Security analysis of contactless communication protocols

David Gerault

To cite this version:

David Gerault. Security analysis of contactless communication protocols. Other [cs.OH]. Université Clermont Auvergne [2017-2020], 2018. English. NNT : 2018CLFAC103 . tel-02536478

HAL Id: tel-02536478
https://theses.hal.science/tel-02536478

Submitted on 8 Apr 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Clermont Auvergne
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
CNRS, UMR 6158, LIMOS, 63173 Aubière, France

Security Analysis of Contactless Communication Protocols

Thèse

pour obtenir le

Doctorat de L'Université Clermont Auvergne

Spécialité informatique

École Doctorale des Sciences Pour l'Ingénieur

présentée et soutenue publiquement par

DAVID GERAULT

le 27/11/2018

devant le jury composé de :

M. Kasper Rasmussen **Department of computer science**	Associate Professor University of Oxford	Rapporteur
Mme María Naya-Plasencia **Equipe SECRET**	Directrice de recherche INRIA Paris	Rapporteure
M. Yves Deville **ICTEAM**	Professeur des universités Université Catholique de Louvain	Examinateur
M. Cédric Lauradoux **Equipe PRIVATICS**	Chargé de recherche INRIA Rhone-Alpes	Examinateur
Mme Marine Minier **LORIA**	Professeure des universités Université de Lorraine	Examinatrice
Mme Christine Solnon **LIRIS**	Professeure des universités INSA Lyon	Examinatrice
M. Pascal Lafourcade **LIMOS**	Maître de conférences Université Clermont Auvergne	Directeur de thèse

Remerciements

En premier lieu, je tiens à remercier Pascal, pour son encadrement tout au long de cette thèse. Il a dépassé de loin ce qu'on est en droit d'attendre d'un directeur de thèse. Il a toujours été disponible et extrêmement réactif, peu importe l'heure, pour travailler, relire ou conseiller. Après ces trois années, je n'ai toujours pas compris comment il arrivait encore à trouver le temps de dormir. J'ai beaucoup appris auprès de lui, tant sur le plan professionel que personnel. Merci Pascal.

Bien que Pascal soit mon directeur de thèse "officiel", j'ai également eu la chance d'être encadré, pour la partie cryptanalyse et programmation par contraintes, par Christine et Marine. Je ne saurais assez les remercier pour la patience et la pédagogie dont elles ont fait preuve.

Au cours de cette thèse, j'ai eu l'occasion de travailler avec plusieurs co auteurs, que je remercie également: Cristina, Ioana (j'attends toujours mon nouveau chapeau en papier), Jean-Marc (grand maître des probabilités), Sébastien, Gildas, Siwei, Hardik, et Manik.

Bien sûr, je remercie Kasper Rasmussen, María Nayya-Plasencia, Yves Deville et Cédric Lauradoux d'avoir bien voulu faire partie de mon jury.

Plus personellement, je remercie ma famille, en premier lieu ma grand mère, pour son soutien inconditionel tout au long de ce parcours: sans elle, cette thèse n'aurait peut être jamais vu le jour. Je remercie aussi ma mère Christine pour son aide et ses encouragements au cours de la période de rédaction, et mon frère Cyril.

Pendant ces trois années, j'ai pu compter sur le soutien et l'amitié de mon grand frère de thèse, Xavier. Des sessions de travail au tableau, ou il m'a initié aux joies de la sécurité prouvable, aux squattages de canapé mutuels, fous rires et autres chansons, cette thèse n'aurait pas été la même sans lui.

Et puis, bien sûr, après le grand frère de thèse, le petit: merci Matthieu, pour tes conseils avisés dans tous les domaines, les leçons de danse en Croatie, et tout le reste.

Je remercie également toute l'équipe du séminaire des doctorants pour cette expérience enrichissante: Hamadoun, Giacomo, Loukhmen, Sylvestre et Mouna.

Je remercie les collègues, plus particulièrement les membres (et anciens membres) du bureau B6, ainsi que les "stagiaires" (Marie-Caroline et Gauthier), et bien sûr Coco, pour la bonne humeur partagée au cours de ces années.

Merci à mes amis danseurs, pour les moments de lâcher prise Ô combien necessaire pendant cette longue aventure qu'est la thèse: Lélie (aka partner), Guillaume (dépositaire de bien trop de dossiers), Alex (courage pour la dernière ligne droite), Gaelle (bretonne chaotique), et tous les autres, et tout particulièrement Mathilde (qui m'a demandé plus souvent ou en était la rédaction que qui que ce soit, y compris Pascal) pour son incroyable joie de vivre, sa capacité à me faire rire en toutes circonstances, et ses incursions dans ma playlist.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Authentication, in which an entity acknowledges another, is one of the most fundamental applications of cryptography. Nowadays, authentication is increasingly often performed with contactless devices. Contactless authentication has become omnipresent in our world. Be it on the workplace, where access cards are replacing traditional keys to open the doors, or in our personal life, when we use contactless payment, most of us are regularly confronted to the contacless technologies. The large scale deployment of contactless authentication brings a strong incentive to scrutinize its security, as attacks against contacless communication protocols could impact all aspects of our lives, from professional (with access cards) to personal (with contactless payment). The goal of this thesis is to study the security of contactless communications, and in particular, contactless authentication.

1.1 Introduction

At the core of the security of contactless communications protocols lies the question of secure *authentication*. Authentication is the set if techniques that permits an entity, referred to as the *verifier*, to validate the legitimacy of another entity, called the *prover*. Authentication is generally performed for the prover to obtain access to a given service from the verifier, for instance, access to a building. Authentication protocols define the messages to be exchanged, and the acceptance rules. If we take a very simple scenario, in which a guard asks visitors a passwords before opening the door for then, the protocol is the following: the guard asks for the passwords, and the visitor responds. If the password given by the visitor is correct, then the guard opens the door. Otherwise, the guard chases the visitor. In this case, the prover is authenticated using something he knows. This is one of the three authentication mechanisms that are usually employed. Authentication is generally achieved by verifying something:

- The prover *owns*, for instance an access card;
- the prover *knows*, for instance, a secret handshake;
- the prover *is*, for instance, through his fingerprints.

Nowadays, in a contactless context, authentication is performed using a device the prover owns. For instance, in the Passive Keyless Entry and Start (PKES) technology, the traditional car key is replaced by a smart key. Smart keys are contactless devices: when a user holding one is close to the car, the car authenticates him as legitimate, and the door open. The technology that is used by contactless authentication protocols is called RFID, for Radio Frequency IDentification. An RFID exchange involves a *tag*, who plays the role of the prover, and a *reader*, who acts as the verifier. The most widely used kind of RFID tags is *passive* tags, which have no battery and receive power from the verifier during the exchange. Their wide usage is due to their very limited cost (a few cents), compared to *active* tags, which have their own power source. Passive tags are typically less powerful, and it is believed that further improvements in technology will not aim at making them more powerful, but cheaper [Juels, 2006]. Hence, authentication protocols designed for RFID need to face the challenge of being secure, while working with low computational power. The design of lightweight authentication protocols, suitable for low power RFID tags, is difficult: most proposals have serious security flaws [Avoine et al., 2015]. At the other end of the spectrum, modern smartphones are generally equipped with an NFC chip. NFC is a subset of RFID, which operates on short distances, and in which the entities involved are not restricted to being either a tag or a reader, but can be both. Therefore, computational power is no longer necessarily a limitation for contactless authentication protocols. It should however still be taken into account, depending on the application. Another challenge for the security of contactless protocols is that the communications between the prover and the reader are sent in the air, so that any passive adversary with an antenna can read the messages exchanged. In contrast, eavesdropping wired communications requires more intrusive techniques, so that there is more surface of attacks on contactless devices. Hence, security is challenging to achieve for contactless authentication protocols.

The security of authentication protocols is defined with regards to an *adversary*, who tries to defeat the protocol. Defeating the protocol can have various meanings. The first one is of course obtaining unauthorised access to a service. The basic goal of an authentication protocol is to prevent this kind of attacks. However, other threats must be taken into account. For instance, legitimate provers could try to cheat on the protocol, by helping an unauthorised accomplice to gain access to a service. For instance, consider a public transportation system, which is accessed to with a contactless card. The card authenticates the user. A user should not be able to permit other people, who did not pay for a card, to authenticate on their behalf and use the public transportation. Finally, authentication protocols are sometimes required to protect the privacy of users. By privacy, we mean that, given the messages exchanged in two executions of the protocol, it should be difficult to determine whether the two executions were performed by the same user or not. We believe that privacy is highly relevant for contactless authentication. By relating two different executions of a protocol to the same user, an adversary can possibly infer a lot about the private life of the user. For instance, if the transaction history of a contactless payment system is leaked, and if the adversary is capable of linking which execution corresponds to a given device, then he practically obtains a location history of an individual. Indeed, if a honest user performed a contactless payment in a shop, then one can deduce he was in that shop. Location history can give a lot of information about a person, and even when no identifier is attached to the logs, it can be sufficient for an attacker to obtain the home or work address, and even identity and social links, of an individual [Krumm, 2009]. Such information can be used for criminal purpose, such as robbery (while the user is at work), or blackmailing, for instance if the adversary identifies suspicious behaviour from the user. The more data the adversary has, the more he can infer about users. In particular, the adversary can be a big company, with access to the logs for several services. Therefore, we believe that contactless authentication protocols should protect the privacy of their users, both with respect to external adversary who obtain some logs, and with respect to the verifiers themselves.

Authentication protocols, such as the Needham-Schroeder [Needham and Schroeder, 1978] protocols, use cryptographic building blocks, for encrypting or authenticating messages. Hence, there are two aspects to the security of an authentication protocol. The first one is the protocol in itself: if it has logical flaws, then it can be defeated. For instance, the Needham-Shroeder protocols are vulnerable to attacks: in [Denning and Sacco, 1981], the authors show how an adversary can defeat one of the protocols by replaying messages observed in a previous execution. The second aspect to the security of authentication protocols is the security of the cryptographic building blocks that compose it. For instance, if the protocol uses encryption, but the encryption scheme is not secure, then the protocol might be insecure. The search for vulnerabilities in cryptographic primitives is called *cryptanalysis*. In this thesis, we study both the security of the authentication protocols themselves, and of the cryptanalysis of the primitives that compose them. In the first part of this manuscript, we consider the case of distance bounding protocols, which were introduced to counter relay attacks, in which an adversary passively relays the messages exchanged between the prover and the verifier to illegally authenticate. In the second part, we propose to use

Constraint Programming (CP), a declarative programming paradigm, to evaluate the security of a cryptographic primitive called block ciphers.

1.2 Part 1: Distance Bounding Protocols

In this section, we introduce the first part of the thesis.

1.2.1 Relay Attacks, or the Chess Grand Master Problem

In 1976, Conway wrote the tale of "*the little girl who played [...] against two Chess Bandmasters [...]. How was it that she managed to win one of the games? Anne-Louise played Black against Spassky. White against Fisher. Spassky moved first, and Anne-Louise just copied his move as the first move of her game against Fisher, then copied Fisher's reply as her own reply to Spassky's first move, and so on*" [Conway, 1976]. This little story, known as the Chess Grand Master Problem, paved the way for a very rich area of research on relay attacks. It sets a very interesting problem: by simply relaying the moves from one player to the other without them being aware of the trick (for instance by having them sit in two different rooms, or playing the games via postal mail), someone can win or obtain a draw even without knowing the rules of chess.

However, relay attacks are not only a problem for chess players. An authentication protocol can be seen as a two-player game between a prover and a verifier. If an adversary relays the messages from one to the other, then he can win the game against the verifier and authenticate successfully. Eleven years after the first mention of the Chess Grand Master Problem, Desmedt applied relay attacks, under the name of Mafia Fraud, against a security protocol [Desmedt et al., 1987]. In his example, the Mafia Fraud is performed against a payment protocol ran on a credit card. The victim wants to pay for his meal in a mafia owned restaurant. The employees of the restaurant, who are malicious, can hijack this payment, and relay it to buy expensive jewellery. To do so, the employees build a fake card reader and a fake card, that are able to communicate with each other from a distance, for instance via a radio link. One of the employees presents the fake reader to the victim for his payment, while a second one presents the fake card at the jewellery store. Both employees then just let the fake card and reader forward to each other the messages they receive. By this manipulation, the victim believes that his card is performing the payment protocol with the restaurant through the reader, when the card is actually communicating with the reader at the jewellery store due to the relaying. As for the Chess Grand Master Problem, the employees do not need to know anything about the secrets involved in the payment protocol in use, as long as they are able to relay the messages.

At the time when it was introduced, this attack was somehow mitigated by the fact that the card had to be inserted into a reader. Performing a relay attack required the victim to be willing to actively take part in the payment, which could limit the large scale applicability.

On the other hand, in contactless payment solutions, no action from the card owner is required. It suffices that the card is close enough to a reader to start the protocol, as it automatically responds when it is solicited. Moreover, relaying messages does not require any suspicious equipment anymore, as smartphones can be used both as fake readers and as contactless payment means. Relay attacks can indeed be performed with off-the-shelf smartphones, on actual payment protocols, as shown by Vila and Rodriguez in [Vila and Rodríguez, 2015]. For instance, an attacker can approach his smartphone close enough to the pocket where the victim stores his card, instead of having to ask the victim to insert his card in a terminal. On the other side, an accomplice of the attacker would just present his own smartphone as a means of payment to the reader of the jewellery store, with both phones communicating through a network. Relay attacks were also used to steal cars, by relaying messages between a smart key and a car [Francillon et al., 2011]. Since then, significant research effort was put in solving relay attacks, in particular through distance bounding, for instance [Dürholz et al., 2011, Boureanu et al., 2015, Debant et al., 2018].

1.2.2 Solving the Chess Grand Master Problem: Distance Bounding Protocols

In their 1991 paper [Beth and Desmedt, 1991], Beth and Desmedt proposed to use time measurements for solving the Chess Grand Master Problem. The idea is to predefine a time interval t between each move. If one of the player does not play his move exactly t units of time after the previous move was played, then the other player detects there is a fraud. The first protocol relying on time measurements was introduced by Brands and Chaum in 1993 [Brands and Chaum, 1994]. The aim of this protocol is to estimate the distance between the prover and the verifier: in the event of a relay attack, it is likely that the victim is far away from the verifier. For instance, for a contactless payment, the prover should be no further than a few centimetres from the reader. To bound the distance, the round trip time of the messages exchanged during the protocol is measured by the verifier. Typically, the verifier starts a clock, sends a cryptographic challenge, and stops the clock when it receives the response. The measured time Δ_t corresponds to twice the time it takes for a message to go from the prover to the verifier (or vice versa), plus the time taken by the prover to reply. To transform this time into a distance, it suffices to observe that no information can travel faster than the speed of light c. Hence, then $d = \frac{\Delta_t \cdot c}{2}$ is an upper bound on the distance between the prover and the verifier. If the prover was any further than d, then it would mean that the messages travelled faster than light, which is assumed impossible. Consequently, if d is short enough, according to a predefined, application dependant bound d_{max}, then the verifier gains the insurance that the prover is close enough. In other words, there is a time bound t_{max}, corresponding to a distance bound d_{max}, such that, if $\Delta_t > t_{\mathsf{max}}$, then the verifier rejects the prover. Time based distance bounding is currently the most studied solution to relay attacks.

However, it is noteworthy that other approaches were proposed, even though they are out of the scope of this thesis. In his seminal paper [Desmedt, 1988], Desmedt proposed that the prover computes his exact location on earth, signs it, and sends it to the verifier. The inconvenient to this approach is that it requires to trust the prover not to cheat. In addition, it requires a safe localisation system, which is not trivial to realise. In particular, using the GPS technology does not seem to be a robust solution [Gorjón and van Iterson, 2015]. Another option is to measure the strength of the signal received by the verifier [Krumm and Horvitz, 2004]: since it decreases as the distance increases, it gives indications on the distance of the prover. However, an attacker can amplify the signal to make the prover appear closer to the verifier, and defeat this system. Similarly, several solutions [Urien and Piramuthu, 2014] based on sensing the local environment (for instance the air temperature) were proposed, with the idea that if the prover was actually close to the verifier, then it would sense similar values. This approach however fails if the adversary is able to manipulate the value that is being sensed, which can be relatively easy. To prevent relay attacks, one can also isolate the prover inside a Faraday cage [Bengio et al., 1991] during the protocol, to make sure that it cannot communicate with external entities. While efficient, this solution is not very user friendly, and limits the usability of the system. Finally, radio frequency fingerprinting [Rasmussen and Capkun, 2007] can be used. It identifies the devices based on variations in the signal features due to imperfections in the manufacturing process. However, such fingerprinting can be counterfeited [Danev et al., 2010], which renders this approach impractical.

Hence, distance bounding protocols seem to be the most promising option to defeat relay attacks. More than 40 distance bounding protocols exist in the literature [Brelurut et al., 2016, Gildas et al., 2017], and most of them are vulnerable to at least one form of attacks. There exists 4 main types of attacks against distance bounding protocols, some performed by outsiders, and some by legitimate users. These attacks are called Mafia Fraud (MF), Distance Fraud (DF), Distance Hijacking (DH) and Terrorist Fraud (TF). A mafia fraud is performed by an external adversary, who tries to authenticate in the presence of a far away honest prover. In a distance fraud, the attacker is a legitimate, far away prover, who tries to authenticate from a distance. In a distance hijacking, the legitimate, far away prover additionally uses honest provers, located close to the verifier, to authenticate from a distance. Finally, in a terrorist fraud, he is helped by an accomplice, located close to the verifier.

1.2.3 Challenges and Contributions

Designing secure protocols is challenging, as shown by the small portion of distance bounding protocols in the literature that are resistant to all attacks. In a survey [Brelurut et al., 2016], we showed that among the 42 studied protocols, only 10 were not known to be vulnerable to at least one of the classical threats against distance bounding protocols. Formal security models, such as DFKO [Dürholz et al., 2011], provide a framework, in which the security of a protocol can be proven. This proof ensures that the protocol is secure with regards to the threats as defined in the model. However, it is notoriously difficult to formalise, terrorist fraud [Fischlin and Onete, 2013a]. Several formal models for terrorist fraud exist, *e.g.*, [Dürholz et al., 2011, Boureanu et al., 2015, Fischlin and Onete, 2013a], and most are designed specifically to prove the security of one particular protocol. Provable terrorist fraud resistance generally involves the introduction of intricate mechanisms, or even backdoors [Fischlin and Onete, 2013a], that only serve for the proof to work.

In the first part of this thesis, we study the problem of provably TF-resistant distance bounding, with or without privacy. We present two of our protocols, SPADE and TREAD [Bultel et al., 2016, Avoine et al., 2017], which use novel techniques to attain provable terrorist fraud resistance, and various degrees of privacy. We then present a generic terrorist fraud, which contradicts the security proofs of most existing distance bounding protocols (to the best of our knowledge, it concerns all protocols but [Igier and Vaudenay, 2016]). This terrorist fraud exhibits attack possibilities that are not considered in the security models. We propose a new terrorist fraud definition, called One-Step Terrorist Fraud (OSTF) that accounts for this attack.

1.3 Part 2: Using Constraint Programming for Cryptanalysis

Authentication protocols, including distance bounding protocols, use cryptographic primitives as building blocks. One of the most versatile cryptographic primitives, which can be used to build other ones, is block ciphers. Block ciphers are symmetric encryption schemes, which encrypt message blocks of fixed size into ciphertexts of fixed size. Block ciphers can be used to build other useful cryptographic primitives, such as hash functions, pseudorandom functions and message authentication codes, which are defined in Chapter 2. We chose to study the security of the primitives used in contactless authentication protocols through the security of block ciphers. For a block cipher, being secure means that it is difficult for an adversary to recover the secret key used to encrypt messages. The cryptanalysis of block ciphers is a very active research field. Cryptanalysing a block cipher typically involves bounding its security against generic, known attacks. For instance, one of the most famous type of attacks is differential cryptanalysis [Biham and Shamir, 1991]. In a differential attack, the adversary exploits biases in the distribution of the XOR difference of ciphertexts obtained by ciphering messages with a given XOR difference under the same key. An example of bias could be the following: if two messages differing only in the last bit are encrypted with the same key (under a given block cipher), then there is a high probability that the two resulting ciphertexts will differ in their last bit too. Most of the time, the biases are not that obvious, so the cryptographers must study the propagation of plaintext differences through the cipher to determine which difference propagation path, or *differential characteristic*, is the most likely to occur. By doing so, they estimate the probability of the best *differential*, *i.e.*, an input difference mapped to an output difference. Modern block ciphers typically operate on 64- or 128-bit plaintexts, so that trying all possible input/output difference pairs is intractable. Hence, the search for optimal differential characteristics is difficult. However, a clever branch and bound approach can tackle it [Matsui, 1994]. It is however more difficult when we also consider differences in the key, as in *related key differential cryptanalysis*. In a related-key differential attacks, the attacker can obtain the encryption of messages of his choice with the secret key, but also with related keys, which have a relation of his choice with the secret key. This kind of attacks is particularly relevant when a block cipher is used to build other primitives. For instance, Microsoft's Xbox uses a hash function built from a block cipher, TEA [Wheeler and Needham, 1995]. An attack on the hash function, due to a vulnerability of TEA in the related-key setting, permitted a hack of the system [ZDNet, 2002].

