(\alpha_p \oplus k_{i,j})$ for all keys $k_{i,j}$ it possesses and checks it against $V[p]$. This is called the *Lookup Process.* The key $k_{i,j}$ that fits on all pairs $(\alpha_p, V[p]), p = 1 \dots q$ is the tag suspected of sending the packets

At the end of the identification part, the reader suspects which tag it is talking to. The authors explain in detail how large q (the number of pairs sent) should be so that with very high probability only the correct key will fit on all pairs $(\alpha_p, V[p]), p = 1 \dots n$.

The choice of parameters by the authors is not explicitly stated in the paper, but we can assume that an 80-bit security was meant, as the authors use a 160-bit hash function for h. The parameter sizes are then $|k_i| = 81$, $|ID_j| = 80$ and $l = 81$ where $|x|$ means the bitlength of x. The parameter q is only defined as being large enough so that the probability of a false positive, calculated by the authors as $n(1/2)^q$ (where n is the number of tags in the system) should be very low.

3.1.2 Tag authentication

During tag authentication the reader would like to make sure that it is indeed talking to tag $\mathcal{T}_i$. This is important since a malicious tag could simply replay an instance of the private identification part of the protocol to the same reader and the reader would not be able to differentiate between the two tags.

The tag authentication part of the protocol is as follows:

1. $\mathcal{R}_j$ sends a nonce n_j to the tag
2. $\mathcal{T}_i$ computes and sends $\omega = h(k_{i,j}||n_j||r_1||k_{i,j})$ to the reader
3. $\mathcal{R}_j$ computes $r_1 = \alpha_1 \oplus k_{i,j}$ and checks ω against $h(k_{i,j}||n_j||r_1||k_{i,j})$. If they match, the tag is authenticated

3.1.3 Reader authentication

After tag authentication the reader authenticates itself to the tag:

1. $\mathcal{R}_j$ computes $r_1 = \alpha_1 \oplus k_{i,j}$ and sends $h(k_{i,j}||r_1||k_{i,j})$ to the tag
2. $\mathcal{T}_i$ computes $h(k_{i,j}||r_1||k_{i,j})$ and checks it against the received hash. If they match, the reader is authenticated

3.2 The DPM function

The DPM function is such that if an even number of majority functions' outputs are flipped, the output is not flipped. This property of the DPM function has two unfortunate consequences: key- and pair-equivalences, which we detail in this section.

3.2.1 Key equivalences

Let us divide the key $k_{i,j}$ into blocks of 3 bits, which we simply call key *blocks*. If an even number of key blocks are inverted, the resulting key is indistinguishable by the reader from the original key using only the $(\alpha_p, V[p])$ pairs. This is because $V[p] = DPM(\alpha_p \oplus k_{i,j}) = DPM(\alpha_p \oplus k_{i,j} \oplus \text{inversions})$ and so the result of the Lookup Process will not depend on whether the blocks were inverted or not. One such pair of key-equivalents is $k_{i,j} =$`[001 000 100]`$\approx$`[110 000 011]`.

Key equivalences mean that any key of size l belongs to a key-equivalence group of size $\sum_{i=0}^{\lfloor (l/3)/2 \rfloor} \binom{l/3}{2i} = 2^{l/3-1}$, i.e. in a keyspace of 2^l there are $2^{l-l/3+1}$ key-equivalence groups (or key-eqgroups for short). Keys in a key-eqgroup are equivalent if the reader only looks at the $(\alpha_p, V[p])$ pairs. Naturally, if the reader checks ω, each of these tags is distinguishable from one another. However, as key sizes of at least 80 bits must be used to thwart brute-forcing of keys, the number of possibilities that could be left after the pairs are processed is $2^{81/3-1} \approx 7$ million, which is impossible for the reader to check, since executing a hash function this many times is too time-consuming for a quick identification session.

Under normal conditions, the keyspace (2^l) is extremely sparsely populated — there may be less tags in the whole system than the size of one such key-eqgroup. However, care must be taken to have a very low ratio of $n : 2^{2l/3+1}$, otherwise the hash $h(k_i||ID_j||k_i)$ could produce many keys that are in the same key-eqgroup for a certain reader (i.e. for a certain ID_j). This can be a problem, as there might be a time-limit for the reader to find which tag it is talking to within the key-eqgroup. We can calculate the chance that for a random reader ID_j there is at least one key-eqgroup with more than one tag inside as $1 - (1 - n/2^{2l/3+1})^{n-1}$. For example, for $n = 10^7$, $l = 81$, this probability is 0.003, which means that if there are 1000 readers deployed, then there is a $1 - (1 - 0.003)^{1000} \approx 95\%$ chance that at least one reader has at least one key-eqgroup with more than one tag inside.

Key-eqgroups also mean that the attacker's keyspace is limited to $2^{l-l/3+1}$ if the attacker is only interested in the key-eqgroup the tag belongs to. If the attacker is interested in the exact tag, she can try to do $2^{l/3-1}$ hash operations to find $k_{i,j}$ using n, r_1 and ω of a session. This is feasible even for $l = 99$ and $h =$SHA-1: the key-eqgroup would be $2^{l/3-1} = 2^{32} \approx 4$ billion large and a Xeon quad-core can do about 4 million SHA-1 operations per second, so in less than half an hour would find the key $k_{i,j}$. Let us note that the hash function on the tag would be more simple than SHA-1 due to hardware constraints, and would be much easier to brute-force.

3.2.2 Pair-equivalences

The design of the DPM function also implies $(\alpha_p, V[p])$ pair-equivalences: for multiple different pairs the same keys are found not to fit (i.e. pruned) during the Lookup Process.

If $V[p] = DPM(\alpha_p \oplus k_{i,j})$ then for any α'_p that has an even number of blocks inverted, $V[p] = DPM(\alpha'_p \oplus k_{i,j})$ will also hold. Therefore the Lookup Process will prune the same keys for these pairs. For example `[010 001]-0`≈`[101 110]-0`

If an odd number of blocks are inverted in α_p then an odd number of blocks in r must have been inverted, so $V'[p] = DPM(r') = \overline{V[p]}$. Therefore, the Lookup Process will prune the same keys for these pairs as well. For example `[100 000]-1`≈`[011 000]-0`.

The property of pair-equivalences implies that the Lookup Process is not running at maximum efficiency since the possibility that two equivalent r-s are produced by the tag during identification is higher than with a DPM function that does not have this property.

3.2.3 The effect of equivalences

It is important to note that during the Lookup Process both pair- and key-equivalences are acting in tandem, and have two different effects: the first slows down the pruning of the keyspace ($k_{i,j}$-s stored in the reader), and the second does not let the pruning go beyond a certain point.

3.3 Private identification

In this section we examine the identification part of the protocol from multiple angles. First, we examine the Lemmas it depends on, then investigate the number of $(\alpha_p, V[p])$ pairs needed for its proper working, and finally we make some practical observations regarding its bandwidth need.

3.3.1 Observations about Lemma 2

In the original paper, Lemma 3 states that for a randomly chosen r, the chance that $DPM(r) = 1$ is 0.5, and the chance that $DPM(r) = 0$ is also 0.5. This means that given a set of tags and their unique random keys, a randomly chosen $(\alpha_p, V[p])$ pair will on average fit on half of the keys. Using this lemma, the authors conclude in Lemma 2 that given q randomly chosen pairs, the probability that at least one key will survive out of n random keys is less than $n(1/2)^q$. For this to hold, the distribution of surviving keys should have been random after one pair. However, the distribution of the surviving keys is not random: for example, two identical pairs prune the keyspace only once.

3.3.2 The true number of $(\alpha_p, V[p])$ pairs needed

We will calculate δq, the additional number of $(\alpha_p, V[p])$ pairs needed for the reader to identify a key-eqgroup with a probability of at least 99% given that there are $n'(\leq n)$ key-eqgroups among the tags in the system.

Since the distribution of key-eqgroups is not random after one or more $(\alpha_p, V[p])$ pairs, we need to investigate whether we need to compensate for this with extra pairs. For the moment, let us use Lemma 2 to calculate an estimate of the number of pairs needed. The conclusion of Lemma 2 is that for an incorrect identification probability $\epsilon \leq 2^{-r}$, the number of pairs sent, q, must be at least $r + \log(n')$ [i]. As an example, for the parameters $n' = 10^6$ and $\epsilon \leq 0.01$, q must be at least 27. We will now investigate how large δq, the additional number of pairs needed should be to compensate for the fallacy of Lemma 2.

Redundant pairs

We call redundant pairs a set of $(\alpha_p, V[p])$ pairs that are not equal in the sense of pair-equivalences, but as a set form a tautology: removing one or more pairs from the set will not reduce the information content of the set. One such set is for instance:

α_p	$V[p]$
011 100	0
011 111	1
010 100	0
010 111	1

In this set, given any 3 of the 4 pairs the 4th pair can be deduced. In other words, the information content of these 4 pairs is only 3 pairs. Theoretically determining the occurrence probabilities of redundant pairs is out of the scope of this analysis. Instead, we ran some tests to observe what is the practical occurrence ratio of them for different keysizes and different number of non-equivalent pairs. For the experiment, the number of tags in the system (n) is irrelevant, the r_p-s were generated using a PRNG with uniform output, and the tag- and reader-specific key $k_{i,j}$ was also randomly drawn. The results of this experiment are present in Fig. 3.1.

Using a logarithmic (ln) scale for the axis z, the equation $9.62l - 16.11q + 24z + 42.18 = 0$ describes the plane. Substituting $q = 27$ and $l = 81$ into this equation yields $z = -16.1$, i.e. the occurrence rate for these parameters is $e^{-16.1} \approx 10^{-7}$. This is so small that it does not need to be compensated with δq. However, for larger tag populations, the occurrence rate can be very high. For example, for 10^9 tags and consequently $q = 40$ the occurrence rate jumps to 10^{-4}, which cannot be ignored and must be compensated with a δq strictly larger than zero.

Pair-equivalents during identification

The chance that among q random $(\alpha_p, V[p])$ pairs there will be at least one that is a pair-equivalent is as follows. There are $2^{2l/3}$ pair-eqgroups, so the chance that at

[i]The authors meant log to be $\log_2$

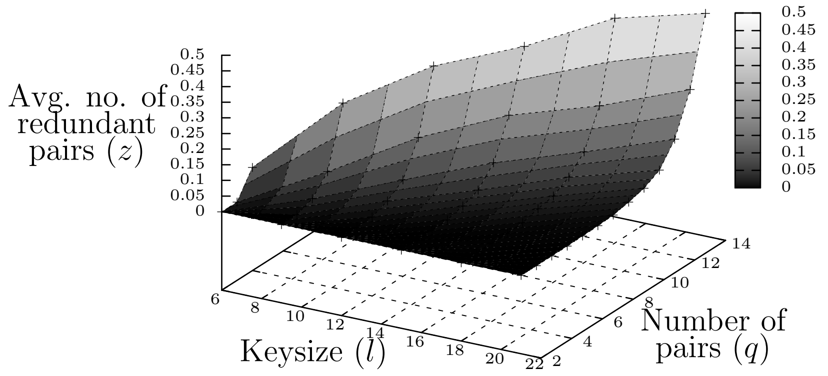


Figure 3.1: Average number of redundant pairs within q non-equivalent pairs for different keysizes. As the number of pairs increases, the occurrence rate of redundant pairs increases at an exponential rate

least two pairs will be from the same pair-eqgroup among q pairs is

$$P_{repeat} \leq \binom{q}{2} 2^{-2l/3} \tag{3.2}$$

For $l = 81$, $n = 10^6$, $\epsilon \leq 0.01$ and $q = 27$, $P_{repeat} \leq \binom{27}{2} 2^{-54} \approx 2 \cdot 10^{-12}$. This probability is so small, that practically it never occurs and so it does not need to be compensated for with additional pairs.

3.3.3 The bandwidth needed in a common setup

Let us measure the total bandwidth cost of the protocol: for an even 80-bit security goal as mentioned in Sect. 3.1.1, the protocol uses $q \cdot 81$ bits for the α_p-s, q bits for V, and $80 + 160 + 160$ bits for the two-way authentication. The total bandwidth cost is thus

$$B = 80 + 81q + q + 80 + 160 + 160 = (6 + q)l + q \text{ bits} \tag{3.3}$$

In the case of $n' = 10^6$, $P_{find} \approx 0.99$, B is 2667 bits, which can be thought of as prohibitively large.

3.3.4 Implementation of the Lookup Process

Although the original paper mentions the processing overhead of the reader and concludes that it is $O(n \log n)$, we investigated the constant hiding behind the big O by implementing the Lookup Process on a Xeon E5345@2.33GHz computer. To speed up the calculations, we used all practical optimisations and programming techniques available to us, such as pure binary operations, loop-unrolling and memory bandwidth minimisation. The results are shown in Table 3.1.

It is clear from Table 3.1 that even if the number of tags in the system is only 10^6, the reader would need to be very powerful — a hand-held reader rarely has the speed of a 2.33GHz Xeon processor. For larger tag populations, the RAM requirement would also become a problem. Therefore, it is more pertinent to speak about backend

Table 3.1: This table shows the average time and RAM required by the Lookup Process to find one tag. The Lookup Process was running on a Xeon E5345@2.33GHz with all optimisations other than assembly-level coding. P_{find} was set to 0.99 and keylength was 81 bits. As the number of tags in the system increases, the time it takes to identify the tag increases in an almost linear manner. Since memory usage mainly consists of storing tag keys, it increases linearly with the number of tags in the system

Number of tags	10^6	10^7	10^8
Time (s)	0.1	1.1	12
Memory (MB)	9.6	96	965

systems that process all incoming identifications and return the tag ID in batch mode. However, if backend systems must be used, then per-reader IDs are not a possibility and per-backend IDs must be used. Thus information confinement — one of the main goals of the paper — cannot be fully achieved.

3.4 Retrieving $k_{i,j}$

In this section we present an algorithm for exhaustive search and an efficient man-in-the-middle (MiM) attack to find $k_{i,j}$. Since the key of the tag is always masked with the ID_j of the reader through $k_{i,j} = h(k_i||ID_j||k_i)$, the attacker can only break the privacy of the tag when the tag is communicating with the same reader. Since both tag-to-reader and reader-to-tag authentication only requires the knowledge of $k_{i,j}$, retrieving it allows the attacker to both authenticate himself to the original reader as a legitimate tag, and to authenticate himself to the original tag as a legitimate reader.

3.4.1 Exhaustive search

The authors strangely forget to mention, indeed they might have overlooked, the simple exhaustive search against the identification part of the protocol, an attack vector that all schemes must exhibit that are not information theoretically secure. Naturally, the identification part of the protocol cannot be information theoretically secure since it sends a secure message (the ID of the tag) possibly infinite number of times while sharing just a few bits of secret information with the reader.

The exhaustive search simply executes the Lookup Process with all possible $2^{2l/3+1}$ key-eqgroups to find the key-eqgroup of $k_{i,j}$. Given different non-redundant $(\alpha_p, V[p])$ pairs, the number of possible keys is reduced by a factor of 2 for each $(\alpha_p, V[p])$ pair. Using the same formula as in Sect. 3.3 and setting $r = 0$, $n' = 2^{2l/3+1}$ we find that we need $2l/3 + 1$ non-equivalent non-redundant pairs to mount the attack. An implementation of the algorithm is present in **Function** `Brute_force`. In practice, executing this algorithm is extremely time-consuming, due to the size of the keys involved — the algorithm runs in $O(n2^l)$ time.

We have implemented the search algorithm and found it to perform as detailed in

Function `Brute_force(`$\alpha_p - V[p]$ *pairs*`)` Given n number of $\alpha_p - V[p]$ pairs harvested from the tag, this function finds a key $k_{i,j}$ that is in the same key-eqgroup as $k_{i,j} = h(k_i, ID_j, k_i)$. It essentially tries all possible r_p-s that the tag could have generated, and reduces the keyspace accordingly. For the function to work, the variable r_p must be represented as an l-bit binary.

```
  Input: $n$ number of $V[p] - \alpha_p$ pairs
1 for $i \leftarrow 0$ to $2^l - 1$ do keys$[i] \leftarrow true$ ;
2 rests $\leftarrow 2^l$;
3 for $k_{i,j} \leftarrow 0$ to $2^l - 1$ do
4     for $p \leftarrow 0$ to $n$ do
5         if $DPM(r_p \oplus k_{i,j}) \neq V[p]$ then keys$[k] \leftarrow false$
6     end
7 end
8 for $k_{i,j} \leftarrow 0$ to $2^l - 1$ do if keys$[k_{i,j}] = true$ then return $k_{i,j}$;
```

Table 3.2: This table shows the performance our implementation of the exhaustive search. For computing the timing values in the table, we used a Xeon E5345@2.33GHz CPU. As the key size increases, the time required to break the privacy of the tag increases at an exponential rate.

Keysize (bits)	27	30	33	36	39	42	45
Time	0.38s	2.9s	27.2s	209s	1462s	13003s	76738s

Table 3.2. on a Xeon E5345@2.33GHz CPU. Due to the time required to execute the algorithm for large keysizes, it can only be used to brute-force a key that the attacker has some information about. Using some supporting information, the exhaustive search can be used to fill out the gaps in (i.e. compute the unknown parts of) the key.

Once the key-eqgroup of $k_{i,j}$ is found, the attacker can simply try to execute the hash function $h()$ implemented in the tag to try each of the $2^{l/3-1}$ combinations left against an ω response to find the exact $k_{i,j}$. As discussed in Sect. 3.2.1 this should not be difficult even for $l = 99$ and $h =$SHA-1.

3.4.2 Man-in-the-middle attack

The man-in-the-middle (MiM) attack gains information about $k_{i,j}$ based on the success or failure of the authentication. Since success or failure can be represented in one bit, at each authentication attempt the attacker learns exactly one bit of information. The attack exploits that with the exception of α_1, none of the α_p-s are authenticated: if an attacker modifies α_2 into α_2' and the Lookup Process still finds the key $k_{i,j}$, then $DPM(k_{i,j} \oplus \alpha_2') = V[2]$, so she either managed not to invert the output of any of the majority functions (M-s) in the DPM, or she managed to invert of an even number of them. However, if $DPM(k_{i,j} \oplus \alpha_2') = \overline{V[2]}$, the authentication fails since the reader fails to find the key $k_{i,j}$ during the Lookup Process, in which case the attacker can be sure that she must have inverted the output of an impair

Table 3.3: This table shows the conclusions that can be drawn by actively modifying an ongoing protocol session: the attacker needs to invert one bit of α_2 at a block's 2nd or 3rd position and observe the outcome of the protocol. If the tag does not get accepted as authentic, then she can deduce the information that is present in the row marked with $\times$, if the tag does get accepted she can deduce the information that is present in the row marked with ✓

Inverted bit	Auth	Original $\alpha_2[x\dots x+2]$ block `000`	`001`
$\alpha_2[x+2]$	✓	$k_{i,j}[x] = k_{i,j}[x+1]$	$k_{i,j}[x] = k_{i,j}[x+1]$
$\alpha_2[x+2]$	$\times$	$k_{i,j}[x] \neq k_{i,j}[x+1]$	$k_{i,j}[x] \neq k_{i,j}[x+1]$
$\alpha_2[x+1]$	✓	$k_{i,j}[x] = k_{i,j}[x+2]$	$k_{i,j}[x] \neq k_{i,j}[x+2]$
$\alpha_2[x+1]$	$\times$	$k_{i,j}[x] \neq k_{i,j}[x+2]$	$k_{i,j}[x] = k_{i,j}[x+2]$
		Original $\alpha_2[x\dots x+2]$ `010`	`100`
$\alpha_2[x+2]$	✓	$k_{i,j}[x] \neq k_{i,j}[x+1]$	$k_{i,j}[x] \neq k_{i,j}[x+1]$
$\alpha_2[x+2]$	$\times$	$k_{i,j}[x] = k_{i,j}[x+1]$	$k_{i,j}[x] = k_{i,j}[x+1]$
$\alpha_2[x+1]$	✓	$k_{i,j}[x] = k_{i,j}[x+2]$	$k_{i,j}[x] \neq k_{i,j}[x+2]$
$\alpha_2[x+1]$	$\times$	$k_{i,j}[x] \neq k_{i,j}[x+2]$	$k_{i,j}[x] = k_{i,j}[x+2]$

number of majority functions in the DPM.

The attacker will take the simplest approach to modifying α_2: she will try to invert one majority function's output. As a simple example, let us consider the block $\alpha_2[x\dots x+2] = 000$. In this case $M(k_{i,j}[x\dots x+2] \oplus \alpha_2[x\dots x+2]) = M(k_{i,j}[x\dots x+2])$, so this block's M will depend solely on the key bits $k_{i,j}[x\dots x+2]$. Let us now invert the x+2nd bit of α_2: M will not change if and only if $k_{i,j}[x] = k_{i,j}[x+1]$, since then no matter what $k_{i,j}[x+2]$ is, $M = k_{i,j}[x] = k_{i,j}[x+1]$. However, M will change if $k_{i,j}[x] \neq k_{i,j}[x+1]$ since then the majority is decided by $k_{i,j}[x+2] \oplus \alpha_2[x+2]$, which the attacker just inverted. Therefore, by inverting the x+2nd bit of α_2, and observing the authentication, the attacker can conclude whether $k_{i,j}[x] = k_{i,j}[x+1]$ or not.

Reasoning this way, all possible α_2 blocks will lead to a conclusion: see Table 3.3 for all the conclusions that can be drawn given any α_2 block and an inversion at either the 2nd or the 3rd bit of the block. From the attack's point of view a block in α_2 behaves the same as the inversion of the same block (i.e. $\texttt{000}\approx\texttt{111}$), so only one of the two is listed.

Using Table 3.3. the attacker only needs two authentication sessions per key block to narrow down the possible key-combinations to $2^{l/3}$. At this point, she will have two possibilities for each key block. It is sufficient for the attacker to simply try the recorded set of $(\alpha_p, V[p])$ pairs on one of the $2^{l/3}$ combinations. If at least one pair does not match, then she simply needs to invert the first block to recover the key-eqgroup of the tag. The whole process thus takes 2 authentication sessions per key block, and less than one millisecond of processing. For $l = 81$ this means the privacy can be broken to a key-equivalence level in a mere 54 protocol sessions.

The MiM attack can be used in tandem with the brute-force attack. If the attacker is willing to invest some time into breaking the scheme, she can use the MiM attack to learn some information about the first x bits of the key, and then use the brute-force attack to learn the rest of the $l - x$ bits. As long as $l - x$ is less than 39, this should take very little time for the attacker, and would let him use less active rounds: in the case of $l = 81$, instead of the normal 54 active rounds needed, she would only need 30 active rounds and a couple of minutes to find the key-eqgroup of $k_{i,j}$.

To find the exact $k_{i,j}$ of the tag, the attacker would need to perform the same actions as in the last stage of the brute-force passive attack, i.e. execute the hash function $h(\cdot)$ for each of the $2^{l/3-1}$ remaining combinations and compare it against an ω response. As discussed in Sect. 3.2.1 this should not be difficult even for $l = 99$ and $h =$SHA-1 — it would take only 2^{32} hash operations, which is possible to do in an hour on a desktop PC.

3.5 Design flaws and their remedies

It is very difficult to design security protocols for RFIDs since the resources available on this platform are extremely limited. Some security properties must always be sacrificed in order to fit the security functions on the tag. However, we believe that the scheme being analysed is not only imperfect due to the limitations of the platform, but it also exhibits limitations that are due to design flaws. In this section we list a set of design flaws exhibited by the scheme and then propose some modifications to overcome the problems detailed.

3.5.1 Design flaws

We have found the following list of design flaws during our analysis of the protocol:

- Identification and authentication boundaries should have been clearly defined. Had identification and authentication been designed and analysed independently, many of the shortcomings described could have been averted
- Identification and authentication keys should have been generated differently. Had this been the case, the attacks presented would only have recovered the identification key and so would have been restricted to breaking the privacy. A simple difference between the generation of the two $k_{i,j}$-s would have been enough
- Given that the identification was not cryptographically secured, the integrity of the data exchanged during identification should have been authenticated during authentication. It is clear that the identification was not cryptographically secured since it only used a xor and a majority function to do its work
- The choice of the DPM function is not clearly motivated and its design is not analysed in a separate paragraph. Such a crucial function of a scheme should have been fully analysed

3.5.2 Remedies for the problems found

We believe that RFID protocols that rely on non-standard crypto-primitives is exceedingly difficult to create in a manner that it sufficiently resists attacks. Therefore, at least in the short term, it is better to use a well-analysed cryptographic function such as AES [7], DES [16], Grain [10] etc. In the long term, with sufficient time to create, analyse, and re-create such non-standard crypto-primitives, their hardness will rise to the same level as currently accepted cryto-primitives, thus allowing their implementation in RFIDs.

Given our reasoning above, the best way to mend the private identification part of the protocol in the short term is to use a standard cryptographic primitive such as hash-chains as in [18], or key-trees as in [15]. Both of these schemes and their improved versions are detailed in Sect. 2.2.3 and 2.2.4, respectively. Authentication can be then added in multiple ways, as detailed in Sect. 2.3. A particularly appropriate choice would be the use of a key-tree protocol as these allow for both authentication and private identification.

3.6 Conclusions

During our analysis of the Molva-Di Pietro scheme we have uncovered multiple flaws, among them tag identification ambiguity, possibility of active attack and unanticipated processing slowness. We have fully analysed the scheme from multiple viewpoints and have found a multitude of obscure features such as pair-equivalences. We presented a detailed list of design flaws that we have found during our analysis and finally, we have given a set of improvement ideas.

In parallel to the publication of the contents of this chapter in [25], another attack by Deursen et al. have emerged [27]. This new attack is a fully passive algebraic attack that requires very little hardware effort to break the key $K_{i,j}$ to the keq-equivalence level using only $2l/q$ communications between a reader and a tag. Since the publication of the contents of this chapter, the designers of the protocol have updated their scheme into the Ff family of protocols [3]. This new version of the protocol counters the shortcomings and attacks demonstrated in this chapter.

The aim of this chapter was to give an example for an ad-hoc protocol that seems to be secure on the surface but if fully analysed, reveleals serious weaknesses. The weaknesses stemmed from the authors overlooking certain attack methods, namely the active attack detailed in this chapter, and the passive attack detailed in [27]. These shortcomings are in no way unique to this ad-hoc protocol: as we shall see in the next chapter, the author of this analysis himself has commited similar mistakes when designing an ad-hoc protocol.

Chapter 4

Secret Shuffling

In this chapter we demonstrate a private identification protocol for very cheap RFID tags. The target of this scheme was to require almost no computation, but to be secure the tag's privacy against moderate attackers. The security of this scheme, called the Probabilistic Identification Protocol (ProbIP), relied on the hardness of a randomly generated NP-complete decision problem. However, after the scheme was published by Castelluccia and Soos [5] it was shown to be insecure by Ouafi et al. [19].

Organisation

We describe the probabilistic identification protocol (ProbIP) in Sect. 4.1. Then, in Sect. 4.2 we describe the attack by Ouafi et al. [19]. Finally, in Sect. 4.3 we conclude this Chapter.

4.1 Probabilistic Identification Protocol

In this section, we describe the Probabilistic Identification Protocol (ProbIP), and provide some implementation details. In ProbIP, each tag $\mathcal{T}_j$ is configured with a unique K-bit long random secret key, k_j. The key is used as a bit-vector, with $k_j[1]$ being the first bit, $k_j[2]$ being the second, etc. The back-end server, $\mathcal{B}$, stores all the keys that are assigned to each of the n tags.

4.1.1 Protocol description

The protocol, between tag $\mathcal{T}_j$, the reader, and the backend $\mathcal{B}$ is as follows:

1. The reader initiates the identification by broadcasting a `HELLO` message
2. Upon reception of a `HELLO` message, $\mathcal{T}_j$ replies with P *packets* and a `FINISHED` message, where P is a system parameter that will be defined in Sect. 5.1.1. A *packet* is a list of $2L$ values, $<a_1, b_1>, <a_2, b_2> \ldots, <a_L, b_L>$, where a_i is a random key index $a_i \overset{r}{\leftarrow} [1, K]$ that is never repeated in the same packet, and b_i is a random bit $b_i \overset{r}{\leftarrow} \{0, 1\}$ such that the equation

$$\sum_{i=1}^{L} \Big[k_j[a_i] \oplus b_i \Big] = L/2 \tag{4.1}$$

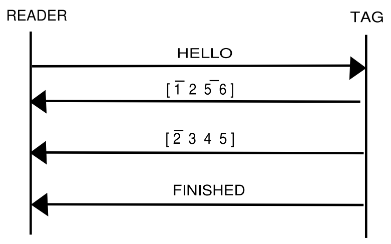
is satisfied, where L is even. Since addition is commutative, as long as the pairs $<a_i, b_i>$ are not changed, the order of the pairs can change. We note these pairs in the following fashion: a_i if $b_i = 1$ and $\neg a_i$ if $b_i = 0$.

3. Upon reception of the packets and the `FINISHED` message, the reader sends the packets it received to $\mathcal{B}$, which computes the result of eq. (4.1) for each packet for every tag's key. The key that fits all the packets is suspected to have been used to send the packets. This information is then relayed back to the reader.

4.1.2 Example protocol run

To illustrate the protocol, let us consider a system that uses the artificially small system parameters $L = 4$, $K = 6$ and $n = 4$. In this example, $\mathcal{T}_1$ is configured with the key k_1 =`011001`, $\mathcal{T}_2$ with the key k_2 =`100101`, $\mathcal{T}_3$ with the key k_3 =`011110` and finally $\mathcal{T}_4$ with k_4 =`001110`.

Let us assume the reader $\mathcal{R}$ is trying to identify tag $\mathcal{T}_2$. An example protocol run between $\mathcal{R}$ and $\mathcal{T}_2$ is the following:



In a step-by-step fashion, the following happens during this protocol run:

1. $\mathcal{R}$ broadcasts a `HELLO` message.

2. Tag $\mathcal{T}_2$ sends two packets and the `FINISHED` message. The first packet is defined by [$\bar{1}$ 2 $\bar{5}$ 6], for which eq. (4.1) wrt. k_2 is $(1\oplus 1)+(0\oplus 0)+(0\oplus 1)+(1\oplus 0) = 2 = L/2$. The second packet is defined by [$\bar{2}$ 3 4 5] for which eq. (4.1) wrt. k_2 is $(0 \oplus 1) + (0 \oplus 0) + (1 \oplus 0) + (0 \oplus 0) = 2 = L/2$.

3. Upon reception of the first packet, the reader computes for each of the 4 tags the eq. (4.1). $\mathcal{R}$ gets that for $\mathcal{T}_1$ it is 4, for $\mathcal{T}_2$ it is 2, for $\mathcal{T}_3$ it is 2 and for $\mathcal{T}_4$ it is 1. The reader, therefore, keeps only tags $\mathcal{T}_2$ and $\mathcal{T}_3$ as possible candidates.

4. Upon reception of the second packet, the reader computes for tags $\mathcal{T}_2$ and $\mathcal{T}_3$ the eq. (4.1). $\mathcal{R}$ gets that for $\mathcal{T}_2$ it is 2 and for $\mathcal{T}_3$ it is 3. At this point, tag $\mathcal{T}_2$ has been successfully identified by $\mathcal{R}$.

4.1.3 Minimum number of packets needed by the reader

Let us compute the minimum amount of packets needed by $\mathcal{R}$ to correctly identify a tag. Since the protocol is probabilistic, there is always a non-zero probability fp that the number of packets sent is not enough for the reader to find the tag sending the packets. However, this probability can be adjusted between $0 < fp < 1$, with the penalty that the closer fp gets to 1, the more packets need to be sent.

The total number of packets possible for *all* keys is $\binom{2K}{L}$, as a_i comes from a set of size K and b_i comes from a set of size 2, whereas for a *given* key, the number of possible packets is only $\binom{K}{L/2}\binom{K-L/2}{L/2}$ since eq. (4.1) must hold and indices cannot be repeated in a packet. The ratio of these two numbers

$$R = \frac{\binom{K}{L/2}\binom{K-L/2}{L/2}}{\binom{2K}{L}} \tag{4.2}$$

is the probability that a random packet is valid for a random tag. As an example, for $K = 400, L = 10,\ R \approx 0.232$.

Given n tags, the false positive probability, fp, that p packets generated by a given tag match another tag's key can be calculated as $fp = n * R^p$. The number of packets sent from the tag to the reader should then be

$$P = \left\lceil \frac{\log(1/n * fp)}{\log(R)} \right\rceil \tag{4.3}$$

which is, for the example parameters of $L = 10$, $fp = 0.1$ and $n = 10^7$, $P = \lceil 12.62 \rceil = 13$. If these packets do not suffice (which has a low chance of happening), repeated identification attempts are carried out by the reader until it finds the correct tag.