1.3.1 Challenges and Contributions

Performing related key differential cryptanalysis requires studying how differences in the message and in the key propagate through the ciphering process, in order to find the optimal related key differential characteristics. The size of the search space is very large: for AES, the encryption standard, the messages are 128 bits, and the key is 128 bits or more, so that exhaustive search is intractable. Dedicated search algorithms, which avoid exhaustive search by using strategies to reduce the search space, exist, *e.g.*, [Biryukov and Nikolic, 2010, Fouque et al., 2013]. However, designing such algorithms is time consuming, and results in large and difficult to maintain code. On the other hand, declarative paradigms, in which the developer simply states the problem, and the search is left to a dedicated solver, are sometimes used for cryptanalysis [Mouha et al., 2012, Sun et al., 2014, Derbez and Fouque, 2016]. The most used paradigm is Mixed Integer Linear Programming (MILP).

In the second part of this thesis, we use Constraint Programming (CP) to perform related key differential cryptanalysis on two block ciphers: the standard AES [Daemen and Rijmen, 2002] and the lightweight block cipher Midori [Banik et al., 2015]. Constraint programming is a declarative framework which was previously not used for cryptanalysis. It is more generic than MILP: in MILP, the problems need to be expressed as set of linear equations, while in CP, there is no such limitation. The resolution methods are also different.

1.4 Outline

In Chapter 2, we introduce the notations we use through the thesis, as well as the cryptographic primitives that we use and their security properties, and some useful mathematical notions. In Chapter 3, we give an introduction to distance bounding protocols, and define the associated attacks. We present the DFKO [Dürholz et al., 2011] framework, in which we perform our security proofs, and the proof methodology. In Chapter 4, we present two of our distance bounding protocols: SPADE [Bultel et al., 2016] and TREAD [Avoine et al., 2017]. The research problem that these protocols aim to solve is provable terrorist fraud resistance, and how to combine it with privacy. SPADE is fully anonymous, and uses a previously known mechanism to grand provable terrorist fraud resistance. TREAD has 3 instances, two of which are concerned with privacy and anonymity, and implements a novel technique for provable terrorist fraud resistance. We give the security proofs of both protocols. In Chapter 5, we present an impossibility result for terrorist fraud resistance, in the form of a generic terrorist fraud. This attacks works for every protocols in which the prover can be cloned. Provers can typically always be cloned, unless hardware measures make them physically uncloneable, for instance by using a temper-proof device which prevents the user from accessing the secret key, or by using physically uncloneable function. We then define a new terrorist fraud resistance notion, which takes this impossibility result into account. Chapter 6 introduces the search for optimal related key differential characteristics using constraint programming. We present the problem that we solve, the methodology that we use, as well as the tools and the experimental setup. Chapter 7 presents our CP models and results for the related key cryptanalysis of the AES. We build up from a naive modelling, with which the search is intractable, to an efficient model, using a different decomposition, that performs significantly better (less than 24 hours versus several weeks) than the state of the art methods. Moreover, we find related key differential characteristic which are better (in terms of probability) than the ones that were previously published. Finally, in Chapter 8, we present our CP models for the related key cryptanalysis of Midori [Banik et al., 2015], a lightweight block cipher [Gérault and Lafourcade, 2016] that comes in two versions: Midori64 and Midori128. Using CP, we find the optimal related key differential characteristics for both versions within a few minutes. With these related-key differential characteristics, we present key recovery attacks with practical complexity: 2^{36} operations for Midori64, and 2^{44} for Midori128.

1.5 Publications

In this section, we present the list of the articles we published during this thesis.

1.5.1 Presented in This Manuscript

The articles that we present in this manuscript are the following:

A Prover-Anonymous and Terrorist-Fraud Resistant Distance-Bounding Protocol [Bultel et al., 2016]

A Terrorist-fraud Resistant and Extractor-free Anonymous Distance-bounding Protocol [Avoine et al., 2017]

Fine-Grained and Application-Ready Distance-Bounding Security [Boureanu et al., 2018]

Constraint Programming Models for Chosen Key Differential Cryptanalysis [Gerault et al., 2016]

Revisiting AES related-key differential attacks with constraint programming [Gérault et al., 2018]

Combining Solvers to Solve a Cryptanalytic Problem [Gerault et al., 2017a]

Using Constraint Programming to solve a Cryptanalytic Problem [Gerault et al., 2017b]

Related-Key Cryptanalysis of Midori [Gérault and Lafourcade, 2016]

1.5.2 Other Publications

Some of our publications are left out of this thesis. We briefly present them. Their titles and abstracts are given below.

Survey of Distance Bounding Protocols and Threats [Brelurut et al., 2016]: NFC and RFID are technologies that are more and more present in our life. These technologies allow a tag to communicate without contact with a reader. In wireless communication an intruder can always listen and forward a signal, so he can mount a so-called worm hole attack. In the last decades, several Distance Bounding (DB) protocols have been introduced to avoid such attacks. In this context, there exist several threat models: Terrorist Fraud, Mafia Fraud, Distance Fraud etc. We first show the links between the existing threat models. Then we list more than forty DB protocols and give the bounds of the best known attacks for different threat models. In some cases, we explain how we are able to improve existing attacks. Then, we present some advice to the designers of the DB protocols and to the intruders to mount some attacks.

Breaking and fixing the HB+DB protocol [Boureanu et al., 2017]: The HB protocol and its HB+ successor are lightweight authentication schemes based on the Learning Parity with Noise (LPN) problem. They both suffer from the so-called GRS-attack whereby a man-in-the-middle (MiM) adversary can recover the secret key. At WiSec 2015, Pagnin et al. proposed the HB+DB protocol: HB+ with an additional distance-bounding dimension added to detect and counteract such MiM attacks. They showed experimentally that HB+DB was resistant to GRS adversaries, and also advanced HB+DB as a distance-bounding protocol, discussing its resistance to worst-case distance-bounding attackers. In this paper, we exhibit flaws both in the authentication and distance-bounding layers of HB+DB; these vulnerabilities encompass practical attacks as well as provable security shortcomings. First, we show that HB+DB may be impractical as a secure distance-bounding protocol, as its distance fraud and mafia-fraud security-levels scale poorly compared to other distance-bounding protocols. Secondly, we describe an effective MiM attack against HB+DB: our attack refines the GRS-strategy and still leads to key-recovery by the attacker, yet this is not deterred by HB+DB's distance bounding. Thirdly, we refute the claim that HB+DB's security against passive attackers relies on the hardness of the LPN problem. We also discuss how (erroneously) requiring such hardness, in fact, lowers HB+DB's efficiency and its resistance to authentication and distance-bounding attacks. Drawing on HB+DB's design flaws, we also propose a new distance-bounding protocol – BLOG. It retains parts of HB+DB, yet BLOG is provably secure, even – in particular – against MiM attacks. Moreover, BLOG enjoys better practical security (asymptotic in the security parameter).

Verifiable Private Polynomial Evaluation [Bultel et al., 2017]: Delegating the computation of a polynomial to a server in a verifiable way is challenging. An even more challenging problem is ensuring that this polynomial remains hidden to clients who are able to query such a server. In this paper, we formally define the notion of Private Polynomial Evaluation (PPE). Our main contribution is to design a rigorous security model along with relations between the different security properties. We define polynomial protection (PP), proof unforgeability (UNF), and indistinguishability against chosen function attack (IND-CFA), which formalises the resistance of a PPE against attackers trying to guess which polynomial is used among two polynomials of their choice. As a second contribution, we give a cryptanalysis of two PPE schemes of the literature. Finally, we design a PPE scheme called PIPE and we prove that it is PP-, UNF- and IND-CFA-secure under the decisional Diffie-Hellman assumption in the random oracle model.

Analysis of AES, SKINNY, and Others with Constraint Programming [Sun et al., 2017]: Search for different types of distinguishers are common tasks in symmetric key cryptanalysis. In this work, we employ the constraint programming (CP) technique to tackle such problems. First, we show that a simple application of the CP approach proposed by Gerault et al. leads to the solution of the open problem of determining the exact lower bound of the number of active S-boxes for 6-round AES-128 in the related-key model. Subsequently, we show that the same approach can be applied in searching for integral distinguishers, impossible differentials, zero-correlation linear approximations, in both the single-key and related-(twea)key model. We implement the method using the open source constraint solver Choco and apply it to the block ciphers PRESENT, SKINNY, and HIGHT (ARX construction). As a result, we find 16 related-tweakey impossible differentials for 12-round SKINNY-64-128 and construct an 18-round attack on SKINNY-64-128 (one target version for the crypto competition https://sites.google.com/site/skinnycipher announced at ASK 2016). Moreover, we show that in some cases, when equipped with proper strategies (ordering heuristic, restart and dynamic branching strategy), the CP approach can be very efficient. Therefore, we suggest that the constraint programming technique should become a convenient tool at hand of the symmetric-key cryptanalysts.

Chapter 2

Cryptographic Tools

In this section, we present the mathematical and cryptographi notions that we use in our algorithms and proofs.

Contents

2.1 Introduction

In Part I of this manuscript, we are concerned with provable security. In provable security, the adversary is assumed to be *polynomially bounded, i.e.*, he can only perform a number of operations that is a polynomial in a *security parameter* λ. Hence, a scheme is said to be secure if the success probability of an adversary diminishes exponentially as the security parameter increases. In other words, a scheme is secure if the success probability of any adversary is a *negligible function* of the security parameter. Intuitively, a negligible function is a function that goes exponentially smaller as the security parameter increases. Stated differently, the function is negligible if it is smaller than the inverse of any polynomial in λ. More formally:

Definition 1. *Negligible function A function $f : \mathbb{N} \to \mathbb{R}^+$ is negligible if for all positive integer c, there exists an integer N_c, such that for all $x > \mathrm{N}_c$, $f(x) < x^c$.*

We use the notation $negl(\lambda)$ to denote the set of negligible functions of λ.

For instance, a distance bounding protocol uses cryptographic keys and nonces (random numbers). Their sizes, as well as the number of rounds, are chosen to be polynomial in the security parameter. Similarly, for the primitives we define in this section, the size of the keys is dependant on the security parameter.

In order to prove that the success probability of an adversary is negligible, we sometimes need to use abstractions, such as the Random Oracle Model (ROM). The ROM is an abstraction in which there exists a random oracle $\mathsf{H}(\cdot)$, which maintains a list HL: this oracle is used to produce random values from its input. When querried with a given input i, it does the following:

- If i is querried for the first time, return a random value o and add the couple (i, o) to a list HL
- If i was querried before, *i.e.*, $\exists(i', o) \in \mathsf{HL}, i = i'$, then it returns the corresponding o.

In other words, it behaves like a hash function that returns truely random values. A hash function in the random oracle model is simply a hash function implemented with a random oracle.

After this brief introduction to provable security, we define the notations, cryptographic primitives and mathematical notions that we use. Note that a more complete definition of most of these primitives can be found in [Goldreich, 2006].

2.2 Notations

We define the notations that we use through the thesis.

$\oplus$ is the bitwise xor operation.

$a||b$ denotes the concatenation of two bitstrings a and b.

$\Big|\Big|_{i\in[0;n]} (a_i)$ denotes the concatenation of the values a_i, for i from 0 to n.

$x \xleftarrow{\$} \mathbb{E}$ denotes the uniform random choice of a value from the set $\mathbb{E}$.

$\mathrm{HW}(a)$ denotes the hamming weight of a bitstring a, *i.e.*, the number of ones in a.

$\mathrm{HD}(a, b)$ denotes the hamming distance between two bitstrings a and b, *i.e.*, $\mathrm{HW}(a \oplus b)$.

$\#\mathbb{X}$ denotes the cardinality of the set $\mathbb{X}$.

$Pr[\mathrm{E}]$ denotes the probability for the event E to occur.

2.3 Encryption

Encryption schemes use a cryptographic key to transform a message m into a ciphertext c, such that only the recipient, knowing the appropriate decryption key, can recover m from c. Encryption schemes can be symmetric or public key. If the scheme is symmetric, the encryption and decryption key are the same. If the scheme is public key, the messages are encrypted with a public key, and decrypted with a secret key..

2.3.1 Symmetric Key Encryption Scheme

A symmetric key encryption scheme is an encryption scheme in which the same key is used for encryption and decryption.

Definition 2 (Symmetric Key Encryption)**.** *A* symmetric key encryption *scheme* SKE *is a triplet of algorithms* $(\mathsf{SKE.gen}, \mathsf{SKE.enc}, \mathsf{SKE.dec})$ *such that :*

$\mathsf{SKE.gen}(1^\lambda)$: *returns a secret key x.*

$\mathsf{SKE.enc}_x(m)$: *returns a ciphertext c from the message m and the key x.*

$\mathsf{SKE.dec}_x(c)$: *returns the plaintext m such that* $\mathsf{E.enc}_x(m) = c$.

2.3.2 Public Key Encryption Scheme

In a public key encryption scheme, the key used to encrypt the messages is public: everyone can use it to encrypt a message. On the other hand, the decryption key is secret.

Definition 3 (Public Key Encryption). *A* public key encryption *scheme* PKE *is defined by:*

$\mathsf{PKE.gen}(1^\lambda)$ *returns a public and private key pair (pk, x).*

$\mathsf{PKE.enc}_{pk}(m)$ *returns the ciphertext c from the message m and the public key pk.*

$\mathsf{PKE.dec}_x(c)$ *returns the plaintext m such that* $\mathsf{E.enc}_{pk}(m) = c$.

2.3.3 $\mathsf{IND\text{-}CCA2}$ security

The $\mathsf{IND\text{-}CCA2}$ notion was formalized for public key encryption schemes in [Bellare et al., 1998], and was later extended to symmetric schemes in [Bellare and Namprempre, 2000]. The authors of [Bellare and Namprempre, 2000] consider a notion they call $\mathsf{IND\text{-}CCA}$, but mention in the paper that it corresponds to the notion of $\mathsf{IND\text{-}CCA2}$ used for public key encryption.

Definition 4 ($\mathsf{IND\text{-}CCA2}$ - Symmetric key). *Let* $\mathsf{SKE} = (\mathsf{SKE.gen}, \mathsf{SKE.enc}, \mathsf{SKE.dec})$ *be a symmetric key encryption scheme.* SKE *is said to be* indistinguishable against adaptive chosen ciphertext attack ($\mathsf{IND\text{-}CCA2}$) *when for any adversary $\mathscr{A} = (\mathscr{A}_0, \mathscr{A}_1)$, the advantage*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathscr{A},\mathsf{SKE}}(1^\lambda) = \left|\Pr\left[\begin{array}{l} \mathsf{k} \leftarrow \mathsf{SKE.gen}(1^\lambda), b \xleftarrow{\$} \{0,1\} \\ b' \leftarrow \mathscr{A}_0^{\overline{\mathsf{SKE}}.\mathsf{enc}_\mathsf{k}(\mathsf{LR}_b), \overline{\mathsf{SKE}}.\mathsf{dec}_\mathsf{k}}(\lambda) \end{array} : b = b'\right] - \frac{1}{2}\right|$$

is negligible, where the oracles $\overline{\mathsf{SKE}}.\mathsf{enc}_\mathsf{k}(\mathsf{LR}_b), \overline{\mathsf{SKE}}.\mathsf{dec}_\mathsf{k}$ *are defined as:*

$\overline{\mathsf{SKE}}.\mathsf{enc}_\mathsf{k}(\mathsf{LR}_b(m_0, m_1))$: *returns* $\mathsf{SKE.enc}_\mathsf{k}(m_b)$ *on the message pair (m_0, m_1), for a random but fixed bit b.*

$\overline{\mathsf{SKE}}.\mathsf{dec}_\mathsf{k}(c)$: *returns $\perp$ if c has been generated by* $\overline{\mathsf{SKE}}.\mathsf{enc}_\mathsf{k}(\mathsf{LR}_b)$, *and* $\mathsf{SKE.dec}_\mathsf{k}(c)$ *otherwise.*

Definition 5 ($\mathsf{IND\text{-}CCA2}$ - Public key). *Let* $\mathsf{PKE} = (\mathsf{PKE.gen}, \mathsf{PKE.enc}, \mathsf{PKE.dec})$ *be a public key encryption scheme.* PKE *is said to be* indistinguishable against adaptive chosen ciphertext attack *when for any adversary $\mathscr{A} = (\mathscr{A}_0, \mathscr{A}_1)$, the advantage*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathscr{A},\mathsf{PKE}}(1^\lambda) = \left|\Pr\left[\begin{array}{l} (pk, x) \leftarrow \mathsf{PKE.gen}(1^\lambda), b \xleftarrow{\$} \{0,1\} \\ b' \leftarrow \mathscr{A}^{\overline{\mathsf{PKE}}.\mathsf{enc_pk}(\mathsf{LR_b}), \overline{\mathsf{PKE}}.\mathsf{dec_x}(pk,\lambda)} \end{array} : b = b'\right] - \frac{1}{2}\right|$$

is negligible, where the oracles $\overline{\mathsf{PKE}}.\mathsf{enc_pk}(\mathsf{LR_b}), \overline{\mathsf{PKE}}.\mathsf{dec_x}$ *are defined as:*

$\overline{\mathsf{PKE}}.\mathsf{enc_pk}(\mathsf{LR}_b(m_0, m_1))$: *returns* $\mathsf{PKE.enc}_{pk}(m_b)$ *on the message pair (m_0, m_1), for a random but fixed bit b.*

$\overline{\mathsf{PKE}}.\mathsf{dec_x}(c)$: *returns $\perp$ if c has been generated by* $\overline{\mathsf{PKE}}.\mathsf{enc_pk}(\mathsf{LR_b})$ *returns $\perp$, and* $\mathsf{PKE.dec}_x(c)$ *otherwise.*

2.4 Message Authentication

We consider three kinds of message authentication schemes: MAC, digital signature and dynamic group signature. In a MAC, a tag authenticating a message is generated and verified with the same secret key. In digital signature schemes, the signing key is private and known only to the signer, whereas the verification key is public. In group signature schemes, each member of the group knows a private signature key, and a public verification key permits to verifiy that the signature was generated by a member of the group. We sometimes abuse language and consider MAC schemes as (symmetric) signature schemes.

The security properties we use are the unforgeability, and the anonymity for group signatures.

2.4.1 Message Authentication Codes

A MAC is an authentication scheme in which the keys used to authenticate and verify identical and secret.

Definition 6 (Message Authentication Code)**.** *A* message authentication code MAC *is a triplet of algorithms* $(\mathsf{MAC.gen}, \mathsf{MAC.tag}, \mathsf{MAC.ver})$ *such that :*

$\mathsf{MAC.gen}(1^\lambda)$**:** *returns a secret key x.*

$\mathsf{MAC.tag}_x(m)$**:** *returns a tag σ from the message m and the key x.*

$\mathsf{MAC.ver}_x(s, m)$**:** *returns a verification bit v from the tag σ and the key x.*

2.4.2 Digital Signature

Digital signatures use a secret signature key, and a public verification key.

Definition 7 (Digital Signature)**.** *A* digital signature *scheme* SIG *is a composed of the three following algorithms:* $(\mathsf{SIG.gen}, \mathsf{SIG.sig}, \mathsf{SIG.ver})$*, such that :*

$\mathsf{SIG.gen}(1^\lambda)$**:** *returns a signature/verification key pair (x, vk), where x is secret and vk is public.*

$\mathsf{SIG.sig}_x(m)$**:** *returns a signature σ from the message m and the signing key x.*

$\mathsf{SIG.ver}_{\mathsf{vk}}(s, m)$**:** *returns a verification bit v from the signature s and the verification key vk.*

2.4.3 Group Signature

In a group signature scheme, the signers are treated as a group. Each member of the group has a personal signing key that he uses to sign a message anonymously on behalf of the group. An entity, called *group manager*, has the power to open, *i.e.*, deanonymise, a signature. A more complete definition is given in [Bellare et al., 2003].