4.1.4 Parameters

The parameter K must be at least $\lceil \log_2(n) \rceil$ bits, but as the security of the system relies on the condition that $n \ll 2^K$, the larger this parameter, the more secure the system. Also, K should be at least an order of magnitude larger than L. The parameter L must be such that $L/2$ is an integer. When deciding the parameters, the number of bits sent in one identification

$$B = P * L * (\lceil \log_2(K) \rceil + 1) \tag{4.4}$$

which is also the minimum amount of random bits that need to be generated during an identification, must be kept in mind. The parameters L, K and n all influence this number. As an example, for $K = 400, L = 10$ and $n = 10^7$, $B = 1300$ bits. It is important to note that sending this information is just a fraction of a second given a 52.969 kb/s label-to-interrogator link in Class 1 EPC tags [9].

4.1.5 Implementation of the protocol in the backend

This subsection describes the algorithm used by the backend, $\mathcal{B}$, to identify the tag using the packets it sent. We first refer to $\mathcal{B}$ as a single entity and later describe how the load can be distributed among multiple entities. It is assumed that $\mathcal{B}$ knows the keys of all the tags in the system $(k_1 \dots k_n)$. These keys are stored in a 'column-like' order $k_1[i], k_2[i], \dots k_n[i]$ for all $i \in K$. We call these columns $Col_1, \dots Col_K$, where Col_1 stores the first bit of all tags' keys.

At each protocol instance, the following is executed by $\mathcal{B}$:

1. $\mathcal{B}$ fills with zeros a temporary n-long byte-vector *temp*. This will store the result of the eq. (4.1) for each tag.

2. $\mathcal{B}$ performs the following for each of the packets' L pairs <a_i, b_i>: For each tag $\mathcal{T}_j$ in the system, *temp*$[j]$ is incremented by one if $Col_{a_i}[j] \oplus b_i = 1$. Iteration through the *temp* and the Col_{a_i} can be parallel, so for a given index, this requires $3n$ processing steps.

3. Once all the pairs in the packet have been considered, all tags $\mathcal{T}_j$ for which *temp*$[j] = L/2$ could have sent the packet.

4. Steps 2-3 are repeated for all packets with different *temp*-s, i.e. $temp_1$ for packet 1, $temp_2$ for packet 2, etc.

5. The identified tag is $\mathcal{T}_j$ for which $temp_p[j] = L/2$ for all $p \in [1, P]$.

In essence, the algorithm computes, using a non-strict notation, $temp_p[j] := [packet_p][k_j]$ for every tag's key and every packet. Then it checks if there exists a $\mathcal{T}_j$ for which $temp_p[j] = L/2$ for all p. The amount of data that needs to be stored in memory by the algorithm is

$$(K * n)/8 + P * n \text{ bytes}$$

which is essentially the keys and some insignificant amount of inter-algorithmic data. The number of processing steps required is

$$P * L * 3n + P * n \tag{4.5}$$

i.e. if P was independent of n, it would be linear in n. However, due to P being dependent on n, it is actually $O(n \log n)$ (see Sect. 5.1.1. for details). As an example, if $K = 400$, $L = 10$, $n = 10^7$, $P = 13$[i] then the overall memory requirement is $(400 * 10^7)/8 + 13 * 10^7$ bytes$\approx$ 600MB and the overall processing requirement is $13 * 10^7 * (3 * 10 + 1) \approx 4e9$ processing steps. Parallelisation of this algorithm is simple: for instance, each packet can be sent to a different server, and an aggregation server can be used to compute the final result (step 5). This would bring down both the memory and processing requirement of individual computers or processor cores.

The algorithm was implemented on a Xeon E5345@2.33GHz in C++. The implemented algorithm used exactly as much memory as predicted and although full assembly-level optimisation and threading was not carried out, it was able to identify a tag in 1.2 s with parameters $L = 10, K = 400, n = 10^7$. The required backend speed can be calculated given single-tag identification processing requirements and the maximum number of identification sessions per second. The required speed can then be achieved using multiple computers or FPGAs with parallelisation. The cost of such a backend should not be a problem given that simply buying 10^7 tags costs about $1 million.

It is important to note that an adversary does not know the configured set of keys, and would need to run this algorithm with $n = 2^K$, which would result in an impossibly large processing requirement.

[i]See Sect. 5.1.1 why $P = 13$

4.1.6 Implementation of the protocol in the tag

To send the packets, the tag needs to run an algorithm that generates packets according to eq. (4.1). Such an algorithm is trivial to implement once a source of random numbers is available. In generating good random numbers, the inherent wireless nature of RFID tags is to our help: tags sense their environment's electromagnetic and temperature-fluctuations and can use it to harvest entropy. Assuming the attacker cannot steal from and imperceptibly return tags to their legitimate owners, tags are rarely in an environment where the attacker can control all parameters, and even if so, the required equipment would be very specialised and costly.

Designing a relatively secure pseudo random number generator (PRNG) is not the goal of this protocol description, but we state some design directions along which such PRNGs could be implemented. Silicon-based Physically Unclonable Functions (PUFs) [4] are ideal for random number generation [17] on RFIDs as per its description in Sect. 2.3.5. In [4] the authors claim to have implemented such a PUF in about 545 gate equivalents. Initial SRAM-states are also exploitable as described in [11], and they could be used as seed or as extra entropy added to the output of the PUF. As a security measure, readers could also offer entropy to tags on each reading, which the tag could XOR into its PRNG's seed thus only increasing its entropy, even if the reader is controlled by the attacker. Although it is hard to know the precise gate-count needed for such a PRNG, we use a conservative estimate of 700 gate equivalents.

Even though the presented ideas for random number generation do not offer suitable safety for high-security applications, we believe they are adequate in the case of RFID tags, where a sufficiently powerful and determined attacker can in any case break the privacy of a tag by simply opening it and examining its contents. The goal is not to completely secure the tag (since it is impossible, at least with current schemes using similar gate-counts), the goal is to defend it to such an extent that it is exceedingly difficult and economically non-viable to break its privacy.

Implementation complexity

The ROM needed to store a 400-bit key is only 400 NOT gates, some of which are blown during tag personalisation to configure the unique tag key. A simple packet-generation algorithm needs about 100 and the PRNG is expected to need at most 700 gate equivalents, which gives a total of about 1200 gate equivalents.

4.2 The attack by Ouafi et al.

Ouafi et al. [19] have broken the security of the ProbIP protocol by using Gaussian elimination on the packets. The main idea of the attack is that a set of packets can be represented as

$$\sum_{i=1}^{L} v_i^1(K[i] \oplus b_i^1) = L/2$$
$$\sum_{i=1}^{L} v_i^2(K[i] \oplus b_i^2) = L/2$$
$$\vdots$$
$$\sum_{i=1}^{L} v_i^l(K[i] \oplus b_i^l) = L/2$$

where l is the number of packets gathered by the attacker, v is the indicator function whether a given key bit is in the packet, and b_i is the random bit in the packet associated with each key index. The authors observe that these equations are solveable using Gaussian elimination once enough packets are collected, which is calculated as P in the previous section. Gaussian elimination only requires about $7 \cdot m^{log_2 7}$ operations using Strassen's algorithm [26] where m is the size of the matrix. Therefore, for the suggested parameter of $K = 400$, solving the system of equations takes at most 2^{24} operations — which takes negligible time on a modern computer.

As a counter-measure against the presented attack, Ouafi et al. propose to update the key at each identification. Such a counter-measure however, would bring about the same problems that plagues the OSK protocol [13]: denial of service and the need of a hash function to update the internal state. Further details on these problems can be found in Sect. 2.2.3 where the OSK protocol family is discussed.

4.3 Conclusions

In this chapter we have described the ProbIP protocol published by Castelluccia and Soos and demonstrated the attack against it by Ouafi et al. In hindsight we can say that the security of the scheme was over-evaluated because the original security analysis was through SAT solvers, which are very slow to execute Gaussian elimination, leading to the authors overlooking this important avenue of attack. However, partially due to this attack, we were encouraged to implement Gaussian elimination into SAT solvers, a description of which is present in Chapter 7. With this important addition to SAT solvers, a similar mistake could be avoided, though a more sophisticated attack than the use of SAT solvers could possibly be found against many schemes. Such a sophisticated attack could, for example, use out some hidden statistical weaknesses, or other, less apparent internal symmetries of the scheme, thereby circumventing the original hard problem the scheme's security was based upon.

The aim of this chapter was to give another example, besides that of the Di Pietro-Molva protocol present in the previous chapter, that ad-hoc protocols are notoriously difficult to design. To break the security of ProbIP, it sufficed for the authors of [19] to simply look at the description of the protocol from another angle (namely, from an algebraic cryptanalysis point of view) to discover its flaws.

Chapter 5

Noisy Secret Shuffling

In this chapter, we describe an improvement to the ProbIP scheme, called Enhanced Probabilistic Identification Protocol (EProbIP). EProbIP adds noise to the original scheme, thereby changing the underlying problem to an NP-hard optimisation problem, strengthening the protocol against attacks in general, and against the attack by Ouafi et al. [19] in particular. We analyse this improved protocol's security using a satisfiability solver that we have modified for this purpose: we devised a way to modify a modern satisfiability solver to solve certain maximum satisfiability problems faster, which we also consider an important contribution of this chapter.

With EProbIP we wish to demonstrate that even very small RFID tags with almost no computation capabilities can still provide some level of privacy protection. By moderate attackers we mean that the attackers have limited access to tags and they are using off-the-shelf equipment. Providing a scheme that is secure against very powerful attackers is, probably, impossible without reasonable processing power and/or large memory. However, we show that *reasonable* security is feasible with little memory and almost no computation. We believe that this level of security is enough for most limited value items (e.g. bottles, fruits) tagged with RFIDs.

Organisation

We describe the enhanced version of the ProbIP, EProbIP, in Sect. 5.1. Next, in Sect. 5.2 provide a security analysis of this enhanced protocol. Finally, in Sect. 5.3 we draw some conclusions.

5.1 Error-introducing ProbIP

In this section we introduce the Error-introducing Probabilistic Identification Protocol (EProbIP), which is similar to ProbIP (see Chapter 4), but the tag introduces some completely random packets into the communication, which changes the attained security level significantly. Conceptually, this is because the reader is less distracted by the erroneous packets, since it knows that there are only a very few keys (at most the no. of tags in the system, n) that could or could not fit the packet. An attacker, on the other hand, is required to consider all $2^K \gg n$ possible keys.

The completely random packets, which may or may not conform to eq. (4.1) and which do not have any repeated indexes are called **Noise**-packets. Normal packets, i.e. packets deliberately conforming to e.q. (4.1) and not having any repeated indexes are called **Valid**-packets. **Valid**- and **Noise**-packets have a Bernoulli distribution of $P(\text{packet is } \textbf{Noise}\text{-packet}) = err$. This change in the scheme overcomes the attack by Ouafi et al., and at the same time keeps the tag implementation footprint low and does not force the backend do significantly more calculations than before.

5.1.1 Minimum number of packets needed by the backend server

In this subsection, we compute the minimum amount of packets needed by backend $\mathcal{B}$ to correctly identify the tag $\mathcal{T}_S$ that is sending the packets. Since the protocol is probabilistic, there is always an adjustable probability $fp > 0$ that the number of packets sent is not enough.

To distinguish $\mathcal{T}_S$ (the tag sending the packets) from the rest, the backend server ranks the tags according to how many packets received satisfy eq. (4.1) wrt. the key of a tag — the more packets satisfy the equation wrt. the key of a tag, the higher the tag's rank. The tag that has the highest ranking is suspected to be $\mathcal{T}_S$. Therefore, we need to calculate the chance fp that any tag is ranked at the same or higher level than $\mathcal{T}_S$. To calculate this, we first calculate the probability R that a **Valid**-packet fits a tag other than $\mathcal{T}_S$. Then we calculate the chances of a tag ranked at a certain rank or higher.

Let us define packet-space $\mathbb{P}$ as all the different **Noise**-packets possible. The size of this space is $\binom{K}{L}2^L$, since in a packet there are L unordered pairs $<a_i, b_i>$, where a_i comes from a set of size K and cannot be repeated, and b_i comes from a set of size 2.

A certain tag emitting **Valid**-packets can only generate a subset $\mathbb{T}$ of the packet-space $\mathbb{P}$. The size of $\mathbb{T}$ can be calculated as follows. Let us group each possible $<a_i, b_i>$ pair into two K-sized sets: in set 0 $a_i \oplus b_i = 0$ and in set 1 $a_i \oplus b_i = 1$. To generate a **Valid**-packet, $L/2$ elements must be selected from each set, making sure no indexes are repeated, so $|\mathbb{T}|=\binom{K}{L/2}\binom{K-L/2}{L/2}$.

Since $\mathbb{T}$ is evenly distributed in $\mathbb{P}$, the ratio $R = \frac{\mathbb{T}}{\mathbb{P}}$, or

$$R = \frac{\binom{K}{L/2}\binom{K-L/2}{L/2}}{\binom{K}{L}2^L}$$

is the probability that a **Noise**-packet by $\mathcal{T}_S$ conforms to eq. (4.1) of a tag. Since $\frac{\mathbb{T}}{\mathbb{P}}$ is relatively large and there are many tags in the system (all with different $\mathbb{T}$-s), the probability that a **Valid**-packet by $\mathcal{T}_S$ conforms to eq. (4.1) of a tag can also be approximated as being R. As an example, for $K = 400, L = 10$, $R = 0.246$.

Let us note the binomial distribution's probability mass function as $\mathtt{Bi}(k, n, p)$ which gives the probability that out of n random events each occurring with probability p, what is the chance that exactly k occurs. Furthermore, let us note the binomial distribution's cumulative distribution function as $\mathtt{BiC}(k, n, p)$ which gives the probability that out of n random events each occurring with probability p, what is the chance that at most k occurs.

The probability that out of P packets sent by $\mathcal{T}_S$, exactly x conform to eq. (4.1) is

$$\texttt{fit\_send}(P,x) = \sum_{i=0}^{x} \Big[\texttt{Bi}(P, x-i, 1-err) * \texttt{Bi}(P-(x-i), i, R)\Big]$$

since **Noise**-packets can also (by chance) conform to eq. (4.1). The probability that at least one tag out of the incorrect $n-1$ tags is ranked at level lev or higher (i.e. at least lev packets conform to eq. (4.1) according to its key) given P packets is

$$\texttt{probrank}(P, lev) = 1 - \Big(\texttt{BiC}(P, lev-1, R)\Big)^{n-1}$$

Therefore the chance that there is any tag ranked at the same or higher level than $\mathcal{T}_S$ given that $\mathcal{T}_S$ sends P packets is:

$$fp := \sum_{i=0}^{P} \Big[\texttt{probrank}(P,i) * \texttt{fit\_send}(P,i)\Big]$$

As an example, for the parameters $K = 400$, $L = 10$, $P = 20$ and $err = 0.1$, $fp = 0.086$. In other words, there is about 91% chance that if a tag sends 20 packets during identification, it will be ranked highest by $\mathcal{B}$ and so correctly identified. As a comparison, in case of the ProbIP protocol (i.e. $err = 0$) for a similar fp only 13 packets sufficed.

From a practical point of view, we note that for any reasonable fp, the err must be limited to around < 0.2 otherwise P starts to get very large and the protocol becomes impractical to use.

5.1.2 Modified backend server and tag implementations

The algorithm executed by the backend is essentially the same as for ProbIP, presented in Sect. 4.1.5. The only difference is that the $temp_1 \ldots temp_P$ blocks need to be processed differently: in step 5. each tag is now given a rank based on how many packets fit on the key of the tag. The tag that has the highest rank is then suspected of sending the packets. There is no real extra memory need to do this, and the processing is only increased by $P \cdot n$ steps to

$$3n \cdot P \cdot L + 2n \cdot P$$

instead of eq. (4.5). As was explained in the previous subsection, for parameters $K = 400$, $L = 10$, $n = 10^7$, $err = 0.1$, P must be 20 (compared to 13 for the ProbIP), and so our reference algorithm implementation identified one tag amongst 10^7 in 2.6 s instead of 1.2 s. For 10^6 tags, the time required shrank to 0.2 s.

The difference between ProbIP's and EProbIP's tag implementation is an invocation of the PRNG before packet-generation. For instance, given that $err = 0.1$, if `PRNG()` $\bmod 10 = 0$ the tag sends a packet filled with data from the PRNG output (making sure no indexes are repeated), otherwise it sends a packet as usual.

5.1.3 Integration of EProbIP into the EPC standard

The EPC standard [9] requires the unique identifier, the EPC code, to be sent after the singulation of the tag as is present in Fig. 1.2 in Chapter 1. In order to protect the tag from loosing its anonymity at a lower level than the level where the EProbIP protocol runs, the EPC singulation protocol must be changed. Therefore, when implementing the EProbIP protocol into an EPC tag, the EPC code of the tag must be changed to a fix value for all tags. An example such EPC code could be the all-zero ID, or if that is reserved by the EPC consortium, then any other fixed value. This way, all tags that implement the EProbIP protocol will belong to an anonymity group that can be distinguished from other tags through their fixed EPC code, but cannot be distinguished from one another, other than through breaking the security of the EProbIP protocol.

5.2 Security analysis of EProbIP

In this section we evaluate the security of the EProbIP scheme using the "strong privacy" model proposed by Juels and Weis in [12]. In EProbIP, *tags' keys are completely independent of each other* thus the corruption of one tag does not affect the security of the rest of the system. Therefore, it is useless for adversary $\mathcal{A}$ to use the SetKey procedure to change the key of tags. It is also useless for $\mathcal{A}$ to examine any other tags than the ones it will pick, i.e. $\mathcal{T}_A$ and $\mathcal{T}_B$. In EProbIP, ReaderInit is a simple fixed `HELLO` message, so it need not be executed by $\mathcal{A}$ at all. Therefore, in view of properties of EProbIP, the privacy experiment of the model can be refined to what is present in Fig. 5.1.

5.2.1 Attack vectors

In order to assess the security level offered by the protocol, a thorough analysis of the **Valid**-packets is needed, since they are the sole source of useful information emitted by the tag. For clarity, let us first define some technical terms used in satisfiability research. SAT is a shorthand for 'satisfiability' or 'satisfiable'. A literal is either a variable or its negation, i.e. a or $\neg a$. A clause is a disjunction (`or`-ing) of literals. CNF, Conjunctive Normal Form, is a conjunction (`and`-ing) of clauses. A SAT-solver is a program that tries to find a solution to a problem described as a CNF – if it fails, the result is UNSAT, if it succeeds, the result is SAT. Pseudo-Boolean or simply PB constraints have the form $\sum_i (a_i * l_i) \geq k$, where $a_i, k \in \mathbb{R}$ and l_i is a literal. The L/2-in-L LSAT problem's input instance is a collection of L-set of literals and the problem is to decide whether there exists an assignment of variables such that in each set exactly $L/2$ literals are true and $L/2$ literals are false.

Every **Valid**-packet,as its definition in eq. (4.1) suggests, can be converted to two PB constraints: e.g. if $K = 10$, $L = 4$, the packet [7 2 8 $\neg$9] becomes:

$$v_7 + v_2 + v_8 + \neg v_9 \geq 2$$
$$v_7 + v_2 + v_8 + \neg v_9 \leq 2$$

so, PB solvers are one of the possible methods for attacking the privacy of tags.

Experiment $\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}}[K, n, x_A + x_B + x_C]$:

Setup:

(1) Generate keys $(k_1, \ldots, k_n)$ uniquely and randomly with GenKey

(2) Initialize $\mathcal{R}$ with keys $(k_1, \ldots, k_n)$

(3) Set each $\mathcal{T}_i$'s key k_i with a SetKey call

Phase 1 (Learning):

(4) Let $\mathcal{A}$ perform x_A TagInit calls with $\mathcal{T}_A$ and let it record the received packets into the set X_A

(5) Let $\mathcal{A}$ perform x_B TagInit calls with $\mathcal{T}_B$ and let it record the received packets into the set X_B

Phase 2 (Challenge):

(6) Let $\mathcal{T}_C \xleftarrow{r} \{\mathcal{T}_A, \mathcal{T}_B\}$

(7) Let $\mathcal{A}$ perform x_C TagInit calls with $\mathcal{T}_C$ and let it record the received packets into the set X_C

(8) Let $\mathcal{A}$ perform calculations on the recorded packets in order to make an educated guess whether $\mathcal{T}_C = \mathcal{T}_A$ or $\mathcal{T}_C = \mathcal{T}_B$.

Exp succeeds if $\mathcal{A}$ guessed $\mathcal{T}_C$ correctly

Figure 5.1: The privacy experiment as proposed by Juels and Weis in [12], refined to the specifics of EProbIP

Looking at packets in another way, every **Valid**-packet is made up of an set of L literals (where one pair $<a_i, b_i>$ describes a literal) and as eq. (4.1) implies, half of the literals must be true and half of them must be false. Therefore, **Valid**-packets represent an $L/2$-in-L LSAT problem, which is NP-hard if $L > 2$ as indicated by Schaefer's dichotomy theorem [23]. Consequently, when emitting only **Valid**-packets, the tags are generating a random $L/2$-in-L LSAT problem on-the-fly and so breaking their privacy would entail finding a solution to an NP-complete decision problem. Since tags are also emitting **Noise**-packets, the attacker is faced with an optimisation version of the original problem.

The analysis of the packets imply that the only tools able to break the scheme are either SAT solvers, Pseudo-Boolean solvers, or specialised methods that are a combination of these methods tailored to the needs of this specific case. The original ProbIP protocol's best attack vector was through Gaussian elimination. For EProbIP, adding the packets received as rows to a matrix (as is present in Sect. 4.2) and solving using Gaussian elimination leads to no results, as the resulting linear system of equations has no solutions. This is because the **Noise**-packets, which are impossible to filter out by the attacker, are also be present in the matrix, but the key of the tag has an exceedingly high chance not to fit the **Noise**-packets.

In this section, we present two methods to break the privacy of the protocol: a computationally-intensive approach, which uses a traditional PB solver and a packet-intensive approach, which uses a custom-built solver. Neither of the approaches need the observation of both $\mathcal{T}_A$ and $\mathcal{T}_B$ in Phase 1 of the privacy experiment, we therefore picked $\mathcal{T}_A$ to observe in Phase 1. In short, the two approaches are the following:

Computationally intensive approach

Execute a PB solver on the packets received from $\mathcal{T}_A$ and $\mathcal{T}_C$ and try to find a solution that satisfies the most packets – if the solution found satisfies less than a certain threshold Th number of packets, then $\mathcal{T}_C \neq \mathcal{T}_A$ (consequently, $\mathcal{T}_C = \mathcal{T}_B$). Otherwise, $\mathcal{T}_C = \mathcal{T}_A$. This is because, if $\mathcal{T}_C = \mathcal{T}_B$ we are expecting that the solver finds the key of the tag and so only a very low rate ($\approx err$) of packets' equations will not be satisfied. However, if $\mathcal{T}_C \neq \mathcal{T}_A$ much more will be unsatisfied by the best solution found, since the keys of the two tags are different.

This approach is a straight-forward approach. However, since the solver needs to find a solution that satisfies the most equations generated from the packets, the problem is an optimisation problem, and belongs to the MAX-SNP-hard class, so it is very difficult to solve. Therefore, the processing for this approach for any reasonable set of parameters is extremely high. A detailed description of the parameters used in this approach is in Sect. 5.2.2.

Packet-intensive approach

This approach exploits the property of the scheme that as soon as there are enough packets, guessing only a relatively small number of K's bits, multiple packets will indicate other bits that must be set, these new bits will indicate further bits, and eventually all bits of the key are recovered. The multiplicity of packets that indicate other bits to be set is what makes this attack possible, as there is only a minor probability that all these packets are **Noise**-packets. In essence, this is an algorithm that takes as an input more packets than would be necessary, and makes decisions based on multiple packets.

5.2.2 Computationally-intensive approach

In this subsection, we compute the parameters needed to win the privacy experiment using the computationally-intensive approach. In a nutshell, the computationally-intensive approach is the following: execute a PB solver on the constraint set defined by $X_A \cup X_C$ and give an objective function to the solver such that the number of packets not satisfied should be minimised. If the solution found satisfies less than a certain threshold Th number of packets, then $\mathcal{T}_C \neq \mathcal{T}_A$ (consequently, $\mathcal{T}_C = \mathcal{T}_B$). Otherwise, $\mathcal{T}_C = \mathcal{T}_A$.

We now determine the value x_A, x_B and the threshold Th for which the chance of the attacker to correctly identify whether $\mathcal{T}_C = \mathcal{T}_A$ or not is a chosen probability $Prob_{att} < 1$. Given there are x **Valid**-packets relating to $\mathcal{T}_A$'s key, y **Valid**-packets relating to $\mathcal{T}_C$'s key, and z **Noise**-packets, the chance that the maximum number of satisfiable packets is less than Th ($Th \leq x + y + z$) can be approximated as

$$\texttt{rel}(x, y, z) = \max \begin{cases} \Big[\texttt{BiC}(x + y + z, Th - 1, r)\Big]^{n} \\ \Big[\texttt{BiC}(x + \min(x, y), Th- \\ \max(x, y), r)\Big]^{1+(n-1)*r^{\max(x,y)}} \end{cases}$$

if $Th > \max(x, y)$ and $Th \leq x + y + z$. Otherwise, if $Th > x + y + z$ it is 1 and if $Th \leq \max(x, y)$ it is 0. Therefore, the chance that the maximum number of satisfiable packets is more or equal to Th is $1 - rel(x, y, z)$.

Using function $\texttt{rel}(x, y, z)$, the chance that if x_A packets are sent by $\mathcal{T}_A$, and x_C by $\mathcal{T}_C$, where $\mathcal{T}_A \neq \mathcal{T}_C$, the maximum number of satisfiable packets is less than Th is

$$\texttt{diff\_less}(x_A, x_C) = \sum_{i=0}^{x_A} \sum_{j=0}^{x_C} \Big[\texttt{Bi}(x_A, i, 1 - err)* \\ \texttt{Bi}(x_C, j, 1 - err) * \texttt{rel}(i, j, (x_A - i) + (x_C - j))\Big]$$

Similarly, the chance that if x_A packets are sent by $\mathcal{T}_A$, and x_C by $\mathcal{T}_C$, where $\mathcal{T}_A = \mathcal{T}_C$, the maximum number of satisfiable packets is more or equal to Th is

$$\texttt{same\_moreeq}(x_A, x_C) = \\ \sum_{i=0}^{x_A + x_C} \Big[\texttt{Bi}(x_A + x_C, i, 1 - err) * (1 - \texttt{rel}(i, 0, x_A + x_C - i))\Big]$$

The goal of the attacker is then to select a number of packets, P_{att}, a corresponding threshold Th, and a distribution of P_{att} packets between x_A and x_C such that

$$\begin{aligned} Prob_{att} &\approx \texttt{diff\_less}(x_A, x_C) \\ &\approx \texttt{same\_moreeq}(x_A, x_C) \end{aligned}$$

For example, using the parameters $K = 400, L = 10, err = 0.1, Th = 141$ and $P_{att} = 266$ distributed evenly between $\mathcal{T}_A$ and $\mathcal{T}_C$ gives a $Prob_{att} \approx 90\%$ chance for the attacker to identify whether $\mathcal{T}_A = \mathcal{T}_C$ or not. This is achieved by running a PB solver on the constraints defined by the packets gathered, and counting how many packets are satisfied by the solution: if more or equal to 241 are satisfied, $\mathcal{A}$ can be 90% sure that $\mathcal{T}_A = \mathcal{T}_C$, otherwise she can be 99% sure that $\mathcal{T}_A \neq \mathcal{T}_C$.

5.2.3 Packet-intensive approach

In a nutshell, the packet-intensive approach takes as input many more packets than would be necessary to win the privacy experiment, and uses out the fact that the majority of the observed packets are **Valid**-packets. The algorithm guesses some bits to all combinations and observes what these guesses indicate when applied to the observed packets. If multiple packets advocate that a certain bit must be set given the guesses, the algorithm sets these bits, and continues. If the final result does not satisfy most packets, the algorithm goes back and guesses the original set of bits

Function `EProbIP_break(packets)`. This function takes as an input the observed set of packets from the two tags $\mathcal{T}_A$ and $\mathcal{T}_C$ given as a challenge to the attacker, and returns whether the two tags are the same or different, winning the privacy experiment. To achieve this, the function guesses a set of key bits to a value, and tries to fill in the rest of the key bits. If none of the 2^k guess combinations give an acceptable result, the two sets of recorded packets must have come from two different tags. If there is a combination that satisfies a good enough portion of the packets, the packets must have come from the same tag, and the key of the tag is recovered.

```
   Input: packets $X_A \cup X_C$
   Output: $\mathcal{T}_A = \mathcal{T}_C$ or $\mathcal{T}_A \neq \mathcal{T}_C$
1  Pick a set of $k$ most prevalent key bits;
2  foreach combination of true-false for the picked bits do
3      picked key bits ← the selected combination;
4      while enough packets indicate that a key bit must be set to a value do
5          key bit ← indicated value;
6      end
7      if all key bits are set and the satisfied portion of packets is about $1 - err$
       then
8          return $\mathcal{T}_A = \mathcal{T}_C$;
9      end
10 end
11 return $\mathcal{T}_A \neq \mathcal{T}_C$;
```

to some other values and starts again. If no combination of guessed values satisfies about $1 - err$ portion of the packets, then the packets must have come from two different tags. But, if the algorithm finds a K that satisfies about $1 - err$ portion of the packets, than the packets have come form the same tag, and the attacker also recovers the key of the tag.

The execution of this algorithm is present in **function** `EProbIP_break`. It first guesses $k > L/2$ bits whose indices were observed the most. There are 2^k different ways to guess these bits, and so the rest of the algorithm needs to be executed at most 2^k times. The next step of the attacker is to collect all packets that have these bits set, and give a mark to all the other indexes in the packet. If a given bit has enough marks, the bit is set to the proposed value, and the algorithm scans through the packets if there are any new that have at least $L/2$ bits set. If so, the remaining indexes in the packets are again marked, and the algorithm continues in this fashion.

Implementation

To properly evaluate the complexity of this algorithm, we needed an implementation that employs all possible optimisations. We have found that the best performance is achieved using a modified SAT solver that takes into account the error rate of the packets in order to win the privacy experiment. To win the privacy experiment, the attacker executes a modified SAT solver on the constraint set defined by $X_A \cup X_C$ that can make decisions based on multiple packets to counter the **Noise**-packets. If the answer of the modified solver is unsatisfiable (UNSAT) then $\mathcal{T}_A \neq \mathcal{T}_C$. Otherwise,

$\mathcal{T}_A = \mathcal{T}_C$.

We decided to modify the MiniSat SAT solver to suit our needs. MiniSat was chosen because it is one of the fastest SAT solvers and also had its source code available. The technique by which we modified MiniSat to solve specially crafted MAX-SAT problems is generic and so can be applied to most modern conflict-driven SAT solvers for various problems.

For input conversion reasons, we first numbered the packets that were obtained during the privacy experiment. To feed the packets to the solver, we had to first convert every packet to a set of *clauses*. Once converted, every clause was marked with the packet number it came from. The conversion used did not add or remove any variables, i.e. it did not simplify the problem or made any alteration to it.

Before getting into details, the definition of some terms used in SAT solvers is necessary.Propagation occurs when all but one literal is assigned in a clause, and all assigned literals are false – in this case, the unassigned literal must be assigned to true in order to satisfy the clause. Conflict occurs when all literals are assigned to false in a clause – in this case, the clause cannot be satisfied, and one of the assignments must be undone.

MiniSat uses the DPLL algorithm [8] to find a satisfying solution to a problem given as a set of clauses. In short, the DPLL algorithm is a backtracking-based, depth-first search algorithm that tries to find a satisfying variable assignment to satisfy the given set of clauses. In more detail, the DPLL works as follows. It picks a variable, assigns it to true or false, and examines the clauses if any propagation can be made from this variable assignment. If so, it makes those propagations and checks if any new propagations can be made from the clauses given the newly assigned variables. It repeats this until there are either no more propagations that can be made, or if it encounters a conflict. If it encountered a conflict, it jumps back, assigns the variable it guessed to its opposite, and starts again. If it does not encounter a conflict, it again picks a variable, assigns it to true or false, and the algorithm continues from here. These steps are repeated until either a solution is found, i.e. the result is SAT, or the whole search-space is exhausted without any solution being found, i.e. the result is UNSAT. A more detailed description of DPLL-based SAT solvers is given in Sect. 7.1.1.

To account for multiple packets advocating the same propagations and conflicts, we modified the DPLL algorithm to what is present in **Function** `Multi-DPLL`. A short description of the modifications follows. Instead of assigning all variables immediately to the propagations to be made, the propagations are stored for the duration of a *round* (steps 3 to 3) in the set prop_votes. A round examines all the clauses that the newly assigned variables could influence to cause a propagation or a conflict. After each round, the votes for the propagations are examined (steps 3 to 3). If a propagation is suggested by more than prop_lim number of packets, the propagation is made (step 3). Otherwise, it is rejected. Therefore, propagations are made in batches. Conflicts are also only trusted during a round if more than a certain number of packets (confl_lim) are suggesting it (step 3).