Definition 8 (Group Signature)**.** *A group signature scheme* $\mathsf{G\text{-}SIG}$*, using a user list* U_i*, is defined by:*

$\mathsf{G.gen}(1^\lambda)$ *returns a group/master key pair* $(\mathsf{gpk}, \mathsf{msk})$ *and sets the user list* UL*. The verification key is* gpk.

$\mathsf{G.sig}_{x_i}(m)$ *returns a group signature σ on the message m, such that* $\mathsf{G.ope}_{\mathsf{msk}}(\sigma, m, \mathsf{UL}, \mathsf{gpk}) = \mathrm{U}_i$

$\mathsf{G.ver}_{\mathsf{gpk}}(\sigma, m)$ *outputs* 1 *if and only if σ is valid for the message m and the key x_i of a non-revoked user, and* 0 *otherwise.*

$\mathsf{G.ope}_{\mathsf{msk}}(\sigma, m, \mathsf{UL}, \mathsf{gpk})$ *outputs the identity of the user U_i who produced σ.*

A group signature can be *dynamic*. In this case, new members can be added to the group, and members can be revoked. More specifically, an entity, called a *group manager*, adds new members. Additionally, a revocation algorithm is needed. A revoked user list RL is added, as well as the following algorithms:

$\mathsf{G.join}_{\mathsf{msk}}(i, \mathsf{gpk}, \mathsf{UL})$ is a protocol between a user U_i (using gpk) and a group manager GM (using gpk and msk). U_i interacts with GM to get his signing key x_i, while GM outputs a value reg_i and adds U_i to UL.

$\mathsf{G.rev}_{\mathsf{msk}}(i, \mathsf{RL}, \mathsf{UL}, \mathsf{gpk})$ computes revocation logs rev_i for user U_i, using reg_i, gpk and msk and moves U_i from UL to RL.

The verification algorithm is modified to include the revoked user list:

$\mathsf{G.ver}_{\mathsf{gpk}}(\sigma, m, \mathsf{RL})$ outputs 1 if and only if σ is valid for the message m and the key x_i of a non-revoked user, and 0 otherwise.

In this thesis, we only use dynamic group signature schemes. When we mention a group signature scheme, we implicitely assume that it is dynamic.

2.4.4 Unforgeability

The unforgeability property of a message authentication scheme states that an adversary cannot generate a valid signature or MAC without knowing the corresponding key. For digital signatures and MAC schemes, this notion is known as Existential UnForgeability under Chosen Message Attack (EUF-CMA). For group signatures, a stronger notion (full-traceability) exists: it states that no user or coallition therof can produce a signature that opens to the identity of a honest user. This distinction needs to be introduced, since in the context of a group signature, a user might try to forge a signature on bealf of another one, for instance to build an alibi. This dishonest user knows more information than a classical adversary, since he knows a singing key. The EUF-CMA notion was first defined for digital signatures in [Goldwasser et al., 1983], and then for MAC schemes in [Bellare et al., 1994]. Full traceability for group signatures was defined in [Bellare et al., 2005].

The unforgeability experiment is defined as follows:

Definition 9 (EUF-CMA (MAC)). *Let $\mathcal{A}$ be an adversary, and* MAC *be a MAC scheme. In the* $\mathsf{Exp}^{\mathit{Forge}}_{\mathcal{A},\mathsf{MAC}}(\lambda)$ *experiment, the challenger creates* (x) *using* $\mathsf{MAC.gen}(1^\lambda)$*, initializes an empty list* Σ*, and gives* $\mathcal{A}$ *access to a signing oracle:*

$\overline{\mathsf{MAC}}.\mathsf{Tag}(\cdot,\cdot)$ *returns a MAC* σ*, using* $\mathsf{MAC.tag} x m$ *and adds the pair* (m, σ) *to* Σ*.*

Then, $\mathcal{A}$ *outputs a message* m^* *and a signature* σ^**.* $\mathcal{A}$ *wins (and the challenger outputs 1) if the two following conditions are satified:* $(m^*, \sigma^*) \not\in \Sigma$ and $\mathsf{MAC.ver}_x(s, m) = 1$

The advantage of an adversary $\mathcal{A}$ *in the unforgeability experiment,* $\mathsf{Adv}^{\mathit{Forge}}_{\mathsf{MAC}}(\lambda)$*, is defined as*

$$\mathsf{Adv}^{\mathit{Forge}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = \Pr[\mathsf{Exp}^{\mathit{Forge}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = 1]$$

and the advantage on the experiment as

$$\mathsf{Adv}^{\mathit{Forge}}_{\mathsf{MAC}}(\lambda) = \max_{\mathcal{A} \in \mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{\mathit{Forge}}_{\mathcal{A},\mathsf{MAC}}(\lambda)\}.$$

A MAC scheme MAC *is unforgeable if* $\mathsf{Adv}^{\mathit{Forge}}_{\mathsf{MAC}}(\lambda)$ *is negligible.*

For digital signature schemes, the security game is essentially the same, except that $\mathcal{A}$ has access to the verification key verk, since it is public.

Definition 10 (EUF-CMA (digital signature)). *Let $\mathscr{A}$ be an adversary, and* SIG *be a digital signature scheme. In the* $\mathsf{Exp}^{Forge}_{\mathscr{A},\mathsf{SIG}}(\lambda)$ *experiment, the challenger creates* (x, pk) *using* $\mathsf{SIG.gen}(1^\lambda)$*, sends* pk *to* $\mathscr{A}$*, initializes an empty list* Σ *which contains all oracle produced signatures, and gives* $\mathscr{A}$ *access to a signing oracle:*

$\overline{\mathsf{SIG}}.\mathsf{Sign}(\cdot,\cdot)$ *returns a signature* σ*, using* $\mathsf{SIG.sig}_x(m)$ *and adds the pair* (m,σ) *to* Σ*.*

Then, $\mathscr{A}$ *outputs a message* m^* *and a signature* σ^**.* $\mathscr{A}$ *wins (and the challenger outputs 1) if the two following conditions are satified:* $(m^*,\sigma^*) \notin \Sigma$ and $\mathsf{SIG.ver}_x(s,m) = 1$

The advantage of an adversary $\mathscr{A}$ *in the unforgeability experiment,* $\mathsf{Adv}^{Forge}_{\mathsf{SIG}}(\lambda)$*, is defined as*

$$\mathsf{Adv}^{Forge}_{\mathscr{A},\mathsf{SIG}}(\lambda) = \Pr[\mathsf{Exp}^{Forge}_{\mathscr{A},\mathsf{SIG}}(\lambda) = 1]$$

and the advantage on the experiment as

$$\mathsf{Adv}^{Forge}_{\mathsf{SIG}}(\lambda) = \max_{\mathscr{A}\in\mathsf{Poly}(\lambda)}\{\mathsf{Adv}^{Forge}_{\mathscr{A},\mathsf{SIG}}(\lambda)\}.$$

A digital signature scheme SIG *is unforgeable if* $\mathsf{Adv}^{Forge}_{\mathsf{SIG}}(\lambda)$ *is negligible.*

For dynamic group signature schemes, the experiment is a bit more complex, since it includes attacks from group members. It is defined as full traceability.

Definition 11 (Full Traceability (Dynamic Group Signatures)). *Let* $\mathsf{G\text{-}SIG} = (\mathsf{G.gen}, \mathsf{G.sig}, \mathsf{G.ver})$ *be a group signature scheme. In the full traceability experiment for an adversary* $\mathscr{A}$ $\mathsf{Exp}^{Trace}_{\mathscr{A},\mathsf{G\text{-}SIG}}(\lambda)$*, the challenger creates* $(\mathsf{UL},\mathsf{RL},\mathsf{msk},\mathsf{gpk})$ *using* $\mathsf{G.gen}(1^\lambda)$*, gives* gpk *to* $\mathscr{A}$*, and sets the lists* CU *and* Σ*, for corrupted users and oracle-issued signatures. During a first phase, phase* $\mathscr{A}$ *has access to the following oracles:*

$\overline{\mathsf{G}}.\mathsf{join}_{\mathsf{msk}}(i,\mathsf{gpk},\mathsf{UL})$ *creates* U_i *using* $\mathsf{G.join}_{\mathsf{msk}}(\mathsf{ID},\mathsf{gpk},\mathsf{UL})$.

$\overline{\mathsf{G}}.\mathsf{join^C}_{\mathsf{msk}}(i,\mathsf{gpk},\mathsf{UL},\mathsf{RL})$ *creates a corrupted user* U_i *by using* $\mathsf{G.join}_{\mathsf{msk}}(\mathsf{ID},\mathsf{gpk},\mathsf{UL})$*, adding* U_i *to* CU *and returning its secret key* x_i *to* $\mathscr{A}$*.*

$\overline{\mathsf{G}}.\mathsf{join}(i)$ *simulates the corruption of a user* U_i *by returning its secret key* x_i *to* $\mathscr{A}$ *and adding it to* CU.

$\overline{\mathsf{G}}.\mathsf{rev}_{\mathsf{msk}}(i,\mathsf{RL},\mathsf{UL},\mathsf{gpk})$ *revokes* U_i *using* $\mathsf{G.rev}_{\mathsf{msk}}(\mathsf{ID},\mathsf{RL},\mathsf{UL},\mathsf{gpk})$.

$\overline{\mathsf{G}}.\mathsf{sig}_{x_i}(m)$ *returns a signature* σ *on behalf of* U_i*, using* $\mathsf{G.sig}_{x_i}(m)$ *and adds the pair* (m,σ) *to* Σ*.*

$\overline{\mathsf{G}}.\mathsf{ope}_{\mathsf{msk}}(\sigma,m,\mathsf{UL},\mathsf{gpk})$ *opens a signature* σ *and returns* U_i *to* $\mathscr{A}$*, using* $\mathsf{G.ope}_{\mathsf{msk}}(\sigma,m,\mathsf{UL},\mathsf{gpk})$.

Then, during a guessing phase, $\mathscr{A}$ *outputs a message* m^* *and a signature* σ^**.* $\mathscr{A}$ *wins (and the challenger outputs* 1*) if and only if:*

$(m^*,\sigma^*) \notin \Sigma$ and $\mathsf{G.ver}_{\mathsf{gpk}}(\sigma^*,m^*,\mathsf{RL}) = 1$,

$\mathsf{G.ope}_{\mathsf{msk}}(\sigma^*,m^*,\mathsf{UL},\mathsf{gpk}) \notin \mathsf{CU} \setminus \mathsf{RL}$.

The advantage of an adversary $\mathscr{A}$ *in the full traceability experiment,* $\mathsf{Adv}^{Trace}_{\mathsf{G\text{-}SIG}}(\lambda)$*, is defined as*

$$\mathsf{Adv}^{Trace}_{\mathscr{A},\mathsf{G}}(\lambda) = \Pr[\mathsf{Exp}^{Trace}_{\mathscr{A},\mathsf{G}}(\lambda) = 1]$$

and the advantage on the experiment as

$$\mathsf{Adv}^{Trace}_{\mathsf{G}}(\lambda) = \max_{\mathscr{A}\in\mathsf{Poly}(\lambda)}\{\mathsf{Adv}^{Trace}_{\mathscr{A},\mathsf{G}}(\lambda)\}.$$

A group signature $\mathsf{G\text{-}SIG}$ *is traceable if* $\mathsf{Adv}^{Trace}_{\mathsf{G\text{-}SIG}}(\lambda)$ *is negligible.*

Anonymity

Anonymity is a property of group signature schemes: if a group signature scheme is anonymous, then no one, except for the opening authority, can find out which user produced a signature within a group. The anonymity experiment for dynamic group signature schemes is defined as follows.

Definition 12. *Let* $\mathsf{G\text{-}SIG} = (\mathsf{G.gen}, \mathsf{G.sig}, \mathsf{G.ver})$ *be a dynamic group signature scheme. In the anonymity experiment for an adversary* $\mathscr{A}$ $\mathsf{Exp}^{Anon}_{\mathscr{A},\mathsf{G\text{-}SIG}}(\lambda)$*, the challenger first uses* $\mathsf{G.gen}(1^\lambda)$ *to create the lists and keys* $(\mathsf{UL}, \mathsf{RL}, \mathsf{msk}, \mathsf{gpk})$*, gives* gpk *to* $\mathscr{A}$*, and sets the lists* CU *and* Σ*. During a first phase,* $\mathscr{A}$ *has access the following oracles:*

$\overline{\mathsf{G}}.\mathsf{join}_{\mathsf{msk}}(i, \mathsf{gpk}, \mathsf{UL})$ *creates* U_i *using* $\mathsf{G.join}_{\mathsf{msk}}(\mathsf{ID}, \mathsf{gpk}, \mathsf{UL})$*.*

$\overline{\mathsf{G}}.\mathsf{join}^{\mathsf{C}}{}_{\mathsf{msk}}(i, \mathsf{gpk}, \mathsf{UL}, \mathsf{RL})$ *creates a corrupted user* U_i *by using* $\mathsf{G.join}_{\mathsf{msk}}(\mathsf{ID}, \mathsf{gpk}, \mathsf{UL})$*, adding* U_i *to* CU *and returning its secret key* x_i *to* $\mathscr{A}$*.*

$\overline{\mathsf{G}}.\mathsf{join}(i)$ *Simulates the corruption of a user* U_i *by returning its secret key* x_i *to* $\mathscr{A}$ *and adding it to* CU*.*

$\overline{\mathsf{G}}.\mathsf{rev}_{\mathsf{msk}}(i, \mathsf{RL}, \mathsf{UL}, \mathsf{gpk})$ *revokes* U_i *using* $\mathsf{G.rev}_{\mathsf{msk}}(\mathsf{ID}, \mathsf{RL}, \mathsf{UL}, \mathsf{gpk})$*.*

$\overline{\mathsf{G}}.\mathsf{sig}_{x_i}(m)$ *returns a signature* σ *on behalf of* U_i*, using* $\mathsf{G.sig}_{x_i}(m)$ *and adds the pair* (m, σ) *to* Σ*.*

$\overline{\mathsf{G}}.\mathsf{ope}_{\mathsf{msk}}(\sigma, m, \mathsf{UL}, \mathsf{gpk})$ *opens a signature* σ *and returns* U_i *to* $\mathscr{A}$*, using* $\mathsf{G.ope}_{\mathsf{msk}}(\sigma, m, \mathsf{UL}, \mathsf{gpk})$*.*

$\mathscr{A}$ *selects two identities* $(\mathsf{ID}_0, \mathsf{ID}_1)$*. If* ID_0 *and* $\mathsf{ID}_1 \in \mathsf{CU}$*, the challenger stops. Otherwise, he picks* $b \xleftarrow{\$} \{0,1\}$*.*

Then, during the guessing phase, $\mathscr{A}$ *cannot use* $\mathsf{G.cor}$ *and* $\mathsf{G.rev}$ *on* ID_0 *or* ID_1 *anymore, but gains access to a new oracle:*

$\overline{\mathsf{G}}.\mathsf{sig}_{\mathsf{b}}$ *simply returns* $\mathsf{G.sig}_{\mathsf{ssk}_{\mathsf{ID}_b}}(m)$*. The signature produced by this oracle cannot be the* $\mathsf{G.ope}$ *oracle.*

$\mathscr{A}$ *outputs* b' *and the challenger returns the boolean value* $(b = b')$*.*

$\mathscr{A}$*'s advantage in this experiment is defined as*

$$\mathsf{Adv}^{Anon}_{\mathscr{A},\mathsf{G}}(\lambda) = \left|\Pr[\mathsf{Exp}^{Anon}_{\mathscr{A},\mathsf{G}}(\lambda) = 1] - \frac{1}{2}\right|$$

and the advantage on the experiment as

$$\mathsf{Adv}^{Anon}_{\mathsf{G}}(\lambda) = \max_{\mathscr{A} \in \mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{Anon}_{\mathscr{A},\mathsf{G}}(\lambda)\}.$$

A group signature $\mathsf{G\text{-}SIG}$ *is anonymous if* $\mathsf{Adv}^{Anon}_{\mathsf{G\text{-}SIG}}(\lambda)$ *is negligible.*

2.5 Others

In this section, we present pseudorandom functions, hash functions, zero-knowledge proofs of knowledge, commitment schemes, and the union bound.

2.5.1 Pseudo Random Functions

A pseudo random function (PRF) is a keyed function the output of which cannot be efficiently distinguished from that of a random function on the same domain. This primitive was defined, and an example (GGM) was given, in [Goldreich et al., 1986].

Definition 13 (Pseudo Random Function)**.** *A family of functions* $PRF_s : \{0,1\}^k \to \{0,1\}^l$*, indexed by a key* $s \in \{0,1\}^n$*, is pseudorandom if it satisfies the following two conditions:*

- *There is an efficient algorithm (polynomial in the security parameter) for computing $PRF_s(x)$, given s and x;*
- *For any $\mathscr{A}$ given oracle access to the function, the probability of distinguishing $PRF_s(\cdot)$ from a random function*[1] *$f: \{0,1\}^k \rightarrow \{0,1\}^l$ is negligible.*

2.5.2 Cryptographic Hash Function

Definition 14 (Hash Function)**.** *A hash function family $\mathbb{H}$ is a function $\mathbb{H}: \mathbb{K} \times \mathbb{M} \rightarrow \mathbb{Y}$, where $\mathbb{K}$ is a key space, $\mathbb{M}$ is a message space and $\mathbb{Y}$ is a digest space.*

A cryptographic hash function $\mathbb{H}_K$ is required to be collision resistant: for an adversary knowing the key K, it should be difficult to find two messages that have the same image. The latest standard hash function is SHA-3 [Bertoni et al., 2013].

Definition 15. *Collision Resistance A hash function family $\mathbb{H}$ is collision resistant if the probability $Pr[K \xleftarrow{\$} \mathbb{K}; (M, M') \leftarrow \mathscr{A}(K) : M \neq M' \wedge \mathbb{H}_K(M) = \mathbb{H}_K(M')$ is negligible.*

2.5.3 Zero-knowledge Proof of Knowledge

Zero-knowledge proofs of knowledge were introduced in [Goldwasser et al., 1989]. A *zero-knowledge proof of knowledge* scheme ZKP is a protocol that allows a prover to convince a verifier that he knows a secrete value x, such that a relation r, involving x, is true. This is denoted $\mathsf{ZKP}[x : r]$. The properties of a ZKP are the following.

Correctness: If the prover knows the solution x, then he is able to convince the verifier.

Soundness: If r is wrong, then the prover is not able to convince the verifier.

Validity: If the prover does not know the secret x, then he is not able to convince the verifier.

Zero-knowledge: No information about the secret x leaks during the protocol.

2.5.4 Commitment

A commitment scheme permits a prover to *commit* to a value v, without revealing v. Later, he can *open* this commitment, *i.e.*, reveal v, and prove that v is the value that was used to generate the commitment. A simple commitment is a hash v: it does not allow to recover v, but can be easily verified once v is revealed. Commitment schemes were first formalised in [Brassard et al., 1988].

Definition 16 (Commitment Scheme)**.** *A* Commitment *scheme* COM *is composed of a pair of algorithms* $(\mathsf{COM.commit}, \mathsf{COM.open})$ *such that:*

$\mathsf{COM.commit}(v, r)$**:** *returns a commitment c to the value v, using the random value r.*

$\mathsf{COM.open}(c, r)$**:** *returns the value v such that* $\mathsf{COM.commit}(v, r) = c$.

2.5.5 Union Bound

The union bound is a useful upper bound for the probability of an event occuring at least once [Kallenberg, 2002]. It states that, for any finite or countable set of events, the probability that at least one of the events happens is no greater than the sum of the probabilities of the individual events. More formally, for a set of events $\{E_i | i \in [1; k]\}$, we have

$$Pr[\bigcup_{i=0}^{k} E_i] \leq \sum_{i=0}^{k} Pr[E_i].$$

[1] By random function, we mean a function chosen at random among all functions on the same interval.

Part I

Distance Bounding Protocols

Chapter 3

Introduction to Distance Bounding

Contents

In a relay attack, an intruder defeats an authentication protocol by relaying the messages between the verifier and a far away prover. Distance bounding protocols are a classical countermeasures to relay attacks. In this chapter, we present the threats associated to distance bounding, and some protocols of the literature. We then present the formalism associated with distance bounding protocols, as well as the DFKO [Dürholz et al., 2011] security model, which we use to prove the security of our protocols, which are presented in Chapter 4.