While modifying the DPLL algorithm, we also introduced the notion of level-history during search: old_level and new_level. Level old_level is the number of assigned variables the last time a round was executed, or the level we jumped back to due to a conflict. Level new_level is the number of assigned variables after the batch of

Function `Multi-DPLL(`*to_examine, old_level, new_level*`)`.This is a modified version of the original DPLL algorithm's propagation&conflict decision function. In this function, both propagations and conflicts are voted for and only those are executed which have enough votes. Propagations are kept in the ballot box prop_votes and are examined in batches, while conflicts are kept in the ballot box confl_votes and are immediately executed if the vote-count is high enough.

```
Input: to_examine, old_level, new_level
Output: conflict_variable, confl_votes
repeat
    prop_votes ←0;
    confl_votes ←0;
    prop_lim ← CalcPropLim(old_level, new_level);
    confl_lim ← CalcConflLim(old_level, new_level);
    while size of to_examine > 0 do
        var ←pop variable from to_examine ;
        foreach clause that var is in do
            packet_no ←the packet this clause belongs to;
            if clause makes propagation on var then AddVote(prop_votes, var,
            sign, packet_no);
            if clause causes conflict due to var then
                AddVote(confl_votes, var, packet_no);
                if confl_votes [ var ] > confl_lim then return Conflict on var
                with votes confl_votes [ var ];
            end
        end
    end
    old_level ←new_level ;
    while size of prop_votes > 0 do
        var, votes, sign ← pop from prop_votes ;
        if votes >prop_lim then
            Bound var to inferred sign ;
            Enqueue var in to_examine ;
            new_level ++;
        end
    end
until old_level < new_level ;
```

Table 5.1: The possible trade-offs in time and calculations for breaking the privacy of the tag. The more packets the attacker can gather, the faster she can break the privacy. The parameters used were $L = 10, Prob_{att} \approx 0.9$ and $err = 0.1$. Times are calculated for a single-core Pentium D@3GHz. P_{att} refers to the minimum packets needed to break the privacy of the EProbIP protocol.

No. of packets	$K = 100$	$K = 200$	$K = 400$	$K = 1000$
$1 \cdot P_{att}$	$1.15e15$ s	$2.53e34$ s	$7.33e72$ s	$1.78e188$ s
$9 \cdot P_{att}$	$1.47e6$ s	$3.16e14$ s	$1.47e31$ s	$1.47e82$ s
$27 \cdot P_{att}$	$5.08e4$ s	$3.87e10$ s	$2.25e22$ s	$4.43e57$ s
$64 \cdot P_{att}$	$2.94e4$ s	$1.77e10$ s	$6.45e21$ s	$3.10e56$ s
$192 \cdot P_{att}$	$1.55e4$ s	$8.96e8$ s	$2.99e18$ s	$1.11e47$ s
$576 \cdot P_{att}$	$1.80e4$ s	$6.29e6$ s	$7.72e11$ s	$1.43e27$ s

propagations made or after randomly bounding a variable to either true or false. These levels are needed to assist with the calculations in the functions `CalcConflLim` and `CalcPropLim`. Since these functions are not central to the understanding of the packet-intensive approach detailed here, they are only present in Appendix 5.A.

5.2.4 Resistance to attacks

The attack resistance of the EProbIP scheme for parameters $L = 10, Prob_{att} \approx 0.90, err = 0.1$ is detailed in Table 5.1. The tests were performed on a Pentium D@3GHz with 2GB of RAM. The number of tags, n, is not directly present anywhere in the results since n only has an effect on P and therefore on identification speed and on the ratio P_{att}/P. The results for the computationally-intensive approach are presented in the first row of the table (i.e. $1 * P_{att}$), the rest of the rows are the times obtainable using the packet-intensive approach. This is because the computationally-intensive approach's performance *degrades* if more packets are given to it than necessary.

The results show that there is a trade-off between the number of packets collected and the hardness of breaking a tag's privacy (i.e. winning the privacy experiment). The more packets an attacker can collect, the easier it is for her to break the tag's privacy. This property is a direct result of the threshold phenomenon: as hypothesised by Cheeseman et al. in [6] and further explained by B. Smith in [24], all NP-hard problems exhibit a so-called phase-transition, which states that given a randomly generated NP-hard constraint satisfaction problem (CSP), there is always a point where it is the hardest to solve the generated problem, and this corresponds exactly to the point where there is a transition from SAT to UNSAT. From this point on, the difficulty of finding a solution decreases at an exponential rate, along with the possibility of having any solution at all. In our case, the randomly generated CSP may have an implanted solution (if $\mathcal{T}_A = \mathcal{T}_C$), so it may happen that it cannot become UNSAT, but the threshold phenomenon still holds — it is simply more difficult to observe, as it cannot be characterised by a simple transition from SAT to UNSAT.

In order to limit packet extraction — thus forcing the attacker to do far more calculations — a *Time-Delay Scheduler* (TDS) could be used. As explained by Van Le et al. in [14] the TDS implements a throttling mechanism on the tag by using small capacitors that power counters that must reach 0 before a new identification session is allowed.

5.3 Conclusions

We have shown that it is possible to create a privacy-preserving identification scheme for RFID tags that fits within the gate-count budget available on these low-power devices. However, it is clear that such schemes are neither easy to create, nor do they offer clear-cut security as other, more conservative designs, such as stream or block ciphers do. For instance, using the suggested parameter sizes, it is possible to attack the EProbIP protocol in about $7.72e11$ s $\approx 2^{70}$ operations if $576 \cdot 267/20 \approx 7000$ identifications' worth of packets are gathered by the attacker. Though extracting this many packets can be made very hard through, for instance, a time delay scheduler, the security of the scheme is not as clear-cut as standard crypto-primitives would offer. However, this trade-off might be one that is useful to make for RFIDs: for instance, if implementing standard crypto-primitives on the RFID would make it so expensive that it would never be used by retailers, the security that EProbIP can offer could be a worthwhile option to consider.

5.A Calculating propagation and conflict probabilities for Function `Multi-DPLL`

The probability that a **Noise**-packet contains i number of variables that are bounded, given that $\mathsf{new\_level}$ are bounded from K is

$$P_{bound}(i) = \frac{\binom{\mathsf{new\_level}}{i}\binom{K-\mathsf{new\_level}}{L-i}}{\binom{K}{L}}$$

since the packet can be divided into two parts: one that is coming from the bounded variables and the other part that is coming from the unbounded variables.

At each DPLL level, given the previous level ($\mathsf{old\_level}$) and the current level ($\mathsf{new\_level}$) in the search tree, if there are i bounded variables in the random packet, the chance that at least one is coming from the newly bounded variables and so the packet will be investigated by the DPLL loop is:

$$P_{newlybound}(i) = \frac{\binom{\mathsf{old\_level}}{i}}{\binom{\mathsf{new\_level}}{i}}$$

Therefore, the probability that a variable will be propagated by a **Noise**-packet, and that this packet will be investigated by the DPLL loop is

$$P_{prop} = \sum_{\substack{i=\max(5,10-\\(K-\mathsf{new\_level}))}}^{\min(9,\mathsf{new\_level})} \left[P_{bound}(i) * P_{newlybound}(i) * \frac{10-i}{K-\mathsf{new\_level}} \right]$$

Similarly, the probability that a variable will cause a conflict by an **Noise**-packet, and that this packet will be investigated by the DPLL loop is

$$P_{confl} = \sum_{\substack{i=\max(6,10-\\(K-\mathsf{new\_level}))}}^{\min(10,\mathsf{new\_level})} \left[P_{bound}(i) * P_{newlybound}(i) * \frac{i}{\mathsf{new\_level}} \right]$$

We will use the cumulative distribution function of the binomial distribution at one standard deviation from the mean to calculate the limits. Therefore, since the variance of the binomial distribution is $\sigma^2 = n * p * (1-p)$, we will use the formula $cumul = n * p + \sqrt{n * p * (1-p)}$.

The probability that a packet picked randomly will not conform to eq. (4.1) is $err_r = err * (1-R)$ since a **Noise**-packet has a chance R that it will conform to it (and **Valid**-packets all conform, by definition). We will thus use err_r in our calculations. Therefore, the number of packets that do not conform to eq. (4.1) is taken to be almost sure to be below

$$errpack = (x_A + x_C) * err_r + \sqrt{(x_A + x_C) * err_r * (1 - err_r)}$$

We can now calculate the number of votes needed to believe propagations/conflicts given that we need good confidence that at most this many wrong votes will occur

$$\texttt{CalcPropLim} = errpack * P_{prop} + \sqrt{errpack * P_{prop} * (1 - P_{prop})}$$

$$\texttt{CalcConflLim} = errpack * P_{confl} + \sqrt{errpack * P_{confl} * (1 - P_{confl})}$$

Bibliography

[1] Bárasz, M., Boros, B., Ligeti, P., Lója, K., and Nagy, D. Breaking LMAP. In *Conference on RFID Security — RFIDSec'07* (Malaga, Spain, July 2007), pp. 69–78.

[2] Bárász, M., Boros, B., Ligeti, P., Lója, K., and Nagy, D. A. Passive attack against the M2AP mutual authentication protocol for RFID tags. In *RFID 2007 — The First International EURASIP Workshop on RFID Technology* (September 2007).

[3] Blass, E.-O., Kurmus, A., Molva, R., Noubir, G., and Shikfa, A. The Ff-Family of Protocols for RFID-Privacy and Authentication. In *Workshop on RFID Security — RFIDSec'09* (Leuven, Belgium, July 2009).

[4] Bolotnyy, L., and Robins, G. Physically unclonable function-based security and privacy in RFID systems. In *PerCom 2007* (March 2007), IEEE, pp. 211–220.

[5] Castelluccia, C., and Soos, M. Secret shuffling: A novel approach to RFID private identification. In *RFIDSec'07* (July 2007), pp. 169–180.

[6] Cheeseman, P., Kanefsky, B., and Taylor, W. M. Where the really hard problems are. In *IJCAI-91* (1991), pp. 331–337.

[7] Daemen, J., and Rijmen, V. Rijndael/aes. In *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed. Springer, 2005.

[8] Davis, M., and Putnam, H. A computing procedure for quantification theory. *J. ACM 7*, 3 (1960), 201–215.

[9] EPCglobal. 13.56 MHz ISM band class 1 radio frequency identification tag interface specification (2003). Tech. rep., Auto-ID cetner, MIT, February 2003.

[10] Hell, M., Johansson, T., and Meier, W. Grain — a stream cipher for constrained environments. In *Proceeding of the Workshop on RFID and Lightweight Crypto* (July 2005), pp. 114–125.

[11] Holcom, D., Burleson, W., and Fu., K. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In *RFIDSec'07* (July 2007), pp. 29–40.

[12] Juels, A., and Weis, S. Defining Strong Privacy for RFID. In *International Conference on Pervasive Computing and Communications — PerCom 2007* (New York City, New York, USA, March 2007), IEEE, IEEE Computer Society Press, pp. 342–347.

[13] Koutarou, M. O., Suzuki, K., and Kinoshita, S. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop* (MIT, Massachusetts, USA, November 2003).

[14] Le, T. V., Burmester, M., and de Medeiros, B. Universally composable and forward-secure RFID authentication and authenticated key exchange. In *Proceedings of the 2nd ACM symposium on Information, Computer and Communications Security — ASIACCS'07* (New York, NY, USA, 2007), ACM, pp. 242–252.

[15] Molnar, D., and Wagner, D. Privacy and security in library RFID: issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), ACM Press, pp. 210–219.

[16] National Bureau of Standards. Data Encryption Standard, 1977.

[17] O'Donnell, C. W., Suh, G. E., and Devadas, S. PUF-based random number generation. In *MIT CSAIL CSG Technical Memo 481* (November 2004).

[18] Ohkubo, M., Suzuki, K., and Kinoshita, S. Efficient hash-chain based RFID privacy protection scheme. In *International Conference on Ubiquitous Computing — Ubicomp 2004, Workshop Privacy: Current Status and Future Directions* (Nottingham, England, September 2004).

[19] Ouafi, K., and Phan, R. C.-W. Privacy of Recent RFID Authentication Protocols. In *Information Security Practice and Experience, 4th International Conference, ISPEC 2008* (Berlin, 2008), Lecture Notes in Computer Science, Springer, pp. 263–277.

[20] Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J., and Ribagorda, A. LMAP: A real lightweight mutual authentication protocol for low-cost RFID tags. In *Proceedings of RFIDSec'06* (Graz, Austria, July 2006), Ecrypt.

[21] Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J., and Ribagorda, A. M2AP: A minimalist mutual-authentication protocol for low-cost RFID tags. In *International Conference on Ubiquitous Intelligence and Computing — UIC'06* (September 2006), vol. 4159 of *LNCS*, Springer-Verlag, pp. 912–923.

[22] Pietro, R. D., and Molva, R. Information confinement, privacy, and security in RFID systems. In *Proceedings of the 12th European Symposium On Research In Computer Security* (September 2007), pp. 187–202.

[23] Schaefer, T. J. The complexity of satisfiability problems. In *STOC'78* (1978), pp. 216–226.

[24] SMITH, B. The phase transition in constraint satisfaction problems: A CLoser look at the mushy region. In *ECAI'94* (1994).

[25] SOOS, M. Analysing the Molva and Di Pietro Private RFID Authentication Scheme. In *Workshop on RFID Security — RFIDSec'08* (Budapest, Hungary, July 2008).

[26] STRASSEN, V. Gaussian elimination is not optimal. *Numerische Mathematik 13* (1969), 354–356.

[27] VAN DEURSEN, T., MAUW, S., AND RADOMIROVIC, S. Untraceability of RFID protocols. In *WISTP* (2008), J. A. Onieva, D. Sauveron, S. Chaumette, D. Gollmann, and C. Markantonakis, Eds., vol. 5019 of *Lecture Notes in Computer Science*, Springer, pp. 1–15.

Part III

Stream ciphers in RFIDs

In the previous part of this thesis we have shown some RFID security protocols that rely on non-standard crypto-primitives. The problem with these protocols is that they require third-party analysis and often even a redesign to make them secure. This can potentially take many years, but the problem of RFIDs is current and very real. Thus, RFIDs need solutions that can solve the challenges they face now and in the near future. To provide such a solution, standard crypto-primitives such as stream ciphers could be used in RFIDs until sufficiently good ad-hoc protocols are found to replace them.

Modern crypto-primitives have been analysed, designed, broken and re-designed for half a century [49, 19, 55, 15, 10]. Therefore it seems a logical conclusion that adapting standard crypto-primitives to the environment of RFIDs brings more security than experimental protocols, at least for the foreseeable future. The NESSIE project, and its successor, the European eSTREAM [56, 2] project's hardware portfolio is one such attempt at bringing standard crypto-primitives, more particularly, stream ciphers to the RFID scene. Two of its Phase-3, latest-round candidates are Trivium [10] and Grain [32], both of which have been designed to be implemented in hardware, and both use an 80-bit key and subsequently aim at offering 80-bit security — the security level universally agreed upon by RFID researchers to be adequate for the security of RFID tags.

The goal of this part of the thesis is to further advance the field of low hardware-complexity stream ciphers through proposing their use in two example RFID protocols and through their examination using SAT solver-based techniques.

Organisation

This third part of the thesis is made up of two chapters: Chapter 6 deals with how stream ciphers can be used in the context of RFID protocols, and Chapter 7 deals with the SAT solver-based analysis of three stream ciphers for RFIDs.

Chapter 6

An example RFID security protocol using low hardware-complexity stream ciphers

In this chapter, we describe low hardware-complexity stream ciphers, why they are uniquely useful for RFIDs, and give two example protocols that use stream ciphers to achieve privacy-preserving identification and authentication for RFIDs. Low hardware-complexity stream ciphers are useful for RFIDs as they are generally much easier to design such that they use less hardware resources than other crypto-primitives such as block- or asymmetric ciphers. For this reason, there are many low hardware-complexity stream ciphers available, many of which have stood the test of time. An example set of tried and tested stream ciphers are the final candidates of the eSTREAM project's hardware portfolio: Trivium [10], Grain [32] and MICKEY [3].

Organisation

In Sect. 6.1 we first describe the general layout and design of low hardware-complexity stream ciphers. Then, in Sect. 6.2 we describe two example security protocols for RFIDs based on stream ciphers. Finally, Sect. 6.3 concludes this chapter.

6.1 Stream ciphers

A stream cipher is a symmetric cryptographic function that allows two parties to communicate privately when they share a secret key. Stream ciphers produce a stream of pseudorandom bits (the *keystream*) given a secret *key* and a public random initialisation vector (IV). This keystream is XOR-ed with a message prior to sending and again XOR-ed after receiving so that the message cannot be read while in transit.

Stream ciphers have two phases: an *initialisation phase* followed by a *keystream generation phase*. During initialisation, key and IV are typically mixed to become the initial state. During keystream generation, the internal state is updated while the keystream is being generated.

6.2 Two stream cipher-based RFID protocols

Stream ciphers are ideal to use in the keytree-based RFID protocol, introduced by Molnar and Wagner [48]. In the keytree protocol, tags are leafs in a tree, as illustrated in Fig. 2.3 in Chapter 2. The original protocol offers both privacy-preserving identification and mutual authentication.

In this section we present two different protocols based on the Molnar-Wagner protocol. The first protocol is simple and more straightforward to use, but only provides privacy-preserving identification for RFID tags. The second is somewhat more complicated, offering both privacy-preserving identification and (optionally mutual) authentication.

Both protocols' security goal is 2^{80}, but this goal is only guaranteed if the protocols are used in a typical RFID environment, where querying the tag large number of times is impossible. From the security standpoint, we mean large number of times as being more than 1 billion ($\approx 2^{30}$) times. Typical RFIDs are slow to compute stream ciphers, therefore if we take an optimistic 0.1 s identification time, querying the tag 1 billion times in a continuous manner would take

$$\frac{10^9 \cdot 0.1}{60 \cdot 60 \cdot 24 \cdot 365} \approx 3.17 \text{ years}$$

which is overly time-consuming, and opening the tag for its secret would be far easier to accomplish for an attacker.

6.2.1 A simplistic protocol

In this simplistic protocol, the original protocol is changed such that instead of sending a nonce to be encoded, the reader sends an IV to be used, and asks for the encoded ID from the tag. Thus, the protocol description changes to that present in Fig. 6.1. Since stream ciphers do not act as Pseudo-Random Functions (PRFs), the original protocol must be changed such that the tag does not reply with a mixture of its key, nonces and ID, rather, it simply encodes its ID with its key and the $IV = IV_1 \oplus IV_2$, where IV_1 is the nonce received from the reader, and IV_2 is its own nonce.

This protocol does not provide authentication like the original Molnar-Wagner protocol. This is because a malicious tag could always set its own nonce, IV_2 to be $old\_nonce \oplus IV_1$, where old_nonce is an old, observed protocol's nonce, and then re-use the old, observed reply σ. For this reason, the adapted protocol only provides privacy-preserving identification.

6.2.2 A more complex protocol

This more complex RFID protocol, based on stream ciphers and the Molnar-Wagner protocol, offers both privacy-preserving identification and (optionally mutual) authentication for the tags and readers. The difference between this protocol and the previous, more simple protocol is essentially in the way the nonces (IV_1 and IV_2) of the two entities are mixed. In the more simple protocol they were simply XOR-ed with each other. This meant that a malicious tag could always control the final outcome of the mixing operation. This is no longer made possible, as the combination

Reader $\mathcal{R}_j$		**Tag** $\mathcal{T}_i$
Generate nonce IV_1		
	$\longrightarrow IV_1$	
		Generate nonce IV_2 and calculate $\sigma = ID \oplus cipher(k, IV_1 \oplus IV_2)$
	$\longleftarrow IV_2, \sigma$	
find $(k, ID) \in L$ s.t. $ID = \sigma \oplus cipher(k, IV_1 \oplus IV_2)$		

Figure 6.1: A less powerful version of the Molnar-Wagner protocol, adapted to the use of stream ciphers. This protocol only provides privacy-preserving identification. L is the database of tags maintained by the RFID reader.

function has been replaced with the cipher function. The mixing is done as follows: the first nonce IV_1 is used as the key, and the second nonce IV_2 is used as the IV. The keystream of the cipher is then stored as M. This new M is then re-loaded into the cipher as the IV and the key k is used to generate again a keystream that is then used to mask the ID. This process is illustrated in Fig. 6.2.

6.3 Conclusions

In this chapter we have described the general layout of low hardware-complexity stream ciphers, and have adopted the Molnar-Wagner keytree-based protocol to their needs. Stream ciphers are uniquely suitable for RFIDs, as they have been researched for a long time, and can be created such as to demand very little hardware resources, essentially only requiring around twice the number of flip-flops than their security rating — for example, the Grain cipher [32] only requires 160 flip-flops to function, while providing 80-bit security.

Reader $\mathcal{R}_j$		**Tag** $\mathcal{T}_i$
Generate nonce IV_1		
	$\longrightarrow IV_1$	
		Generate nonce IV_2 and calculate $M = \mathit{cipher}(IV_1, IV_2)$ $\sigma = ID \oplus \mathit{cipher}(k, M)$
	$\longleftarrow IV_2, \sigma$	
calculate $M = \mathit{cipher}(IV_1, IV_2)$ find $(k, ID) \in L$ s.t. $ID = \sigma \oplus \mathit{cipher}(k, M)$		
	optional — only for mutual authentication	
calculate $\tau = ID \oplus \mathit{cipher}(k, M \oplus \mathtt{1})$		
	$\longrightarrow \tau$	
		check $\tau \overset{?}{=} ID \oplus \mathit{cipher}(k, M \oplus \mathtt{1})$

Figure 6.2: A more complex version of the Molnar-Wagner protocol adapted to stream ciphers. This protocol provides privacy-preserving identification and (optionally mutual) authentication. L is the database of tags maintained by the RFID reader.

Chapter 7

Using SAT solvers to analyse low hardware-complexity stream ciphers

Cryptographic functions are at the base of computer security with encryption ciphers ensuring confidentiality and authenticity. Despite their importance, many practical cryptographic functions rely on unproven assumptions about the complexity of their underlying mathematical problems. When these assumptions are found to be incorrect, new theoretical and practical attacks are constructed that sharpen the understanding of a specific problem and advance the evolution of cryptography in general. SAT solvers have been shown to be a powerful tool in testing mathematical assumptions. In this chapter, we extend SAT solvers to better work in the environment of cryptography.

In this chapter we analyse two stream ciphers used in RFIDs and a low hardware-complexity stream cipher that is a close relative to a stream cipher targeted for RFIDs. In order to achieve these goals we bring SAT solvers and stream ciphers closer to each other by adapting both the solver to stream ciphers and the representation of stream ciphers to SAT solvers. Previous work on solving cryptographic problems with SAT solvers has concentrated on the best mathematical representation of ciphers [43, 13, 4, 54, 24, 46]. To further improve the potential of SAT solvers, we adapted a SAT solver to better suit cryptographic problems and then manipulated the representation of some cryptographic problems to best fit this modified solver. We refined SAT solvers to understand the XOR operation, which is common in cryptography, besides functions in the conjunctive normal form (CNF) that is native to many SAT solvers. We further added dynamic behaviour analysis to more thoroughly understand the workings of SAT solvers on cryptographic primitives.

The first two stream ciphers analysed, Crypto-1 [18] and HiTag2 [51], are weak stream ciphers, widely used in electronic payment, access control and car immobilisers. The third cipher, Bivium B [53], is a simplified version of the eSTREAM standard stream cipher Trivium [10] known for its simple description and hardware-oriented design. As is customary [46, 24] for SAT solver-based analysis we will express our results by comparing the estimated number of seconds to solve the ciphers on a desktop computer before and after our improvements. Solving the above mentioned ciphers with an unmodified MiniSat SAT solver and with only basic improvements to their CNF representation reveals the secret on a desktop machine[i] state within 170 hours for Crypto-1, a week for HiTag2 and in $2^{42.7}$ s [46] for Bivium B. With our adapted SAT solver and tuned cipher description techniques, the average attack time using SAT solver-based techniques on a desktop PC drops to 40 seconds for Crypto-1, 6.5 hours for HiTag2 and $2^{36.5}$ s for Bivium B.

Until now, our techniques give the best SAT solver-based attack time for the above mentioned ciphers. For Crypto-1 there exists an algebraic attack by Garcia et al. [28] that solves for the key by inverting the filter function to arrive at approx. 2^{16} candidate keys, which takes only approx. $O(2^{26})$ operations to break, which is only a fraction of a second to carry out on a desktop computer.

Organisation

We first introduce the problem domains' state-of-the-art in Sect. 7.1. We then optimise a standard SAT solver for cryptographic problems in Sect. 7.2. The SAT solver now handles XOR operations natively to faster solve cryptographic problems and the solver's execution is visualised to allow insight into its inner workings. Based on these improvements, guidelines are derived on how to convert ciphers to a description that can be quickly solved in Sect. 7.3. Finally, three ciphers are solved using the adapted SAT solver faster than was previously possible with other SAT solver-based techniques, in Sect. 7.4.

7.1 Background

Our results build on research in stream ciphers, SAT solvers, and algebraic cryptanalysis. In this section, we present the current state of research in SAT solvers, algebraic cryptanalysis and stream ciphers.

7.1.1 SAT solvers

Satisfiability solvers are programs that employ highly optimised mathematical algorithms to decide whether a formula has a solution. In this chapter we concentrate on one widely-used formula description, the clausal normal form (CNF). The CNF formula φ on n binary variables $v_1, \ldots, v_n$, is a conjunction (`and`-ing) of m clauses $\omega_1, \ldots, \omega_m$ each of which is the disjunction (`or`-ing) of literals, where a literal is the occurrence of a variable e.g. v_1 or its complement, $\neg v_1$. Any formula can be transformed to CNF by introducing new variables to label the subformulas. A formula can be converted to clausal form by introducing fresh variables for each compound subformula and adding suitable clauses, e.g. in converting $\neg p \vee (\neg q \wedge r)$,

[i]Intel Xeon E5345@2.33GHz, 4MB cache, 4GB memory, using one core

we label $\neg q \wedge r$ as b and $\neg p \vee b$ as a to obtain the clauses

$$a$$

$$\left.\begin{array}{r} a \vee p \\ a \vee \neg b \\ \neg a \vee \neg p \vee b \end{array}\right\} \neg p \vee b \Leftrightarrow a$$

$$b$$

$$\left.\begin{array}{r} b \vee q \vee \neg r \\ \neg b \vee \neg q \\ \neg b \vee r \end{array}\right\} \neg q \wedge r \Leftrightarrow b$$

When transforming a given function to CNF, the best known way is the Karnaugh-map method [35]. However, there is a multitude of ways, which are further elaborated upon in Sect. 7.3.4.

SAT solvers are mostly used in electronic design automation (EDA), though they are also used in a growing number of other domains. State-of-the-art solvers have been extended or adopted to meet the specific characteristics of different problem domains, for example temporal induction in [23], ontology matching [29] and knowledge compilation [17].

The basic DPLL algorithm

In this chapter we concentrate on SAT solvers that are based on the DPLL algorithm [16]. The DPLL-algorithm is a depth-first search algorithm to find a satisfying variable assignment to the formula φ in CNF. The algorithm recursively chooses a variable, assigns a truth value to it, simplifies the formula, and if the new formula φ' is satisfiable, the original formula was satisfiable. If the simplified formula is not satisfiable, it goes back, assigns the opposite truth value to the variable, simplifies the formula, and if this new formula φ'' is satisfiable, the original formula was satisfiable. However, if even this second attempt fails, then the original formula was also unsatisfiable. This method is known as the *splitting rule*.

Clauses that contain at least one literal assigned to `true` are satisfied, therefore they are removed from the problem set and are not treated by the DPLL algorithm. For the remaining clauses, the DPLL algorithm uses the *unit clause* and *empty clause* rules for formula simplification. Clauses that do not contain any unassigned literals are called empty clauses. A formula that contains such a clause is unsatisfiable. If such an event occurs, it is called a *conflict*. Clauses that contain exactly one unassigned literal are called unit clauses. These clauses assign a variable to `false` if the unassigned literal is negated and to `true` if the literal is non-negated. If such a variable assignment occurs, it is called a *propagation*.

A description of the recursive DPLL algorithm is in **Function** `DPLL`. The function takes formula φ as a parameter and returns either SAT or UNSAT. The algorithm starts by making all possible propagations, then checks for empty clauses. If one or more are found, it returns UNSAT. Otherwise, it checks if there are any clauses remaining that have unassigned literals. If so, it assigns an unassigned variable to `true`, and recursively calls itself. If the function call returns UNSAT, it tries to assign `false` to the same variable, and again recursively calls itself. If the function call again returns UNSAT, then the original φ formula must have been unsatisfiable.

If at least one of the function calls have returned SAT, all clauses must have been satisfied in either one or the other part of the split, and the DPLL function returns SAT. It is easy to verify that the algorithm is complete: it always terminates as there is only a finite number of variables that can be branched upon.

Function `DPLL(`φ`)` The basic DPLL algorithm. The algorithm first makes all possible propagations, then checks if there are any conflicts. If all clauses are satisfied, it returns `SAT`. If some clauses are still unsatisfied, it tries to branch on a variable by assigning it first one value then to the other, recursively calling itself. If both branches returned with `UNSAT`, the function returns with `UNSAT`

```
Input: Clausal Normal Form formula $\varphi$
Output: SAT or UNSAT
foreach unit clause $\omega$ in $\varphi$ do
    assign var in clause to satisfy it;
end
if $\varphi$ contains an empty clause then
    return UNSAT;
end
if there are unsatisfied clauses then
    Take unassigned variable var ;
    var $\leftarrow$ true;
    if DPLL($\varphi'$) $\neq$ SAT then
        var $\leftarrow$ false;
        if DPLL($\varphi''$) $\neq$ SAT then
            // Neither branch is satisfiable
            return UNSAT;
        end
    end
end
// All clauses are satisfied
return SAT;
```

Improvements over the original DPLL scheme

The first major improvement over the standard DPLL algorithm was by Silva and Sakallah [57] who introduced conflict analysis which allows non-chronological backtracking and learning. In the paper, the authors describe GRASP, a SAT solver that records all propagations made and if a conflict arises, builds an *implication graph* to determine which decisions contributed to the conflict. The solver then generates a conflict-induced clause, or *learnt clause* for each conflict, and adds this new clause into the database. The authors observe that using such a technique, the generated learnt clause can lead to *non-chronological backtracking*, that is, backtracking to a level lower than the level just below the current level. To overcome the problem of too many learnt clauses, the authors suggest to use a heuristic to keep only a certain set of learnt clauses. Finally, the authors propose an algorithm to more carefully analyse the implication graph through *Unique Implication Points* thereby reducing the size of the generated learnt clauses.

The second major improvement over the original scheme is by Gomes et al. [30], who introduced random *restarts*. Observing the runtime distribution of DPLL-based backtrack search methods on a variety of problem instances, they found that these are often characterised by very long tails or "heavy tails" [42]. They then used heavy-tailed distributions to provide a formal framework explaining the large variance and the erratic behaviour of the mean of backtrack search and proposed the solution of randomly restarting the solver to effectively eliminate this behaviour. Using rapid restarts improved the performance of SAT solvers by two orders of magnitude, thereby significantly reducing running times.

The most recent major improvement over the original DPLL algorithm is that by Malik et al. [41] who introduced the solver Chaff. Chaff's innovations are its method of *watched literals* to effectively monitor clauses for variable changes and its *Variable State Independent Decaying Sum (VSID)* variable decision heuristic.

The method of watched literals effectively eliminates looking through all clauses for propagations and conflicts by watching for changes exactly two literals in every clause. If a clause is only one-long it automatically causes a propagation and the DPLL does not need to treat it. The remaining clauses thus all have at least two literals, two of which are always watched. Since propagation needs exactly one unassigned literal, if propagation is to be made by a clause, it must have at least one literal assigned `false` and so is detected. Since conflict needs all literals assigned `false`, it too is detected. If a watched literal is assigned `false`, all clauses in which the literal is watched is investigated for propagation or conflict. If neither a conflict or a propagation occurs in a clause, then the clause must have at least two unassigned literals, one of which is added to the watch-list. This way, all propagations and conflicts are detected by watching only two literals in all clauses.

The Variable State Independent Decaying Sum (VSID) method of selecting decision variables essentially uses the learnt clauses to guide the solver towards the most important variables, using randomness to look through as-yet undiscovered areas of the search space.

MiniSat

MiniSat by Eén and Sörensson [22] is a prime example of modern DPLL-based SAT solvers. It employs all the previously detailed techniques in one well-documented package. MiniSat's primary advantage is its design simplicity: it is well built and well-implemented, using data structures that optimise cache and memory use. MiniSat is also specifically made to be extended: its source code is freely available under the MIT licence, a strictly more permissive licence than the GPL or the LGPL licences.

Many have taken advantage of the extensibility of MiniSat to enhance it in multiple ways. MiniMarch, by Wieringa et al. [62] adds branch direction heuristics and some extra logic to make the solver less sensitive to input formula shuffling. MidiSat by Kibria [36] adds a new binary clause representation method and equivalent variable detection to the solver. Finally, MiniMaxSat by Heras et al. [33] is a completely re-designed variation of MiniSat that solves weighted Max-SAT problems.

7.1.2 Algebraic Cryptanalysis

Algebraic cryptanalysis [5] is a family of attacks that can be loosely grouped according to their technique of converting the original problem into polynomial systems of equations, which are then solved using systems such as SINGULAR [31], MAGMA [9], or SAGE [60]. These systems employ a wide range of algorithms to resolve the systems of polynomials, among them algorithms such as the F4 [26] and F5 [27] algorithms by Faugère, or standard SAT solvers such as MiniSat [22] using the conversion method by Bard et al. [6].

Algebraic cryptanalysis have successfully been applied to break a number of ciphers secure against other forms of cryptanalysis. The first SAT solver-based algebraic cryptanalysis was by Massacci et al. [44], experimenting with the Data Encryption Standard (DES) using DPLL-based SAT solvers. More recent work by Courtois and Bard has produced attacks against KeeLoq [11, 4], DES [13] and stream ciphers with linear feedback [12]. Algebraic cryptanalysis has also been used on modern stream ciphers, such as the reduced version of Trivium, Bivium A and Bivium B [53].

7.1.3 Stream Ciphers

A general description of low-complexity stream ciphers can be found in the previous chapter, in Sect. 6.1. In this chapter, we restrict ourselves to stream ciphers that are based on one or more shift registers with a linear or non-linear feedback function as well as a filter function that maps the shift register states to a keystream bit. In these stream ciphers, the keystream is generated by clocking the shift registers (updating their content) and mapping the new state to a new keystream bit. An example cipher using this process is illustrated in Fig. 7.1.

Some ciphers fall into the category of stream ciphers that has been detailed in Fig. 7.1 but have their registers irregularly clocked. Such a cipher is the A5/1 cipher [1] (used in mobile phones), where the irregular clocking is used to introduce non-linearity. Such ciphers are very complicated to model using SAT solvers thus making their solving slow using SAT solver-based techniques. For this reason, irregularly clocked stream ciphers are not discussed in this chapter.

7.2 Adapting the SAT solver

To take full advantage of the power of SAT solving we adapted and optimised MiniSat, a state-of-the-art DPLL-based SAT solver, for algebraic cryptanalysis. We further added visualisation to the solver to help identify bottlenecks and improve the solving by altering the problem representation. Among the many choices for modern SAT solvers, we chose MiniSat for its competitive performance, code availability, and a design that specifically encourages extensions to its input language.

7.2.1 Full pre-simplification

Clause pre-simplification is an algorithm that activates itself inside MiniSat when a direct variable assignment is either given through the input file, or is found through

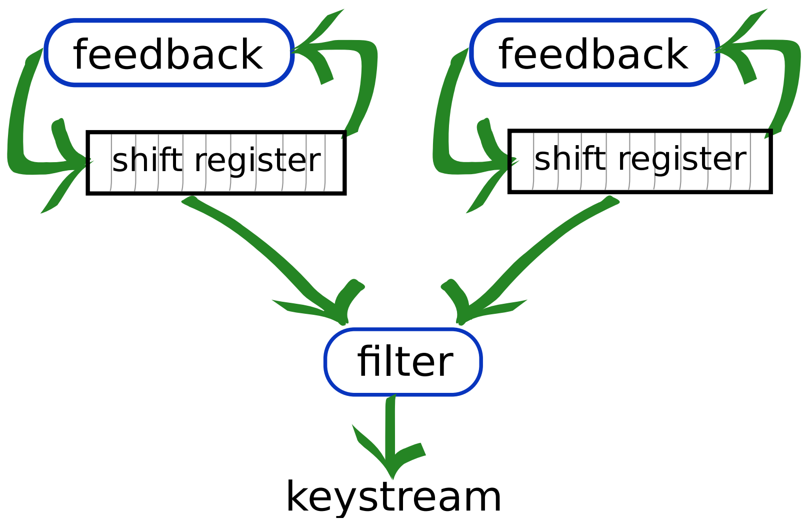


Figure 7.1: Illustration of the keystream generation process of an example stream cipher that uses two shift registers and the process of updating its state with feedback functions and generating the keystream using filter functions. Functions are illustrated in rounded boxes and shift registers in sharped-edged boxes. The two feedback functions can be different for the two shift registers, and the size of the shift registers can also vary. The input of the feedback functions may also vary and may not only come from register it is updating. Typically, there is only one keystream bit generated per cycle, though with modern ciphers, higher throughput can usually be achieved by multiplying the number of feedback and filter functions, updating the state of the shift registers in groups of 2,3, etc. bits, generating 2,3, etc. keystream bits per cycle.

learning. Such a direct variable assignment, expressed as a single-literal clause, has the form "a" or "$\neg a$". Such a single-literal clause assigns the variable to either true or false. This assignment can then be propagated through the clauses *before* any dynamic behaviour (i.e., branching, learning, etc.) of the solver. If through such an assignment a literal in a clause becomes true, MiniSat automatically removes that clause form the watched clause list. Also, if such assignments cause propagations, these propagations are automatically made and the propagating clause is again removed from the watched clause list.

However, if through such an assignment a literal in a clause becomes false, the literal is left inside the clause. Neither the source code nor the research paper on MiniSat [22] documents why this omission was made, and we can only speculate that the designers simply overlooked it. Since the false literals are left inside the clauses, they must be examined for change (in vain) during the execution of the solver, which unnecessarily slows down the solver.

We implemented into MiniSat a function that removes false literals from clauses. The fixed overhead from this *full pre-simplification* technique is more than compensated for by the faster solving if the function is only called at the beginning of the search and once after every 3-4 restarts. The observable overall speedup using this technique depends on the number of false literals, and can range from unobservable to 1.5x for specific cases such as a large number of internal state bits given as unit clauses to help solve a stream cipher.

7.2.2 XOR support

Cryptographic building blocks such as filter and feedback functions lead to equations with many XORs. These XOR constraints, when converted to CNF representation without further elaboration, grow exponentially in size. This is because an n-long XOR constraint's Karnaugh table contains 2^{n-1} minterms, and hence needs 2^{len-1} clauses to describe in CNF.

Previous work

Previous research extended the Satz solver to reason about 2- and 3-long XOR constraints, which its authors called equivalency reasoning [39]. Research has also been conducted on using XORs as a black-box part of the search [40], but apart from a will to implement such a construction by Massacci [7, Sect. 9], as of yet, there is no published successful native implementation of the XOR function in SAT solvers. Current techniques therefore focus on circumventing the exponential explosion of XORs when converted to CNF, by cutting up the XOR function [4, Sect. 6.4] into groups of smaller XORs, each setting an additional variable. The full XOR is then represented as a XOR of the additional variables. While cutting up XORs allows MiniSat to work on long XOR chains, this approach forces the solver to watch and examine many clauses for variable changes, when in fact only one XOR constraint should be watched.

Our approach

To mitigate this limitation, we implemented the XOR constraint natively into MiniSat. Each XOR constraint is represented by a single *xor-clause*[ii]. A xor-clause behaves as a regular clause towards all unchanged parts of the solver: it dynamically changes appearance when propagating or causing a conflict by appearing as a different regular clause depending on the current assignment of variables.

For example, the xor-clause

$$a \oplus b \oplus c$$

represents all the regular clauses

$$a \vee \neg b \vee \neg c \quad (1) \qquad \neg a \vee \neg b \vee c \quad (2)$$
$$a \vee b \vee c \quad (3) \qquad \neg a \vee b \vee \neg c \quad (4)$$

and if, for example $a =$ `true` and $b =$ `true`, then it changes its appearance to the regular clause (2), and causes the propagation $c =$ `true` just as its regular representation would. If, however, $a =$ `false`, $b =$ `true` and $c =$ `true`, the xor-clause changes its appearance to regular clause (1) and causes a conflict just as its regular representation would.

Generating a conflicting or propagating clause from a xor-clause is done as follows. All variables that are assigned to `false` are included as-is, and all variables that are assigned to `true` are included in a negated form. If propagating, the single unassigned variable is also included, its negation depending on the values of the other variables in the xor-clause.

Pseudo-code for XOR support in MiniSat

The pseudo-code for xor-propagation and xor-conflict generation in MiniSat is presented in **Function** `xor-DPLL`. The queue `to_examine` contains all the variables that were assigned recently. A variable `var` is popped from this queue and is examined whether it causes any propagations or conflicts. The technique of watched literals (first used in Chaff [41]) is used by xor-clauses. In the case of xor-clauses however, since they are never satisfied by a single true literal, variables are watched instead of literals. This entails managing a separate variable watch-list for xor-clauses. Although this produces some overhead, one must take into account that if the xor-clause was converted to 2^{n-1} regular clauses, then the number of watched literals for a single xor-clause would be 2^n — a far greater number than the two necessary with the new solution.

Since watchlists (see Sect. 7.1.1 on watchlists) are different for xor- and regular clauses, they cannot be examined in a mixed fashion. The order, however, does make a difference. In our implementation, the regular clauses are examined first by the regular `DPLL` function. If they cause any propagations, the requested assignments are carried out. If they cause a conflict, the `DPLL` function returns a non-null `clause` which in turn is returned by the `xor-DPLL` function. Only after the regular clauses have been examined does the algorithm examine the xor-clauses. The order regular-then-xor is preferred as xor-clauses are usually more difficult to examine and it helps to have all possible information (i.e. assignments by the regular clauses) at hand before executing the relatively expensive xor-clause examination procedure.

[ii]This is the same name as given by Massacci [7] to such clauses

Function `xor-DPLL(`*to_examine*`)`. The function is an extension of the original DPLL function to handle xor-clauses in a manner that hides the complexity from the rest of the SAT solver. Between lines 5.-5. the function iterates through all the xor-clauses that the currently investigated variable, var, could potentially affect. If var causes a propagation (line 5.), the propagated variable is assigned and the xor-clause takes on the form of the clause that would have caused this propagation had each and every 2^{n-1} clause been inserted into the solver. If var causes a conflict (line 5.), the xor-clause takes on the form of the regular clause that would have generated this conflict, and returns this clause. If the assignment of the variable var did not cause either a propagation or a conflict (line 5.), a new variable is set to be watched in the xor-clause, just like with regular clauses.

```
   Input: to_examine
   Output: Conflict_clause
1  while size(to_examine) > 0 do
2      var ←pop variable from to_examine ;
3      clause ←DPLL(var);
4      if clause != null then return clause ;
5      foreach xor-clause that var is watched in do
6          if xor-clause makes propagation on var then
7              Bound var to propagated value;
8              Enqueue var in to_examine ;
9              xor-clause ←Regular_clause_from_xor(xor-clause);
10         else
11             if xor-clause causes conflict then
12                 xor-clause ←Regular_clause_from_xor(xor-clause);
13                 return xor-clause ;
14             else
15                 Find new unassigned variable to be watched in xor-clause ;
16             end
17         end
18     end
19 end
20 return NULL;
```

Results

Solving cryptographic functions is accelerated considerably by integrating xor-clauses into MiniSat. For the stream ciphers Crypto-1 and Grain solving is up to twice as fast with xor-clauses and memory usage is decreased by at least an order of magnitude. Besides speeding up the solving, native XOR support leads to more concise input file and internal data structures, which simplify analysing the dynamic behaviour of the solver. Lastly, xor-clauses enable a straightforward implementation of Gaussian elimination into MiniSat as explained in the next section.

7.2.3 Gaussian elimination

Gaussian elimination is an efficient algorithm for solving systems of linear equations. Since each xor-clause is a linear equation, we can use this algorithm to solve the system of equations described by the xor-clauses. Some linear problems with as many as 100 variables can be trivially solved with Gaussian elimination but take an excessive amount of time when solved with SAT solvers. This phenomenon is due to the fact that SAT solvers solve by guessing variables and determining if there is any equation that gives a result given the current assignments. If the set of linear equations is dense (i.e. all equations contain many variables), almost all variables need to be guessed before any equation gives a result. Thus, for a system with 100 variables, it is not uncommon that 80 variables need to be guessed before any equation gives a result, i.e. the search space is on the order of 2^{80}. When using Gaussian elimination, on the other hand, the same problem can be solved in less than 2^{20} operations.

Previous work

Including Gaussian elimination into MiniSat is part of the general move towards *SAT Modulo Theories* (SMT). A tutorial by Moura et al. [38] is a good introduction to SMT. Essentially, an SMT instance is a generalisation of a Boolean SAT instance in which various sets of variables are replaced by predicates from a variety of underlying theories. SMT formulas provide much richer modelling language than is possible with Boolean SAT formulas. Such richer modelling language includes, but is not limited to equivalence, congruence closure, difference arithmetic, linear arithmetic, and a combination of these. Xor-clauses allow linear arithmetic which Gaussian elimination can understand and reason about, tightly integrating its conclusions into the DPLL algorithm of MiniSat.

Implementation of the Gaussian elimination as a black-box subroutine into the DPLL algorithm has been done many times before, for example in [40, 61]. These methods pay off when the affine logic portion is overwhelming, for example in the bit parity problem, but are ineffective if the affine logic part is just a small part of the original formula. However, this latter is the case for many problem domains, such as model checking [40] and when encoding cryptographic ciphers. For example, in the encoding of DES, XOR-s make up only 4% of the original problem [43]. To overcome this difficulty, Massacci has proposed a calculus with affine-logic reasoning in [7], which essentially ports the Gaussian elimination procedure to the world of DPLL-based SAT-solving, but left the implementation as future work. Our implementation can be thought of as specific instantiation of this calculus.

Our approach

To benefit from Gaussian elimination during solving, whenever the SAT solver cannot perform any further propagations and would need to guess (i.e. branch on) a variable, the Gaussian elimination routine is asked if there is any information it could extract from the xor-clauses. If the Gaussian elimination finds a combination of the xor-clauses that is a unit or empty under the current assignments, then this combination of xor-clauses is treated as if it was a new xor-clause. The new xor-clause is transformed into a regular clause (as described in Sect. 7.2.2), and produces a propagation or a conflict. Therefore, the extracted new clause takes part in building the implication graph and analysing conflicts.

It is possible that the Gaussian elimination gives more than one result, or that the results are special, e.g. a xor-clause that is 1- or 0-long. The following list of rules are applied in a top-down fashion to deal with the results of the Gaussian elimination routine:

1. *A 0-length non-inverted xor-clause* is found. Such a xor-clause, a combination of the xor-clauses originally in the system, is unsatisfiable under any variable assignment, and thus the original system must be unsatisfiable. The solver outputs UNSAT and stops.

2. *A 1-length xor-clause* is found. Such a xor-clause indicates an unconditional variable assignment, which must be done at the beginning of the search tree. All assignments are reverted until depth 0, the indicated propagation is made, and the algorithm starts again.

3. *One or more empty xor-clauses* are found. The conflict analyser expects that during solving, the first moment that a conflict can be found, it is found. However, Gaussian elimination is not called at every variable-assignment (see parameter d below) and so conflicts are not found as early as would otherwise be possible. To circumvent this limitation, the *conflict-level*, the highest assumption level of all variables in the xor-clause is calculated for all xor-clauses, and the xor-clause is picked that has the lowest such level. Then, all variable assumptions are reverted until the conflict-level of the selected xor-clause. Only then is the conflict analyser given the selected xor-clause to generate the learnt clause.

4. *One or more unit xor-clauses* are found. All propagations indicated by the unit xor-clauses are made. Note that rules dealing with empty clauses are above this rule, since if a conflict is found, it is of no benefit to propagate variables, as the current branch is necessary wrong. Further examination of the current branch would therefore be unnecessary.

In cases (3) and (4) the solver needs the xor-clause, which when evaluated with the variable assignments, gives the empty or unit clause, respectively. Calculating this xor-clause is important, as it signals the solver what variable was propagated by what clause (in case of a propagation), or what clause caused the conflict (in case of a conflict). Without the calculation of this xor-clause, the results obtained from

Gaussian elimination could not take part in conflict analysis and thus would not be fully integrated into the DPLL algorithm.

To calculate the empty/unit xor-clauses and the xor-clauses that evaluate to the empty/unit xor-clauses, the implementation keeps two matrixes at all times. The matrixes' rows represent xor-clauses and all but their last column represent variables, the last column being reserved for the traditional augmented column used in Gaussian elimination. Both matrixes are initially loaded with the same xor-clauses and the same augmented column storing the inversion of the xor-clause. The assigned matrix, or *A-matrix* is updated with the current assignments, which simplifies the task of evaluating whether a combination of xor-clauses is unit or empty. This is the matrix the Gaussian elimination routine treats. The non-assigned or *N-matrix* is not updated with the current assignments, and only mirrors the A-matrix with its row-swap and row-xor operations. Having two matrixes allows the xor-clause to be read from the N-matrix, and its assigned form to be read from the A-matrix. For example, if the two matrixes are:

A-matrix
with $v8$ assigned to `true`

$$\begin{array}{cccc|c} v10 & v8 & v9 & v12 & \text{aug} \\ 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}$$

N-matrix

$$\begin{array}{cccc|c} v10 & v8 & v9 & v12 & \text{aug} \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}$$

then the second to last row of the A-matrix indicates propagation of $v12 =$ `false`. The xor-clause can be read from the N-matrix: it is $v8 \oplus v12 \oplus 1$. The A-matrix, used by the Gaussian elimination algorithm, is upper triangular, but the N-matrix is only upper triangular for the columns representing variables that are not assigned.

Tweaks to the Gaussian elimination

Linear algebra and Gaussian elimination in particular have large literature on possible optimisations. Our problem, however, is somewhat special, as we need to optimise for solving similar matrices, that are updated only minimally, many times. Furthermore, we need to keep not one, but two matrixes at all times, thus certain optimisations only apply to one of the two matrixes. These challenges make our problem somewhat different from that covered by currently existing implementations of the Gaussian elimination algorithm, requiring solutions adapted to our specific needs.

Since the sum of variables in all the xor-clauses is in general much larger than the average length of any given xor-clause, most of the matrix will be filled with zeros at the beginning of the search. To keep this advantageous property of sparseness during the run of the algorithm, we need to adopt to our needs the well-known optimisation of choosing the pivot during Gaussian elimination [34]. Choosing the best row and column on which to pivot during Gaussian elimination can keep the density of the matrix low for a longer amount of time. The resulting lower density helps, as is lowers the total number of row-xor operations as there are less '1'-s in the matrix that need to be eliminated by xor-ing with another row. Since we have two matrixes, we have the choice to minimise the density in the A- or the N-matrix, but we can optimise for only one, and so a choice must be made:

- The advantage of keeping the A-matrix sparse is that less back-substitution is needed to find a propagating or conflicting row, as the probability of rows that only contain one or no '1'-s in them after the Gaussian elimination is increased. This advantage is specific to our case, as normally back-substitution is used to eliminate such remaining '1'-s, but this is less of a concern since making the matrix upper triangular is usually the most difficult part. For us, the matrix is small, therefore the time taken for back-substitution is also significant.

- The advantage of keeping the N-matrix sparse is that with a more sparse N-matrix, the generated xor-clauses are on average smaller. It follows that the conflict analysis routine has to work less and the generated learnt clauses are smaller.

With simulations we have found that it is best to optimise for the sparsity of the A-matrix to reduce the amount of time taken for back-substitution.

To save time, the A-matrix is incrementally normalised as the solver travels down the search tree and assignments are made. We save both matrixes at every d-th search depth, and in case the solver has to jump back (due to a conflict), we re-load the matrixes from the state saved at that (or, if unavailable, somewhat earlier) depth. Saving at every depth point (i.e. $d = 1$) requires many matrix savings, but the A-matrix has to be updated only with the assignments at the current depth, and so the Gaussian elimination has to be done on a matrix with a small delta. On the other hand, saving at comparatively far depth points (i.e. $d \gg 1$) requires less savings, but the A-matrix has to be updated with, on average, $d/2 + 1$ levels' assignments, and the Gaussian elimination has to normalise a matrix with a large delta. We have found that saving at every $d = 2$-nd search depth is a good trade-off between having too many matrix-saves and too large matrix updates and thus working on matrixes with large deltas.

To further save time during Gaussian elimination, we keep record of the last '1' in the A-matrix for each column. When the A-matrix has to be incrementally Gauss-eliminated, we only look for candidate pivot rows in that column until the indicated row. This optimisation reduces the search time for the next pivot, and only requires a very small list of numbers to be stored along with the A-matrix.

A trade-off parameter for the Gaussian elimination is the cut-off depth until which it is worthwhile to execute the algorithm. Cutting off branches at the top reduces the search space more than cutting at the bottom, but it takes approximately the same time to execute the algorithm. However, if the cut-off depth is too shallow, the constant overhead is more than the benefit, but if too deep, the dynamic overhead is more than the benefit. In the end, we made the cut-off depth configurable, and ran tests to decide for each cipher which depth gave the most benefit.

Results

Using Gaussian elimination, solving Crypto-1, HiTag2 and Bivium B is faster by 1–5% if we restrict the search depth to 2–3, depending on the number of guessed bits. The results detailing the speedups are present in Table 7.1. For other instances

Table 7.1: The table shows the speedups achievable with the Gaussian elimination for different stream ciphers and for different maximum search levels. Each cipher was executed 500 times with randomly picked and randomly set help bits (see Sect. 7.4.1 on details). All times in the table are the sum of time it took to execute the solver 500 times. For Bivium B, 177 keystream bits were given. For both Crypto-1 and HiTag2, 56 keystream bits were given. All calculations were carried out on a laptop-class computer.

	No. help bits	Gaussian elimination active until level			
		Inactive	2	3	4
Crypto-1	12	26.95 s	25.76 s	26.54 s	27.70 s
HiTag2	18	34.83 s	33.85 s	29.47 s	38.50 s
Bivium B	60	174.02 s	165.12 s	171.14 s	176.05 s

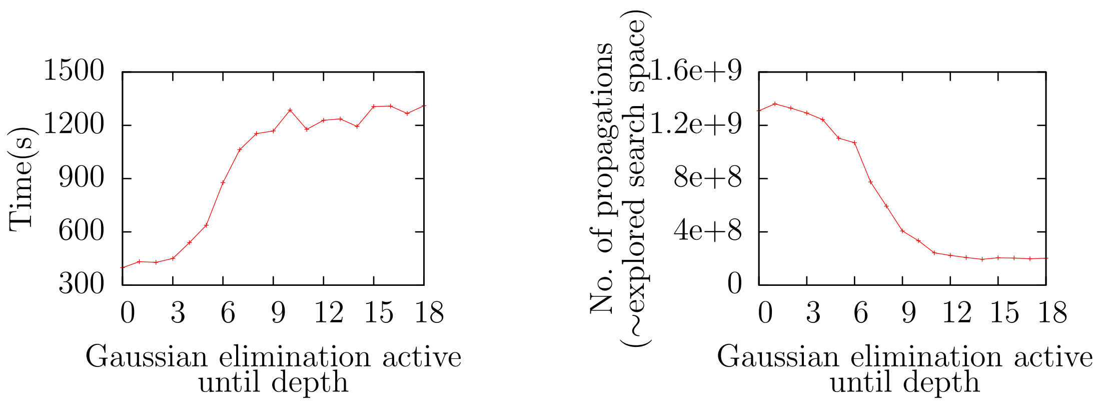


Figure 7.2: Comparison between the time and the number of propagations (∼explored search space), relative to the depth until which the Gaussian elimination was active. Each point in the graphs represent 2000 random examples of the Bivium B cipher, given 56 randomly guessed state bits

derived from other ciphers, Gaussian elimination does not appear to decrease the overall solving time. A comparative figure for Bivium B, showing the speed of solving and the explored search space versus the depth until which the algorithm was active is present in Fig. 7.2. It is apparent from the graphs that using Gaussian elimination reduces the explored search space (in the example, by up to 83%), but the algorithm takes more and more time to execute as the cut-off depth is increased.

Apart from the marginal speedup that Gaussian elimination brings for certain cipher instances, it is a useful tool for multiple other reasons. First of all, it demonstrates that SAT solvers ignore certain characteristics of the problem they are dealing with, and by exploiting these properties the search space could be significantly reduced. Secondly, the combined solver works much faster on problems that have large parts that can benefit from Gaussian elimination. A trivial example of such a problem is for example a SAT problem that encodes a dense linear problem, mentioned at the introduction to Gaussian elimination. Lastly, our implementation of Gaussian elimination can likely be improved upon leading to greater speedups.

7.2.4 Dynamic behaviour analysis

The dynamic behaviour of SAT solvers is hard to follow since branching and propagation occur far too many times to be traceable by hand. Understanding the solver's dynamic behaviour, however, is essential for estimating a cipher's complexity and for improving the solver's performance. To better understand how MiniSat reaches solutions, we implemented search tree tracing into the solver. The output of our MiniSat trace extension can be analysed visually and statistically.

Visual dynamic behaviour analysis

Visualising the operation of DPLL-based SAT solvers was introduced by Sinz [58]. In [58] the author introduced DPvis, a tool separately detailed in [59], to visualise the structure of SAT instances and runs of the DPLL procedure. Although the DPvis tool has multiple interesting features, such as manual intervention during the solving and an interface to MiniSat, we have found it to lack two features that are useful to analyse the solving of cryptographic functions.

Firstly, since DPvis was designed to examine general SAT problems for hidden structures, it is not well-suited to analyse problems arising from cryptographic functions, as these problems have a structure that is known, which does not need to be discovered. Instead, this known structure must be integrated into the way the search graph is plotted. Secondly, since DPvis shows the DPLL instance step-by-step, thereby having to use a placement algorithm that is efficient enough to plot the graph step-by-step. We, on the other hand, wished to plot the search graph of a whole restart, and therefore were interested in an algorithm that plots a search graph using more demanding algorithms that produced a better final graph placement.

To overcome these shortcomings of DPvis, we implemented into MiniSat a dynamic search-tree description generator which can be visualised using the Graphviz [25] software tool. Our extension to MiniSat allows for variables to be named and for clauses to be grouped and named, which is useful when multiple clauses are used to represent one logical entity (e.g. a feedback function). An example search tree of the Crypto-1 cipher is in Fig. 7.3. The visualisation allowed us, for instance, to identify the regularly placed filter function taps of Crypto-1 as its largest weakness over an improved variant found in HiTag2 tags.

Statistical dynamic behaviour analysis

During solving, the following statistics are calculated for each restart of the solver:

- *Search tree statistics.* These include the number of branches visited, the average branch depth, the number of decisions per branches and the number of propagations per branches.

- *Clause group propagation statistics.* This statistic lists the clause groups that propagate the most, i.e. the clause groups that set most of the internal variables. In case of stream ciphers, the clause groups that cause the most propagations are usually the groups that represent the feedback function.

- *Clause group conflict statistics.* This statistic gives information on which clause groups were responsible for stopping the propagation of badly guessed

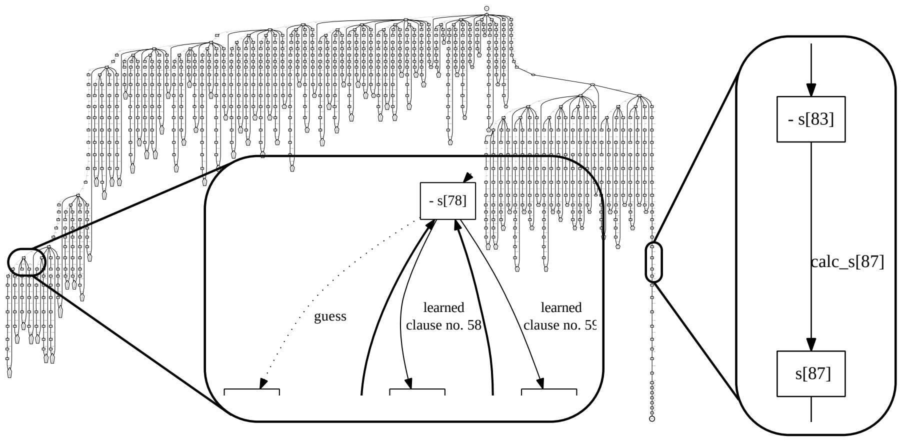


Figure 7.3: Graphviz [25] visualisation of an example search for the Crypto-1 cipher's states. The tree is read from left to right, top to bottom: the left- and bottommost pentagon is the first conflict clause, the right- and bottommost circle is the satisfying assignment.

variables (key bits or state variables), i.e. the clause groups that made the most conflicts. In case of stream ciphers, these clause groups were consistently the clause groups that represented the filter functions, i.e. the clause groups that connected the known (observed) keystream bits with the state or key bits.

- *Variable assignment statistics.* Displays the depth at which variables were assigned on average during the restart. For instance, if variable $v1$ was assigned once at depth 10, and once at depth 20, then $v1$ appears with value 15 on this list. The variables that are guessed at the top of the search tree are the variables deemed the most important from the solver's point of view. Unless the solver gets lost at a wrong part of the search space, the important variables for stream ciphers are correctly recognised to be the key or state variables.

Data sets to plot the distribution of branch depths are also provided. An example of the produced graph is present in Fig. 7.4., where the branch depth distribution statistics of the Grain cipher is shown. Essentially, the distribution resembles a binary distribution cut in half. The reason for this is that it is almost trivial to guess a small set of variables such that they do not immediately cause a conflict, but as the number of guessed variables increases, the chance to have a conflict increases rapidly. Therefore, the number of branches that are short is much larger than the number of branches that are long. On the figure, the evolution of the search graph over the restarts is also perceivable. Fundamentally, the distribution does not change but both the maximum search depth and the maximum number of branches of a certain length increases.

Data sets to plot learnt clause statistics is also provided: the solver counts the number and length of learnt clauses and outputs these informations in a structured text format. The number of learnt clauses of certain lengths can then be plotted using the Graphviz [25] program. An example plot showing the distribution of the learnt clauses for Grain is present in Fig. 7.5. For the plots of all ciphers' learnt clause length statistics, the interested reader is referred to Appendix 7.A.

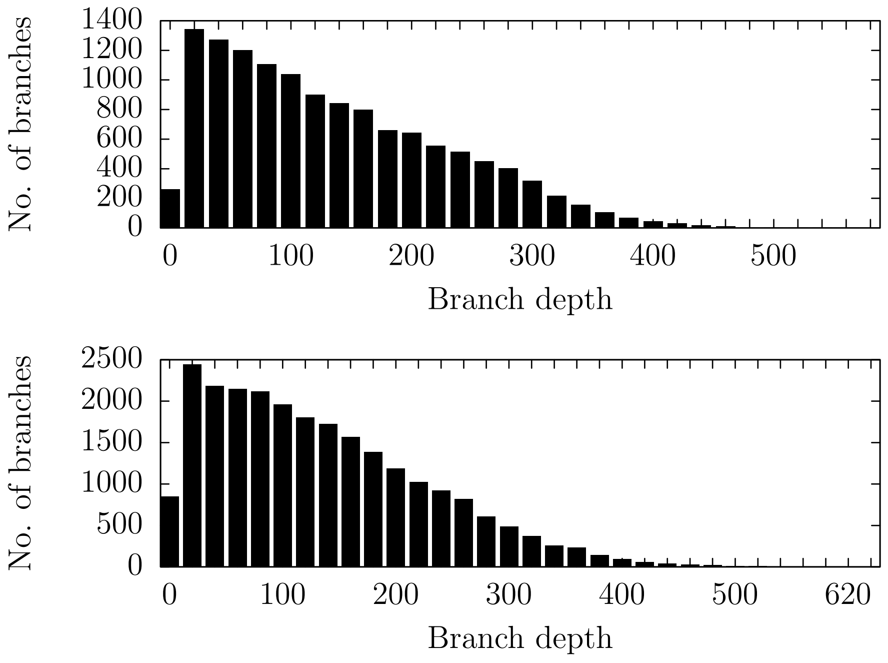


Figure 7.4: Tree branch depth distribution of restart numbers 13 (at the top) and 16 (at the bottom) when solving Grain. The distribution is essentially the same, however, since the number of allowed conflicts in the restart is more for the 16th restart, the numbers on the x axis are higher and the maximal branch depth is larger.