3.1 Introduction

Distance bounding protocols were introduced to counter relay attacks, in which an adversary relays the messages between a prover and a verifier. Since the introduction of the first protocol [Brands and Chaum, 1994], many designs for distance bounding appeared in the literature. In a survey [Brelurut et al., 2016], we showed that among the 42 studied protocols, only 10 had no known vulnerabilities against any distance bounding attack yet. Formal models for distance bounding were introduced [Avoine et al., 2009, Dürholz et al., 2011, Boureanu et al., 2015] to increase the confidence in the protocols. The models define precisely which frauds concern distance bounding, and provide a framework for performing security proofs. In this chapter, we introduce the security notions used in distance bounding. We first describe the general structure of distance bounding protocols, before giving a high level overview of their security properties and exemplifying with classical protocols from the literature. Finally, we present the DFKO [Dürholz et al., 2011] formal model, and the proof methodology that we use to prove the security of our protocols in the next chapter.

3.2 General Structure of Distance Bounding Protocols

Distance bounding protocols are subject to very tight timing constraints: in one nanosecond, a radio wave travels 30 centimetres. As contacless applications, such as payments, are generally designed to work at some centimetres, distance bounding protocols typically need to be able to reliably perform challenge response rounds within nanoseconds. This places strong constraints on the hardware that is used, and is believed to be feasible only with single bit messages. For this reason, most distance bounding protocols are decomposed into slow (also called lazy) phases, during which messages of arbitrary length can be exchanged, and time-critical ones, during which the messages are single bits and the time is measured. The classical structure of a Distance Bounding protocol includes a slow initialisation phase, followed by a time-critical distance bounding phase, and a final slow authentication phase, even though some protocols do not have this final slow phase. The time measurement is performed during the distance bounding phase, in which the verifier sends a challenge, to which the prover immediately replies with a response.

In addition, to make the measurement reliable, the computation of the response by the prover should take a predictable, constant, and short time. If it is not predictable, then the estimated distance is meaningless, since the verifier cannot assess which part of the time was spent computing, and which was actually the time of flight of the response. If the computation time is not constant, the same kind of problems appears. Finally, if it is not almost instantaneous, frauds can be performed, for instance by overclocking the devices. If the computation time is very close to the propagation time to begin with, this makes little difference, but if it is much longer than the propagation time, then gaining even a few nanoseconds permits to cheat on the distance. To satisfy these constraints, the usual approach is to have the prover compute both response vectors in advance, so that it only needs to perform a table lookup to respond the challenges. In addition, distance bounding protocols, except for some notable exceptions, such as [Meadows et al., 2007], traditionally use one bit challenges and one bit responses.

In order to keep the challenges and responses one bit long, it is not feasible to add error correcting codes to them, which makes the distance bounding protocols vulnerable to channel noise. Hence, protocols sometimes include an error tolerance: they allow n_{err} errors in the responses to avoid rejecting legitimate provers due to transmission errors.

The general structure of a distance bounding protocol is shown on Figure 3.1. The prover and the verifier agree on two response vectors a^0 and a^1 during an initialisation phase, and then run a distance bounding phase. The distance bounding phase is composed of n rounds. At each of these rounds, the verifier picks a random bit c_i, starts a clock and sends c_i to the prover. The prover immediately responds with $r_i = a^{c_i}$, and the verifier stops his clock after receiving r_i. After the n rounds, the prover computes a tag τ to authenticate the transcript, *i.e.*, the concatenation of the messages exchanged during the protocol, with the shared key x. The verifier accepts if all the time measures were correct, the responses correspond to the response vectors chosen during the initialisation phase, and the tag is correct.

To agree on the response vectors a^0 and a^1, many modern distance bounding protocols use a PseudoRandom Function (PRF). Typically, both the prover and the verifier generate a random nonce, and these two nonces are used in a PRF keyed with their shared secret to generate the re-

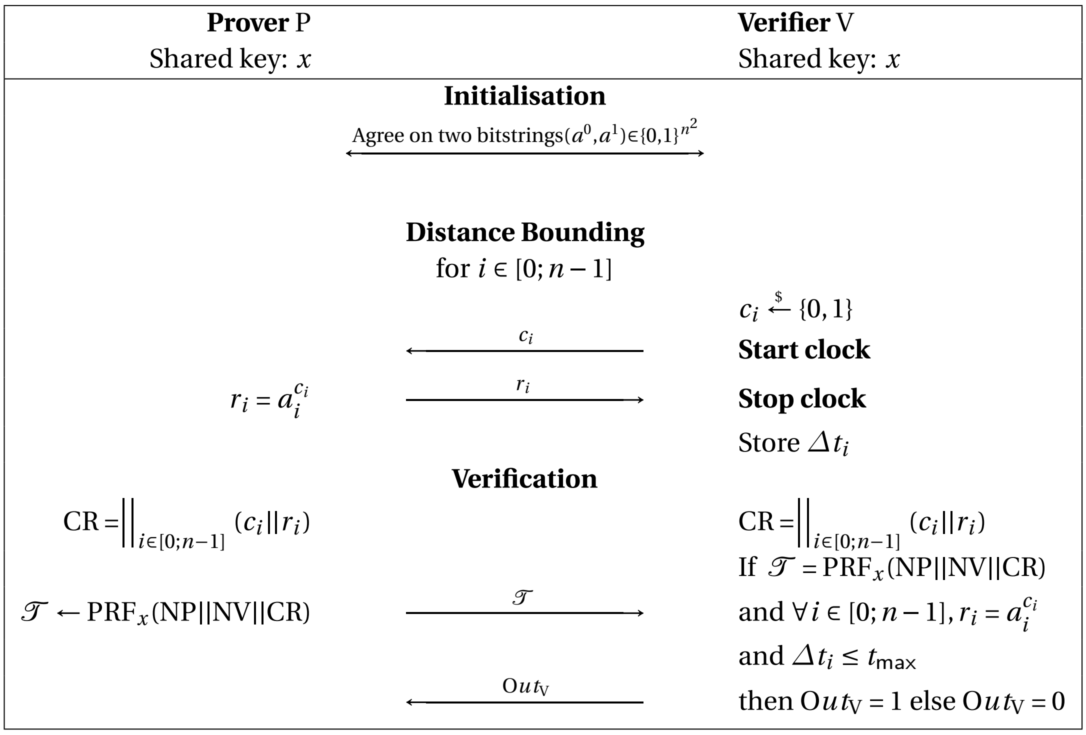
Prover P
Shared key: x
Verifier V
Shared key: x
Initialisation
Agree on two bitstrings$(a^0,a^1) \in \{0,1\}^{n^2}$
Distance Bounding
for $i \in [0; n-1]$
$c_i \xleftarrow{\$} \{0,1\}$
c_i
Start clock
$r_i = a_i^{c_i}$
r_i
Stop clock
Store Δt_i
Verification
$\mathrm{CR} = \big\|_{i\in[0;n-1]} (c_i \| r_i)$
$\mathrm{CR} = \big\|_{i\in[0;n-1]} (c_i \| r_i)$
If $\mathcal{T} = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV}\|\mathrm{CR})$
$\mathcal{T} \leftarrow \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV}\|\mathrm{CR})$
$\mathcal{T}$
and $\forall i \in [0; n-1], r_i = a_i^{c_i}$
and $\Delta t_i \leq t_{\max}$
Out_V
then $Out_V = 1$ else $Out_V = 0$

Figure 3.1: The general structure of a Distance Bounding protocol, where PRF is a pseudorandom function.

sponse vectors. The interest of this method is its lightweight aspect, compared to the commitment schemes and digital signatures used in prior protocols. The first use of a PRF for unilateral distance bounding protocols (in which the prover does not need to authenticate the verifier) is the protocol of Hancke and Kuhn [Hancke and Kuhn, 2005], depicted on Figure 3.2. In this protocol, during the initialisation phase, the prover generates a nonce NP, and the verifier a nonce NV. They then exchange the nonces, and both compute $(a^0 \| a^1) = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV})$, where x is a shared secret key. No message is exchanged during the verification phase.

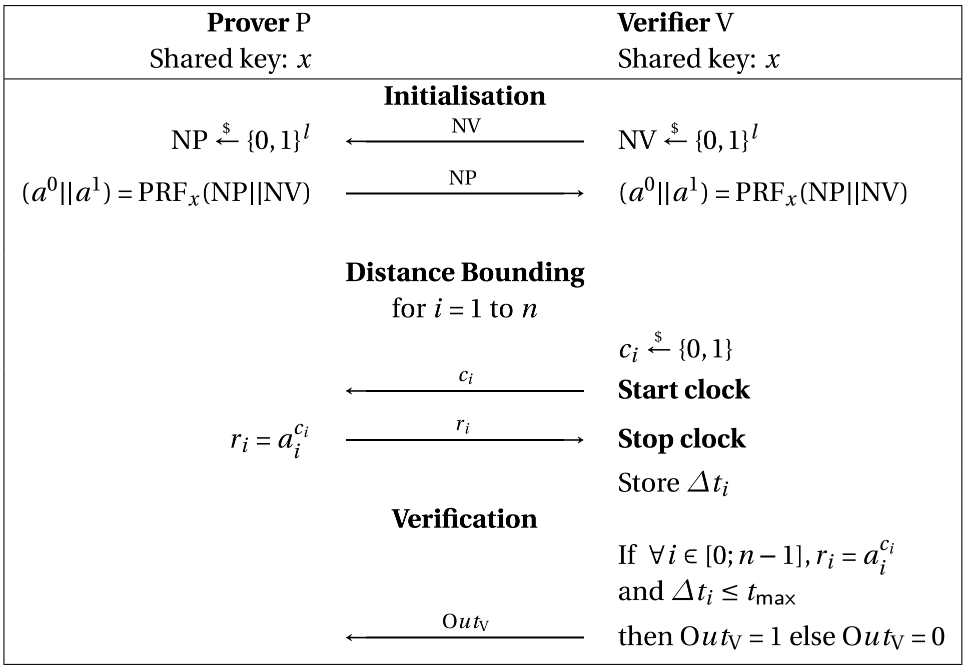
Prover P
Shared key: x
Verifier V
Shared key: x
Initialisation
$\mathrm{NP} \xleftarrow{\$} \{0,1\}^l$
NV
$\mathrm{NV} \xleftarrow{\$} \{0,1\}^l$
$(a^0 \| a^1) = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV})$
NP
$(a^0 \| a^1) = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV})$
Distance Bounding
for $i = 1$ to n
$c_i \xleftarrow{\$} \{0,1\}$
c_i
Start clock
$r_i = a_i^{c_i}$
r_i
Stop clock
Store Δt_i
Verification
If $\forall i \in [0; n-1], r_i = a_i^{c_i}$
and $\Delta t_i \leq t_{\max}$
Out_V
then $Out_V = 1$ else $Out_V = 0$

Figure 3.2: The Distance Bounding protocol proposed by Hancke and Kuhn in [Hancke and Kuhn, 2005]

While this structure is used in most protocols, there are some exceptions that exploit different strategies for secure distance bounding. For instance, instead of just having two response vectors, some protocols (such as [Avoine and Tchamkerten, 2009]) use a response tree, where the prover picks one branch or the other depending on the challenge. Similarly, graph based response functions were proposed, for instance in [Trujillo-Rasua et al., 2014]. Some protocols, such as [Kleber et al., 2015], rely on a Physically Uncloneable Function (PUF)[1] instead of other cryptographic primitives, such as PRFs or signatures. Finally, while the challenges are usually sent at fixed interval, it is possible to randomise the time at which they are sent to improve the security, as in [Kilinąnd Vaudenay, 2015].

These different designs were proposed in an attempt to build more secure protocols. Indeed, while classical relay attacks are easy to thwart with simple protocols, more advanced attacks exist, which we present in the next section.

3.3 Threats Against Distance Bounding Protocols

Distance bounding protocols aim at solving the problem of relay attacks. On the other hand, they are additionally subject to attacks that do not affect authentication protocols. Indeed, any attack that makes the prover appear closer than it actually is defeats the purpose of a distance bounding protocol, which is to compute a correct upper bound on this distance. In this section, we survey the existing threats against distance bounding protocols, as well as the classical defence mechanisms against them.

The first of these attacks is Mafia Fraud:

Mafia Fraud (MF) [Desmedt, 1988]: In a Mafia Fraud, an adversary $\mathscr{A}$ passes the protocol in the presence of a honest prover P, who is far away from the verifier V. The adversary is typically composed of two entities: one (the leech) that is close to the prover and acts as a verifier, and another one (the ghost) that is close to the verifier and that acts like a prover. The fraud succeeds if the verifier accepts the authentication of the prover. While the adversary can physically not relay both the challenges and the responses during the timed bit exchanges without being detected, he can safely relay the messages exchanged during the untimed part of the protocol. Additionally, it is generally considered that there is a small delay before the verifier begins the distance bounding phase, during which the adversary can send random challenges to the prover. Similarly, there is a delay after the end of the distance bounding phase. Hence, there are two main strategies to perform a mafia fraud: either send a random challenge in advance (pre-ask) and relay the (possibly modified) response, or relay the (possibly modified) challenge, and send a random response (post-ask). If the prover is within the distance bound, it might be possible for the adversary to modify the challenges and responses on the fly without being detected. For this reason, a generalisation of mafia frauds, labelled Man-in-the-Middle attack, was proposed in [Boureanu et al., 2015]. In this attack, the adversary can play a learning phase beforehand, during which the prover is located close to the verifier, in order to try to learn something, such as a secret key. Hence, in this attack, the adversary has all the capabilities of a classical man-in-the-middle adversary. A common defence against mafia fraud and man-in-the-middle attacks is for the prover to authenticate the transcript after the distance bounding phase, so that modifications in the challenges are detected.

Attacks on the distance. The other kinds of attacks are performed by dishonest provers, who cheat on their distance with the verifier. Making the verifier believe a prover is close when it is actually far permits, for instance, to build a fake alibi. For example, in some companies, the employees have to use their contactless badge as they arrive and leave, to prove that they spent the day at work. By performing a distance fraud, a lazy employee could make believe that he spent the day at work, when he actually used his badge from home. This type of attacks comes in three

[1] A PUF is a function maps challenge bitstrings to response bitstrings, and cannot be cloned [Maes and Verbauwhede, 2010].

flavours: distance frauds, where the cheating prover receives no help, distance hijacking, where he exploits a honest prover located close to the verifier, and terrorist fraud, where he is helped by an accomplice who is close to the verifier.

3.3.1 Distance Fraud (DF)

In a Distance Fraud [Brands and Chaum, 1994], a lonely malicious prover located far away (at a distance more than t_{max} from the verifier) passes the protocol, thus defeating the distance measurement. There are several possible motivations for committing such a fraud, other than simply breaking the protocol. For instance, if distance bounding on electronic bracelets is used for home arrest, an inmate who would perform a distance fraud would be able to pretend that he is within the restricted area, while actually being far away. Note that this attack can possibly require to amplify the signal, so that the prover and the verifier can communicate even though they are far away. If a far away prover sends his response after receiving the challenge, its propagation time will be too long to meet the bound t_{max}. Hence, a common attack strategy is to try making both response vectors equal, so that the response is independent of the challenge. When this is the case, a malicious prover can send his response bit in advance (before receiving the challenge), so that it arrives to the verifier on time.

To prevent distance frauds, one would ideally want to have $r^0 \neq r^1$ for as many rounds as possible. If k is the number of rounds for which $r^0 \neq r^1$, then the prover needs to guess k challenges, so he succeeds with probability $\frac{1}{2^k}$.

In some cases, the malicious prover can gain some advantage in cheating on his distance if honest provers are present near the verifier: this attack is known as Distance Hijacking.

3.3.2 Distance Hijacking (DH)

Distance Hijacking [Cremers et al., 2012] is quite similar to DF: a malicious, far away prover passes the protocol. The difference is that in a DH, there are honest provers running the protocol near the verifier. This gives the malicious prover more surface of attack, so that some protocols are resistant to distance fraud, while being vulnerable to distance hijacking. A Distance Hijacking attack is generally performed by letting a close-by honest prover run the protocol, and then sending the final authenticated message on his behalf before he does. This is feasible when the rest of the protocol is independent of the secret key: if the verifier has no other means of verifying the identity of the prover than the final message, then it is possible to cheat. Preventing Distance Hijacking can be done by involving the secret key of the prover early in the protocol, for instance by computing the response vectors as a the output of a PRF keyed with the secret key of the prover. This way, if the final authenticated message is hijacked, the verifier will notice that the key used to authenticate it was not the one used to compute the responses.

3.3.3 Terrorist Fraud (TF)

A malicious prover trying to cheat on his distance might also receive the help from an accomplice located near the verifier: this kind of attacks is known as Terrorist Fraud [Desmedt, 1988]. Terrorist Fraud is an attack in which a malicious prover, located far away from the verifier, is helped by an accomplice to pass the protocol. The accomplice can be located near the verifier. Of course, a trivial attack in this scenario would be that the prover simply gives all his secret keys to his accomplice, so that the accomplice can pass the protocol on his own. Since there is not much to be done against a prover willing to give away his secret material to an accomplice, a terrorist fraud is only considered successful if the accomplice does not gain any significant advantage in authenticating by himself after the session in which he was helped by the malicious prover. To perform a terrorist fraud, a prover can ask his accomplice to relay the messages of the slow phases, and give him both response vectors, so that the accomplice can reply on time during the distance bounding phase. If it is possible for a prover to extract the secret key of his prover device, then it is unfeasible to prevent him from giving it away. Here, secret key is used in the broad sense: it can refer to any material that would help the accomplice to impersonate the prover. Hence, we can only defend against provers that are not willing to give their secret key to an accomplice. Consequently, a strategy to prevent a terrorist fraud is to make sure that giving both r^0 and r^1 to the accomplice leaks the secret key. An example of this strategy is illustrated by Figure 3.3.

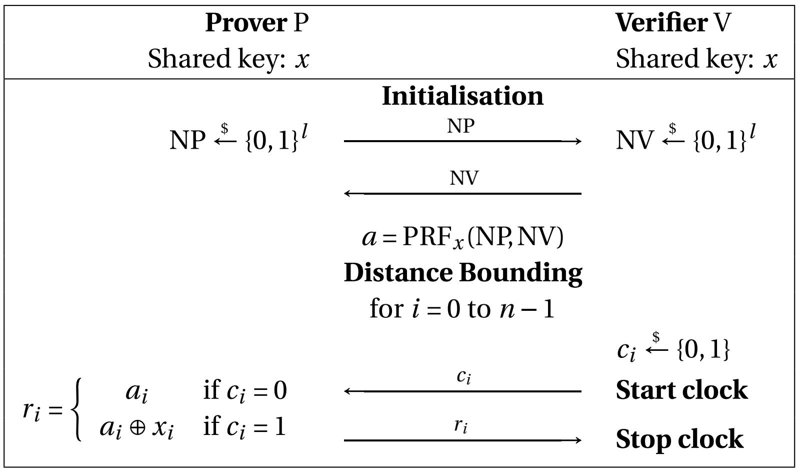
Prover P
Shared key: x
Verifier V
Shared key: x
Initialisation
$NP \xleftarrow{\$} \{0,1\}^l$
NP
$NV \xleftarrow{\$} \{0,1\}^l$
NV
$a = PRF_x(NP, NV)$
Distance Bounding
for $i = 0$ to $n-1$
$c_i \xleftarrow{\$} \{0,1\}$
$r_i = \begin{cases} a_i & \text{if } c_i = 0 \\ a_i \oplus x_i & \text{if } c_i = 1 \end{cases}$
c_i
Start clock
r_i
Stop clock

Figure 3.3: The classical countermeasure against terrorist fraud: if the prover gives both possible responses, *i.e.*, a_i and $a_i \oplus x_i$ to his accomplice for a given c_i, he leaks one bit of his long-term authentication secret x. Here, n is the numer of rounds, and l is the size of the nonces.

Some protocols are designed for the adversary to recover the secret key if the prover helps him once, for instance [Reid et al., 2007], but for other protocols, such as SKI[Boureanu et al., 2013], the accomplice needs to be helped several times by the same prover to recover the key. The strategy of forcing the prover to leak his secret key to his accomplice is the one employed in the security models [Dürholz et al., 2011, Boureanu et al., 2015] which formally define the threats against distance bounding protocols. In Chapter 5, we show that there exists an attack which permits the prover to help his accomplice without leaking his secret key for most protocols. However, this attack is out of the scope of the security models. Hence, before Chapter 5, when we write that a protocol is terrorist fraud resistant, we mean that it is terrorist fraud resistant according to the adversarial capabilities described in the models, and do not account for attack defined in Chapter 5.

If the protocol does not feature resistance to transmission error, the strategy described on Figure 3.3 is sufficient. However, if communication noise resistance is taken into account in the design of the protocol, then this strategy fails, as we develop in the next section.

3.3.4 Terrorist frauds in noise resistant protocols

To make the protocols tolerant to transmission errors, the common approach is to make the verifier accept the authentication of the prover if no more than n_{err}rounds contain errors during the distance bounding phase. The threshold value n_{err} should be polynomial in n, and a typical value is 5% of n. If we make the assumption that an adversary can reduce the error rate of the channel, for instance by being very close to the verifier, then this adversary can have up to n_{err}errors in his responses and still be accepted. This is not really an issue for a classical adversary, since he still has to guess $n - n_{err}$ response bits, and this becomes exponentially more difficult as the number of rounds n increases. On the other hand, it can allow for terrorist frauds against protocols in which combining the two response vectors reveals the secret key, such as Reid et al.'s protocol [Reid et al., 2007], as pointed out in [Hancke, 2012]. The malicious prover can give two response vectors to his accomplice, but deliberately introduce errors in n_{err} rounds. With this strategy, the accomplice can successfully pass the protocol due to the bit error tolerance, but recovering the key of the prover is difficult. Indeed, by XORing the two vectors, the accomplice can only recover a bitstring with Hamming Distance n_{err} to the secret key of the prover. Since he does not know the positions of the incorrect bits, he has to try flipping each possible n_{err}-uple of bits, which requires $\binom{n}{n_{err}}$ operations, and is unfeasible for a sufficiently large number of rounds n.

To counter such attacks, the accomplice can be given a way to pass the protocol on his own even with a noisy version of the secret key. For instance, a form of special mode can be used, in which one can use a noisy version of the secret key to authenticate [Fischlin and Onete, 2013a]. Other solutions let the accomplice obtain the whole key from a noisy version of it, for instance through a leakage scheme as in [Boureanu et al., 2015], or using a scheme called an extractor, as in [Boureanu and Vaudenay, 2014].

This attack works because real world implementations differ from the perfect environment considered while designing the protocols, due to the necessity of handling communication errors in real life. Similarly, while it is tempting to abstract the PRF used in the protocol as a random oracle, this simplification opens the door for attacks when real PRFs are used, as shown in the next subsection.

3.3.5 PRF programming attacks

In 2012, Boureanu et al. [Boureanu et al., 2012] exhibit new attacks on PRF based distance bounding protocols. The key idea to these attacks is that the underlying assumption for the security of such protocols is that the prover cannot control the output of the PRF. If he could for instance force $r^0 = r^1$ for all rounds, then he could mount a distance fraud by sending his responses in advance, as they would become independent of the challenge. However, the security that is expected from a PRF is that, to an adversary who does not know its key, it is indistinguishable from a truly random function. On the other hand, in distance bounding protocols, the prover does know the key, so the security of the PRF alone may not be sufficient. Indeed, for instance, a good block cipher is expected to behave as a secure PRF, but its use as a PRF could lead to problems in a distance bounding protocol where the only input would be a prover generated nonce NP. Indeed, in this case, the prover could simply obtain the nonce corresponding to an all-zero bitstring output by computing the decryption of an all-0 bitstring with his secret key. A less trivial example is the following PRF f, used to compute $\text{PRF}_x(\text{NP}||\text{NV})$ (where x is a secret key, NP the prover's nonce, and NV the verifier's nonce). It returns $g_x(\text{NP}||\text{NV})$ (where g is a regular PRF) when $\text{NP} \neq x$, and $r^0||r^0$ (for a random r^0) when NP is equal to x. This is a secure PRF, since an adversary who does not know the key has a negligible probability of finding a NP such that $\text{NP} = x$, which would trigger the non random behaviour, but a malicious prover could exploit it by choosing $\text{NP} = x$ to mount a distance fraud.

Similarly, Boureanu et al. [Boureanu et al., 2012] presented Mafia Frauds that work when the PRF used contained a trapdoor, and are efficient when part of the output of the PRF are XORed with parts of the key. They show that there exist intricate PRF constructions which allow a Mafia Fraud adversary to extract the secret key of the prover. In practice, it means that a malicious manufacturer could implement backdoors in the PRF used by the distance bounding devices, in order to be able to extract the key of any prover later.

For this reason, the designers of Distance Bounding protocols should be extremely careful on how the PRF is used. Preventing distance frauds based on such PRFs can be done by XORing a random mask picked by the verifier to the response vectors, so that they cannot anymore be completely controlled by the prover. The Mafia Fraud attacks can be prevented by using two separate keys, one for the PRF and one for TF resistance, as in [Fischlin and Onete, 2013b], or by using PRFs that are safe to use when XORed with parts of the key, for instance circular-secure PRFs (as defined in [Boureanu et al., 2012]).

The successive apparition of all these new attack model forced the design of many new protocols. In the next section, we present some classical ones.

3.4 Classical Protocols

In this section, we present a selection of Distance Bounding protocols from the literature. Note that this list of protocols is far from being exhaustive, since more than 40 distance bounding protocols appear in the literature [Brelurut et al., 2016, Gildas et al., 2017].

3.4.1 Brands and Chaum: Protocol 1

In their seminal paper, "Distance Bounding Protocols" [Brands and Chaum, 1994], Brands and Chaum build the first distance bounding protocols. They build up to a secure protocol through several examples. The first one simply handles Mafia Frauds, and is shown on Figure 3.4. This protocol, similarly to most of those that followed, is composed of three parts, and only one, the Distance Bounding phase, is timed. During the first period, called the initialisation phase, the prover draws a random bitstring β of length n, where n is a number of rounds, chosen according to a security parameter. Then, during the distance bounding phase, a two-message exchange is repeated n times: the verifier draws a random challenge bit c_i, and the prover immediately responds with $r_i = \beta_i$ (the i^{th} bit of β). The verifier stores the time elapsed between sending c_i (time t_0) and receiving r_i (time t_1) as $\Delta t_i = t_1 - t_0$. After that, during the verification phase, the prover generates a digital signature on the concatenation of all challenges and responses, and sends it to the verifier. If the signature is correct, and it holds that $\forall i \in [1..n], \Delta t_i \leq t_{\max}$, then the protocol succeeds, and the verifier outputs an accepting bit $\mathrm{O}ut\mathrm{V} = 1$. This protocol is resistant to Mafia Fraud: if an adversary relays the communications between the verifier and a far away prover, the time measurements will show it. Hence, the adversary can either send a random challenge to the prover in advance, or forward the correct challenge to the prover and try to guess the response. In both cases, at each round, his probability to guess properly is $\frac{1}{2}$. If the signature scheme used in the protocol is unforgeable, then the adversary only has a negligible probability of forging a valid final signature if he did even just one mistake among the n rounds.

Brands and Chaum however note a flaw in this protocol: the response bits during the Distance Bounding phase do not depend of the challenges at all. Due to this, a far away malicious prover can send his responses in advance, so that $\Delta_t \leq t_{\max}$ despite the distance.

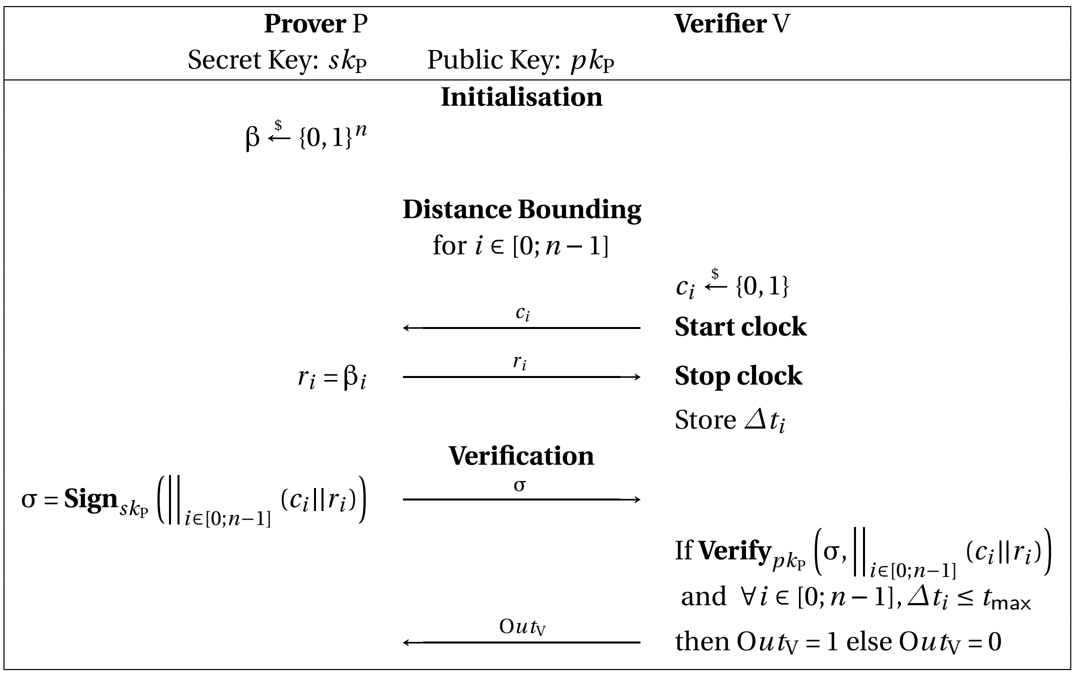
Prover P
Secret Key: sk_P
Verifier V
Public Key: pk_P
Initialisation
$\beta \xleftarrow{\$} \{0,1\}^n$
Distance Bounding
for $i \in [0; n-1]$
$c_i \xleftarrow{\$} \{0,1\}$
c_i
Start clock
$r_i = \beta_i$
r_i
Stop clock
Store Δt_i
Verification
$\sigma = \mathbf{Sign}_{sk_P}\left(\Big\|_{i\in[0;n-1]} (c_i \| r_i)\right)$
σ
If $\mathbf{Verify}_{pk_P}\left(\sigma, \Big\|_{i\in[0;n-1]} (c_i \| r_i)\right)$
and $\forall i \in [0; n-1], \Delta t_i \leq t_{\max}$
then $\mathrm{O}ut_V = 1$ else $\mathrm{O}ut_V = 0$
$\mathrm{O}ut_V$

Figure 3.4: The first Distance Bounding protocol proposed by Brands and Chaum in [Brands and Chaum, 1994]. It is resistant to mafia fraud, but vulnerable to distance fraud.

3.4.2 Brands and Chaum: Protocol 2

In the first protocol, a malicious prover could defeat the distance bound by sending responses in advance. To counter this attack, called a Distance Fraud, Brands and Chaum propose a second protocol, depicted on Figure 3.5. This protocol is only designed to resist Distance Fraud, hence, the bitstring β is public and accessible by anyone. Then, during the Distance Bounding phase, the prover replies with $r_i = \beta_i \oplus c_i$ upon receiving a challenge. The verifier then checks that all the responses are correct, as well as the time measurements. In this protocol, the prover cannot send the response in advance reliably: he only has a probability of $\frac{1}{2}$ of guessing the correct challenge, which is needed to send the correct response. On the other hand, this protocol is not resistant to Mafia Fraud: since the response string is public, and no computations involve a secret value, an adversary can authenticate successfully.

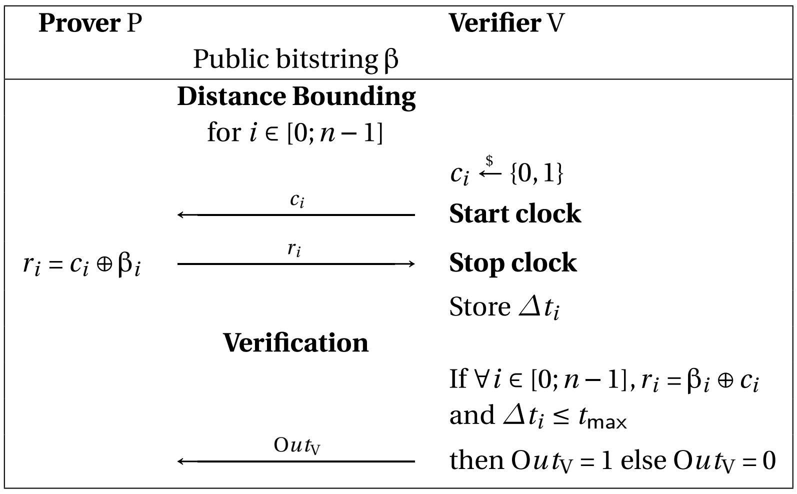
Prover P
Verifier V
Public bitstring β
Distance Bounding
for $i \in [0; n-1]$
$c_i \xleftarrow{\$} \{0,1\}$
c_i
Start clock
$r_i = c_i \oplus \beta_i$
r_i
Stop clock
Store Δt_i
Verification
If $\forall i \in [0; n-1], r_i = \beta_i \oplus c_i$
and $\Delta t_i \leq t_{\max}$
Out_V
then $Out_V = 1$ else $Out_V = 0$

Figure 3.5: The second Distance Bounding protocol proposed by Brands and Chaum in [Brands and Chaum, 1994]. It is resistant to distance fraud, but vulnerable to mafia fraud.

3.4.3 Brands and Chaum: Protocol 3

Finally, Brands and Chaum combined the idea of these two protocols to build one that is resistant to both Distance and Mafia fraud. It is depicted on Figure 3.6. Note that further in this thesis, when we mention Brands and Chaum's protocol, it is this one that is concerned. During the initialisation phase, the prover generates a random bitstring β, and sends a cryptographic commitments on its value. Then, during the Distance Bounding phase, the prover replies to the challenges c_i with $r_i = \beta_i \oplus c_i$. Finally, during the verification phase, the prover opens the commitment, so that the verifier can check that the β_i values were the ones the prover had committed to, as well as a signature on the concatenation of challenges and responses. This protocol is secure against both Distance Fraud and Mafia Fraud. It is however vulnerable to a terrorist fraud: a prover can sent the response vector β to his accomplice without leaking any secret information. Brands and Chaum left this as an open problem for future work.

3.4.4 Reid et al.

In [Reid et al., 2007], Reid et al. propose a Terrorist Fraud resistant protocol. This protocol is based on the same idea as the one proposed by Bussard and Bagga [Bussard and Bagga, 2005], except that the protocol of Bussard and Bagga uses computationally expensive public key primitives, while the protocol of Reid et al. only relies on symmetric primitives. Both protocol aim at making it impossible for a malicious prover to permit an accomplice to pass the protocol without revealing its secret key. Remember that for terrorist fraud resistance, we assume that provers want to protect

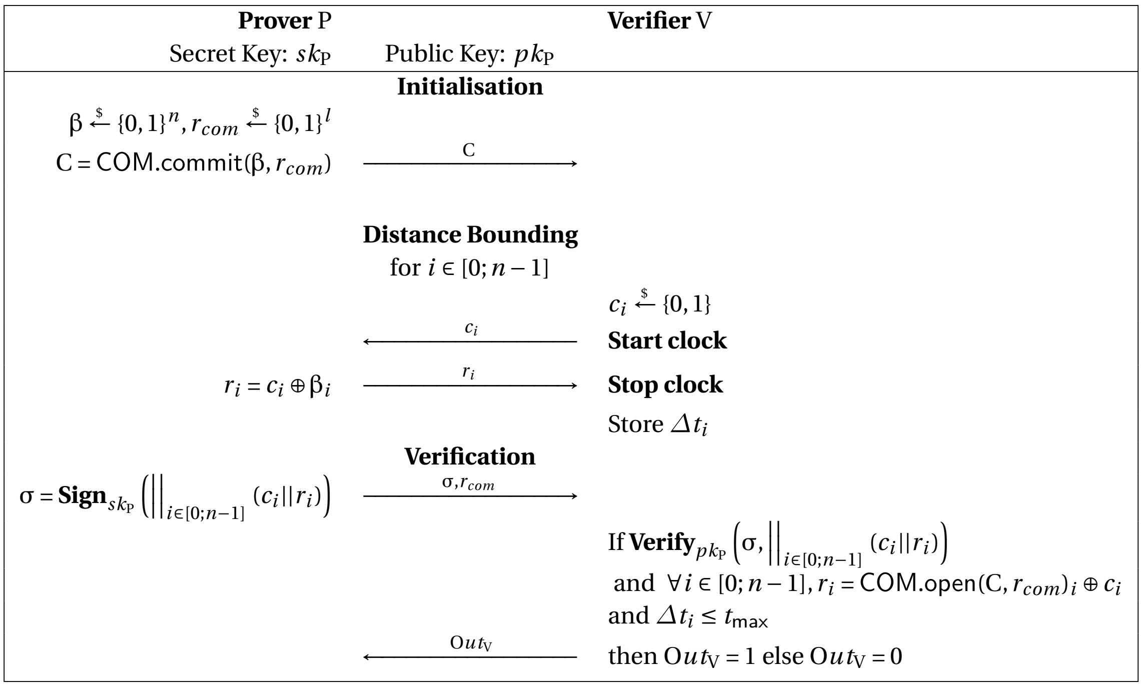
Prover P
Secret Key: sk_P
Public Key: pk_P
Verifier V
Initialisation
$\beta \xleftarrow{\$} \{0,1\}^n, r_{com} \xleftarrow{\$} \{0,1\}^l$
$C = \mathsf{COM.commit}(\beta, r_{com})$
$\xrightarrow{C}$
Distance Bounding
for $i \in [0; n-1]$
$c_i \xleftarrow{\$} \{0,1\}$
$\xleftarrow{c_i}$
Start clock
$r_i = c_i \oplus \beta_i$
$\xrightarrow{r_i}$
Stop clock
Store Δt_i
Verification
$\sigma = \mathbf{Sign}_{sk_P}\left(\Big|\Big|_{i\in[0;n-1]} (c_i||r_i)\right)$
$\xrightarrow{\sigma, r_{com}}$
If $\mathbf{Verify}_{pk_P}\left(\sigma, \Big|\Big|_{i\in[0;n-1]} (c_i||r_i)\right)$
and $\forall i \in [0; n-1], r_i = \mathsf{COM.open}(C, r_{com})_i \oplus c_i$
and $\Delta t_i \leq t_{\max}$
then $Out_V = 1$ else $Out_V = 0$
$\xleftarrow{Out_V}$

Figure 3.6: The third Distance Bounding protocol proposed by Brands and Chaum in [Brands and Chaum, 1994], in which COM is a commitment scheme. It is resistant to both Mafia Fraud and Distance Fraud.

themselves against later impersonation. In the protocol of Reid et al., depicted on Figure 3.7, the prover and verifier each pick a nonce, respectively NP and NV, and exchange them. They use these nonces to build a response vector $a = \mathsf{MAC.tag}_x(\text{NP}||\text{NV})$, where MAC is a MAC scheme. In the original paper, the requirement if for the output of MAC to be pseudorandom, so that it is actually used as a PRF. Both the prover and the verifier then compute $e = x \oplus a$. During the distance bounding phase, the prover responds with a_i if the challenge is 0, and e_i if the challenge is 1. In this protocol, if a malicious prover wants to help an accomplice to pass the protocol by giving him the two response strings, then the accomplice can recover $x = a \oplus e$, so trivial terrorist frauds are avoided. al. [Bay et al., 2013] still

However, this protocol, and others relying on the same idea, are vulnerable to terrorist frauds when noise resistance is added to the protocol, as developed in Section 3.3.4. The next protocol we present is resistant to terrorist frauds in the presence of noise.

3.4.5 Fischlin and Onete

In 2013, Fischlin propose a Distance Bounding protocol designed to have provable Terrorist Fraud resistance, even in a noisy communication environment, and to additionally resist PRF programming attacks [Fischlin and Onete, 2013b]. This protocol is shown on Figure 3.8. It uses two keys, x and y: x is used to compute the PRF, and y is the one that is XORed with the output of the PRF, in order to prevent Mafia Frauds based on PRF programming.