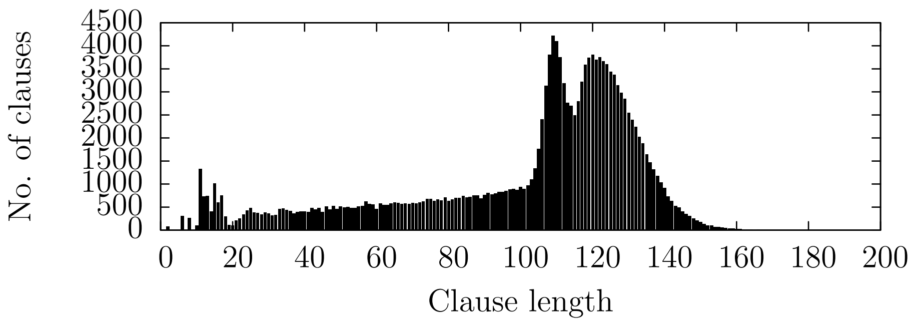


Figure 7.5: Plot showing the distribution of the size of the learnt clauses while solving the Grain stream cipher with our techniques. To produce the plot, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

Conclusions regarding stream ciphers

It is clear from the statistics and the visualised search trees that during solving, the most important variables are picked automatically by MiniSat, which for stream ciphers are always the state or key bits. By examining smaller search graphs and the statistics on the most conflicted clause groups, we further found that once the important variables have been guessed, the results of these assignments are propagated to the clause groups representing the filter functions, and if the keystream bits do not match, a conflict occurs, a guess is reversed, and the algorithm starts again.

The solver's strategy is therefore similar to a brute force search in which all key or state bits are tried but with three key features that make the solver's strategy markedly distinct from that of a brute-forcing attack:

1. During the search, the solver learns new clauses (as detailed in Sect. 7.1.1). These learnt clauses help the solver evade dead-ends later during the search, thus reducing the search tree. As observed in our experiments with stream ciphers, these learnt clauses are sometimes very short, going even so far as being unit length (see Appendix 7.A for details).

2. During the search, if the guessed variables do not work out and a conflict is found, not all of the stream cipher needs to be re-calculated, as is the case with brute-forcing. The solver automatically determines the point in history where it needs to jump back, thereby evading the re-computation of the internal state until that point. Therefore, it is possible to quickly search part of the search space by re-evaluating only a small part of the cipher. Furthermore, the dynamic variable ordering logic of the solver picks the variables such as to minimise the jump in history, therefore localising the search and maximising the advantage of the solver compared to brute-forcing.

3. The solver evaluates the cipher's internals only partially instead of blindly evaluating all parts of the cipher, as is the case with brute-forcing. If the cipher has a relatively short calculation path from its state or key bits to its keystream bits, the solver finds and exploits it to reduce the average search tree depth.

7.2.5 Optimal attack method

The lessons learnt from search-tree tracing are as follows. It is best not to include long initialisation sequences (such as that used by Grain) in the equations since after initialisation all keystream bits depend on all key bits. This forces the solver to calculate a large part of the cipher in an ineffective way, as its description and subsequent evaluation in the solver is more complicated than the way the cipher was originally meant to be calculated.

There are a number of criteria that secure stream ciphers must adhere to. Among these criteria is that its state should not be recoverable at any point during keystream generation. Therefore, to break stream ciphers, instead of making initialisation part the problem, for the best results, we have found it useful to treat its state at a suitable point as the unknown. This way, we could take the most advantage of the partial evaluation property of SAT solvers. Although the state of a stream cipher is larger than the key for all modern ciphers, it is relatively easy to solve a large part

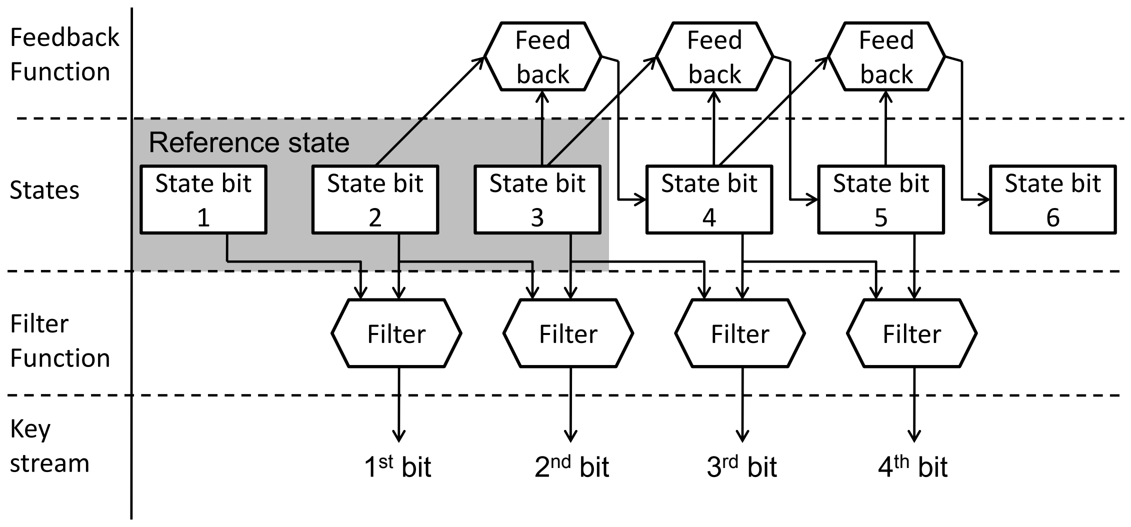


Figure 7.6: Logical circuit representation of an example stream cipher: The cipher has a 3-bit shift register, whose filter function depends on the first two bits in the register, and whose feedback function depends on the last two bits in the register.

of it, as the keystream bits depend much more directly on the selected state's bits — the search tree for these state bits is thus shallow.

7.3 Adapting the cipher representation

Finding the best representation of stream ciphers in regular and xor-clauses is a crucial step in breaking a cipher with SAT solvers [4, Sect. 8]. For the techniques in this chapter, a cipher is described as a logical circuit with functions, variables, the known keystream, and known inputs.

7.3.1 Logical circuit representation

In the logical circuit representation used in our approach, the unknown is the reference state's bits, and functions are expressed in regular and xor-clauses, using variables as input. An example logical circuit for a 3-bit state stream cipher is given in Fig. 7.6. In the figure, the cipher produces four keystream bits, and the shift register is shifted three times, using the feedback function. Functions are shown as hexagons, variables as simple boxes, and the reference state is marked in grey.

The *depth* of a keystream bit is the number of distinct functions (resp. hexagons) traversed on the way from the keystream bit to the reference state bits. For example, on Fig. 7.6. the 1st keystream bit's depth is one, while the 4th keystream bit's is four. Since the solver guesses the reference state's bits, the depth of the circuit indicates the number of functions that must be evaluated by the solver to realise that a wrong guess was made for a given keystream bit. Therefore, the shallower the overall depth of the circuit, the faster the solving.

The difficulty hidden behind the functions (resp. hexagons) is also relevant, as they must be calculated whenever traversed. If the number and length of clauses

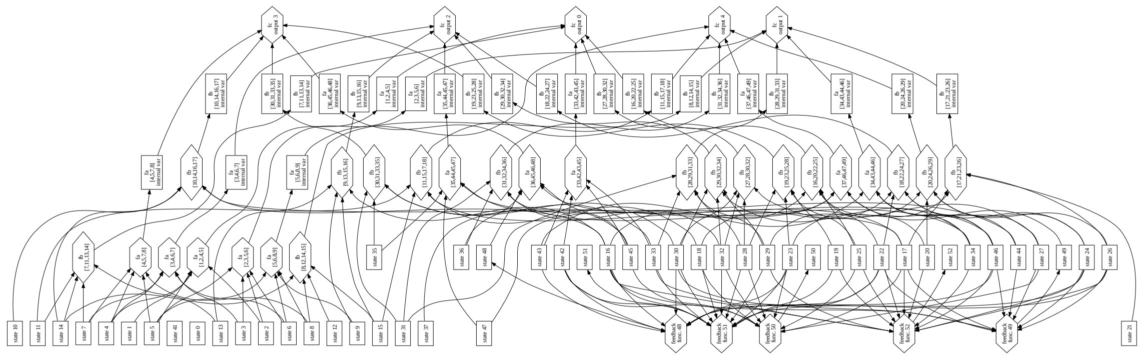


Figure 7.7: Clause- and variable-dependency graph of HiTag2. Clause groups are represented as hexagons, and variables as boxes. The known keystream bits are the 5 final filter functions at the top, and the feedback functions are the 5 hexagons at the bottom right.

representing these hexagons are large, the solver is slowed down considerably. Finally, the number of reference state bits each keystream bit depends on plays an important role during solving, as a large part of these must be guessed before evaluation can take place. This *dependency number* can be calculated by simply traversing the graph in a breath-first search fashion from the keystream up. The lower this number, the faster the solving.

To summarise, when attempting to represent a cipher, the depth of the resulting logical circuit, the number of reference state bit dependencies and the complexity of the traversed functions' representations must all be optimised to maximise solving speed.

7.3.2 Generating the logical circuit representation

To evaluate the effectiveness of different representations of the same stream cipher, we extended MiniSat by a tool that generates the logical circuit's description. Given some additional information in the input language, the circuit is visualised with `Graphviz` or statistically analysed to calculate keystream bit depths and state-bit dependencies. In the generated circuit, just as in the search tree, clauses are grouped into logical elements (such as a filter function), and variables are named (such as reference state bit). An example visualisation of HiTag2's logical circuit representation is in Fig. 7.7.

Having the logical circuit representation allowed us to implement a dependency-tree walker that removes functions whose output does not contribute to any keystream bits, e.g. the last feedback function in Fig. 7.6. The method used is in essence the same that is used in electric circuit design to remove unused elements, applied to the domain of SAT solver-based cryptanalysis. Removing useless functions gives only a minor speedup of about 1%, however, unnecessary functions no longer show up on the dynamic behaviour analysis statistics, which helps in understanding the inner workings of the solver.

7.3.3 Optimising the representation of LFSRs

Most stream ciphers contain one or more linear feedback shift registers (LFSR). For these ciphers, the state bits not in the reference state can either be deduced by continually applying the forward and backward feedback functions or be directly calculated from the reference state's bits. This latter option increases the interdependency of the resulting equations, which helps the solver generate learnt clauses that are useful for a larger part of the search tree. These learnt clauses are then used later to avoid useless branches of the search tree, reducing the overall search time.

To generate r keystream bits, r distinct states are needed since generating the n-th keystream bit requires the filter function to be applied to the n-th state. For the solving to be fast, we need to choose the reference state that generates the least complex logical circuit representation. In particular, we must minimise both the average depth and the reference state bit dependencies. According to our experience, this optimal reference state is usually near the $r/2$-th state. As an example, if we had taken state 2 (i.e. state bits 2 to 4) as reference in Fig. 7.6., the overall depth of the circuit would have been reduced.

7.3.4 Optimising the representation of non-linear functions

For efficient solving, the number of clauses, the average clause length, and the number of variables should all be low, but often there exists a trade-off between the three properties.

As an example, the simple $\mathbb{GF}(2)$ polynomial

$$x_1 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_3$$

has a Karnaugh table presentation in CNF of

$$\neg x_1 \lor \neg x_3 \qquad \neg x_2 \lor x_3 \qquad x_1 \lor x_2$$

However, the same polynomial can be represented with each non-single monomial expressed as a function, setting additional variables $i_1 \ldots i_3$. The polynomial then becomes

$$x_1 \oplus i_1 \oplus i_2 \oplus i_3$$

Using this representation, the number of clauses increases to 3×3 regular + 1 xor-clause, and the total clause length increases to 10 from 6. Three extra variables also need to be added, diluting the possible learnt clauses with extra variables, thus reducing the effectiveness of learning.

On the other hand, for the $\mathbb{GF}(2)$ polynomial

$$x_1 \oplus x_2 \oplus x_2x_3 \oplus x_4$$

the Karnaugh table presentation in CNF is

$$\begin{array}{rl} \neg x_1 \lor x_2 \lor x_4 & x_1 \lor \neg x_3 \lor x_4 \\ x_1 \lor x_2 \lor x_4 & x_1 \lor \neg x_2 \lor x_3 \lor \neg x_4 \\ \neg x_1 \lor \neg x_2 \lor x_3 \lor x_4 & \end{array}$$

While the same polynomial's other representation, with the non-single monomial expressed as a function, setting additional variable i_1 is

$$x_1 \oplus x_2 \oplus i_1 \oplus x_4$$

With this representation, the number of clauses drops to 3 regular + 1 xor-clause, and the total clause length drops to 7 from 17, while only one extra variable needs to be added.

The trade-offs between the two representation methods are non-trivial. From our experience with Grain, Trivium, Crypto-1 and other ciphers, we find that the Karnaugh-table representation works well for functions that contain few (up to 5-6) variables and where these variables are often repeated in many monomials. For instance, solving HiTag2 and Crypto-1 are both sped up by a factor of up to 9x using the Karnaugh table representation.

When a polynomial can be broken up into sub-functions that do not share variables among themselves, such as the polynomials representing the filter functions of Crypto-1 and HiTag2, then these sub-functions must be modelled separately. This increases the overall depth of the resulting logical circuit, however, the complexity of the individual functions traversed during solving is much lower, which is crucial for the solver.

7.4 Implemented Attacks

The extended SAT solver can solve many stream ciphers. Attacks against three ciphers have been implemented that are faster than any previous SAT solver-based attacks. In this section, we first describe how the attack times were calculated and how the attack could be parallelised then detail the results for each cipher.

7.4.1 Calculating the expected running time

For almost all ciphers, the time to solve for their states is extremely long, even if it takes much less time than the claimed security level would suggest. Therefore, to predict how much time it would take to break a cipher, we first present the *Monte-Carlo method* we used. We then present an improved version of the method of *Guess-and-determine* that lets the attacker efficiently distribute the computational load to multiple independent processes.

Monte-Carlo method

The Monte-Carlo method, first introduced by Metropolis and Ulam [47] is used in many areas of research such as integration and computer security (e.g. the Rabin primality test [52]). It is essentially a randomised algorithm that samples a tiny part of the possibly immense space and processes the results to approximate an unknown value for the whole space. In case of the Rabin primality test, the Monte-Carlo algorithm uses a randomised test to decide if a positive integer is a prime or not. The algorithm has a certain chance ($< 1/4$) to give a false negative result, but running the algorithm many times essentially eliminates the chance that a number is composite. In our case, to let the solver finish within reasonable time, we randomly guess some randomly picked reference state bits and do many runs of these random configurations. The time it takes to solve these randomly picked instances is then averaged.

To improve the randomisation of each run, we replaced MiniSat's pseudorandom number generator (PRNG) with the Mersenne Twister algorithm [45] and used a different randomly picked seed for each run. The original PRNG in MiniSat is a Linear Congruential Generator (LCG) that used the prime 1389796 with period length $2^{32} - 1$, originally suggested by L'Ecuyer and Hellekalek in [37]. The new PRNG, the Mersenne Twister is specifically designed to rectify many of the shortcomings of LCG-based random number generators. In particular, it has a period of $2^{19937} - 1$ and has a very high order of dimensional equidistribution (while LCG-s are limited to 5). The overhead of using this somewhat more complicated algorithm is minimal ($\ll 1\%$), but the benefits are important as they improve the confidence interval of the final results.

To further randomise each run, we also randomly permutate the order of the clauses in MiniSat's memory for each run. This is a very important step in the randomisation, as the order of clauses can (and most of the time do) contribute to the selection of the first, second, etc. propagations and conflicts, and thus are instrumental in deciding the flow of the algorithm.

These randomisation steps have helped us to effectively generate high-quality random instances, and thus evaluate with high confidence the time it takes to break a cipher given a certain number of variables. To approximate the time it takes to attack the cipher without giving any variables we use the following technique. The Monte-Carlo algorithm is run given $G = n, n-1, \ldots n-k$ number of reference state variables, where n is small enough such that the algorithm is not trivial to solve, and $n-k$ is as small as possible such that the resulting system is still solved within a reasonable amount of time. The average time is then plotted against the number of reference state variables given, and the plot is extrapolated to the point where there are no reference state variables given.

Although there is no proof that at any point during $G = n-k-1 \ldots 0$ the graph does not suddenly change, we believe this to be extremely unlikely. For the explication of the reasons, let us first define some notions. Let us define two problems for a given cipher: problem A is when $G = x$, and problem B is where $G = x - 1$, where $n \geq x > 0$ but otherwise x is irrelevant. Let us assume, without loss of generality, that V is the set of reference variables, and V' is the set of reference variables selected to be assigned in B. Let the set of reference variables assigned in A be $V' \bigcup v$. We can now list the reasons why we believe the graph does not deviate from a straight line if the time is plotted in a logarithmic scale:

- Every problem in B can be directly mapped to $2|V \setminus V'| = 2(n-x+1)$ problems in A. Since the underlying algorithm of DPLL-based SAT solvers is essentially an intelligent brute-force, we can safely assume it does not behave worse than a brute-force, and solves the problem B in at most twice the time than solving any problem in A. This is further underlined by our observation that SAT solvers branch on the reference state variables — thus the first branching of the solver when solving B will indeed be a variable from $|V \setminus V'|$
- The more choice of variables a SAT solver has to branch on, the better the dynamic variable branch ordering will work. This means that it is expected of

the solver to solve in less than twice the time problem B with a choice of $n - x$ branch variables, than two problems in A with a choice of $n - x - 1$ branch variables

- Clauses learnt during the solving of A that are independent of the setting of variable v cannot be reused between the solving instances. Therefore, it is expected that problem B can be solved faster than two problems in A, as the solver in the former case does not need to re-learn these same clauses

We have calculated that for ciphers HiTag2 and Crypto-1 the graph showing the solving times versus the number of randomly guessed bits is as predicted until $G = 0$. The interested reader is referred to Appendix 7.B for these graphs.

We note that the possibility of extrapolation is an advancement over previous attempts. Previous attempts failed, as they did not introduce sufficient randomness into the system. This lack of suitable randomisation meant that their results were not extrapolateable [46, 24].

Guess-and-determine

Guess-and-determine is a method to effectively parallelise the solving of ciphers into many independent sub-processes. This method entails iterating a certain, fixed set of reference variables V to all different combinations and executing the SAT solver for each combination. At each execution, the SAT solver's input system has these V variables set, and the solver must be executed with every possible $2^{|V|}$ true-false combination to use the correct combination at least once. The system whose result is satisfiable is the solution seeked by the attacker.

The advantage of the guess-and-determine method is that the attacker can successfully divide the original, difficult problem into $2^{|V|}$ distinct sub-problems, permitting her to run them in parallel, with a master process evaluating the results. This is extremely useful, as it is much cheaper to buy large quantities of commodity hardware than to make a very expensive, fast machine to execute the original problem faster. The expected time to solve the problem is then the average time to solve one system with the V variables set, multiplied by $2^{|V|}$ and divided by the number of processor cores available.

The challenge of the guess-and-determine method is to select the best V variables. The SAT solver automatically selects the variables to branch on, however, the perfection of this ordered list of variables is done throughout the running of the solver. The attacker cannot wait for the perfection to be done for too long, and must determine the best variables with relative speed, as it is just a pre-cursor to her real attack algorithm.

To find the best guess bits, we implemented the function `calcClauseDifficulty` into MiniSat that weights the difficulty of each clause by its length and type (xor or regular). The following weights were found to closely correlate with the impact of different state bits on the attack time:

$$\text{difficulty} = 2 \cdot \text{ no.clauses} + \sum_{\substack{\forall \text{ regular} \\ \text{clauses}}} \text{no. of literals} + 7 \cdot \sum_{\substack{\forall \text{ xor-} \\ \text{clauses}}} \text{no. of literals}$$

Table 7.2: Running times for solving Crypto-1, HiTag2, and Bivium B using various optimisations presented in this chapter

	Vanilla MiniSat	Karnaugh optimisation	Karnaugh and xor-clause optimisations
Crypto-1	500 s	72 s	40 s
HiTag2	$2^{17.8}$ *s*	2^{15} s	$2^{14.5}$ s
Bivium B	$2^{36.7}$ s	$2^{36.7}$ s	$2^{36.5}$ s

The best V variables to set are selected as follows. The CNF system is written down with a candidate variable v assigned. Then the CNF system is reduced using the pre-simplification as explained in Sect. 7.2.1. The above metric is then calculated for both $v =$ `true` and $v =$ `false` and the difficulties are summed. This sum is calculated for all possible candidate variables, and the candidate variable with the lowest sum wins entry into the collection V. To pick the next best variable, this whole process is re-iterated, but the variables in V are always set to some random combination, and the sum is done for many (up to a thousand) different combination settings of variables in V.

7.4.2 The attacks

Our first two targets, Crypto-1 and its relative HiTag2, are weak ciphers used in contactless cards and car immobilisers. The third target, Bivium B, is a simplified version of Trivium, a modern cipher standardised through the eSTREAM competition. The solving times for Crypto-1, HiTag2, and Bivium B are in Table 7.2, and their detailed discussion is below.

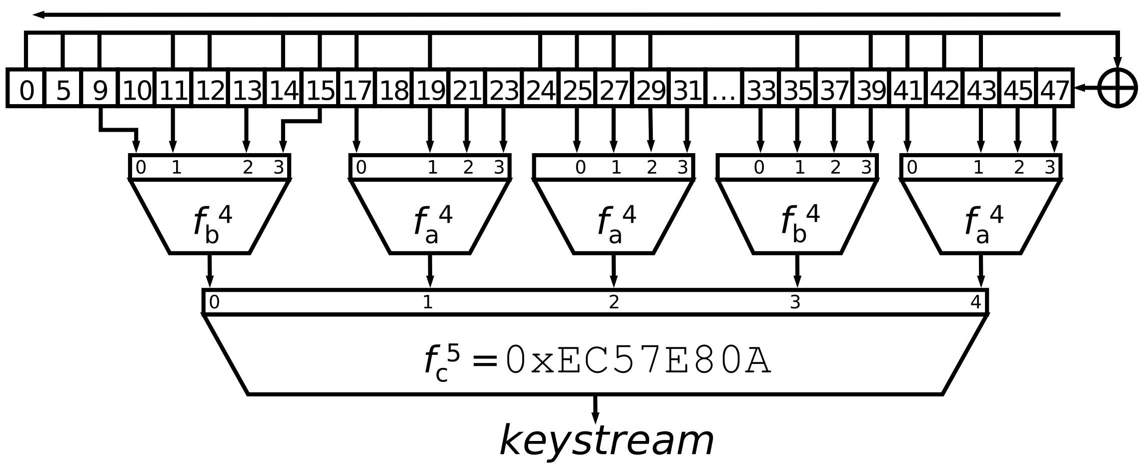
Crypto-1

The Crypto-1 algorithm is a 48-bit stream cipher used on the NXP *Mifare Classic* card. The cipher was designed to have a particularly small hardware footprint consisting of a 48-bit LFSR and a network of small binary functions that form the filter function. Mifare Classic is a contactless card with a maximum reading range of 10 cm. The card is widely used for micropayment in public transport[iii] and building access control.

As detailed in Fig. 7.8., Crypto-1 uses a 48-bit LFSR with 18 feedback taps and a network of small functions making up the filter function that feed off from 20 regularly placed taps. The fact that the taps of the filter function are regularly placed makes the equations particularly easy to solve.

The security of the Mifare Classic encryption has been broken several times already. An early attack by Nohl [50] and an attack by researchers from the Radboud University [20] exploit the card's weak random number generator in combination with cryptographic weaknesses. Courtois et al. reported an algebraic attack [14]

[iii]Mifare Classic cards are used in the public transport systems of the Netherlands, London, Rio de Janeiro, São Paulo, Madrid, Valencia, Oslo, Sydney, Hamilton, Delhi, Nanjing, Shanghai, Taipei, Kuala Lumpur, Atlanta, St. Paul, Houston, Los Angles, Bangkok, and many others



$$f_a^4 = \mathtt{0x9E98} = (a+b)(c+1)(a+d)+(b+1)c+a$$

$$f_b^4 = \mathtt{0xB48E} = (a+c)(a+b+d)+(a+b)cd+b$$

Figure 7.8: NXP *Mifare Classic* Crypto-1 stream cipher. The network of small functions f_a^4, f_b^4 and f_c^5 form the filter function. The numbers on top represent the LFSR states. The $\oplus$ mark at the top right denotes the update function

against the cipher that works independently of the quality of the random numbers. They did not publish the details of their attack but only stated that secret keys can be found within 200 seconds on average on a desktop PC. Their attack modifies the cipher representation in a mathematical way, which makes their techniques mostly orthogonal to ours. Combining the two methods would most likely lead to further improvements. A non-SAT solver-based attack has also been published on Crypto-1 by Garcia et al. [28] that solves for the key by inverting the filter function to arrive at approx. 2^{16} candidate keys, which takes only approx. $O(2^{26})$ operations to break — taking only a fraction of a second on a desktop computer.

On a general-purpose single-core PC, using $r = 56$ bits of known keystream, our SAT solver-based attack breaks the cipher in 40 seconds on average. As the attack can be fully parallelised, the same task only takes 10 seconds on a quad-core processor.

HiTag2

The HiTag2 stream cipher, illustrated in Fig. 7.9., is closely related to Crypto-1 and is used in RFID tags by NXP. HiTag2 and Crypto-1 share the same general structure and key length, but differ in their feedback and filter functions, and their filter function taps. In particular, the irregularly placed taps to the filter function make HiTag2 much harder to solve than Crypto-1: the SAT solver needs to guess more variables before the equations start leading to contradictions. Since HiTag2 succeeds Crypto-1, it can be assumed that the changes to the cipher were made to improve the cryptographic complexity without increasing hardware costs.

The HiTag2 cipher is harder to break in comparison with Crypto-1: using $r = 56$

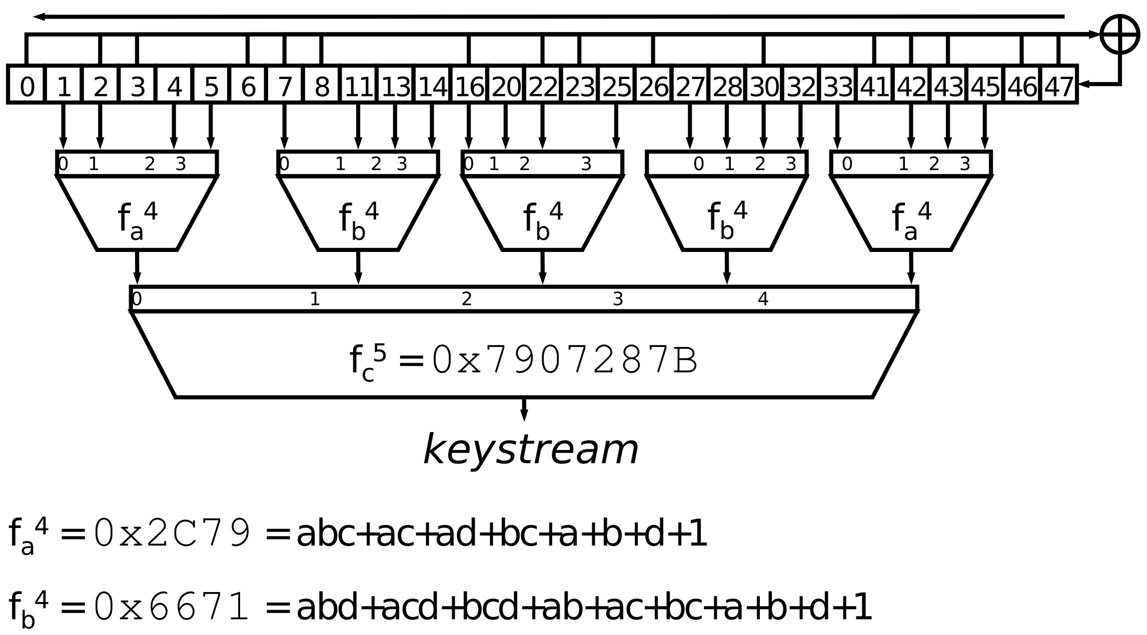


Figure 7.9: NXP HiTag2 stream cipher. The network of small functions f_a^4, f_b^4 and f_c^5 form the filter function. The numbers on top represent the LFSR states. The $\oplus$ mark at the top right denotes the update function

bits of known keystream, recovering the LFSR state takes about 6.5 hours on average using a single-core desktop PC. As with Crypto-1, the attack can be parallelised and so on a quad-core Xeon it only takes about 1h 40m on average.

Bivium B

The Bivium B stream cipher, put forward by Raddum [53] for research purposes is illustrated in Algorithm 6. It is a reduced version of the original Trivium cipher by Cannière [10], and is intended to be used solely as a research tool to analyse the original cipher. The research papers that try to solve Bivium B using SAT solver-based techniques [53, 24, 46] improve on each other's results, the best of which is solving on average in $2^{42.7}$ s on a desktop machine. There are research papers that attack Bivium B in other ways, mainly through other algebraic techniques. One such attack is by Borghoff et al. [8] who treat Bivium B as a mixed-0-1 linear programming problem and estimate to find its state in about $2^{64.5}$ s. Another algebraic attack is the cube attack by Dinur and Shamir [21] who give results only for Trivium, but mention Bivium B as a possible attack candidate, as their techniques should be applicable to Bivium B in such a way as the total time[iv] to break it would be lower than our results with the adapted MiniSat.

Bivium B can be solved by describing it in MiniSat using the enhancements and insight presented in this chapter. Due to the reasons outlined in Sect. 7.4.1 we can extrapolate the graph showing its solving times in Fig. 7.10., giving the result that solving the cipher's state given 177 keystream bits takes about $2^{36.5}$ s on average.

To generate this result, Gaussian elimination was turned off, as it proved to slow

[iv]Including pre-processing, the most time-consuming phase of the cube attack

Algorithm 6: Description of Bivium B's keystream generation. Bivium B has 177 states, (s_1,s_2,..., s_{177}), and the keystream generated is (z_1, ..., z_r)

```
   Input: ($s_1$,$s_2$,..., $s_{177}$), $N$
   Output: $z_1$, $z_2$, ..., $z_r$
1  for $i = 1$ to $r$ do
2      $t_1 \leftarrow s_{66} \oplus s_{93}$;
3      $t_2 \leftarrow s_{162} \oplus s_{177}$;
4      $z_i \leftarrow t_1 \oplus t_2$;
5      $t_1 \leftarrow t_1 \oplus (s_{91} \wedge s_{92}) \oplus s_{171}$;
6      $t_2 \leftarrow t_2 \oplus (s_{175} \wedge s_{176}) \oplus s_{69}$;
7      $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_2, s_1, \ldots s_{92})$;
8      $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots s_{176})$;
9  end
```

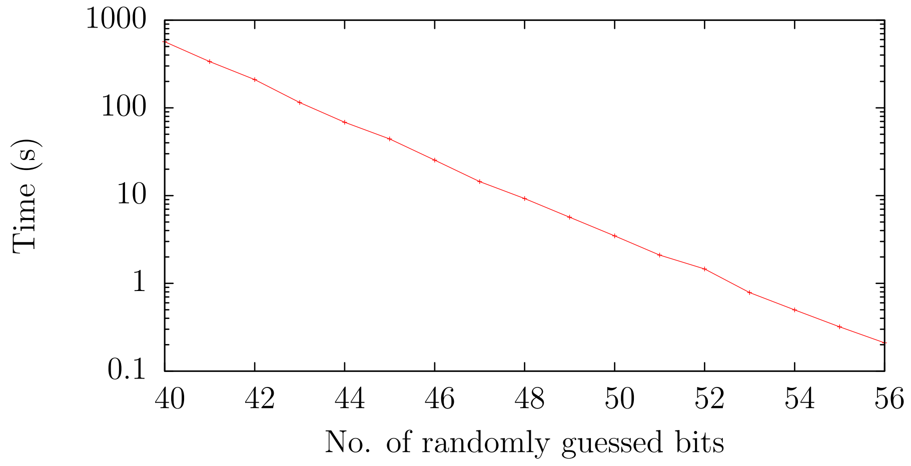


Figure 7.10: Solving the Bivium B cipher 1000 times and averaging the results. Guess bits were randomly selected and assigned. The time to solve is exponential in the number of guess bits. Extrapolating the graph to zero on the x axis, the time to solve is $2^{36.5}$ s

down the solver if less than 58 reference state bits were guessed — for more than 58 guessed bits however, Gaussian elimination with cut-off depth 8 gave an average 5% speedup.

7.5 Conclusions

SAT solvers are powerful tools in the analysis of mathematical assumptions, including cryptographic hardness and complexity assumptions. The full potential of SAT solving can only be achieved by matching the problem description to the solver language. For cryptographic ciphers, matching the solver and the problem requires extensive changes to the solver itself. We implemented several steps towards a specialised SAT solver for cryptography including native support for the XOR operation, Gaussian elimination, and logical circuit generation.

The extended solver solves problems from its target domain, simple and complex stream ciphers, faster than any other known SAT-solver based techniques. The Crypto-1 cipher is solved in 40 seconds, HiTag2 in $2^{14.5}$ s, while Bivium B takes $2^{36.5}$ s, 2^6 times less than the previous best SAT solver-based attack [46]. Stream ciphers can be strengthened against the attacks presented in this chapter through the use of larger states, more complex feedback functions, and through longer initialisation phases.