It starts with the verifier sending a nonce NV, and the prover sending a nonce NV, along with a value b. A honest prover always sends $b = 0$, and a value I such that $\text{PRF}_x(\text{NP}||\text{NV}) = \text{I}||a^0$. In this case, both then compute $a^1 = a^0 \oplus y$, and start the Distance Bounding phase. The prover responds to the challenges with $r_i = a_i^{c_i}$ at each round. Finally, the prover authenticates the transcript during the verification phase. To be Terrorist Fraud resistant even in the presence of noise, it includes a special mode which allows someone (typically the accomplice $\mathscr{A}$ of a dishonest prover after a Terrorist Fraud) knowing a bitstring with a low Hamming Distance to a secret key y to pass. This mode is activated when $\mathscr{A}$ sends $b = 1$. He can then run a kind of degraded version of the protocol,

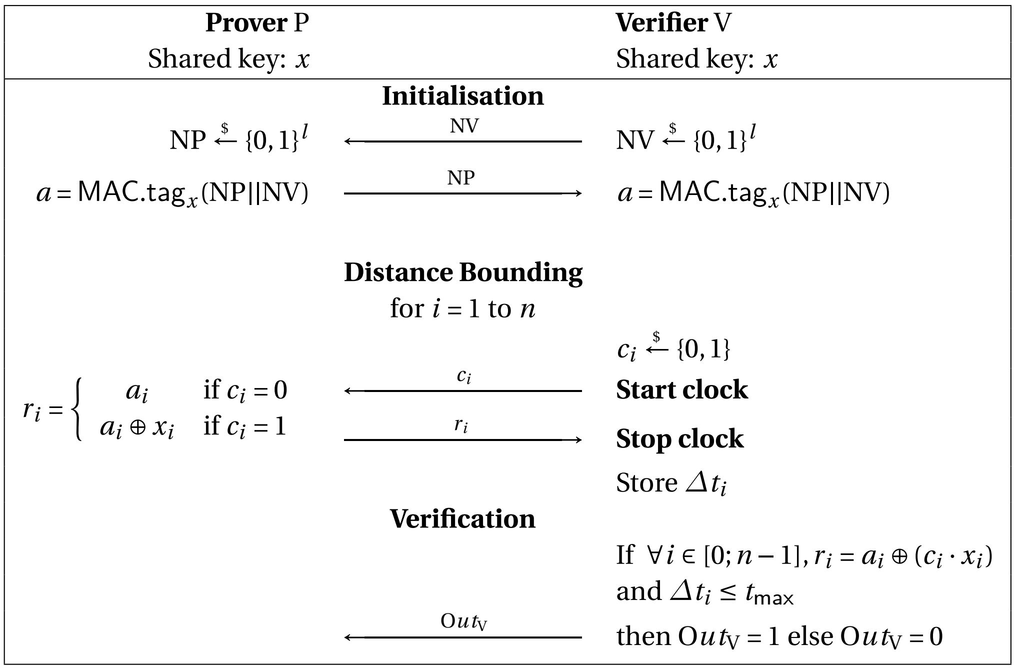
Prover P
Shared key: x
Verifier V
Shared key: x
Initialisation
$NP \xleftarrow{\$} \{0,1\}^l$
NV
$NV \xleftarrow{\$} \{0,1\}^l$
$a = \mathsf{MAC.tag}_x(NP||NV)$
NP
$a = \mathsf{MAC.tag}_x(NP||NV)$
Distance Bounding
for $i = 1$ to n
$c_i \xleftarrow{\$} \{0,1\}$
$r_i = \begin{cases} a_i & \text{if } c_i = 0 \\ a_i \oplus x_i & \text{if } c_i = 1 \end{cases}$
c_i
Start clock
r_i
Stop clock
Store Δt_i
Verification
If $\forall i \in [0; n-1], r_i = a_i \oplus (c_i \cdot x_i)$
and $\Delta t_i \leq t_{\mathsf{max}}$
Out_V
then $Out_V = 1$ else $Out_V = 0$

Figure 3.7: The Distance Bounding protocol proposed by Reid et al. in [Reid et al., 2007], where MAC is a MAC scheme used as a PRF.

in which the responses are equal to the challenges during the Distance Bounding phase. If he responds correctly, $\mathscr{A}$ is accepted with probability $min(1, 2^{\mathrm{HD}(y,\mathrm{I})+\mathrm{TMAX}+\mathrm{EMAX}})$, where TMAX+EMAX is the maximum number of errors tolerated in the protocol.

Some of these protocols come with proofs, which give a certain level of confidence in their security. In order to carry these proofs, a formal model is needed. This model precisely defines the attacker capabilities, as well as the threats that the protocol aims at resisting. In the next section, we present one of these formal models, which we used to prove the security of our protocols.

3.5 Formalism for Distance Bounding

In this section, we formally define the notion of distance bounding protocols, as well as the associated threat models. While several formal security models for distance bounding were proposed [Dürholz et al., 2011, Boureanu et al., 2015, Avoine et al., 2009], we only present the one by Dürholz, Fischlin, Kasper and Onete [Dürholz et al., 2011], which we used for our proofs.

3.5.1 Formal Definition of DB Protocols

Formally, DB protocols are interactive protocols running between two participants. The objective of the *prover* P is to prove to the *verifier* V that he is legitimate and located at a distance at most d_{max}. The participants interact during *rounds*, defined as sequences of messages. For some of these rounds, the verifier uses a *clock* to measure the time elapsed between the emission of a challenge c_i and the reception of the corresponding response r_i. These back-and-forth rounds are referred to as *time-critical* rounds while otherwise they are referred to as *slow phases* or *lazy phases*. In most protocols, the *DB phase* of a protocol is composed of either n independent time-critical rounds or only one combined time-critical round. Having measured the elapsed time at the end of each time-critical round, the verifier then compares this value to a threshold t_{max} associated with the maximal allowed distance d_{max}. If at least one of these tests fails, the prover will not be considered in the vicinity of the verifier.

Without loss of generality, the DFKO model considers a single verifier. This verifier is assumed

Prover P Shared keys: x, y		**Verifier** V Shared keys: x, y
	Initialisation	
$b = 1, \mathrm{NP} \xleftarrow{\$} \{0,1\}^l$	$\xleftarrow{\mathrm{NV}}$	$\mathrm{NV} \xleftarrow{\$} \{0,1\}^l$
$(\mathrm{I}' \| a^0) = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV})$	$\xrightarrow{b, \mathrm{I}', \mathrm{NP}}$	$(\mathrm{I} \| a^0) = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV})$
$a^1 = a^0 \oplus y$		$a^1 = a^0 \oplus y$
	Distance Bounding for $i \in [0; n-1]$	
		$c_i \xleftarrow{\$} \{0,1\}$
$r_i = \begin{cases} a_i^0 & \text{if } c_i = 0 \\ a_i^1 & \text{if } c_i = 1 \end{cases}$	$\xleftarrow{c_i}$	**Start clock**
	$\xrightarrow{r_i}$	**Stop clock**
		Store Δt_i
	Verification	
$\mathrm{CR} = \Big\|_{i \in [0;n-1]} (c_i \| r_i)$		$\mathrm{CR} = \Big\|_{i \in [0;n-1]} (c_i \| r_i)$
$\mathbb{T} = \mathrm{PRF}_x(\mathrm{NP}\|\mathrm{NV}\|\mathrm{I}'\|b\|\mathrm{CR})$	$\xrightarrow{\mathbb{T}}$	
	Acceptance conditions	
$b = 0$		$b = 1$
$\wedge \mathrm{I}' = \mathrm{I}$		$r_i = c_i$
$\wedge \mathbb{T}$ correct		$\wedge \Delta t_i \leq t_{\max}$
$\wedge r_i = a_i^{c_i}$		
$\wedge \Delta t_i \leq t_{\max}$		

Figure 3.8: The Distance Bounding protocol propoposed by Fisclin and Onete in [Fischlin and Onete, 2013b]. It has 2 modes: If $b = 0$, it behaves as a regular Distance Bounding protocol, but if $b = 1$, it switches to a special mode in which it accepts the authentication probability $p_{\mathrm{HW}(\mathrm{I},y)}$ (a probability that gets higher as the hamming distance between I and y decreases) if the responses are equal to the challenges.

to behave honestly during the authentication of a prover. However, he may try to lift the *anonymity* of a prover if this is possible. On the other hand, provers can potentially behave maliciously and attempt to fool the verifier, either by themselves or by using (voluntary or unwilling) accomplices.

Most distance bounding protocols are *revocable*: it is possible to deny access to a given user if he needs to be banned from the system. For non anonymous protocols, revocation is trivial: the verifier knows with which prover it is interacting, so he can consult a revoked user list to check whether he should accept or not. On the other hand, for anonymous protocols, revocation is more tricky. Hence, we include a revocation mechanism in the features of a distance bounding protocol, even though it only needs to be explicitly defined for anonymous protocols. Note that for some protocols, such as PDB [Ahmadi and Safavi-Naini, 2014], the revocation is not defined.

When a protocol uses nonces, their length is denoted k.

The features distance bounding protocol, using a user list UL, and a revoked user list RL, can be defined as follows:

Definition 17 (DB protocol)**.** *A* distance-bounding *protocol DB is defined by the following algorithms:*

$\mathsf{DB.gen}(1^\lambda)$ *is the algorithm run by an honest party, setting up the cryptographic primitives for a security parameter* λ*. It also sets the number of the time-critical rounds* n*, and the nonce length* l*, which are polynomial in the security parameter* λ*.*

$\mathsf{DB.prover}(\mathbb{K}_{\mathsf{ID}})$ *is the algorithm run by a honest prover, with identity* ID *and secret key(s)* $\mathbb{K}_{\mathsf{ID}}$.

$\mathsf{DB.verifier}(\mathbb{K}_V, \mathsf{UL}, \mathsf{RL})$ *is the algorithm run by the verifier, with the secret key(s)* $\mathbb{K}_V$, *with a user list* UL *and a revoked user list* RL. *At the end of its execution, it outputs a bit* Out_V, *which is 0 if the authentication is accepted, and 1 otherwise.*

$\mathsf{DB.join}(\mathsf{ID}, \mathsf{UL})$ *is the algorithm used to register a new prover with identifier* ID *in the list* UL. *It generates the keys for this new prover and initialises it with the corresponding values* $\mathbb{K}_P$.

$\mathsf{DB.revoke}(\mathsf{ID}, \mathsf{UL}, \mathsf{RL})$ *is the algorithm to revoke a prover with identifier* ID *in* UL *and transfer him to the revoked users list* RL.

Additionally, revocable protocols with anonymity need a functionality that allows a trusted authority (for instance, the entity who runs $\mathsf{DB.gen}(1^\lambda)$*) to deanonymize a transcript:*

$\mathsf{DB.open}(\mathsf{id}_P)$ *is the algorithm that returns the identifier of the prover who was accepted in the session* id_P.

3.5.2 Formal models: The DFKO Framework

In this section, we describe the DFKO model for defining the classical threats against these protocols.

High Level Overview. The DFKO model defines the security properties of distance bounding protocols as security games [Shoup, 2004]. In a security game, an adversary plays according to some rules, and a winning condition, which define a successful attack. In the DFKO formalism, the interactions between the participants are modelled as sessions. For each security property, the capabilities of the adversary in these sessions are defined. While the adversary is generally allowed to do anything, even relaying messages between far away participants, some of these actions are said to *taint* the session in which they are performed. A *tainted* session intuitively denotes a session in which the adversary did something that is forbidden, either by the laws of physics or by the attack scenario. In order to win, the adversary must be able to run a session that is not tainted, and satisfies the winning conditions.

Adversary Model. In this DFKO model, an adversary is a probabilistic polynomial time algorithm, who can interact with provers and verifiers in three kinds of sessions:

- *Prover-verifier* sessions, in which he observes an honest execution of the protocol between a prover and a verifier.
- *Prover-adversary* sessions, in which he interacts with a honest prover as a verifier.
- *Adversary-verifier* sessions, in which he interacts with a legitimate verifier as a prover.

Each session is associated with a unique identifier sid.

The adversaries are quantified in terms of their computational power t, the number of prover-verifier sessions q_{obs} they may observe, the number q_{v} of adversary-verifier sessions and the number q_{p} of prover-adversary sessions that they initiate as well as their winning advantage for the corresponding security games.

Each game has its own definition for a *tainted* message exchange. A tainted exchange indicates that an attack scenario is ruled out by definition, due for instance to the verifier's ability to detect pure relays through his accurate clock. In other words, the adversary can perform any action, but some of these actions may taint the session. An adversary cannot win a game in a tainted session. A session is tainted if any of its time critical phases is tainted.

Following the terminology introduced by Vaudenay [Vaudenay, 2007] and later re-used to define prover-anonymity [Hermans et al., 2013a], if an adversary is assumed to know the final result of an authentication session (*i.e.*, accept or reject), he is said to be *wide*, while otherwise he is *narrow*. Orthogonally, if the adversary may never corrupt provers, he is considered to be *weak* while if a corruption query is only followed by other such queries, the adversary is *forward*. Finally, if there is no restriction on the corruption queries, the adversary is said to be *strong*. In our proofs, we consider the strongest adversary model possible, namely *wide-strong* adversaries.

Communications and time. Communications involving honest entities are always part of a session. To capture the notion of relays, the DFKO framework uses an abstract clock keeping track of the sequence of the adversary's actions. It is given as a function $\mathsf{marker} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, such that $\mathsf{marker}(\cdot, \cdot)$ is strictly increasing. In the following definitions, $\Pi_{\mathsf{sid}}[i, \dots, j]$ denotes a sequence of messages $(m_i, \dots, m_j)$ exchanged during the session sid of the protocol.

A timed exchange is called a time-critical phase, it starts when the clock starts, and stops when the clock stops. A time-critical phase might contain several challenge response exchanges, each of which is called a round. In most protocols of the literature, each phase is composed of only one round, *i.e.*, there is only one challenge and one response between the moment the clock starts and when it stops. Hence, except when imposed by the protocol, we sometimes use the word time-critical round instead of phase. The untimed message exchanges of the protocol are called *lazy* (or *slow*) phases.

Game structure The threat models are represented as security games involving an adversary $\mathscr{A}$ and a challenger simulating the environment for him. All these game-based proofs start with the challenger building the simulation environment using $\mathsf{DB.gen}(1^\lambda)$. For clarity, this step is omitted in their descriptions. The adversary interacts with the simulated environment through oracles that he is allowed to run concurrently. These include a prover oracle (for prover-adversary sessions), a verifier oracle (for adversary-verifier sessions) as well as a session oracle to simulate an honest exchange between a prover ID and the verifier. The challenger can simulate the following oracles, to which $\mathscr{A}$ can access a polynomial number of times, for a prover identifier ID of his choice:

$\overline{\mathsf{DB}}.\mathsf{Verifier}(\cdot)$ *runs the protocol* $\mathsf{DB.verifier}(\mathbb{K}_\mathsf{V}, \mathsf{UL}, \mathsf{RL})$.

$\overline{\mathsf{DB}}.\mathsf{Prover}(\cdot)$ *runs the protocol* $\mathsf{DB.prover}(\mathsf{ID}, \mathscr{K}_\mathsf{ID})$.

$\overline{\mathsf{DB}}.\mathsf{Session}(\cdot)$ *returns the transcript of a new honest run of the protocol.*

$\overline{\mathsf{DB}}.\mathsf{Join}^h(\cdot)$ *simulates the join of a honest prover* ID *by running* $\mathsf{DB.join}(\mathsf{ID}, \mathsf{UL})$.

$\overline{\mathsf{DB}}.\mathsf{Join}^c(\cdot)$ *simulates the join of a corrupted prover* ID *by running* $\mathsf{DB.join}(\mathsf{ID}, \mathsf{UL})$ *and returning the secret keys.*

$\overline{\mathsf{DB}}.\mathsf{Corrupt}(\cdot)$ *simulates the corruption of a prover* U_i *by returning his secret keys.*

$\overline{\mathsf{DB}}.\mathsf{Revoke}(\cdot)$ *on input* ID, *it runs* $\mathsf{DB.revoke}(\mathsf{ID}, \mathsf{RL}, \mathsf{UL})$ *to revoke the prover* P_ID.

Note that the adversary always has access to $\overline{\mathsf{DB}}.\mathsf{Prover}(\cdot), \overline{\mathsf{DB}}.\mathsf{Verifier}(\cdot)$ and $\overline{\mathsf{DB}}.\mathsf{Session}(\cdot)$, used to emulate the 3 kinds of sessions, but can only access $\overline{\mathsf{DB}}.\mathsf{Join}^h(\cdot), \overline{\mathsf{DB}}.\mathsf{Join}^c(\cdot), \overline{\mathsf{DB}}.\mathsf{Corrupt}(\cdot)$ and $\overline{\mathsf{DB}}.\mathsf{Revoke}(\cdot)$ oracles for the privacy and anonymity game.

We now give the definitions for the security games corresponding to the different attack scenarios.

Mafia Fraud (MF)

During a mafia fraud, an active MiM adversary, interacting with a single prover and a single verifier during several sessions, tries to authenticate.

The adversary is not able to purely relay information between the verifier and the prover during the time-critical phases. On the other hand, the model allows him to relay a modified message, which allows for something equivalent to the learning phase of [Boureanu et al., 2015]. Accordingly, the tainted time-critical phases are defined as follows.

Definition 18 (Tainted session (MF)). *A time-critical phase $\Pi_{\mathsf{sid}}[k, k+1] = (m_k, m_{k+1})$, for $k \geq 1$, of an adversary-verifier session* sid, *with the message m_k being received by the adversary as the k^{th} challenge from the verifier, is* tainted by the time-critical phase $\Pi_{\mathsf{sid}^*}[k, k+1] = (m^*_k, m^*_{k+1})$ of a prover-adversary session sid^* *if the 3 following conditions are satisfied:*

$$(m_k, m_{k+1}) = (m^*_k, m^*_{k+1}),$$
$$\mathsf{marker}(\mathsf{sid}, k) < \mathsf{marker}(\mathsf{sid}^*, k),$$
$$\mathsf{marker}(\mathsf{sid}, k+1) > \mathsf{marker}(\mathsf{sid}^*, k+1).$$

Once this definition is given, the game-based definition of MF resistance notion can be stated as follows.

Definition 19 (MF Resistance). *For a DB authentication scheme DB, a $(t, q_{\mathsf{v}}, q_{\mathsf{p}}, q_{\mathsf{obs}})$-MF adversary $\mathcal{A}$ wins against DB if the verifier accepts $\mathcal{A}$ in one of the q_{v} adversary-verifier sessions* sid, *which does not have any critical phase tainted by a prover-adversary session* sid^*. *The advantage of an adversary $\mathcal{A}$ $\mathsf{Adv}^{\mathsf{MF}}_{\mathcal{A},DB}(\lambda)$ in this experiment is the probability that $\mathcal{A}$ wins the MF game. We define the advantage on the MF experiment as*

$$\mathsf{Adv}^{\mathsf{MF}}_{DB}(\lambda) = \max_{\mathcal{A} \in \mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{\mathsf{MF}}_{\mathcal{A},DB}(\lambda)\}.$$

A protocol is MF-resistant if $\mathsf{Adv}^{\mathsf{MF}}_{DB}(\lambda)$ is negligible.

Distance Fraud and Distance Hijacking (DF and DH)

Distance Hijacking [Cremers et al., 2012] was not defined in the original DFKO framework. However, the framework was extended to include this threat in [Avoine et al., 2017], as a generalisation of the distance fraud property. In DF attacks, the adversary is a malicious prover who aims to authenticate from a distance greater than d_{max}. In DH attacks, the adversary attempts to do the same, but he uses the unintentional help of *legitimate provers* located close to the verifier. The remote adversary may initiate the DB protocol and let the nearby prover complete the DB phase.

To capture DH in the DFKO framework, we consider an adversary (here a malicious prover) able to use the help of an honest prover in the verifier's proximity and having two choices.

- In the DB phase, he commits to a response in advance, before the challenge of that specific round, and sends that commitment. These commitments are not cryptographic commitments, with the properties of binding and hiding, but rather they indicate the prover's choice with regards to a response, which he must transmit to the verifier.
- In any phase, he commits to a special message Prompt, triggering the response by a close-by honest prover.

The prompting oracle also works in lazy phases. If the adversary either (1) fails to commit or prompt for one specific phase, or (2) sends a different value than committed *after* receiving the challenge, he taints the phase and the session. More formally, when the adversary opens a verifier-adversary session sid, he also opens two associated dummy sessions $\mathsf{sid}_{\mathsf{Commit}}$ for committed responses and $\mathsf{sid}_{\mathsf{Prompt}}$ for the responses prompted from the prover. Technically, such an adversary is more powerful than in a typical DH attack [Boureanu and Vaudenay, 2014], since the adversary can intercept time-critical responses that are sent by the honest prover, and replace them with his own committed responses. More precisely, the formal definition of tainted phases is as follows.