7.A Learnt clause length statistics

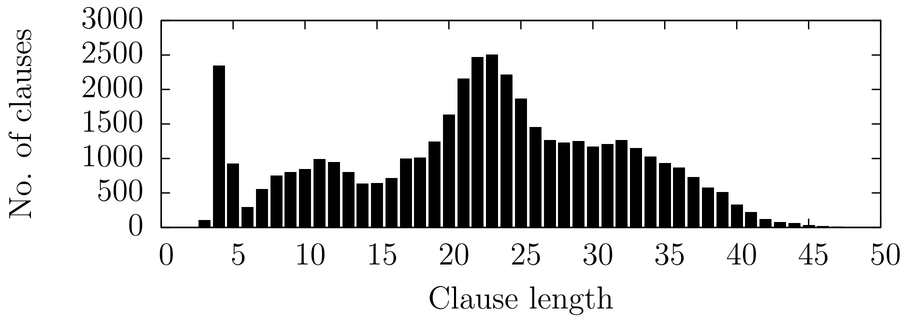


Figure 7.11: Plot showing the distribution of the size of the learnt clauses while solving the HiTag2 stream cipher with our techniques. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

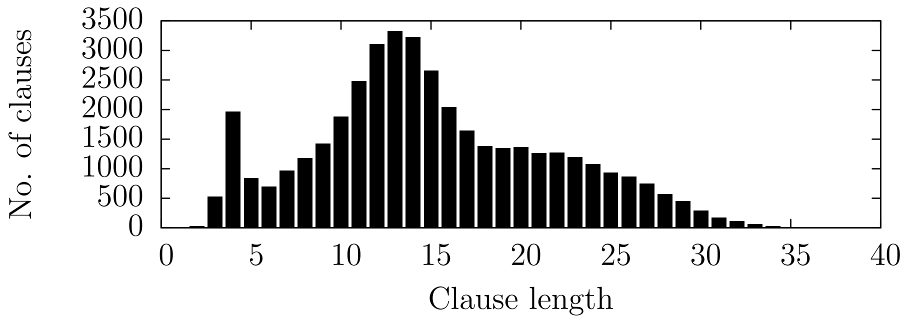


Figure 7.12: Plot showing the distribution of the size of the learnt clauses while solving the HiTag2 stream cipher with our techniques when given 10 randomly set and randomly picked reference state bits. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

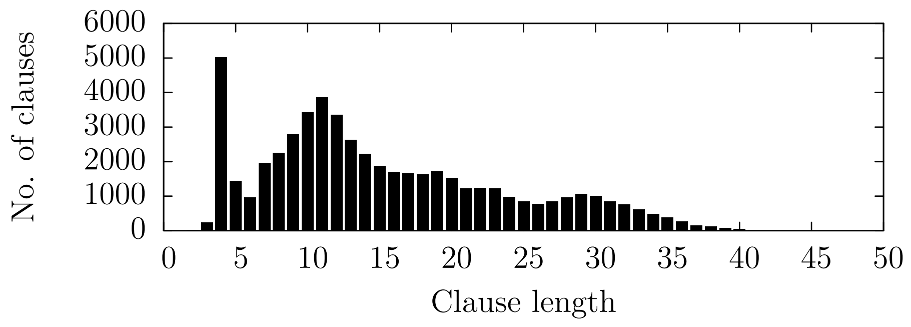


Figure 7.13: Plot showing the distribution of the size of the learnt clauses while solving the Crypto-1 stream cipher with our techniques. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

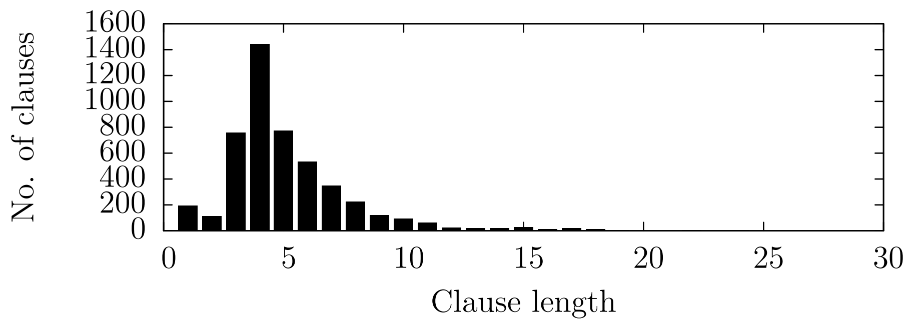


Figure 7.14: Plot showing the distribution of the size of the learnt clauses while solving the Crypto-1 stream cipher with our techniques when given 10 randomly set and randomly picked reference state bits. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

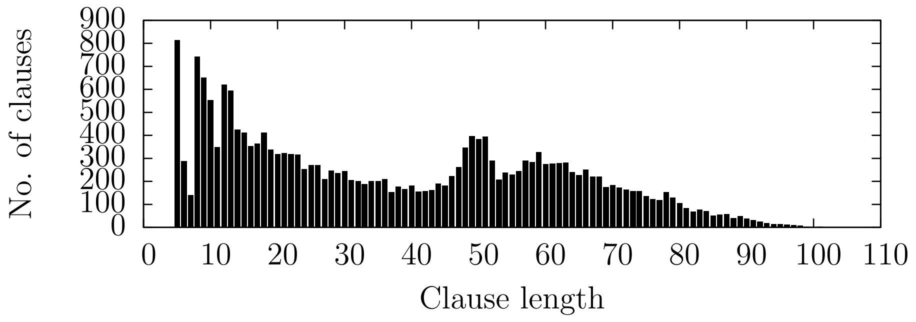


Figure 7.15: Plot showing the distribution of the size of the learnt clauses while solving the Bivium B stream cipher with our techniques. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

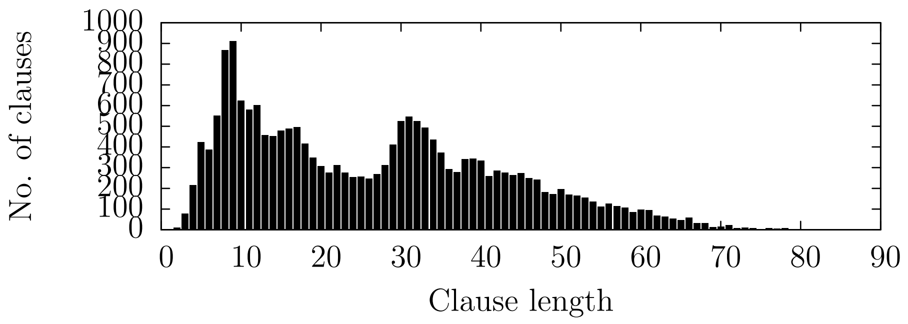


Figure 7.16: Plot showing the distribution of the size of the learnt clauses while solving the Bivium B stream cipher with our techniques when given 20 randomly set and randomly picked reference state bits. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

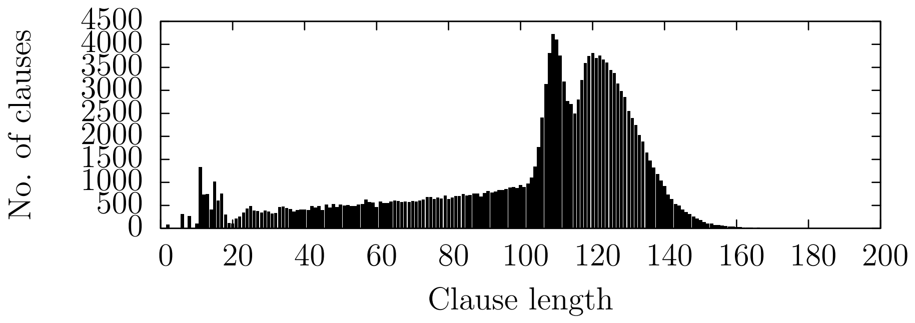


Figure 7.17: Plot showing the distribution of the size of the learnt clauses while solving the Grain stream cipher with our techniques. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

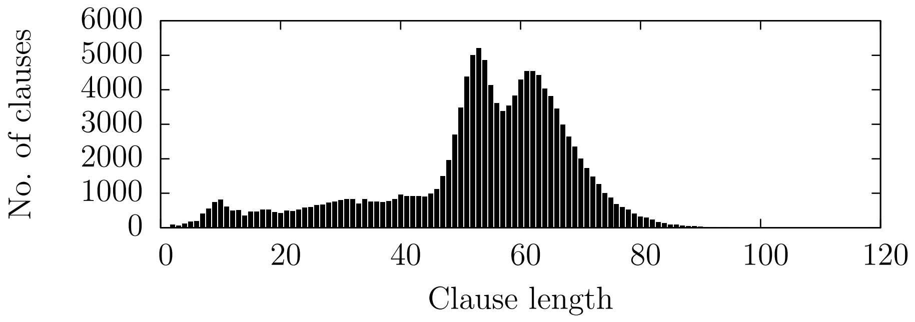


Figure 7.18: Plot showing the distribution of the size of the learnt clauses while solving the Grain stream cipher with our techniques when given 60 randomly set and randomly picked reference state bits. To produce the figure, solving of the cipher was started 20 times, each time interrupting the solving process after 13 restarts. The number and length of the learnt clauses were then summed up for the runs and plotted using Graphviz.

7.B HiTag2 and Crypto-1 extrapolation examples

It is interesting to observe that the time it takes to solve both HiTag2 and Crypto1 is less if we give $G = 0$ help bits than the extrapolation of the graphs in Fig. 7.19 and Fig. 7.20 would indicate. This is because at $G = 0$, all problem instances are satisfiable, but at $G = 1$ only half of them, and at $G = x$, only 2^{-x} portion of them are satisfiable. Instances that are UNSAT take approximately twice the time to solve than satisfiable instances as have been observed by others [46] — this is because on average, the satisfying solution is found in the middle of the search space, while to find UNSAT, all of the search space must be exhausted. If every point in the graphs are compensated for this, graphically they all form a straight line.

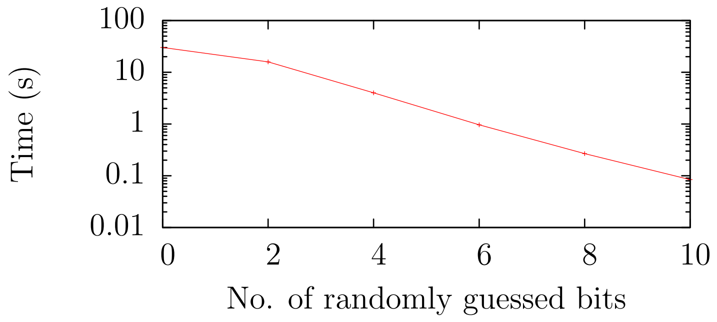


Figure 7.19: Plot showing the average time it took to solve the Crypto-1 stream cipher given different numbers of randomly picked and randomly set reference bits. We used one core of a desktop computer with the following specifications: Intel Xeon E5345@2.33GHz, 4MB cache, 4GB memory

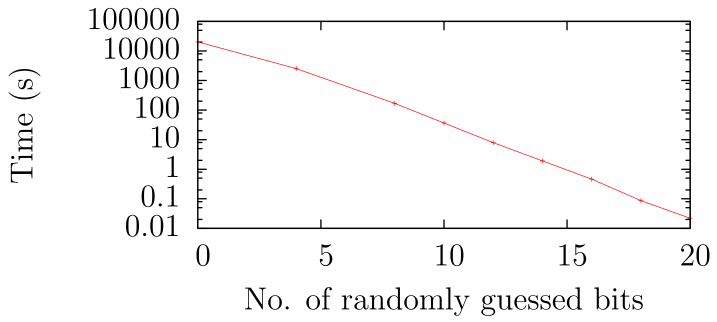


Figure 7.20: Plot showing the average time it took to solve the HiTag2 stream cipher given different numbers of randomly picked and randomly set reference bits. We used one core of a desktop computer with the following specifications: Intel Xeon E5345@2.33GHz, 4MB cache, 4GB memory

Bibliography

[1] Anderson, R. A5 (was: Hacking digital phones). Newsgroup Communication, 1994.

[2] Babbage, S., Canniere, C. D., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Paar, C., Parker, M., Preneel, B., Rijmen, V., Robshaw, M., and Wu, H. The eSTREAM portfolio. Tech. rep., eStream Project, September 2008.

[3] Babbage, S., and Dodd, M. The MICKEY stream ciphers. In *New Stream Cipher Designs: The eSTREAM Finalists* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 191–209.

[4] Bard, G. V. Algorithms for the solution of polynomial and linear systems of equations over finite fields, with an application to the cryptanalysis of KeeLoq. Tech. rep., University of Maryland Dissertation, April 2008. Ph.D. Thesis.

[5] Bard, G. V. *Algebraic Cryptanalysis*, vol. XXXIV of *Security and Cryptology*. Springer, 2009.

[6] Bard, G. V., Courtois, N. T., and Jefferson, C. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. Cryptology ePrint Archive, Report 2007/024, http://eprint.iacr.org/2007/024, 2007.

[7] Baumgartner, P., and Massacci, F. The taming of the (X)OR. In *Computational Logic — CL 2000* (2000), vol. 1861/2000 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 508–522.

[8] Borghoff, J., Knudsen, L. R., and Stolpe, M. Bivium as a mixed-0-1 linear programming problem. In *WEWoRC — Western European Workshop on Research in Cryptology* (Graz, Austria, July 2009).

[9] Bosma, W., Cannon, J., and Matthews, G. Programming with algebraic structures: design of the MAGMA language. In *Proceedings of the international symposium on Symbolic and algebraic computation — ISSAC '94* (New York, NY, USA, 1994), ACM, pp. 52–57.

[10] Cannière, C. D. Trivium: A stream cipher construction inspired by block cipher design principles. In *ISC* (2006), S. K. Katsikas and et al, Eds., vol. 4176 of *LNCS*, Springer, pp. 171–186.

[11] Courtois, N., Bard, G. V., and Wagner, D. Algebraic and slide attacks on KeeLoq. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, pp. 97–115.

[12] Courtois, N. T. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology — CRYPTO 2003* (2003), vol. 2729/2003 of *LNCS*, Springer, pp. 176–194.

[13] Courtois, N. T., and Bard, G. V. Algebraic cryptanalysis of the Data Encryption Standard. In *IMA Int. Conf.* (2007), S. D. Galbraith, Ed., vol. 4887 of *Lecture Notes in Computer Science*, Springer, pp. 152–169.

[14] Courtois, N. T., Nohl, K., and O'Neil, S. Algebraic attacks on the Crypto-1 stream cipher in Mifare Classic and Oyster cards. Tech. Rep. 2008/166, Cryptology ePrint Archive, 2008.

[15] Daemen, J., and Rijmen, V. Rijndael/aes. In *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed. Springer, 2005.

[16] Davis, M., and Putnam, H. A computing procedure for quantification theory. *J. ACM 7*, 3 (1960), 201–215.

[17] Dawiche, A. New advances in compiling CNF to decomposable negation normal form. In *Proc. of European Conference on Artificial Intelligence* (2004), pp. 328 – 332.

[18] de Koning Gans, G., Hoepman, J.-H., and Garcia, F. D. A practical attack on the MIFARE Classic. In *CARDIS* (2008), G. Grimaud and F.-X. Standaert, Eds., vol. 5189 of *Lecture Notes in Computer Science*, Springer, pp. 267–282.

[19] Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (1976), 644–654.

[20] Digital Security group, Radboud University Nijmegen. Security flaw in Mifare Classic. Press release, March 2008. http://www.ru.nl/english/general/radboud_university/vm/security_flaw_in/.

[21] Dinur, I., and Shamir, A. Cube attacks on tweakable black box polynomials. In *EUROCRYPT* (2009), A. Joux, Ed., vol. 5479 of *Lecture Notes in Computer Science*, Springer, pp. 278–299.

[22] Eén, N., and Sörensson, N. An extensible SAT-solver. In *SAT* (2003), E. Giunchiglia and A. Tacchella, Eds., vol. 2919 of *LNCS*, Springer, pp. 502–518.

[23] Eén, N., and Sörensson, N. Temporal induction by incremental SAT solving. In *Proc. of First Intrenational Workshop on Bounded Model Checking* (2003), vol. 89 of *ENTCS*, Elsevier.

[24] Eibach, T., Pilz, E., and Völkel, G. Attacking Bivium using SAT solvers. In *SAT* (2008), H. K. Büning and X. Zhao, Eds., vol. 4996 of *LNCS*, Springer, pp. 63–76.

[25] Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., and Woodhull, G. Graphviz — open source graph drawing tools. In *Graph Drawing* (2001), P. Mutzel, M. Jünger, and S. Leipert, Eds., vol. 2265 of *Lecture Notes in Computer Science*, Springer, pp. 483–484.

[26] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra 1*, 139 (June 1999), 61–88.

[27] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *ISSAC '02* (2002), ACM, pp. 75–83.

[28] Garcia, F. D., and et al. Dismantling MIFARE Classic. In *ESORICS* (2008), S. Jajodia and J. López, Eds., vol. 5283 of *LNCS*, Springer, pp. 97–114.

[29] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. S-Match: an algorithm and an implementation of semantic matching. In *Semantic Interoperability and Integration* (2005), Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, Eds., no. 04391 in Dagstuhl Seminar Proceedings, IBFI. http://drops.dagstuhl.de/opus/volltexte/2005/37.

[30] Gomes, C. P., Selman, B., Crato, N., and Kautz, H. A. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning 24*, 1/2 (2000), 67–100.

[31] Greuel, G.-M., Pfister, G., and Schönemann, H. SINGULAR — a computer algebra system for polynomial computations. In *Symbolic computation and automated reasoning* (Natick, MA, USA, 2001), A. K. Peters, Ltd., pp. 227–233.

[32] Hell, M., Johansson, T., and Meier, W. Grain — a stream cipher for constrained environments. In *Proceeding of the Workshop on RFID and Lightweight Crypto* (July 2005), pp. 114–125.

[33] Heras, F., Larrosa, J., and Oliveras, A. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research 31* (2008), 1–32.

[34] Hsieh, H. Y., and Ghausi, M. S. On optimal-pivoting algorithms in sparse matrices. *IEEE Trans. Circuit Theory CT-19* (January 1972), 93–96.

[35] Karnaugh, M. The map method for synthesis of combinational logic circuits. *Transactions of American Institute of Electrical Engineers part I 72*, 9 (November 1953), 593–599.

[36] Kibria, R. MidiSat - An extension of MiniSAT. Tech. rep., Department of Electrical and Computer Engineering, Darmstadt University of Technology, April 2005.

[37] L'Ecuyer, P., and Hellekalek, P. Random number generators: Selection criteria and testing. In *Random and Quasi-Random Point Sets* (New York, 1998), vol. 138 of *Lecture Notes in Statistics*, Springer-Verlag, pp. 223–266.

[38] Leonardo de Moura, B. D., and Shankar, N. A tutorial on satisfiability modulo theories. In *Computer Aided Verification* (2007), vol. 4590/2007 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 20–36.

[39] Li, C. M. Equivalency reasoning to solve a class of hard SAT problems. In *Information Processing Letters* (1999), pp. 76–1.

[40] Li, C. M. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* (2000), AAAI Press / The MIT Press, pp. 291–296.

[41] Malik, S., Zhao, Y., Madigan, C. F., Zhang, L., and Moskewicz, M. W. Chaff: Engineering an efficient SAT solver. *Design Automation Conference* (2001), 530–535.

[42] Mandelbrot, B. B. The pareto-lévy law and the distribution of income. *Internat. Econom. Rev. 1* (1960), 79–106.

[43] Massacci, F. Using Walk-SAT and Rel-sat for cryptographic key search. In *Proc. of IJCAI-99* (1999), M. Kaufmann, Ed., pp. 290–295.

[44] Massacci, F., and Marraro, L. Logical cryptanalysis as a SAT-problem: Encoding and analysis. *Journal of Automated Reasoning 24* (2000), 165–203.

[45] Matsumoto, M., and Nishimura, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul. 8*, 1 (January 1998), 3–30.

[46] McDonald, C., Charnes, C., and Pieprzyk, J. Attacking Bivium with MiniSat. Tech. Rep. 2007/040, ECRYPT Stream Cipher Project, 2007.

[47] Metropolis, N., and Ulam, S. The Monte Carlo method. *Journal of the American Statistical Association 44*, 247 (1949), 335–341.

[48] Molnar, D., and Wagner, D. Privacy and security in library RFID: issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), ACM Press, pp. 210–219.

[49] National Bureau of Standards. Data Encryption Standard, 1977.

[50] Nohl, K. Cryptanalysis of Crypto-1. Press release, March 2008. http://www.cs.virginia.edu/~kn5f/Mifare.Cryptanalysis.htm.

[51] Nohl, K. Description of HiTag2. Press release, March 2008. http://cryptolib.com/ciphers/hitag2/.

[52] Rabin, M. O. Probabilistic algorithm for testing primality. *J. Number Theory 12*, 1 (1980), 128–138.

[53] Raddum, H. Cryptanalytic results on Trivium. Tech. Rep. 2006/039, ECRYPT Stream Cipher Project, 2006. www.ecrypt.eu.org/stream/papersdir/2006/039.ps.

[54] Raddum, H., and Semae, I. New technique for solving sparse equation systems, January 2006. eprint.iacr.org/2006/475/.

[55] Rivest, R. L., Shamir, A., and Adleman, L. M. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21* (1978), 120–126.

[56] Robshaw, M., and Billet, O., Eds. *The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[57] Silva, J. P. M., and Sakallah, K. A. GRASP-a new search algorithm for satisfiability. In *ICCAD'96* (1996), IEEE Computer Society, pp. 220–227.

[58] Sinz, C. Visualizing SAT instances and runs of the DPLL algorithm. *J. Autom. Reason. 39*, 2 (2007), 219–243.

[59] Sinz, C., and Dieringer, E.-M. DPvis — A tool to visualize the structure of SAT instances. In *SAT* (2005), F. Bacchus and T. Walsh, Eds., vol. 3569 of *Lecture Notes in Computer Science*, Springer, pp. 257–268.

[60] The SAGE Group. SAGE Mathematics Software (Version 3.1.2), 2008.

[61] Warners, J. P., and Maaren, H. V. A two phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters 23*, 3–5 (1999), 81–88.

[62] Wieringa, S. MiniMarch — Embedding lookahead direction heuristics in a conflict driven solver. Tech. rep., Technische Universiteit Delft, 2007. Research Report.

Conclusions

In this thesis we have described the RFID hardware and software layers, iterating through all major RFID security protocols, highlighting their insecurities and shortcomings. Next, we analysed the Di Pietro-Molva scheme, uncovering some insecurities and obscure features of the protocol. Then, we proposed two new RFID identification schemes, the ProbIP and EProbIP schemes, highlighting how EProbIP has been secured against attacks that were possible against ProbIP. Next, we demonstrated the use of stream ciphers in RFIDs, and finally, adopted SAT solvers to analyse low hardware-complexity stream ciphers, used in RFID protocols.

We believe that in the coming years, the implemented RFID protocols will be based on lightweight versions of established crypto-primitives, but in the long run, they may well be based on protocols that are currently in an experimental state, such as the Di Pietro-Molva or the EProbIP schemes that have been elaborated upon in this thesis. These experimental protocols could offer tailor-made solutions to RFIDs, while protocols that are based on standard crypto-primitives must always retain some of their disadvantages. However, as we have seen in both the case of the Di Pietro-Molva and the ProbIP protocols, experimental protocols are at the moment quite fragile from a security standpoint, thus their large-scale adoption will have to wait until they have been shown to stand the scrutiny of cryptographers.

It is possible, that when experimental RFID protocols finally achieve the status of being well-tested and secure, they will resemble lightweight crypto-primitives that are available today. The reasoning behind this is that as experimental protocols get redesigned again and again, their complexity gradually increases to counter the attacks found against them. This kind of feature-creep is in fact very characteristic of experimental protocols. For example, HB^+ started out as an extremely lightweight protocol, but its subsequent re-designs HB^{++} and $HB^{\#}$ have each added an extra layer of complexity, going as far as having a very large (though special) matrix stored inside the tag for $HB^{\#}$. With so many re-designs and extra layers of complexity added, experimental protocols can easily loose their edge over traditional cipher designs.

However, even if ad-hoc protocols finally loose their advantage over standard ciphers, the research activity invested in such protocols has not been in vain. Without such research, it would be impossible to tell if the research community has not missed out on a design that could have revolutionised the field. For example, the HB^+ class of protocols have brought a fresh idea into the area of cryptographic research.

The RFID protocol that is finally selected and implemented can make a large difference on the acceptability of RFID systems. For instance, if the implemented protocol is found to be insecure but is already active and in widespread use (i.e. the tags have not been deactivated as is now customary), RFIDs could suffer a large backlash, with potentially fateful consequences. On the other hand, if the protocol is found to be reasonably secure over time, RFIDs could enjoy a large boom with tags in all, and readers in many consumer products such as intelligent washing machines, refrigerators, etc. The choice of the protocol is therefore crucial for the long-term use of RFIDs in sectors other than supply chain management, the sole sector EPC tags are used in at the moment.

If the choice of the RFID protocol finally selected is wise, many day-to-day chores of people could be eliminated. For example, checkout counters in large supermarkets could be replaced with self-checkout counters, where the RFIDs on the items bought could indicate the final price to be paid by the customers. Similar repetitive tasks, such as checking the "best before" tags on items in the refrigerator, or making sure that colours are not mixed in the drum of the washing machine, could be eliminated.

Although such uses seem innocuous, users and researchers must always watch out for insecurities and feature-creeps in the use of RFIDs: there is a large responsibility on researchers in the field for investigating and making sure that schemes that are not secure, such as the Crypto-1 cipher, are discovered and dealt with. Without proper investigation of these protocols, businesses could tell half-truths about their products, endangering other businesses' and people's lives and livelihoods. For example, the Mifare card is marketed as an access-control device, and many institutions have installed it to such effect, even though its security material (the cryptographic key) can be copied in the matter of seconds, without prior approval (or even a remote chance of discovery) by the owner. As another, somewhat more disturbing example, passports have recently been tagged with RFIDs, which later have been shown to be leaking information even from a relatively large distance. After this discovery, American passports have been augmented with an integrated wire mesh, acting as a Faraday cage, but European passports still do not implement this feature. These and similar problems occur when RFIDs are implemented either wrongly, or for the wrong purposes. Therefore, public scrutiny and extensive, public research on RFIDs is of essence not only to ensure the security of RFIDs, but to ensure the security and liberty of society as a whole.

Annexe A

Protocoles de protection de la vie privée et de sécurité pour les RFIDs

A.1 Introduction

Les puces RFID, c'est-à-dire les systèmes d'identification à radio-fréquence se montrent capables de substituer à l'avenir les systèmes de codes barres appliqués de nos jours. Ce changement implique de nombreuses possibilités : un contrôle plus minutieux des stockages, une lecture plus rapide des codes barres ou bien un service de facture sans papier inutile. En revanche, les systèmes RFID utilisés actuellement présentent de multiples inconvénients susceptibles d'entraver leur extension à l'avenir : entre autres, ils peuvent s'avérer dangereux pour la vie privée, de plus, la sécurité de leur authenticité n'est pas résolue.

Durant mon doctorat, j'ai fait connaissance avec les protocoles qui concernent les problèmes soulevés par les RFID (Premier chapitre). Un article dans lequel j'ai créé un nouveau protocole de sécurité a été publié (Deuxième chapitre). Ensuite, j'ai publié un article présentant les imperfections d'un protocole déjà publié (Troisième chapitre). Finalement, j'ai publié un article dans lequel je transforme les algorithmes SAT de telle sorte qu'ils parviennent à résoudre les problèmes posés par des chiffrements par flot au faible coût d'implementation utilisables potentiellement dans les RFID.

Organisation

Ce résumé de thèse est organisé comme il suit. Dans la première section nous décrivons les protocoles de sécurité les plus courants. Puis, dans la seconde section, les protocoles ProbIP et EProbIP sont décrits, leurs différences sont mises en évidence. En particulier nous montrons pourquoi EProbIP est plus sûr. Dans la Section 3 nous décrivons et analysons le protocole d'identification anonyme Di Pietro-Molva. Puis, dans la section 4, nous décrivons comment les systèmes de chiffrement par flot dédiés aux implémentations matérielles légères, qui sont souvent utilisés dans les RFID, peuvent être analysés grâce a des solveurs de problèmes SAT adaptés. Finalement, nous concluons cette thèse.

A.2 Les protocoles RFID

Les protocoles RFID tentent de résoudre deux problématiques importantes : l'identification privée et l'authentification. Nous allons diviser les protocoles selon ces deux parties cruciales.

A.2.1 Identification

Le problème d'identification est le suivant : l'entité A souhaite signaler à l'entité B, d'après un système de signe établi à l'avance, qu'elle est A. L'identification selon le standard EPC fonctionne de la façon suivante : la puce EPC envoie son numéro d'identification par un canal de radio. C'est bien une solution du problème de base, néanmoins il permet de suivre la puce EPC et à la fois son soutien (même de loin) contre un attaquant à l'écoute.

La solution idéale de ce problème est l'identification privée, dans laquelle une puce EPC dévoile son identité au lecteur de sorte que personne d'autre ne puisse la décoder. L'identification privée possède de nombreuses définitions. Une définition bien connue a été forgée par Juels et al. [23]. Dans celle-ci, ils définissent une expérience d'identification privée qui doit être gagnée par l'attaquant selon une probabilité non négligeable afin qu'il s'avère que le protocole RFID d'identification privée ne protège pas la vie privée et que la puce peuvent être suivie.

Nous distinguons quatre types différents de protocole RFID d'identification privée : les types basés sur la théorie d'information, sur le hachage, sur l'arbre des clés et les protocoles expérimentaux.

Protocoles de théorie d'information

Le meilleur exemple des protocoles de théorie d'information est le protocole à la chaîne de pseudonymes instauré par Juels [21]. Ce protocole fonctionne de telle sorte que la puce RFID retient n numéros d'identification (ID) différents, et elle les émet les uns après les autres en cas de demande d'identification. Dans la mesure où la puce RFID rencontre un lecteur avec lequel elle parvient à créer une authenticité, ils se mettent d'accord sur n ID qui peuvent être utilisés de nouveau. Le protocole exige extrêmement peu de calculations, et il suffit n banques de mémoire pour l'implementation. En revanche, le protocole recèle un problème, c'est que l'attaquant a l'occasion d'entrer plusieurs fois ($> n$) en contact avec la puce et de cette façon, le numéro d'identification de la puce ne sera pas inconnu après un certain temps, par conséquent, il deviendra possible de suivre la puce.

Protocoles basés sur le hachage

L'exemple le plus connu des protocoles basés sur le hachage est le protocole OSK établi par Ohkubo et al. [24]. Le protocole OSK fonctionne de la manière suivante :

1. La puce envoie une réponse d'identification : $a_i = G(s_i)$
2. La puce met à jour son état : $s_{i+1} = H(s_i)$

où G et H sont deux fonctions de hachage différentes qui peuvent être générées à partir d'une même fonction de hachage, par exemple de la façon suivante : nous

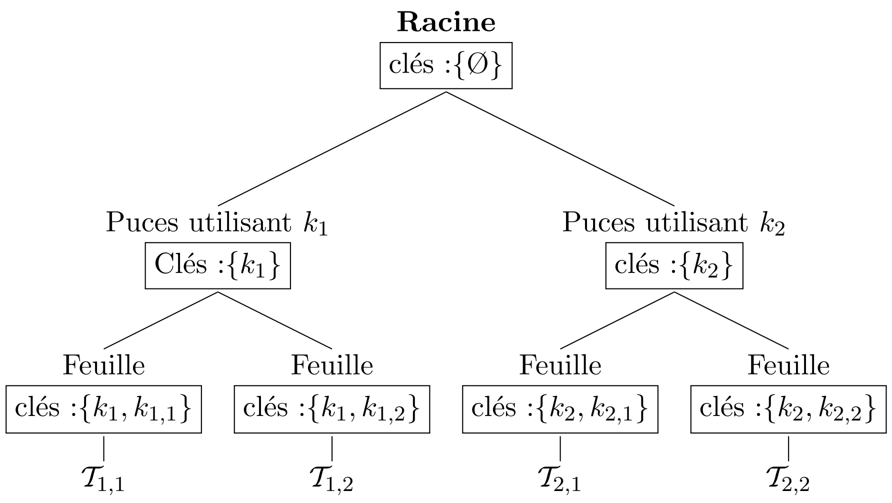


FIG. A.1 – Un exemple de l'arbre des clés de Molnar-Wagner. La facteur de branchement de l'arbre est b deux et sa profondeur est également d deux. Il y a ainsi quatre puces au total dans le système.

mettons '1' à la première entrée et '0' à l'autre. Ce protocole s'appelle forward-secure, du moment que si l'attaquant connaît n'importe quand l'état de la puce, il ne parvient pas à en décoder les identifications anciennes. Le protocole sollicite du lecteur de suivre l'état des puces et de mettre à jour l'état correspondant à la puce. L'inconvénient du protocole se montre dans la mesure où l'attaquant identifie la puce à plusieurs reprises. Dans ce cas-là, l'état de la puce est mis à jour tant de fois que le lecteur doit mettre beaucoup de temps de le chercher.