Definition 20 (Tainted Time-Critical Phase (DH)). *A time-critical phase denoted $\Pi_{\mathsf{sid}}[k, k+2 \cdot l - 1] = (m_k, m_{k+2 \cdot l-1})$ of an adversary-verifier session* sid, *with the message m_k being received by the adversary as the k^{th} challenge from the verifier, is* tainted *if one of the following conditions holds.*

The maximal j with $\Pi_{\mathsf{sid}_{\mathsf{Commit}}}[j] = (\mathsf{sid}, k+1, m^*_{k+2\cdot l-1})$ for $m^*_{k+1} \neq \mathsf{Prompt}$ and $\mathsf{marker}(\mathsf{sid}, k) > \mathsf{marker}(\mathsf{sid}_{\mathsf{Commit}}, j)$ satisfies $m^*_{k+1} \neq m_{k+1}$ (or no such j exists).

The maximal j with $\Pi_{\mathsf{sid}_{\mathsf{Commit}}}[j] = (\mathsf{sid}, k+1, m^*_{k+1})$ for $m^*_{k+1} = \mathsf{Prompt}$ satisfies $m^{k+1} \neq m^{\mathsf{Prompt}}_{k+1}$, in which $m^{\mathsf{Prompt}}_{k+1}$ denotes the message m_{k+1} in $\mathsf{sid}_{\mathsf{Prompt}}$.

This definition rules out some potential actions of attackers. Once this is done, the game-based definition of DH resistance notion can be stated as follows.

Definition 21 (DH Resistance)**.** *For a DB authentication scheme DB with DB threshold $t_{\max}$, a $(t, q_{\mathsf{p}}, q_{\mathsf{v}}, q_{\mathsf{obs}})$-DH adversary $\mathscr{A}$ (with $\mathsf{id}_{\mathscr{A}}$) wins against DB if the verifier accepts $\mathsf{id}_{\mathscr{A}}$ in one of q_{v} adversary-verifier sessions, which does not have any critical phase tainted.*

The advantage of an adversary $\mathscr{A}$ $\mathsf{Adv}^{\mathsf{DH}}_{\mathscr{A},DB}(\lambda)$ in this experiment is the probability that $\mathscr{A}$ wins the DH game. We define the advantage on the DH experiment as

$$\mathsf{Adv}^{\mathsf{DH}}_{DB}(\lambda) = \max_{\mathscr{A} \in \mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{\mathsf{DH}}_{\mathscr{A},DB}(\lambda)\}.$$

A protocol is DH-resistant if $\mathsf{Adv}^{\mathsf{DH}}_{DB}(\lambda)$ is negligible.

In this thesis, we present distance bounding protocols that are anonymous. In these protocols, the verifier does not know which prover is authenticated, so that it could seem pointless to perform a distance hijacking. However, in anonymous protocols, an entity called opening authority is allowed to deanonymize a session. Hence, a malicious prover could still mount an alibi, by performing a distance hijacking, and then asking the opening authority to verify that he did authenticate. For this reason, we still consider distance hijacking relevant for anonymous protocols, with regards to the opening authority.

Terrorist-Fraud (TF)

Terrorist fraud resistance is a notion that is difficult to capture. Many different definitions for it exist in the literature. The DFKO model defines the notions of GameTF [Fischlin and Onete, 2013a] and strSimTF [Dürholz et al., 2011]. There also exists another definition, SimTF, which is similar to strSimTF, except that in SimTF, the prover and his accomplice are forbidden to communicate with each other during the time critical phases, while in strSimTF, this limitation is lifted.

In both strSimTF and GameTF, a far away, dishonest prover P* helps an accomplice $\mathscr{A}$ to pass the protocol. Both attacks are defined as a two phase security game, which are identical for the first phase, but different for the second. In the first phase, the prover helps the adversary (his accomplice), so that the adversary obtains some information, referred to as view. This adversary is accepted by V with probability $p_{\mathscr{A}}$. In the second phase, the information view is used in an authentication attempt. For GameTF, an adversary $\mathscr{B}$(view) uses it, in the presence of P*, who is still far away but behaves honestly. He wins if he passes the protocol with a probability greater than the probability of the best MF adversary. For strSimTF, a simulator, using only view but without access to the prover, tries to authenticate, and wins if he passes with a probability greater than $p_{\mathscr{A}}$.

We first define the tainted session notion for both strSimTF and GameTF. For SimTF, the tainted phase definition is different, and we also give it for completeness, even though we do not use it in our proofs.

Definition 22 (Tainted session - GameTF and strSimTF)**.** *A time-critical phase $\Pi_{\mathsf{sid}}[k, k+2\cdot l-1] = (m_k, m_{k+2\cdot l-1})$, for $k, l \geq 1$, of an adversary-verifier session* sid*, with the message m_k being received by the adversary as the k^{th} challenge from the verifier, is* tainted *if there exists a prover-adversary session* sid^**, and messages $(m^*_k, m^*_{k+2\cdot l-1})$ such that for all $i \in \{0, l-1\}$ we have*

$$\begin{aligned} &\mathsf{marker}(\mathsf{sid}, k+2\cdot i) < \mathsf{marker}(\mathsf{sid}^*, k+2\cdot i), \\ \textit{and} \quad &\mathsf{marker}(\mathsf{sid}, k+2\cdot i+1) > \mathsf{marker}(\mathsf{sid}^*, k+2\cdot i+1). \end{aligned}$$

In other words, the adversary cannot relay both the challenge and its response.

In the SimTF definition, the malicious prover is not allowed to communicate with his accomplice at all during the time-critical phases. Thus, any communication between them during any time-critical phase taints the session, which is formalised by the following definition:

Definition 23 (Tainted Session - SimTF). *An adversary-verifier session* sid*, containing the time-critical phases* $\Pi_{\mathsf{sid}}[k, k+2\cdot l-1] = (m_k, m_{k+2\cdot l-1})$*, for* $k, l \geq 1$*, with the* k*-th message being received by the adversary, is* tainted *if there is a session* sid' *between the adversary and* P *such that, for any* i*,*

$$\mathsf{marker}(\mathsf{sid}, k) < \mathsf{marker}(\mathsf{sid}', i) < \mathsf{marker}(\mathsf{sid}, k+2\cdot l-1).$$

Let view be the internal state of $\mathscr{A}$ after colluding with P.

For GameTF, we are interested in whether this help was *helpful* to $\mathscr{A}$, *i.e.*, whether the help allows $\mathscr{A}$ to win the MF game with a better probability than an unhelped adversary.

Definition 24 (Helpful adversary). *A TF adversary* $\mathscr{A}$ *against a distance-bounding authentication protocol DB is said to be* helpful *to an adversary* $\mathscr{B}$ *playing the MF game if, given* view*,* $\mathscr{B}$ *wins the MF game with a probability at least* $\mathsf{Adv}_{DB}^{MF}(\lambda)$.

The GameTF definition follows:

Definition 25 (gametf-security). *For a DB protocol DB, a* $(t, q_{\mathsf{v}}, q_{\mathsf{p}}, q_{\mathsf{obs}})$*-terrorist-fraud adversary pair* $(\mathscr{A}, \mathrm{P})$*, a DB protocol DB is* GameTF*-secure if, for* $\mathscr{A}$ *authenticating with probability* $p_{\mathscr{A}}$ *with the help of* P^**, at least one of the two following conditions holds:*

- $p_{\mathscr{A}}$ *is negligible with respect to* n*,*
- *there exists a MF adversary* $\mathscr{B}$ *running* $\mathrm{O}(q_{\mathsf{obs}})$ *prover-verifier sessions and* $\mathrm{O}(q_{\mathsf{v}})$ *adversary-verifier sessions to which* $\mathscr{A}$ *is helpful.*

In $\mathsf{strSimTF}$, we are interested in the success probability of a simulator Sim, given view, and without any interaction with any legitimate prover. Let p_{Sim} denote this success probability. The TF attack by the pair $(\mathrm{P}, \mathscr{A})$ is *successful*, if the help of P^* during the attack does make any difference in the success probability of ~ (*i.e.*, if $p_{\mathscr{A}} > p_{\mathsf{Sim}}$).

The definition of the TF-resistance notion $\mathsf{strSimTF}$ follows:

Definition 26 (TF Resistance $\mathsf{strSimTF}$). *For a DB authentication scheme DB, a* $(t, q_{\mathsf{v}}, q_{\mathsf{p}}, q_{\mathsf{obs}})$*-terrorist-fraud adversary pair* $(\mathscr{A}, \mathrm{P})$ *and a simulator* Sim *running in time* t_{Sim}*, the malicious prover* P *and his accomplice* $\mathscr{A}$ *win against DB if* $\mathscr{A}$ *authenticates in at least one of* q_{v} *adversary-verifier sessions* without tainting it *with probability* $p_{\mathscr{A}}$*, and if* Sim *authenticates in one of* q_{v} *sessions with the view of* $\mathscr{A}$ *with probability* p_{Sim}*, then* $p_{\mathscr{A}} \leq p_{\mathsf{Sim}}$.

TF resistance is a binary property. Indeed, the accomplice (*i.e.*, the simulator) is either able to impersonate independently the prover with at least the same probability in later sessions having the initial information received from the prover (*i.e.*, TF-resistant) or not.

Slow-phase impersonation

The notion of slow phase impersonation was defined when the hardware used for distance bounding only allowed for a limited number of time-critical rounds. If the number of rounds is low, then an attacker can more easily guess the responses. This security property aims at expressing resistance to this kind of attacks: the slow phases of the protocol should be sufficient for a safe authentication. Hence, a protocol is said to be slow-phase impersonation resistant if it is unfeasible for an adversary able to relay the messages during the protocol to authenticate without purely relaying the messages of the slow phases.

Let the IF experiment $\mathsf{Exp}_{\mathscr{A},\mathsf{DB}}^{\mathsf{IF}}(\lambda)$ for an adversary $\mathscr{A}$ on a protocol DB be defined as follows.

Definition 27 (Impersonation Fraud Resistance)**.**
Let DB be a distance bounding protocol, and $\mathscr{A}$ be an adversary. $\mathscr{A}$ has access to the DB-oracles $\mathsf{Join}^h(\cdot)$, $\mathsf{Join}^c(\cdot)$, $\mathsf{Revoke}(\cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{Prover}(\cdot)$, $\mathsf{Verifier}(\cdot)$, $\mathsf{Session}(i)$, $\mathsf{H}(\cdot)$ *and* $\mathsf{Taint}(\cdot)$. *$\mathscr{A}$ wins if and only if* $\exists\, \mathsf{trans} \in \mathsf{TL_V}$ *such that* trans *is the concatenation of all the messages exchanged during a DB session, and both the following conditions are satisfied.*

- $OutV = 1$,
- $\nexists t \in \mathsf{TL_V}$ *such that* $t \neq \mathsf{trans}$ *and the lazy phases of t and* trans *are equal.*

The advantage of an adversary $\mathscr{A}$ $\mathsf{Adv}^{\mathsf{IF}}_{\mathscr{A},DB}(\lambda)$ *in this experiment is the probability that $\mathscr{A}$ wins the IF game. We define the advantage on the DH experiment as*

$$\mathsf{Adv}^{\mathsf{IF}}_{DB}(\lambda) = \max_{\mathscr{A}\in\mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{\mathsf{IF}}_{\mathscr{A},DB}(\lambda)\}.$$

DB is IF-resistant *if* $\mathsf{Adv}^{IF}_{DB}(\lambda)$ *is negligible.*

We have chosen to discard the slow-phase impersonation-security threat in our analysis. Initially, this notion has been introduced for resource-limited provers and states that the authentication of a prover should be difficult even if only a reduced number of time-critical rounds is supported. This notion is relatively ambiguous: it states that the protocol should remain secure even if the adversary is capable of guessing all the time-critical responses. However, the number of rounds depends on the security parameter, just like the size of the keys and nonces. Therefore, if the adversary is capable of guessing all the responses, with probability 2^{-n}, then he should also be capable of guessing the key, and the protocol is not secure. Therefore, we believe that the need for slow-phase authentication is no longer a constraint for the design of DB protocols, and we do not take this property into account in our analysis.

MiM-Privacy

MiM-privacy (which we sometimes refer to as privacy) is a property that deals with the ability of an adversary to distinguish which prover is running the protocol, and was added to DFKO in [Gambs et al., 2014]. It is formalised as follows:

Definition 28 (MiM-Privacy Protection)**.** *Let DB be a DB scheme. The MiM-privacy experiment* $\mathsf{Exp}^{MiM\text{-}Priv}_{\mathscr{A},DB}(\lambda)$ *for an adversary $\mathscr{A}$ on DB is defined as follows. $\mathscr{A}$ interacts with a challenger who runs the algorithm* $\mathsf{DB.gen}(1^\lambda)$ *and sends all the public parameters to $\mathscr{A}$. During the experiment, $\mathscr{A}$ can access the usual oracles, but also* $\overline{DB}.\mathsf{Corrupt}(\cdot)$, $\overline{DB}.\mathsf{Join}^c(\cdot)$, $\overline{DB}.\mathsf{Join}^h(\cdot)$, $\overline{DB}.\mathsf{Revoke}(\cdot)$. *After interacting with these oracles, $\mathscr{A}$ sends the pair of provers* $(\mathsf{ID}_0, \mathsf{ID}_1)$ *to the challenger. If* ID_0 *or* ID_1 *is in* CU*, the challenger aborts the experiment. Otherwise, he picks* $b \xleftarrow{\$} \{0,1\}$*. Then, $\mathscr{A}$ loses access to* $\mathsf{Revoke}(\cdot)$ *and* $\mathsf{Corrupt}(\cdot)$ *on* $\imath_0$ *and* $\imath_1$ *(the oracles return $\perp$ if $\mathscr{A}$ uses these inputs).*

$\mathscr{A}$ is then given access to the following oracle:

$\overline{DB}.\mathsf{Prover}(\cdot)_b$ *simulates a session by the prover* $\mathsf{P}_{\mathsf{ID}_b}$ *using* $\mathscr{K}_{\mathsf{ID}_b}$.

Finally, $\mathscr{A}$ returns b'. If $b = b'$, the challenger returns 1 *(*i.e.*, the guess of $\mathscr{A}$ is correct), while otherwise he returns* 0.

We define $\mathscr{A}$'s advantage on this experiment as

$$\mathsf{Adv}^{MiM\text{-}Priv}_{\mathscr{A},DB}(\lambda) = \left| \mathsf{Pr}[\mathsf{Exp}^{MiM\text{-}Priv}_{\mathscr{A},DB}(\lambda) = 1] - \frac{1}{2} \right|,$$

and the advantage on the privacy experiment as

$$\mathsf{Adv}^{MiM\text{-}Priv}_{DB}(\lambda) = \max_{\mathscr{A}\in\mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{MiM\text{-}Priv}_{\mathscr{A},DB}(\lambda)\}.$$

DB is MiM-privacy preserving *if* $\mathsf{Adv}^{Priv}_{DB}(\lambda)$ *is negligible.*

The next property is anonymity. It deals with the ability of the verifier to identify which prover is running the protocol. For this property, the adversary is even stronger, so the privacy game is included in it. Stated differently, an anonymous protocol is also private.

3.5.3 Prover Anonymity

Prover anonymity is a property that is stronger than privacy. A protocol is prover anonymous if even the verifier is not capable of distinguishing which prover is running the protocol.

Its definition is exactly the same as the one for the privacy game, except that $\mathscr{A}$ is additionally given the keys of the verifier $\mathscr{K}_V$ as input.

Definition 29 (Prover Anonymity)**.** *Let DB be a DB scheme. The anonymity experiment* $\mathsf{Exp}^{Anon}_{\mathscr{A},DB}(\lambda)$ *for an adversary* $\mathscr{A}$ *on DB is defined as follows.* $\mathscr{A}$ *interacts with a challenger who runs the algorithm* $\mathsf{DB.gen}(1^\lambda)$ *and sends all the public parameters to* $\mathscr{A}$*. During the experiment,* $\mathscr{A}$ *can access the usual oracles, but also* $\overline{DB}.\mathsf{Corrupt}(\cdot)$, $\overline{DB}.\mathsf{Join}^c(\cdot)$, $\overline{DB}.\mathsf{Join}^h(\cdot)$, $\overline{DB}.\mathsf{Revoke}(\cdot)$*, and receives the set of verifier keys* $\mathscr{K}_V$*.*

After interacting with these oracles, $\mathscr{A}$ *sends the pair of provers* $(\mathsf{ID}_0, \mathsf{ID}_1)$ *to the challenger. If* ID_0 *or* ID_1 *is in* CU*, the challenger aborts the experiment. Otherwise, he picks* $b \xleftarrow{\$} \{0,1\}$*. Then,* $\mathscr{A}$ *loses access to* $\mathsf{Revoke}(\cdot)$ *and* $\mathsf{Corrupt}(\cdot)$ *on* ID_0 *and* ID_1 *(the oracles return* $\perp$ *if* $\mathscr{A}$ *uses these inputs).*

$\mathscr{A}$ *is then given access to the following oracle:*

$\overline{DB}.\mathsf{Prover}(\cdot)_b$ *simulates a session by the prover* $\mathrm{P}_{\mathsf{ID}_b}$ *using* $\mathscr{K}_{\mathsf{ID}_b}$*.*

Finally, $\mathscr{A}$ *returns* b'*. If* $b = b'$*, the challenger returns* 1 *(*i.e.*, the guess of* $\mathscr{A}$ *is correct), while otherwise he returns* 0*.*

We define $\mathscr{A}$*'s advantage on this experiment as*

$$\mathsf{Adv}^{Anon}_{\mathscr{A},DB}(\lambda) = \left|\Pr[\mathsf{Exp}^{Anon}_{\mathscr{A},DB}(\lambda) = 1] - \frac{1}{2}\right|$$

and the advantage on the PA experiment as

$$\mathsf{Adv}^{Anon}_{DB}(\lambda) = \max_{\mathscr{A}\in\mathsf{Poly}(\lambda)} \{\mathsf{Adv}^{Anon}_{\mathscr{A},DB}(\lambda)\}.$$

DB is prover anonymous *if* $\mathsf{Adv}^{Anon}_{DB}(\lambda)$ *is negligible.*

3.5.4 Introduction to Proofs

These security games define the capabilities and aims of an adversary against a given security property. Given an adversary against a property of a protocol, the game is ran by a challenger, in such a way that $\mathscr{A}$ cannot distinguish between the environment provided by the challenger, and an actual execution of the protocol. These games can be used to build security proofs: from a security game, and a protocol, the aim is to prove that an adversary has a negligible success probability in winning this game. A success probability is negligible if it is defined by a negligible function of the security parameter. The size of the nonces and keys used in the protocol, as well as the number of rounds n, depends on the security parameter.

Since the adversaries we consider are bounded to perform a polynomial number of operations, if their success probability is negligible, then the protocol is secure. In our proofs, we generally prove that the success probability is negligible in n, instead of λ. However, since n is chosen according to λ, this is equivalent to showing that the success probability is negligible in λ.

Some proofs are too complex to carry them on the complete protocol directly. In order to make them simpler and easier to follow, we use the game-hopping technique formalised by Shoup in [Shoup, 2004]. In this formalism, the initial security game, denoted G0, is transformed to a simplified game G1. Let $Pr[Gi]$ denote the success probability of the adversary in the game Gi: in order for the transformation to be possible, we need to prove that $|\Pr[G1] - \Pr[G0]| \in negl(\lambda)$, *i.e.*, that the transformation does not alter the success probability of $\mathscr{A}$ in a significant way. Then, this game G1 can in turn be transformed into a simpler game, under the same conditions, and so on, until we reach a *final game*, in which the proof is easier to perform. The transition between games, or game hops, can for instance be used to replace pseudorandom values by actually random values, in order to finish the proof.

3.6 Conclusion

Relay attacks can be used by two colluding adversaries to defeat a security protocol, by making the prover and the verifier believe that they are communicating directly with each other, even though they are actually communicating with attackers. This can be prevented by measuring the time of flight of the messages, in an attempt to bound the distance of the prover, using Distance Bounding protocols. These protocols are typically composed of untimed initialisation phase, during which the prover and the verifier agree on two response vectors, followed by a timed distance bounding phase, during which the verifier sends one bit challenges and expects a response from the corresponding response vector. Some protocols have a final, untimed phase, in which additional messages are exchanged, typically commitment openings or signatures on the transcript. While most distance bounding protocols are resistant to relay attacks, generalised under the name of Mafia Fraud, they might be vulnerable to frauds from dishonest, distant provers trying to cheat on their distance: Distance Fraud, Distance Hijacking and Terrorist Fraud. In the first one, the prover acts on his own. In the second one, he exploits honest provers that are located near the verifier. In the last one, he uses the help of an accomplice, located near the verifier, but with the limitation that he does not want the accomplice to be able to impersonate him later (otherwise, the prover could perform a trivial attack, by simply giving the accomplice his credentials). These security properties, along with privacy and prover anonymity, were formalised in formal models, including the DFKO framework. Distance Bounding is a very rich research topic, as more than 40 protocols were published in the past three decades. The current challenges are provable security, in particular against Terrorist Fraud, and the preservation of privacy for the users of the protocols. In the next chapter, we present solutions for attaining various levels of privacy, while preserving provable security.