À part le protocole OSK, il existe de nombreux protocoles basés sur le hachage qui ont été, presque tous, développés à partir d'OSK. Entre autres, c'est YA-TRAP de Tsudik [39], YA-TRAP+ et O-TRAP de Burmester et al. [8] et RIPP-FS de Conti et al. [11].

Protocoles basés sur l'arbre des clés

Le fondement des protocoles basés sur l'arbre des clés est le protocole Molnar-Wagner [26]. Ce protocole ordonne les puces dans un arbre des clés où les feuilles de l'arbre sont les puces et une clé différente correspond à chaque branche. Une partie des clés des puces se trouvant près de l'une des autres est identique, tandis que les clés des puces se situant loin de l'une des autres ne sont pas ou à peine identiques. Ce système de partage des clés (voir la Figure A.1.) est profitable, car le lecteur peut identifier (et optionnellement authentifier) les puces en $\log n$ temps. Le partage des clés aboutit, en revanche, au dévoilement des clés des puces intactes, vu que certaines clés se trouvent dans plusieurs puces. Alors le dévoilement d'une puce (par exemple par le biais de l'ouverture physique) peut entraîner le dévoilement d'autres puces. De cette manière, l'attaquant a la possibilité d'endommager l'identification privée des puces qu'il n'a jamais ouvertes, voire, qu'il n'a jamais vues.

Protocoles expérimentaux

Les protocoles expérimentaux sont établis sur ce que les conditions des puces RFID et leurs modes de fonctionnement diffèrent de ce qui est habituel dans la cryptographie à tel point que l'on a besoin de nouveaux protocoles jusqu'ici totalement inconnus susceptibles de fournir une solution aux questions du domaine de recherche très spécifique. Les deux exemples célèbres de tels protocoles sont LMAP de Peris et al. [32] et DPM de Pietro et Molva [2].

L'avantage de ces protocoles est qu'ils sont de faible demande matérielle, par conséquent, une puce RFID peut en contenir un. Leur désavantage, par contre, c'est qu'ils ne s'avèrent pas toujours assez fiables : c'est ce qui est advenu à LMAP [1] et à DPM [2].

A.2.2 Authentification

Bien que l'identification soit un problème de premier ordre attendant d'être résolu par les protocoles RFID afin qu'ils puissent prendre de l'extension dans la vie quotidienne, la capacité d'authentification des RFID pourrait être utilisable plusieurs fois. Elle serait incontestablement apte à entraver l'extension de faux médicaments et des contrefaçons des produits de luxe, ainsi qu'à assurer la réparation des produits garantis sans besoin de ticket de caisse et de facture. Nous distingue quatre protocoles d'authentification RFID différents : les protocoles appliquant le chiffrement symétrique, les protocoles utilisant le chiffrement public, les protocoles basés sur PUF et ceux basés sur LPN.

Protocoles appliquant le chiffrement symétrique

Dans la mesure où il y a un un chiffrement symétrique tel qu'AES n'exigeant que 3500 GE [15] ou bien PRESENT [5] de Bogdanov et al., qui exige uniquement 1570, le problème de l'authentification peut être aisément résolu. C'est la raison pour laquelle de nombreuses recherches se penchent sur les protocoles appliquant le chiffrement symétrique de faible demande matérielle. Par exemple, le projet européen NESSIE et eSTREAM, qui l'a développé, se préoccupent de ce sujet : Trivium [9] de Canniere et Grain [19] de Hell et al. ont vu le jour dans le cadre du deuxième projet.

Protocoles utilisant le chiffrement public

L'utilisation du chiffrement publique aux RFID présente de multiples avantages, entre autres, il devient possible de partager la partie publique de la clé à n'importe qui, même aux entités en lesquelles le planificateur de système n'a pas de confiance totale. Son inconvénient est, part contre, que ce protocole est de grande demande matérielle. Cette grande demande matérielle se laisse corriger de deux façons : soit nous appliquons des algorithmes standards avec des coupons utilisables juste une fois, soit nous transformons l'algorithme afin qu'il puisse être plus facilement implémenté aux RFID. Un exemple de la première solution est GPS [25] de McLoone et Robshaw et un exemple de la deuxième est SQUASH [35] de Shamir.

Famille de protocole d'authentification HB$^+$

Le protocole HB$^+$ basé sur le problème LPN (Learning Parity with Noise) [12] et publié premièrement par Juels et Weis [22] a inspiré de nombreux protocoles [7, 27, 37]. Ce succès est dû au fait que le problème LPN peut être facilement analysé et que les demandes matérielles de HB$^+$ étaient faibles. En revanche, les demandes matérielles ont augmenté avec le temps vu que les procédés d'attaque se sont également renforcés contre le protocole [17] et contre le problème LPN [16, 40, 30].

Protocoles basés sur PUF

Le PUF, c'est-à-dire Physically Uncloneable Function, est un circuit dont la sortie ne peut être définie à l'avance, car des retardements sont mis dans le circuit lors de la fabrication et ils rendent le système imprévisible. Étant donné que la sortie ne peut pas être modelées en fonction de l'entrée, le PUF fonctionne à l'instar d'une fonction de hachage. Ce sont Bolotnyy et Gabriel qui ont lancé l'idée du PUF pour les RFID [6], mais il n'a malheureusement pas été réalisé en nombre, c'est pourquoi nous ne sommes pas en mesure de savoir s'il fonctionnait bien dans de plus grands systèmes réels.

Tableau de comparaison

Ce tableau de comparaison des services de sécurité des lesdits protocoles se situe dans le Tableau A.1. Il est intéressant de remarquer que, sauf le protocole basé sur l'arbre des clés de Molnar-Wagner, les services de sécurité des protocoles RFID ne se sont pas avérés fiables en dépit des attentes de leur inventeur. Dans la technologie de sécurité, c'est habituel puisque les procédés d'attaque sont toujours en train de progresser, ils ne seront jamais plus mauvais.

A.3 Noisy Secret Shuffling

Dans ce chapitre, nous allons présenter deux protocoles d'identification privée qui peuvent être implémentés aux RFID et qui, en même temps, sont assez sécurisés afin qu'une puce RFID et son propriétaire ne puissent pas être suivis. Le premier protocole à présenter, le Probabilistic Identification Protocol (ProbIP), se sert d'une clé K bit sur laquelle il renseigne le lecteur. Le lecteur, connaissant la clé de chaque puce du système, est en mesure de deviner quelle puce a envoyé les informations, tandis qu'un outsider en est incapable, vu qu'il ne connaît pas la clé des puces du système. Le deuxième protocole, l'Enhanced Probabilistic Identification Protocol (EProbIP) présente des ressemblances avec le ProbIP, mais il fournit quelque fois des informations divergentes au lecteur dans le but de tromper l'attaquant.

A.3.1 Probabilistic Identification Protocol

Le fonctionnement du Probabilistic Identification Protocol (ProbIP) se déroule entre le lecteur RFID et la puce RFID précédant ainsi toutes les autres communications. Le déroulement du protocole est le suivant :

1. Le lecteur RFID envoie un message `HELLO`.

TAB. A.1 – Aperçu des services de sécurité des lesdits protocoles RFID. Les services de sécurité, qui se sont avérés défectueux, sont marqués par une étoile.

Protocole	Unlinkable ident.	Untraceable ident.	Tag auth.	Reader auth.
ISO14443A coll.-avoidance [20]	Non	Non	Non	Non
EPC coll.-avoidance [14]	Non	Non	Non	Non
Pseudonym-rotation [21]	**Oui***	**Oui***	Non	Non
ProbIP [10]	**Oui***	**Oui***	Non	Non
OSK [24]	**Oui***	**Oui***	**Oui**	Non
YA-TRAP [39]	**Oui***	**Oui***	**Oui**	Non
YA-TRAP+ [8]	**Oui***	**Oui***	**Oui**	Non
O-TRAP [8]	**Oui***	**Oui***	**Oui**	Non
RIPP-FS [11]	**Oui***	**Oui***	**Oui**	**Oui**
Molnar-Wagner [26]	**Oui**	**Oui**	**Oui**	**Oui**
DPM [33]	**Oui***	**Oui***	**Oui**	**Oui**
SQUASH-0 [35]	Non	Non	**Oui***	Non
WIPR [29]	Non	Non	**Oui**	Non
HB^+ [22]	Non	Non	**Oui***	Non
$HB^\#$ [18]	Non	Non	**Oui***	Non
PUF [6]	Non	Non	**Oui**	Non

2. Dès qu'elle a reçu le message `HELLO`, la puce RFID j ($\mathcal{T}_j$) envoie le *paquet* P et le message `FINISHED`. P est un paramètre de système et le *paquet* est une liste de donnée de $2L$ de long, $<a_1, b_1>, <a_2, b_2> \dots, <a_L, b_L>$, où a_i est un indice de clé aléatoire $a_i \overset{r}{\leftarrow} [1, K]$ ne se répétant pas dans un même paquet, et b_i est un bit aléatoire $b_i \overset{r}{\leftarrow} \{0, 1\}$ de sorte que l'équation

$$\sum_{i=1}^{L} \Big[k_j[a_i] \oplus b_i \Big] = L/2 \qquad (A.1)$$

soit satisfaite (L est un numéro pair). Comme l'addition est une opération commutative, jusqu'à ce que les paires $<a_i, b_i>$ ne changent pas, l'ordre des paires peut se modifier. Ces paires sont marquées de la façon suivante : a_i dans la mesure où $b_i = 1$ et $\bar{a}_i$ si $b_i = 0$.

3. Dès que le lecteur a reçu les paquets, il les envoie au système d'arrière-plan, c'est-à-dire à $\mathcal{B}$, qui calcule l'équation (A.1) pour chaque paquet et pour la clé de chaque RFID. La clé correspondant à tous les paquets est probablement la clé qui a été utilisé par la RFID, c'est pourquoi $\mathcal{B}$ peut dévoiler la puce RFID appartenant aux paquets. $\mathcal{B}$ renvoie le numéro d'identification de la puce RFID ainsi dévoilée au lecteur.

Le nombre minimum de paquets

Le nombre de tous les paquets possibles est $\binom{2K}{L}$, du moment que a_i provient d'un ensemble de la taille K et que b_i vient d'un ensemble de la taille 2. Le nombre de paquets possibles correspondants à une clé donnée n'est que $\binom{K}{L/2}\binom{K-L/2}{L/2}$ étant donné que l'équation (A.1) doit être satisfaite et qu'il est impossible de répéter les indices dans un paquet. La proportion de ces deux chiffres

$$R = \frac{\binom{K}{L/2}\binom{K-L/2}{L/2}}{\binom{2K}{L}} \tag{A.2}$$

est la probabilité de ce qu'un paquet choisi aléatoirement s'applique à la clé d'une puce sélectionnée aléatoirement. Par exemple, dans la mesure où $K = 400, L = 10$, alors $R \approx 0.232$.

Dans la mesure où le nombre des puces RFID est n, dans ce cas-là le "faux positif" est probablement $fp = n * R^p$. Alors le nombre des paquets envoyés doit être :

$$P = \left\lceil \frac{\log(1/n * fp)}{\log(R)} \right\rceil \tag{A.3}$$

Ce qui, pour les paramètres $L = 10$, $fp = 0.1$ et $n = 10^7$, est $P = \lceil 12.62 \rceil = 13$. Si autant de paquets ne s'avèrent pas suffisants, le protocole peut être réitéré.

Implémentation

Le protocole peut être facilement implémenté au système de fond, il n'y a besoin que d'une capacité de processeur convenable. Ceci peut être résolu par des FPGA aussi, qui peuvent rendre le système beaucoup plus rapide.

L'implémentation du protocole à une puce est également simple vu qu'elle ne nécessite qu'un calcul presque négligeable et extrêmement peu de mémoire. Le seul défi est lancé par le générateur de nombres aléatoires utilisé par le système pour générer les paquets. Dans le même but, les Physically Unclonable Functions (PUFs) [6] peuvent être appliqués. Ils se montrent idéaux pour générer des nombres aléatoires [28] sur les puces RFID, vu qu'ils perçoivent les changements de l'environnement et en y réagissant, mais pas d'une façon calculable, ils génèrent une sortie aléatoire.

Si nous calculons avec le paramètre $K = 400$, alors l'exigence ROM du protocole sur la puce est 400-bit correspondant grosso modo à une porte 400 NOT. L'algorithme de génération de paquets demande environs 100 GE ("Gate Equivalent") et le générateur de nombres aléatoires (PRNG) exige environ 700 GE, pour un total de 1200 GE d'espace sur le tag RFID.

A.3.2 L'analyse de sécurité du ProbIP

Ouafi et al. [31] ont prouvé avec succès que la sécurité du protocole ProbIP peut être brisée de l'élimination de Gauss, dans la mesure où l'attaquant parvient à rassembler suffisant de paquets. L'idée de l'attaque est que les paquets peuvent être représentés de la façon suivante :

$$\sum_{i=1}^{L} v_i^1(K[i] \oplus b_i^1) = L/2$$
$$\sum_{i=1}^{L} v_i^2(K[i] \oplus b_i^2) = L/2$$
$$\vdots$$
$$\sum_{i=1}^{L} v_i^l(K[i] \oplus b_i^l) = L/2$$

où le nombre des paquets obtenus par l'attaquant est n, et v est la fonction d'indication montrant si un bit de clé donné se trouve dans le paquet, et b_i est est bit aléatoire associé au bit de clé. S'il y a une telle représentation, alors l'élimination de Gauss n'exige que $7 \cdot m^{log_2 7}$ opérations dans la mesure où nous utilisons l'algorithme de Strassen [38] où la taille de la matrice est m.

A.3.3 Le protocole EProbIP

Le protocole Error-Intoroducing Probabilistic Identificaiton protocol (EProbIP) est un tel protocol qui se base sur le ProbIP, mais il le protège de l'attaque publiée par Ouafi et al. [31] de telle sorte que la puce RFID envoi de temps en temps des paquets aléatoires. Cette modification change dans une large mesure le degré de sécurité assuré par le protocole.

Les paquets aléatoires dans le protocole EProbIP ont la même apparence que les paquets normaux sauf que la donnée qu'ils renferment ne satisfait pas nécessairement l'équation (A.1.), du moment que les indices a_i, de même que les bits b_i ont été sélectionnés de manière aléatoire en veillant à ce que les indices ne se répètent pas. Les paquets envoyés aléatoirement s'appellent paquets **Noise**, et les paquets normaux se nomment paquets **Valid**. La probabilité de l'occurrence des paquets **Noise** peut être réglée, elle est marquée par err.

Les faux paquets n'occasionnent pas de problème au serveur de fond $\mathcal{B}$: par rapport au ProbIP, la différence est que le serveur de fond est obligé de tenir compte de plus d'informations. Il doit compter le nombre des paquets correspondant aux clés des puces. La puce, à la clé de laquelle les plus de paquets ont correspondu, est celle qui a très probablement effectué l'identification.

Le nombre minimum de paquets

Dans ce chapitre, nous allons calculer le nombre minimum de paquets qui doivent être envoyés par la puce afin que le lecteur ait la chance de l'identifier. Vu que le protocole envoi également des paquets sélectionnés aléatoirement, la probabilité de l'identification ne sera jamais 100%, nous avons néanmoins la possibilité d'approcher à volonté de 100%. Ceci consiste dans l'envoi de plusieurs paquets, comme nous allons le voir. Sans éclairer tous les détails, la probabilité que du paquet P, envoyé par la puce, exactement x satisfasse l'équation est la suivante :

$$\texttt{fit\_send}(P,x) = \sum_{i=0}^{x} \Big[\texttt{Bi}(P, x-i, 1-err) * \texttt{Bi}(P-(x-i), i, R) \Big]$$

où

$$R = \frac{\binom{K}{L/2}\binom{K-L/2}{L/2}}{\binom{K}{L}2^L}$$

Par conséquent, la chance qu'une fausse puce se situe plus en haut dans l'hiérarchie que la puce recherchée est la suivante :

$$\texttt{probrank}(P, lev) = 1 - \Big(\texttt{BiC}(P, lev-1, R)\Big)^{n-1}$$

La chance que la puce recherchée ne se trouve pas au sommet de l'hiérarchie est donc la suivante :

$$fp := \sum_{i=0}^{P} \Big[\texttt{probrank}(P,i) * \texttt{fit\_send}(P,i) \Big]$$

Par exemple, dans la mesure où nous calculons avec les paramètres $K = 400$, $L = 10$, $P = 20$ et $err = 0.1$, dans ce cas-là $fp = 0.086$. Par conséquent, il y a 91% de chance que, si la puce a envoyé 20 paquets lors de l'identification, elle soit tout en haut, au sommet de l'hiérarchie établie par le lecteur et de cette façon, elle sera exactement identifiée. Pour comparer, dans le protocole ProbIP (alors lorsque $err = 0$) un tel fp n'exige que 13 paquets. Nous avons besoin de 7 paquets de plus vu qu'il est fort possible que la puce envoie également de faux paquets et cela doit être compensé.

A.3.4 L'analyse de sécurité de l'EProbIP

La sécurité de l'EProbIP a été examinée d'après "Privacy Experiment", l'expérience d'identification privée, décrite par Juels et Weis [23]. Dans cette expérience, l'attaquant se voit avoir le droit d'ouvrir certaines puces et certains lecteurs et de retenir leur contenu. Nous allons choisir deux puces qui ne sont pas connues par l'attaquant. L'attaquant devra distinguer les deux puces après un certain nombre d'expériences d'identification avec une probabilité non négligeable de plus de 50%.

L'EProbIP peut être le plus facilement brisé de telle sorte que nous convertissons un nombre convenant de paquets à la notation CNF, et nous confions à l'algorithme MaxSAT de retrouver la combination de bit de clé contenant le moins de faux paquets. Dans la mesure où nous avons donné assez de paquets à l'algorithme MaxSAT, il retrouve la clé K de la puce. Le problème de ce procédé c'est qu'il exige beaucoup de temps et qu'il ne se sert pas de la possibilité que les puces peuvent être interrogées en nombre : l'algorithme MaxSAT fonctionne *plus lentement* s'il possède plus d'informations.

Afin que l'attaquant puisse se servir de ce que les puces peuvent être interrogées à maintes reprises, nous avons développé un algorithme SAT spécial. Celui-ci se base

No. de paquets	$K = 100$	$K = 200$	$K = 400$	$K = 1000$
$1 \cdot P_{att}$	$1.15e15$ s	$2.53e34$ s	$7.33e72$ s	$1.78e188$ s
$9 \cdot P_{att}$	$1.47e6$ s	$3.16e14$ s	$1.47e31$ s	$1.47e82$ s
$27 \cdot P_{att}$	$5.08e4$ s	$3.87e10$ s	$2.25e22$ s	$4.43e57$ s
$64 \cdot P_{att}$	$2.94e4$ s	$1.77e10$ s	$6.45e21$ s	$3.10e56$ s
$192 \cdot P_{att}$	$1.55e4$ s	$8.96e8$ s	$2.99e18$ s	$1.11e47$ s
$576 \cdot P_{att}$	$1.80e4$ s	$6.29e6$ s	$7.72e11$ s	$1.43e27$ s

TAB. A.2 – Le tableau indique la rapidité avec laquelle l'EProbIP peut être brisé par l'algorithme SAT en cas d'utilisation d'un ordinateur Pentium D@3GHz. Les paramètres utilisés étaient $L = 5$ et $fp \approx 90$. P_{att} représente le nombre de paquets dont l'attaquant a besoin dans le but de briser le système. Le tableau fait la démonstration de la capacité de l'algorithme SAT modifié à profiter de la quantité des paquets émis par la puce à l'inverse de l'algorithme MaxSat

sur le système MiniSat [13], mais il le développe encore de telle sorte qu'il soit plus ferme : si un paquet donné ne correspond pas à une combination de clé donnée, il ne jette pas la combination de clé, mais il calcule la quantité de paquets non conformes, et si cette qualité est inférieure à une limite, il ne s'intéresse plus à ce problème. Ainsi, il devient capable de fonctionner aussi vite que les algorithmes SAT sur un problème qui exigerait d'ailleurs un algorithme MaxSAT. La rapidité de bris présentée par ce procédé est résumée dans le Tableau A.2.

A.3.5 Conclusion

Nous avons présenté la façon dont il est possible de créer un protocole d'identification privée gardant le droit à la vie privée du propriétaire, mais qui, à la fois, est apte à être implémenté aux RFID. Il est évident aussi qu'il est relativement difficile de créer un tel protocole : la première version du protocole présenté ne s'est pas avérée fiable comme Ouafi et al.[31] l'ont démontré. Le nouveau protocole, l'EProbIP fournit, en revanche, une solution à la méthode d'attaque publié par Ouafi et al., et il rend le système plus fiable contre des attaquants s'efforçant de la façon présentée.

A.4 L'analyse du protocole d'authentification et d'identification RFID de Di Pietro et Molva

Dans ce chapitre, nous allons nous pencher de plus près sur le protocole d'authentification et d'identification RFID de Di Pietro et Molva [33], qui résout l'identification des RFID d'une façon radicalement nouvelle de sorte que les demandes matérielles restent faibles. Nous allons nous focaliser dans ce chapitre sur la partie du protocole consacrée à l'identification, vu que sa partie vouée à l'authentification applique la fonction SHA célèbre par sa fiabilité qu'il est impossible de briser de la technologie actuellement connue.

A.4.1 Le protocole Di Pietro-Molva

Pour l'identification privée, le protocole utilise la fonction DPM. L'entrée de la DPM est l bit où l est divisible en 3, et sa sortie est 1 bit. La DPM est définie par l'équation suivante :

$$DPM(x) = \bigoplus_{i=0}^{l/3} M(x[3i], x[3i+1], x[3i+2]) \tag{A.4}$$

où M est la fonction de majorité : son entrée est 3 bits, sa sortie est 1 bit. La fonction M constat s'il y a plus de `1` dans l'entrée que `0`.

La partie du protocole consacrée à l'identification privée est la suivante :

1. $\mathcal{R}_j$ envoie son ID_j à la puce
2. $\mathcal{T}_i$ calcule $k_{i,j} = h(k_i || ID_j || k_i)$. Ensuite, il génère une fréquence des bits de q l bit aléatoire, r_p $(p = 1 \ldots q)$. Par la suite, il envoie q α_p, où $\alpha_p = r_p \oplus k_{i,j}$ et il envoie le vecteur de q bit de long, V, qui a été généré de cette manière : $V[p] = DPM(r_p)$
3. $\mathcal{R}_j$ calcule $DPM(\alpha_p \oplus k_{i,j})$ sur toutes les clés $k_{i,j}$ qui sont dans sa base de données, et il le compare avec $V[p]$. Le protocole appelle ceci : *Lookup Process*. La clé $k_{i,j}$ convenant à chaque paire $(\alpha_p, V[p]), p = 1 \ldots n$ est ce qui a été utilisée par la puce.

A.4.2 Équivalences de clés

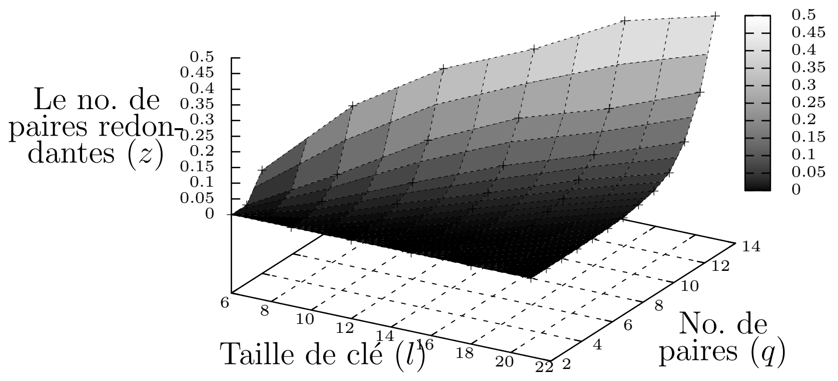
Divisons la clé $k_{i,j}$ en blocs de clé de 3 bits de long, que l'on va désormais nommer des blocs. Si nous invertissons un bloc pair, le lecteur ne sera pas capable de distinguer la clé de la clé originale, étant donné que le lecteur n'applique que des paires $(\alpha_p, V[p])$ pour la distinction. C'est parce que $V[p] = DPM(\alpha_p \oplus k_{i,j}) = DPM(\alpha_p \oplus k_{i,j} \oplus \text{inversions})$ et ainsi, le "Lookup Process" ne dépendra pas de ce que les blocs ont été inversés. Un exemple de l'équivalences de clés est : $k_{i,j} =$`[001 000 100]`$\approx$`[110 000 011]`. Les équivalences de clés signifient que chaque clé de l de long fait partie d'un groupe d'équivalence de clés de $\sum_{i=0}^{\lfloor (l/3)/2 \rfloor} \binom{l/3}{2i} = 2^{l/3-1}$.

Les équivalences de clés comportent de nombreux inconvénients pour le protocoles. D'une part, l'attaquant ne doit pas déchiffrer tout la taille de clé, mais il lui suffit de déchiffrer les deux tiers de la clé, le reste n'a pas d'importance du point de vue de l'identification privée. D'autre part, le lecteur est incapable dans certains cas de distinguer deux puces RFID de l'une de l'autre même s'il effectue plusieurs identifications sur les puces. C'est une faute qui rend le système inapproprié pour un grand nombre d'utilisation, vu que la fonction essentielle de la puce RFID se blesse ainsi.

A.4.3 Remarques en relation avec le deuxième lemme

Dans l'article original, le troisième lemme affirme qu'avec un r choisi aléatoirement la probabilité que $DPM(r) = 1$ soit exactement 0.5 et que $DPM(r) = 0$ soit également 0.5. Appliquant ce lemme, les auteurs ont conclu dans le deuxième lemme que si q donné est une paire choisie d'une manière aléatoire $(\alpha_p, V[p])$, la probabilité

FIG. A.2 – Le nombre moyen des paires redondantes (constituant une tautologie) en cas de paires non équivalentes et de tailles de clé différentes.



qu'au moins une clé survive au "Lookup Process" est moins que $n(1/2)^q$. En revanche, afin que cela s'avère vrai, la distribution du reste des clés après une paire $(\alpha_p, V[p])$ aurait dû rester également aléatoire. Mais ce n'est pas ainsi, puisque si l'on utilise deux fois la même paire, les mêmes clés restent sur la liste des puces possibles, comme si l'on avait utilisé une des deux.

Cette remarque laisse penser que le nombre minimum de paires $(\alpha_p, V[p])$ calculé par les auteurs ne suffit pas souvent. Nous avons calculé le nombre moyen des paires redondantes (constituant une tautologie), et nous avons parvenus à l'image de la Figure A.2.

A.4.4 Retrouver $k_{i,j}$

Dans ce chapitre, nous allons présenter une méthode d'attaque qui retrouvera $k_{i,j}$. Étant donné que la clé de la puce est toujours masquée par ID_j du lecteur (vu que $k_{i,j} = h(k_i||ID_j||k_i)$), l'attaquant ne peut briser la sécurité de la puce que si la puce communique avec un certain lecteur. Du moment qu'il y besoin de la clé $k_{i,j}$ pour l'identification tant de la puce que du lecteur, dans la mesure où l'attaquant entre en possession de cette clé, il devient capable d'authentifier vers la puce et le lecteur.

L'attaque Man-in-the-middle

Il est possible de lancer une attaque man-in-the-middle (MiM) contre le protocole. Lors de cette attaque, l'attaquant peut s'informer à l'aide de $k_{i,j}$ en utilisant l'information si l'authentification effectuée après l'identification a réussi. L'attaque profite de ce qu'à l'exception de α_1 aucun α_p n'est authentifié : dans la mesure où l'attaquant modifie α_2 en α'_2 et "Lookup Process" retrouve quand même la clé $k_{i,j}$, alors dans ce cas-là $DPM(k_{i,j} \oplus \alpha'_p) = V[p]$ de telle sorte que, soit l'attaquant n'a inversé la sortie d'aucune fonction de majorité (M) dans DPM, soit il a inversé un sortie paire. En revanche, si $DPM(k_{i,j} \oplus \alpha'_p) = \overline{V[p]}$, l'authentification ne réussira pas, car le lecteur ne retrouvera pas la clé $k_{i,j}$ sous "Lookup Process", et dans ce cas-là l'attaquant peut être sûr d'avoir inversé la sortie de nombre impair de fonctions de majorité dans DPM.

Au lieu de changer α_1, l'attaquant transformera α_2 de telle sorte qu'il inversera

TAB. A.3 – Le tableau représente les conclusions qui peuvent être déduites par l'attaquant s'il modifie les paquets du protocole lors de l'envoi. L'attaquant ne doit changer que le deuxième ou le troisième bit du bloc α_2 et qu'observer si l'authentification réussit.

Bit inversé	Auth	Bloc $\alpha_2[x \dots x+2]$ original `000`	Bloc $\alpha_2[x \dots x+2]$ original `001`
$\alpha_2[x+2]$	✓	$k_{i,j}[x] = k_{i,j}[x+1]$	$k_{i,j}[x] = k_{i,j}[x+1]$
$\alpha_2[x+2]$	$\times$	$k_{i,j}[x] \neq k_{i,j}[x+1]$	$k_{i,j}[x] \neq k_{i,j}[x+1]$
$\alpha_2[x+1]$	✓	$k_{i,j}[x] = k_{i,j}[x+2]$	$k_{i,j}[x] \neq k_{i,j}[x+2]$
$\alpha_2[x+1]$	$\times$	$k_{i,j}[x] \neq k_{i,j}[x+2]$	$k_{i,j}[x] = k_{i,j}[x+2]$
		Bloc $\alpha_2[x \dots x+2]$ original `010`	Bloc $\alpha_2[x \dots x+2]$ original `100`
$\alpha_2[x+2]$	✓	$k_{i,j}[x] \neq k_{i,j}[x+1]$	$k_{i,j}[x] \neq k_{i,j}[x+1]$
$\alpha_2[x+2]$	$\times$	$k_{i,j}[x] = k_{i,j}[x+1]$	$k_{i,j}[x] = k_{i,j}[x+1]$
$\alpha_2[x+1]$	✓	$k_{i,j}[x] = k_{i,j}[x+2]$	$k_{i,j}[x] \neq k_{i,j}[x+2]$
$\alpha_2[x+1]$	$\times$	$k_{i,j}[x] \neq k_{i,j}[x+2]$	$k_{i,j}[x] = k_{i,j}[x+2]$

un certain bit (par exemple le deuxième et le troisième), et il observera si l'authentification se produit. Chaque bloc α_2 parviendra à une conclusion différente : le Tableau A.3. énumère toutes les conclusions envisageables qui peuvent être tirées à chaque bloc α_2 dans la mesure où l'attaquant a inversé le deuxième ou le troisième bit.

Utilisant le Tableau A.3. l'attaquant n'a besoin que de deux déroulements de protocole afin de déchiffrer le contenu d'un bloc de telle sorte que la puce puisse être identifiée. C'est-à-dire que par exemple l'identification privée $l = 81$ peut être tout à fait brisée seulement pendant le déroulement de 54 protocoles.

De la façon présentée, l'attaquant n'arrive à briser que les deux-tiers de la clé, le reste se brise par une simple méthode d'attaque intitulé “brute force” qui n'exige que $2^{K/3}$ opérations. En cas de taille de clé normale (par exemple une RFID possède généralement des clés de 80 bits de long), cela ne demande que peu de temps, avec un ordinateur moderne, il ne faut compter qu'avec quelques secondes.

A.4.5 Conclusion

Le protocole d'identification et d'authentification Di Pietro et Molva peut être brisé par un attaquant très facilement à travers étonnamment peu de tentatives d'attaque. L'attaque devient possible du fait que la fonction DPM utilisée par le protocole d'identification n'a pas été analysé d'une manière suffisante et qu'elle dispose d'une faute entraînant deux problèmes différents. D'un côté, les puces ne peuvent pas être toujours distinguées l'une de l'autre, de l'autre côté, un attaquant malveillant a la possibilité d'en servir pour briser la clé. Ce protocole pourrait être amélioré de telle sorte que l'on substitue la fonction DPM à une fonction pouvant être aussi simplement implémentée, mais qui s'avère plus fiable au niveau de la cryptographie.

A.5 L'application des algorithmes SAT pour briser les chiffrements par flot

Les systèmes utilisant les nouveaux primitifs de cryptographie jusqu'ici inconnus peuvent être brisés avec une grande probabilité d'après ce que l'on a présenté jusqu'à ce chapitre. Pour éviter cela, nous allons analyser les primitifs de cryptographie bien connus ayant une faible demande matérielle pour pouvoir prendre place sur une puce RFID. En raison de ce qu'ils utilisent une construction connue depuis longtemps, ils s'analysent plus facilement.

Le but de ce chapitre est de rapprocher les algorithmes SAT de la cryptographie et de rapprocher la description des chiffrements par flot des algorithmes SAT. Par conséquent, les résultats de ces deux domaines de recherches deviennent plus facilement utilisables.

A.5.1 L'adaptation des algorithmes SAT à l'environnement des chiffrements par flot

Le problème des clauses XOR

Les chiffrement par flot utilisent dans une large mesure la fonction XOR. Ces fonctions XOR, lorsque l'on les convertit à la notation CNF connue par les algorithmes SAT, seront de la taille exponentielle : elle se convertissent exactement à 2^{len-1} clauses, dans la mesure où la longueur de la XOR originale était *len*. Rappelons que ce problème a été résolu de plusieurs façons. Le travail purement théorique de Massacci [4, Sect. 9] fournit une solution complète au problèmes des clauses XOR. Cette solution purement théorique n'a malheureusement jamais été implémentée. Nous tentons de remplir ce vide.

Nous avons complété l'algorithme SAT MiniSat [13] par des clauses XOR de telle sorte que nous avons laissé toutes les fonctions du MiniSat intactes. Nous n'avons complété que la fonction "propagate" de telle sorte qu'elle soit capable de traiter les clauses XOR aussi. Nous sommes parvenus à ce que la clause XOR se comporte dans tous les cas telle une clause normale. Par contre, si elle provoque une propagation ou un conflit, elle change de telle sorte qu'elle ait l'apparence de la clause de 2^{len-1} qui a causé la propagation ou le conflit.

Par exemple la clause XOR $a \oplus b \oplus c$ représente toutes ces clauses

$$\begin{array}{llll} a \vee \neg b \vee \neg c & (1) & \neg a \vee \neg b \vee c & (2) \\ a \vee b \vee c & (3) & \neg a \vee b \vee \neg c & (4) \end{array}$$

et elle se modifie en utilisant polymorphisme de C++ afin qu'elle corresponde à la situation donnée.

L'analyse du déroulement de l'algorithme SAT

L'algorithme SAT effectue extrêmement beaucoup d'opérations pendant très peu de temps. Cela provoque le problème suivant : le déroulement de l'algorithme ne peut pas être suivi, et cela rend plus difficile à faire des analyses postérieures sur le temps trop long de la recherche. Afin de résoudre ce problème, nous avons

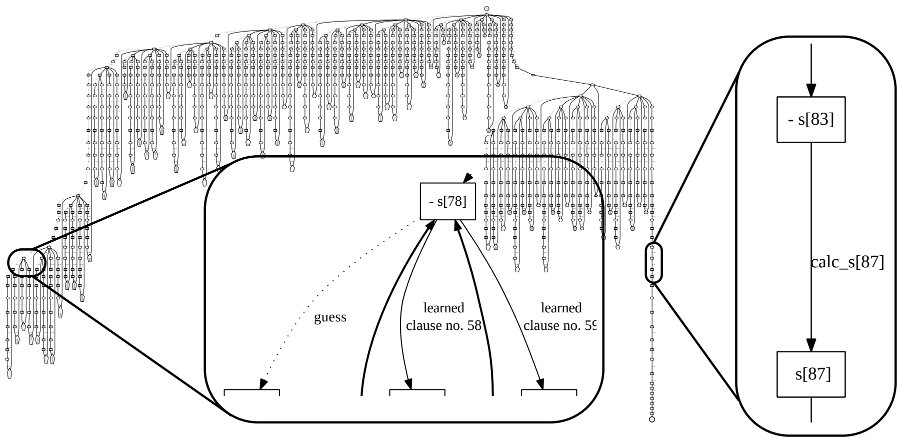


Fig. A.3 – Visualisation d'un arbre de recherche ayant cherché et retrouvé l'état de l'algorithme de chiffrement Crypto-1. L'arbre se lit de gauche à droit et de haut en bas. Le premier conflit est le pentagone régulier qui se trouve à l'extrémité de gauche et de bas. La solution recherchée est le cercle se situant à l'extrémité de droite et de bas.

modifié l'algorithme SAT de telle sorte qu'il soit capable de créer un arbre de recherche pendant qu'il cherche la solution et que parallèlement à cela, il établisse des statistiques sur son déroulement.

Bien qu'il y ait eu jusqu'ici aussi des compléments faisant des analyses dynamiques pour l'algorithme MiniSat [36], ils ne pouvait pas se servir de l'information du problème original, soit l'ensemble des fonctions et des variables. Nous avons transformé l'algorithme de telle sorte que nous allons nommer les variables et les fonctions qui se représentent dans l'arbre de recherche. Il s'ensuit que l'arbre de recherche devient plus clair et plus interprétable. Un arbre de recherche généré par l'algorithme modifié se trouve dans la Figure A.3.

L'algorithme rassemble pendant la recherche les statistiques suivantes :

– *Statistiques des arbres de recherche.* Le nombre de branches, la profondeur moyenne d'arbre, la quantité moyenne de bifurcation de branches.
– *Statistiques des clauses.* La liste des clauses les plus actives. Cette liste permet d'aider à retrouver les clauses qui sont les plus importantes pour la solution.
– *Statistiques variables.* La liste des clauses dont les variables ont été le plus souvent assignées lors du déroulement d'algorithme SAT. Cette liste contribue à retrouver les variables qui sont les plus importantes du point de vue de la solution.

L'algorithme établit également des statistiques illustrant la distribution de la profondeur d'arbre qui peuvent être traitées par le logiciel "gnuplot" aussi. La distribution de la profondeur d'arbre est illustrée dans la Figure A.4.

A.5.2 Adaptation de la représentation de l'algorithme de chiffrement

La bonne représentation des chiffrements par flot dans les clauses régulières et XOR est une démarche importante du bris des algorithmes de chiffrement [3, Sect. 8]. Dans ce chapitre, nous allons briser les chiffrements par flot de sorte que nous allons dévoiler leur état – l'algorithme SAT va donc chercher l'état de l'algorithme de chiffrement.

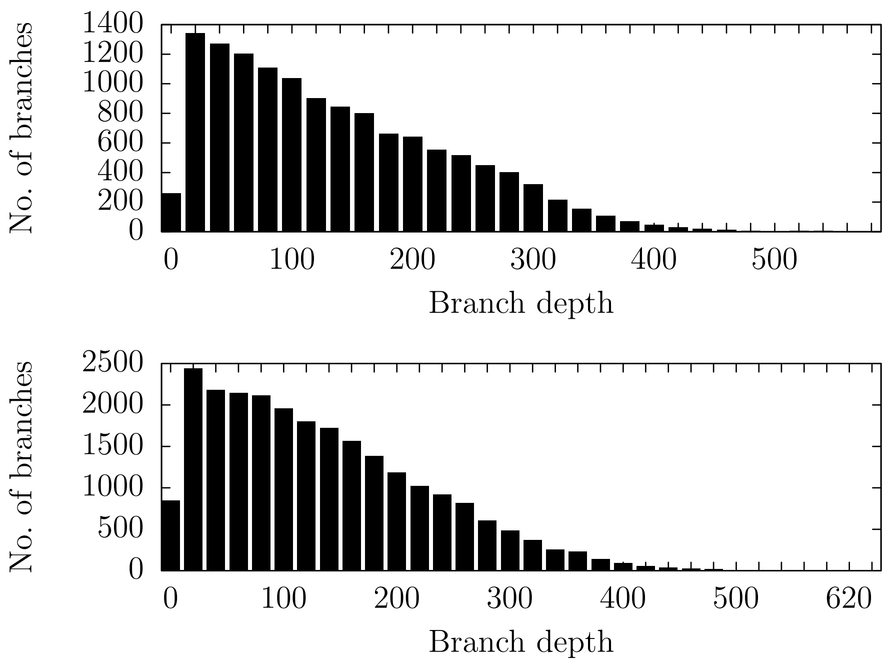


Fig. A.4 – La statistique de la profondeur d'arbre de l'arbre de recherche lors du treizième et seizième redémarrage pendant que l'algorithme SAT était à la recherche des états du chiffrement par flot. Les distributions de la profondeur d'arbre des deux recherches se correspondent.

La représentation du circuit logique

La représentation du circuit logique sert à nous fournir une image globale sur la façon dont l'algorithme de chiffrement est encodé du pont de vue de l'algorithme SAT. Les parties du circuit logique sont les bits d'état de l'algorithme de chiffrement, les fonctions de mise à jour d'état et les fonctions de sortie. Un exemple du circuit logique se trouve dans la Figure A.5. Les fonctions sont indiquées par des hexagones, et les bits d'état sont représentés par les carrés.

La *profondeur* du bit de sorti est le nombre des fonctions qui doit être calculé par l'algorithme SAT afin qu'il parvienne des bits d'état au bit de sortie. Par exemple dans la Figure A.5, la profondeur du bit de sortie est une, tandis que la profondeur du quatrième bit de sortie est quatre. Étant donné que l'algorithme tente de dévoiler les bits d'état et qu'il se dirige vers les bits de sortie, la profondeur des bits de sortie est un facteur très important de la rapidité de l'algorithme.

Les fonctions (marquées par des hexagones dans la figure) doivent être dans tous les cas calculées par l'algorithme SAT lorsqu'il a besoin de leur sortie. C'est-à-dire que l'encodage des fonctions (par exemple, par des clauses XOR ou des tables de Karnaugh etc.) n'est pas une partie négligeables du circuit logique. Il faut diminuer sa complexité au plus possible, éventuellement par des solutions hybrides (par exemple l'ensemble de clause XOR et de table de Karnaugh).

Finalement, comme l'algorithme SAT fixe d'abord les bits d'état et il se dirige de ceux-ci vers les bits de sortie, il est exceptionnellement important que les bits de sortie dépendent de très peu de bits d'état. En effet, s'ils dépendent d'une multitude de bits d'état, l'algorithme se voit dans l'obligation de fixer beaucoup de bits d'état

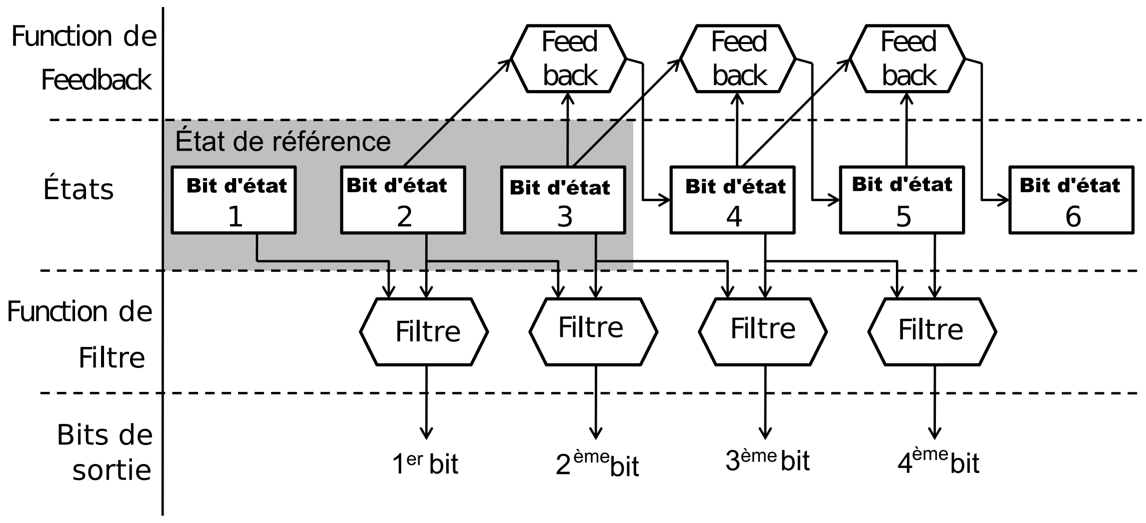


FIG. A.5 – La représentation du circuit logique d'un chiffrement par flot imaginaire

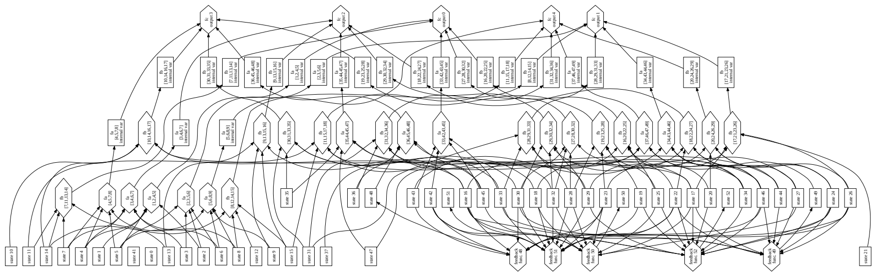


FIG. A.6 – La représentation graphique du circuit logique du chiffrement par flot HiTag2. Les fonctions sont représentées par des hexagones, les bits d'état sont indiqués par des carrés. Les bits de sortie connus sont représentés par les cinq hexagones se situant en haut de la figure, les cinq fonctions de mise à jour d'état se trouvent à droite, en bas.

avant de réaliser les avoir fixés sur une mauvaise combination de bit et il n'a d'autre possibilité que recommencer une partie du calcul. Le nombre indiquant la quantité de bits d'entrée dont un bit de sortie dépend s'appelle nombre de dépendance.

En fin de compte, le but est de diminuer au minimum la profondeur du circuit logique, la taille des nombres de dépendance et la complexité de la description des fonctions en réduisant ainsi le travail de l'algorithme SAT.

Création de la représentation du circuit logique

Dans le but de pouvoir analyser l'utilité de la représentation du circuit logique mentionnée ci-dessus, nous avons implémenté une génératrice du circuit logique dans l'algorithme SAT MiniSat. Le circuit logique ainsi fait peut être illustré par le logiciel `Graphviz`, ou bien il est possible d'en établir des statistiques permettant d'aider à illustrer et diminuer les variables mentionnées (nombre de dépendance, profondeur, etc.). À titre indicatif, il y a la représentation du circuit logique du chiffrement par flot HiTag2 dans la Figure A.6.

La représentation du circuit logique a également permis d'effectuer une analyse

TAB. A.4 – Le temps nécessaire pour résoudre les chiffrements Crypto-1, HiTag2, et Bivium

	Vanilla MiniSat	Karnaugh optimization	Karnaugh and xor-clause optimizations
Crypto-1	500 s	72 s	40 s
HiTag2	$2^{17.8}$ s	2^{15} s	$2^{14.5}$ s
Bivium	$2^{36.7}$ s	$2^{36.7}$ s	$2^{36.5}$ s

de l'arbre de dépendance. L'arbre de dépendance est destiné à éviter que l'on ajoute à la représentation des fonctions qui ne dépendent pas des bits de sortie, c'est-à-dire qui n'aident pas à retrouver les bits d'état. Naturellement, de telles fonctions peuvent être enlevées sans problème de la représentation et cela rend l'algorithme SAT plus rapide. Un exemple simple pour une telle fonction est la dernière fonction de mise à jour d'état de la Figure A.5. La sortie de cette fonction n'est reliée à aucun bit de sortie, elle est ainsi inutile.

A.5.3 Attaques réalisées

L'algorithme SAT augmenté est apte à résoudre une multitude de chiffrements par flots. Les attaques contre trois algorithmes de chiffrement ont été implémentées : contre HiTag2, Crypto-1 et Bivium. Le tableau comparatif de ces attaques se trouve dans A.4.

Crypto-1

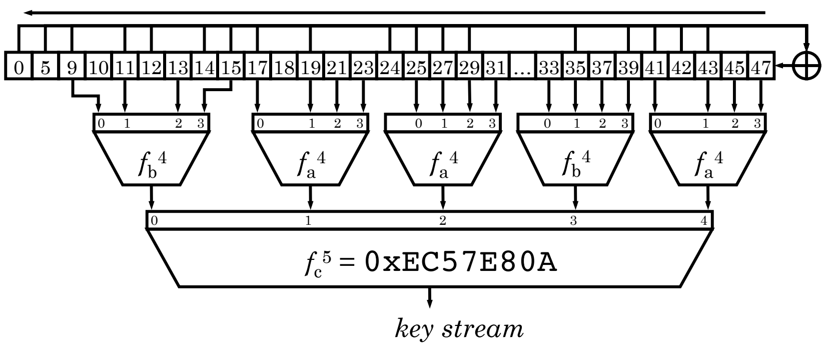
L'algorithme de chiffrement par flot Crypto-1 est un LFSR possédant un état de 48 bits et il est relié à une fonction de sortie non-linéaire. Le chiffrement par flot s'utilise à la carte *Mifare Classic*, dont on se sert partout dans le monde dans le transport en commun afin de contrôler les entrées (par exemple à Londrès et à Rio de Janeiro).

Comme la Figure A.7 montre, Crypto-1 utilise un LFSR de 48 bits, dont 18 bits d'état sont examinés par la fonction de mise à jour d'état. La fonction de sortie se compose de plusieurs petites fonctions en formant ainsi une fonction plus complexe et d'une plus grande taille.

Sur un ordinateur de bureau typique, si nous n'utilisons pas un processeur multi-cœur, et dans la mesure où $r = 56$ bits sont disponibles, l'attaque se basant sur l'algorithme SAT présenté par nous retrouve l'état de l'algorithme de chiffrement en 40 secondes.

HiTag2

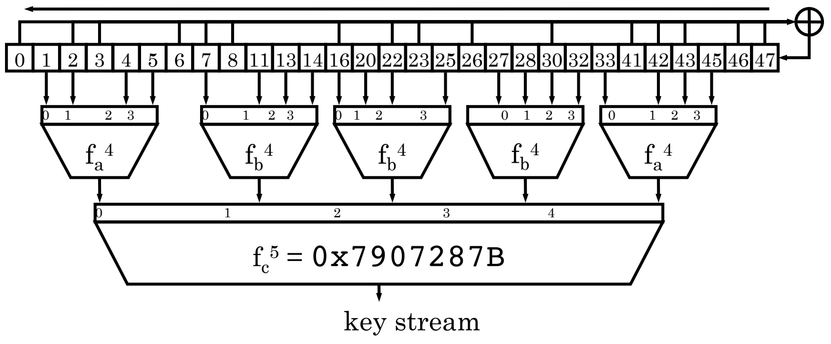
Le chiffrement par flot HiTag2 ressemble beaucoup à Crypto-1. La seule différence est que la fonction de sortie n'est pas pareille et surtout que les entrées de la fonction de sortie ne sont pas symétriques. Bien que cette différence soit petite, elle a de la première importance : l'attaque se basant sur l'algorithme SAT présenté par nous est plus lente à l'égard de cet algorithme, mais elle ne demande que six heures et demie.



f_a^4 = `0x9E98` = (a+b)(c+1)(a+d)+(b+1)c+a

f_b^4 = `0xB48E` = (a+c)(a+b+d)+(a+b)cd+b

Fig. A.7 – NXP *Mifare Classic* Le chiffrement par flot Crypto-1. La fonction de sortie se compose du réseau de petites fonctions (de f_a^4, f_b^4 et f_c^5).



f_a^4 = `0x2C79` = abc+ac+ad+bc+a+b+d+1

f_b^4 = `0x6671` = abd+acd+bcd+ab+ac+bc+a+b+d+1

Fig. A.8 – Le chiffrement par flot HiTag2. La fonction de sortie se compose du réseau de petites fonctions (de f_a^4, f_b^4 et f_c^5).

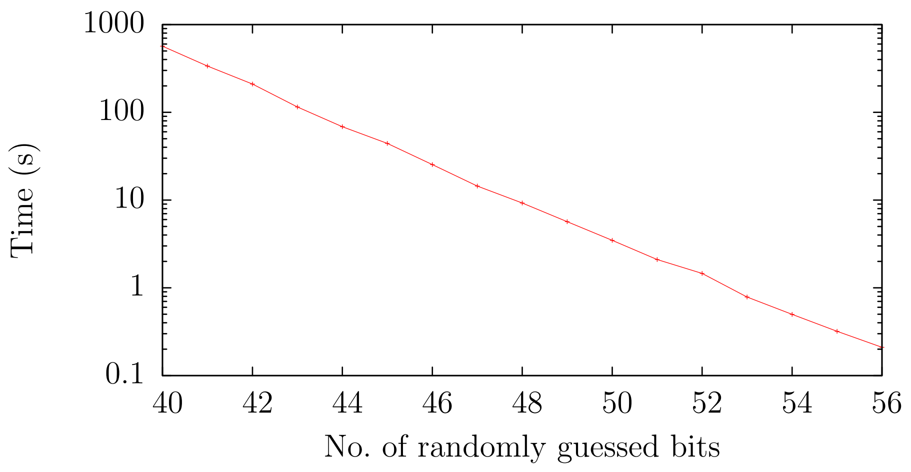


FIG. A.9 – La recherche des états du chiffrement par flot Bivium. Tous les poins sur la courbe montre le temps de mille exécutions. Le temps de le recherche est exponentiel au nombre des bits donnés – moins il y a des bits d'aide pour l'algorithme SAT, le déroulement de l'algorithme est moins rapide. Dans le mesure où l'on extrapole le graphique sur l'axe x jusqu'à zéro, le temps d'une exécution sera $2^{36.5}$ secondes.

Bivium

Le chiffrement par flot Bivium [34] est une version simplifiée du chiffrement par flot Trivium, et il n'a été créé que pour des buts de recherche. Les articles publiés en matière de cet algorithme de chiffrement parviennent aux solutions qui sont toujours de plus en plus meilleures dans le bris de l'algorithme de chiffrement. La version la plus rapide brise l'algorithme de chiffrement en $2^{42.7}$ secondes sur un ordinateur de bureau.

Il est possible de dévoiler les 177 bits d'état de Bivium à partir de 177 bits de sortie en $2^{36.5}$ secondes en moyenne dans le mesure où nous utilisons l'algorithme SAT MiniSat. La méthode de solution présentée par nous est, par conséquent, 2^6 fois plus rapide en moyenne que la méthode de solution la plus rapide publiée jusqu'ici.

A.5.4 Conclusion

Les algorithmes SAT gardent de nombreuses possibilités pour les cryptographes, et l'on a commencé à en profiter dans les dernières années. Cet article poursuit cette direction de telle sorte qu'il rapproche les frontières de ces deux domaines de recherche (soit les algorithmes SAT et la cryptographie) par des méthodes différentes. Les méthodes présentées récemment sont capables de briser l'algorithme de chiffrement Crypto-1 uniquement en 40 secondes en utilisant un ordinateur de bureau, tandis que l'algorithme Bivium, dont nous avons constaté qu'il peut être plus difficilement brisé, ne peut être brisé qu'en $2^{36.5}$ secondes en moyenne, 2^6 fois plus rapidement que la méthode de solution la plus rapide connue jusqu'ici.

A.6 Conclusion

Dans cette thèse, nous avons parvenus à présenter les protocoles RFID les plus importants et analysé un protocole déjà publié. Nous avons fait connaître le protocole ProbIP et sa version améliorée, le protocole EProbIP, et nous avons finalement analysé à l'aide des algorithmes SAT les chiffrements par flot de faible demande matérielle qui sont idéaux pour être implémentés aux RFID.

Nous pouvons constaté que les protocoles expérimentaux, tel le protocole de Di Pietro-Molva, sont fortement fragiles du point de vue de la sécurité, et jusqu'à ce qu'un bon protocole expérimental ne voie le jour, nous nous voyons dans l'obligation d'utiliser des primitifs cryptographiques standards dans le but de maintenir la sécurité des puces RFID. Le dernier chapitre de la thèse a analysé des versions de ce primitif cryptographique standard qui sont particulièrement idéaux pour les appliquer sur des RFID.

Le protocole RFID finalement choisi et utilisé peut influencer dans une large mesure la propagation des RFID. Dans la mesure où il s'avère que le protocole choisi et implémenté est défectueux et qu'ainsi tous les produits marqués par ce protocole deviennent traçables, cela peut provoquer un tel désastre aux RFID qu'il peuvent être retirées de la vie quotidienne. En revanche, si le protocole choisi est convenable, fiable et s'il ne peut pas être facilement brisé, la puce RFID peut passer à chaque produit commercial et un lecteur RFID peut passer à tous les outils quotidiens.

Bibliographie

[1] Bárasz, M., Boros, B., Ligeti, P., Lója, K., and Nagy, D. Breaking LMAP. In *Conference on RFID Security — RFIDSec'07* (Malaga, Spain, July 2007), pp. 69–78.

[2] Bárász, M., Boros, B., Ligeti, P., Lója, K., and Nagy, D. A. Passive attack against the M2AP mutual authentication protocol for RFID tags. In *RFID 2007 — The First International EURASIP Workshop on RFID Technology* (September 2007).

[3] Bard, G. V. Algorithms for the solution of polynomial and linear systems of equations over finite fields, with an application to the cryptanalysis of KeeLoq. Tech. rep., University of Maryland Dissertation, April 2008. Ph.D. Thesis.

[4] Baumgartner, P., and Massacci, F. The taming of the (X)OR. In *Computational Logic — CL 2000* (2000), vol. 1861/2000 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 508–522.

[5] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., Seurin, Y., and Vikkelsoe, C. PRESENT : An ultra-lightweight block cipher. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2007* (Vienna, Austria, September 2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 450–466.

[6] Bolotnyy, L., and Robins, G. Physically unclonable function-based security and privacy in RFID systems. In *PerCom 2007* (March 2007), IEEE, pp. 211–220.

[7] Bringer, J., Chabanne, H., and Dottax, E. HB^{++} : a lightweight authentication protocol secure against some attacks. In *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006 — SecPerU 2006* (June 2006), pp. 28–33.

[8] Burmester, M., Le, T. v., and Medeiros, B. d. Provably secure ubiquitous systems : Universally composable RFID authentication protocols. In *Conference on Security and Privacy for Emerging Areas in Communication Networks — SecureComm '06* (Baltimore, Maryland, USA, August-September 2006), IEEE.

[9] Cannière, C. D. Trivium : A stream cipher construction inspired by block cipher design principles. In *ISC* (2006), S. K. Katsikas and et al, Eds., vol. 4176 of *LNCS*, Springer, pp. 171–186.

[10] Castelluccia, C., and Soos, M. Secret shuffling : A novel approach to RFID private identification. In *RFIDSec'07* (July 2007), pp. 169–180.

[11] Conti, M., Pietro, R. D., Mancini, L. V., and Spognardi, A. RIPP-FS : an RFID identification, privacy preserving protocol with forward secrecy. In *International Workshop on Pervasive Computing and Communication Security — PerSec '07* (New York City, New York, USA, March 2007), IEEE, IEEE Computer Society Press, pp. 229–234.

[12] Crawford, J. M., Kearns, M. J., and Shapire, R. E. The minimal disagreement parity problem as a hard satisfiability problem. Tech. rep., Computational Intelligence Research Laboratory and AT&T Bell Labs, February 1994.

[13] Eén, N., and Sörensson, N. An extensible SAT-solver. In *SAT* (2003), E. Giunchiglia and A. Tacchella, Eds., vol. 2919 of *LNCS*, Springer, pp. 502–518.

[14] EPCglobal. 13.56 MHz ISM band class 1 radio frequency identification tag interface specification (2003). Tech. rep., Auto-ID cetner, MIT, February 2003.

[15] Feldhofer, M., Wolkerstorfer, J., and Rijmen, V. AES implementation on a grain of sand. In *Information Security* (2005), IEEE, pp. 13–20.

[16] Fossorier, M. P. C., Mihaljević, M. J., Imai, H., Cui, Y., and Matsuura, K. A novel algorithm for solving the LPN problem and its applicatio to security evaluation of the HB protocol for RFID authentication. In *INDOCRYPT* (2006), R. Barua and T. Lange, Eds., vol. 4329 of *Lecture Notes in Computer Science*, Springer, pp. 48–62.

[17] Gilbert, H., Robshaw, M., and Sibert, H. An active attack against HB^+ - a provably secure lightweight authentication protocol. In *IEE Electronic Letters 41, 21* (2005), pp. 1169–1170.

[18] Gilbert, H., Robshaw, M. J. B., and Seurin, Y. $HB^{\#}$: Increasing the security and efficiency of HB^+. In Smart [37], pp. 361–378.

[19] Hell, M., Johansson, T., and Meier, W. Grain — a stream cipher for constrained environments. In *Proceeding of the Workshop on RFID and Lightweight Crypto* (July 2005), pp. 114–125.

[20] ISO/IEC. 14443-3 — Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 3 : Initialization and anticollision, 2001, Stage : 90.92 — 2007-12-11.

[21] Juels, A. Minimalist cryptography for low-cost RFID tags. In *International Conference on Security in Communication Networks — SCN 2004* (Amalfi, Italia, September 2004), C. Blundo and S. Cimato, Eds., vol. 3352 of *LNCS*, Springer-Verlag, pp. 149–164.

[22] Juels, A., and Weis, S. Authenticating pervasive devices with human protocols. In *Advances in Cryptology — CRYPTO'05* (Santa Barbara, California, USA, August 2005), V. Shoup, Ed., vol. 3126 of *LNCS*, IACR, Springer-Verlag, pp. 293–308.

[23] Juels, A., and Weis, S. Defining Strong Privacy for RFID. In *International Conference on Pervasive Computing and Communications — PerCom 2007* (New York City, New York, USA, March 2007), IEEE, IEEE Computer Society Press, pp. 342–347.

[24] Koutarou, M. O., Suzuki, K., and Kinoshita, S. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop* (MIT, Massachusetts, USA, November 2003).

[25] McLoone, M., and Robshaw, M. J. B. Public key cryptography and RFID tags. In *CT-RSA* (2007), M. Abe, Ed., vol. 4377 of *Lecture Notes in Computer Science*, Springer, pp. 372–384.

[26] Molnar, D., and Wagner, D. Privacy and security in library RFID : issues, practices, and architectures. In *CCS '04 : Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), ACM Press, pp. 210–219.

[27] Munilla, J., and Peinado, A. HB-MP : A further step in the hb-family of lightweight authentication protocols. *Comput. Netw. 51*, 9 (2007), 2262–2267.

[28] O'Donnell, C. W., Suh, G. E., and Devadas, S. PUF-based random number generation. In *MIT CSAIL CSG Technical Memo 481* (November 2004).

[29] Oren, Y., and Feldhofer, M. WIPR — a public key implementation on two grains of sand. In *Workshop on RFID Security 2008* (2008), S. Dominikus, Ed., pp. 15 – 27.

[30] Ouafi, K., Overbeck, R., and Vaudenay, S. On the security of HB# against a man-in-the-middle attack. In *Advances in Cryptology — Asiacrypt 2008* (Melbourne, Australia, December 2008), vol. 5350 of *Lecture Notes in Computer Science*, Springer, pp. 108–124.

[31] Ouafi, K., and Phan, R. C.-W. Privacy of Recent RFID Authentication Protocols. In *Information Security Practice and Experience, 4th International Conference, ISPEC 2008* (Berlin, 2008), Lecture Notes in Computer Science, Springer, pp. 263–277.

[32] Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J., and Ribagorda, A. LMAP : A real lightweight mutual authentication protocol for low-cost RFID tags. In *Proceedings of RFIDSec'06* (Graz, Austria, July 2006), Ecrypt.

[33] Pietro, R. D., and Molva, R. Information confinement, privacy, and security in RFID systems. In *Proceedings of the 12th European Symposium On Research In Computer Security* (September 2007), pp. 187–202.

[34] Raddum, H. Cryptanalytic results on Trivium. Tech. Rep. 2006/039, ECRYPT Stream Cipher Project, 2006. www.ecrypt.eu.org/stream/papersdir/2006/039.ps.

[35] Shamir, A. SQUASH — a new MAC with provable security properties for highly constrained devices such as RFID tags. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, pp. 144–157.

[36] Sinz, C. Visualizing SAT instances and runs of the DPLL algorithm. *J. Autom. Reason. 39*, 2 (2007), 219–243.

[37] Smart, N. P., Ed. *Advances in Cryptology — EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings* (2008), vol. 4965 of *Lecture Notes in Computer Science*, Springer.

[38] Strassen, V. Gaussian elimination is not optimal. *Numerische Mathematik 13* (1969), 354–356.

[39] Tsudik, G. YA-TRAP : Yet another trivial RFID authentication protocol. In *International Conference on Pervasive Computing and Communications — PerCom 2006* (Pisa, Italy, March 2006), IEEE, IEEE Computer Society Press, pp. 640–643.

[40] Éric Levieil, and Fouque, P.-A. An improved LPN algorithm. In *Security and Cryptography for Networks — SCN* (2006), R. D. Prisco and M. Yung, Eds., vol. 4116 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 348–359.