Chapter 4

Provably Secure Distance Bounding Protocols

Contents

The number of distance bounding protocols vulnerable to attacks brought a strong incentive to build provably secure distance bounding protocols [Dürholz et al., 2011, Boureanu et al., 2015]. Additionally, growing concerns about privacy preservation motivate the need for protocols that are anonymous, in the sense that even the verifier cannot trace which prover ran the protocol, as long as this prover belongs to a list of legitimate users. In this chapter, we present our two provably secure distance bounding constructions: SPADE [Bultel et al., 2016] and TREAD [Avoine et al., 2017]. They both rely on the idea of leaking an authenticated session key, instead of a long term secret, for terrorist fraud resistance. This terrorist fraud resistance corresponds the DFKO security model [Dürholz et al., 2011], even though it can be defeated when specific hardware is used, as developed in Chapter 5.

We prove the security of both SPADE and TREAD in the DFKO [Dürholz et al., 2011] formalism.

4.1 Introduction

The Snowden revelations about mass surveillance brought more incentive for the research community to focus on privacy preservation. In particular, it is important that the users of security systems can have the insurance that they will not be trackable, either by eavesdroppers, the company providing them with a service, ad companies, or government agencies. On the other hand, companies must protect their interests, so they could be reluctant to propose anonymous authentication if no revocation mechanism exists to ban misbehaving users.

Introducing anonymity with revocation in distance bounding protocols is challenging. Indeed, for terrorist fraud resistance, the responses need to be intertwined with the secret key of the prover. It seems difficult to make the verifier able to verify the responses if he is not capable of identifying which secret key is intertwined with the responses. If he is capable of doing it, however, he is able to identify the prover, which contradicts the need for anonymity.

While challenging, an anonymous and secure distance bounding protocol is appealing. It could for instance be used for membership cards in a public transportation service. In such a system, a user would pay every month to access the city transportation services, and present his card to a contactless reader to be allowed in subway stations. This system should of course be protected against relay attacks, since non paying users should not be able to use the subway. Additionally, a far away card holder should never be allowed to use his card, since it would permit him to let illegitimate users in. Terrorist Fraud resistance also makes sense in this context: we could imagine a rogue user who would offer people to use his own access for money (at a lower rate than the usual price). Such a rogue user, if he is rationale, would not want the other users to be able to access his card on their own, but only when they pay him, so he would like to retain his credentials. Additionally, many things can be inferred from the location history of an individual [Krumm, 2009], such as his home or work address, or even social links between individuals. Hence, the public transportation history of a person can be very sensitive information and should be protected, so the company running the subway should not be able to determine the whereabouts of a given user. On the other hand, if the company suspects that a user gave away his credentials, or that they were stolen, it should be able to ask an administrator to deanonymize a given protocol session, in order to be able to ban these credentials. Finally, the verifiers should be capable of determining whether an authentication attempt is performed by a revocated user or a legitimate one.

In this chapter, we solve the problem of designing protocols suitable for such use cases.

4.2 Contributions

We describe two constructions, both provably resistant to Mafia Fraud, Distance Hijacking and Terrorist Fraud (as per the security models). The first one, SPADE(for Secure Prover Anonymous Distance-bounding Exchange), is a distance bounding protocol that is prover anonymous: no verifier can identify which prover is running the protocol. It is also revocable, in the sense that a misbehaving user can be banned: the verifier can check whether the user running the protocol is legitimate or revocated, and deny this user access if he is revocated. Our second construction is TREAD (for Terrorist-fraud Resistant and Extractor-free Anonymous Distance-bounding), a generic construction that can be instantiated with an encryption scheme and a signature scheme (in the broad sense: a MAC can be used, for instance). Depending on the choice of encryption and signature, which can both be either public key or private key, the resulting distance bounding protocol can reach different levels of privacy. We describe three instances of TREAD: a lightweight one based on symmetric primitives (TREAD$_{sym}$), with no privacy goals, one that is MiM-private (an eavesdropper cannot determine which prover ran the protocol from the transcript, but the verifier can) and uses public key primitives (TREAD$_{priv}$), and finally one that is prover anonymous and revocable, and uses more demanding cryptographic primitives (group signatures): TREAD$_{ano}$.

Both of these construction share a new design principle: instead of ensuring that a terrorist fraud adversary has to leak his long term secret, they make him leak a prover-chosen temporary value, as well as a signature on it. In the event of a terrorist fraud attempt, the accomplice can play the protocol with the same value. Hence, if he learnt enough information about it, he can pass the protocol.

The two constructions however differ on the method used to grant provable terrorist fraud resistance. While SPADE uses a traditional method based on a key extraction mechanism, TREAD is the first provably terrorist fraud resistant protocol that does not need secret extraction or a backdoor. In fact, SPADE can be seen as a first step towards TREAD, which is more generic and elegant.

4.3 Related Work

Provable Terrorist Fraud Resistance Proving that the help received by an accomplice from a dishonest prover during a terrorist fraud is sufficient for this accomplice to authenticate on his own late is a not trivial. To do so, the usual strategy is to add specific mechanisms to the protocol. Using these mechanisms, an accomplice who received enough help to pass the protocol once with a non negligible probability can exploit the potentially incomplete information he obtained to pass the protocol whenever he wants after the initial help.

There are two strategies for such mechanisms: the first one is to use a special mode, or backdoor, as it is done in the protocol of Fischlin and Onete [Fischlin and Onete, 2013b]. This protocol includes a special mode allowing the adversary to authenticate if he knows a string containing enough bits in common with the secret key of the prover.

The second strategy is to permit the accomplice to extract the whole secret key from the partial information he obtained during the session in which he was helped, if this help was sufficient for him to authenticate with a non negligible probability the first time. Extracting the secret key makes the accomplice able of impersonating his companion prover at will. Hence, these mechanisms deter rational provers from helping illegitimate users authenticate.

In their SKI protocols [Boureanu et al., 2013], Boureanu, Mitrokotsa and Vaudenay employed a *leakage scheme* allowing an adversary to retrieve the long-term secret key after authenticating with the help of a prover several times. The same technique is also used in the DB$_{opt}$ protocols [Boureanu and Vaudenay, 2014].

Similarly, Vaudenay [Vaudenay, 2015b] used *extractor schemes* to recover a string close to the long-term secret key from the view of all nearby participants after a TF attempt.

The first of our constructions, SPADE, uses a form a backdoor similar to the one used by Fischlin and Onete [Fischlin and Onete, 2013b]. The second one, TREAD, does not rely on such additional artificial mechanisms at all, which makes it simpler and somewhat more natural.

While a lot of effort has gone into designing secure DB protocols, the research community has only recently investigated *privacy issues* linked to distance bounding. Considering the amount of information that can be inferred from the location history of an individual [Krumm, 2009], protecting privacy becomes a critical issue for the wide acceptance of such technology.

Privacy To address this concern, two aspects have to be considered: (1) the protection of the privacy of the provers with respect to eavesdroppers and (2) the protection of the anonymity of the provers with respect to verifiers.

In the first distance bounding protocol offering privacy against external eavesdropper, Swiss Knife [Kim et al., 2009], the prover does not send his identifier, but the output of a PRF keyed with a key he shares with the verifier, computed on the nonces. By checking one by one which key matches the result, the verifier is capable of identifying the prover, while no eavesdropper can. Another private protocol, HPO, was introduced in [Hermans et al., 2013b], in which the verifier does not have to go through all the keys it knows. The next year, Gambs, Onete and Robert extended this protocol to deal with *honest-but-curious* and *malicious* verifiers trying to track the users by linking their sessions. They proposed an extension of the HPO protocol, called GOR [Gambs et al., 2014], in which the provers are managed as a group. A prover running the protocol simply proves that he belongs to the group. These two protocols do however not resist to terrorist fraud.

In 2015, Vaudenay [Vaudenay, 2015a] proposed a generic construction to add privacy against external eavesdropper to distance bounding protocols. His solution is to perform an authenticated key exchange before a one-time secure distance bounding protocol, and use the obtained key during the protocol. He does however not treat terrorist fraud in this work.

In 2016, Kilinç and Vaudenay extended Proprox to obtain a privacy preserving and terrorist fraud resistant protocol, eProprox [Kılınç and Vaudenay, 2016].

Finally, Ahmadi and Safavi-Naini [Ahmadi and Safavi-Naini, 2014] proposed the first anonymous distance bounding protocol to resist terrorist fraud. Their protocol is an updated version of the the DBPK-log protocol [Bussard and Bagga, 2005], which fixes a flaw exhibited by Bay $et al.$ in [Bay et al., 2013]. They added an anonymity mechanism to it: the prover uses a *zero-knowledge proof* to prove he knows the secret key used during the protocol, as well as a blind signature on it issued by a trusted authority. This protocol does however not permit to revoke the users whose key was stolen. In addition, it seems to be vulnerable to terrorist frauds in the presence of noise, as described in Section 3.3.4. In [Ahmadi et al., 2018], the same authors proposed two new protocols, DB$i2an^{\mathrm{P}}$ and DB$i2an^{\mathrm{P}}$, which additionally resist to an hardware-based attack they present in the same paper. It is presented as directional TF in Chapter 5.

In opposition, our protocols grant both anonymity and provable terrorist fraud resistance. A comparison of our protocols to other protocols of the literature is given in Table 4.1.

Protocol	TF	MF	DH	Private	Anonymous	Revocable
Not formally proven						
Swiss Knife[Kim et al., 2009]	✓	✓[1]	✓	✓	✗	✓
Proven- Not TF-resistant						
HPO[Hermans et al., 2013b]	✗	✓	✓	✓	✗	✓
GOR[Gambs et al., 2014]	✗	✓	✓	✓	✓	✓
privDB[Vaudenay, 2015a]	✗	✓	✓	✓	✗	✓
Proven- TF-resistant[2]						
PDB[Ahmadi and Safavi-Naini, 2014]	✓[3]	✓	✓	✓	✓	✗
SPADE[Bultel et al., 2016]	✓	✓	✓	✓	✓	✓
eProprox[Kılınç and Vaudenay, 2016]	✓	✓	✓	✓	✗	✓
TREAD$_{sym}$ [Avoine et al., 2017]	✓	✓	✓	✗	✗	✓
TREAD$_{priv}$ [Avoine et al., 2017]	✓	✓	✓	✓	✗	✓
TREAD$_{ano}$ [Avoine et al., 2017]	✓	✓	✓	✓	✓	✓
DB$i2an^{\mathrm{P}}$[Ahmadi et al., 2018]	✓	✓	✓	✓	✓	✓
DB$i2an^{\mathrm{GM}}$[Ahmadi et al., 2018]	✓	✓	✓	✓	✓	✓

Table 4.1: Comparison of privacy preserving protocols. It indicates which of them are resistant to terrorist fraud (TF), Mafia Fraud (MF) and Distance Hijacking (DH, including DF). A ✓ indicates the protocol is secure for a given property, and a ✗ indicates that this property was not considered in the design of the protocol. Private and Anonymous respectively denote privacy with respect to an eavesdropper and anonymity with respect to a malicious verifier. Revocable denotes if a user can be revoked.

4.4 Common Features and Differences of the Two Designs

Both constructions are resistant to MF, DH and TF, and both have an anonymous version.

In contrast to most protocols in the literature, SPADE and TREAD are not ran with a long-term secret shared between a prover and the verifier, but with temporary, session specific values that are chosen by the prover. The use of long-term shared secrets as part of the distance bounding protocol constitute a serious burden to overcome to provide anonymity for the prover, as these secrets can be easily used to link different sessions of a user. On the other hand, using temporary secrets gives more freedom in the design of the protocol. Both protocols are built in such a way that an adversary can replay session specific values if he gets access to it through a TF. This approach constitutes an important shift in the way to handle terrorist fraud resistance. Previous protocols

[1]Even though there exists a PRF programming attack, see [Boureanu et al., 2012] for more details

[2]In the models: hardware-based attacks presented in Chapter 5 not considered.

[3]Only in a noiseless environment

relied on a long term secret being leaked to the accomplice during a terrorist fraud. Our protocols show, on the other hand, that terrorist fraud resistance can be achieved by forcing the prover to leak a session specific value, and making it possible for the accomplice to exploit this value in a new session.

More specifically, in SPADE, the prover picks a session key NP, and sends it encrypted and signed with a dynamic group signature scheme. Group signature schemes allow a user to produce a signature anonymously on behalf of a group of users, so that the verifier cannot distinguish which member of the group signed. This session key is then used to generate the response vector a. In the original article [Bultel et al., 2016], a is generated with a PRF keyed with NP. However, this exposes the protocol to PRF programming attacks, so we replace the PRF by a specific construction in the random oracle model (as defined in Section 3.5.4): more details are given in the description of the protocol. During a terrorist fraud attempt, the accomplice needs to have access to both a_i and $a_i \oplus \mathrm{NP}_i$ to be able to respond in time. If he learns NP, then he can replay the initial encrypted, signed message in order to run a new session using NP as the session key.

In TREAD, the prover directly picks two response vectors a and b and send them, encrypted and signed (with a dynamic group signature scheme for the anonymous version) to the verifier. This provides a simple way to avoid the use of a PRF, and to protect against the associated attacks.

In SPADE, the mechanism used to achieve provable terrorist fraud resistance is quite classic, except for being based on a session key. It uses a technique that permits the accomplice to extract the whole key NP from his view after a successful authentication, namely, a backdoor provided by the verifier. On the other hand, in TREAD, we eliminated the need to extract anything, and focused on a rewinding technique for the proof instead.

4.5 SPADE

4.5.1 Protocol Description

In Figure 4.1, we present SPADE. Its structure is rather similar to the generic structure presented on Figure 3.1. In order to gain anonymity, the provers are authenticated as members of a group, instead of using their actual identity. In essence, the prover first picks a random session key NP, and signs it using a dynamic group signature scheme. The resulting signature can be verified with the public key corresponding to the group, but it cannot be deanonymized, except by a trusted party called *opening authority*. Then, the prover sends a message e encrypted with the public key of the verifier, containing both NP and its signature. The verifier responds with a random nonce NV and a random n-bit value m (where n is the number of rounds of the distance bounding phase), and both compute $a = \mathsf{H}(\mathrm{NP}||\mathrm{NV})$ (instead of $\mathrm{PRF}_{\mathrm{NP}}(\mathrm{NV})$, as proposed in the original paper [Bultel et al., 2016]). Then, the distance bounding phase begins: in n similar rounds, the verifier sends a one-bit challenge c_i, and the prover responds with a one-bit response r_i, which is either a_i if $c_i = 0$, or $a_i \oplus \mathrm{NP}_i$ if $c_i = 1$. Then, during the verification phase, the prover computes $\tau = \mathsf{H}(\mathrm{NP}||\mathrm{NV}||m||\mathrm{C}||\mathrm{R})$ (instead of $\mathrm{PRF}_{\mathrm{NP}}(\mathrm{NV}||m||\mathrm{C}||\mathrm{R})$), where C and R respectively denote the concatenation of all the challenges and the responses, and sends it to the verifier. In the end, the verifier accepts the authentication of the prover if the signature in the initial message was valid and not issued by a revocated user, all the responses were sent in time and properly formed, and τ corresponds to the actual transcript of the protocol.

In order to achieve provable Terrorist Fraud resistance, SPADE includes a form of backdoor. This backdoor is a special function of the verifier, which can be queried by anyone. Before running the protocol, honest provers send a bit $b = 0$, but if the verifier receives $b = 1$, the backdoor is activated, and the user can send a message e, along with NP'. If NP' is sufficiently close to the NP encrypted in the message e, then the verifier returns NP. A terrorist fraud accomplice, to have a chance to successfully authenticate when he is helped, should be able to rebuild a bitstring NP' which is very close to NP, and the backdoor helps him recover the missing bits. By close, we mean that there exists a threshold value t such that t is a fraction of n, and the Hamming distance between NP and NP' is at most t. The probability for an adversary to obtain NP by querying the backdoor with random bitstrings is denoted $p_{back}(n,t)$. More details about the backdoor settings are given in Section 4.5.3.

While most distance bounding protocols focus on lightweight primitives, we made the choice of using group signatures, which are known to be computationally demanding. This might make the protocol difficult to use on devices with limited computation capacity, such as smartcards.

However, this protocol could be used on more powerful devices, such as smartphones. Additionally, since the group signature is computed on input coming exclusively from the prover, it can be computed in advance. The prover can pick NP and compute the group signature beforehand. Alternatively, if the proving device is not capable of computing a group signature, it can be pre-loaded with some NP, σ_p couples computed on a more powerful device, such as a personal computer.

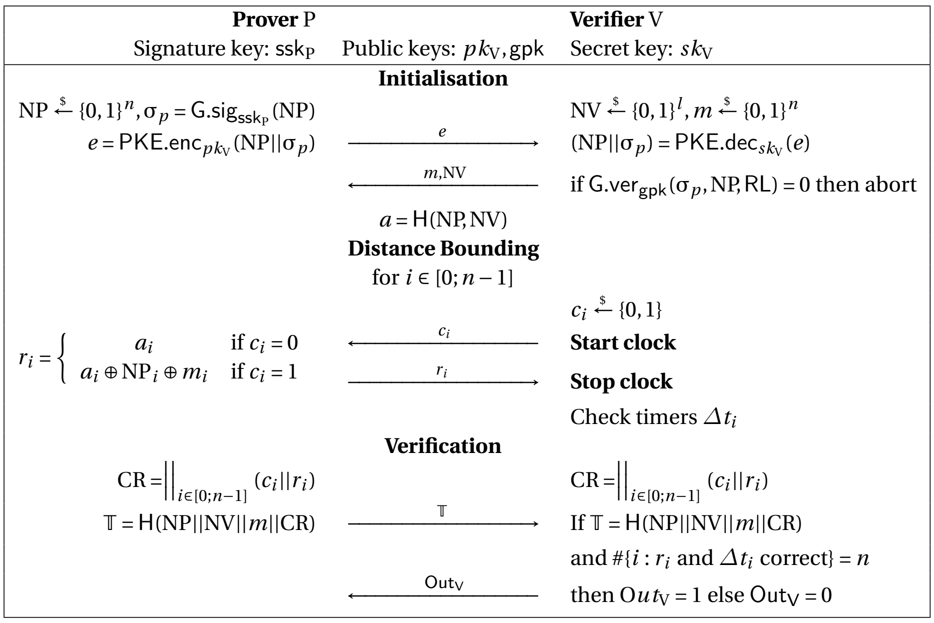
Prover P
Signature key: $\mathsf{ssk_P}$
Public keys: pk_V, gpk
Verifier V
Secret key: sk_V
Initialisation
$\mathrm{NP} \xleftarrow{\$} \{0,1\}^n, \sigma_p = \mathsf{G.sig}_{\mathsf{ssk_P}}(\mathrm{NP})$
$e = \mathsf{PKE.enc}_{pk_V}(\mathrm{NP}||\sigma_p)$
$\xrightarrow{e}$
$\mathrm{NV} \xleftarrow{\$} \{0,1\}^l, m \xleftarrow{\$} \{0,1\}^n$
$(\mathrm{NP}||\sigma_p) = \mathsf{PKE.dec}_{sk_V}(e)$
$\xleftarrow{m,\mathrm{NV}}$
if $\mathsf{G.ver}_{\mathsf{gpk}}(\sigma_p, \mathrm{NP}, \mathrm{RL}) = 0$ then abort
$a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$
Distance Bounding
for $i \in [0; n-1]$
$c_i \xleftarrow{\$} \{0,1\}$
$r_i = \begin{cases} a_i & \text{if } c_i = 0 \\ a_i \oplus \mathrm{NP}_i \oplus m_i & \text{if } c_i = 1 \end{cases}$
$\xleftarrow{c_i}$
Start clock
$\xrightarrow{r_i}$
Stop clock
Check timers Δt_i
Verification
$\mathrm{CR} = \big\|_{i \in [0;n-1]} (c_i||r_i)$
$\mathbb{T} = \mathsf{H}(\mathrm{NP}||\mathrm{NV}||m||\mathrm{CR})$
$\xrightarrow{\mathbb{T}}$
$\mathrm{CR} = \big\|_{i \in [0;n-1]} (c_i||r_i)$
If $\mathbb{T} = \mathsf{H}(\mathrm{NP}||\mathrm{NV}||m||\mathrm{CR})$
and $\#\{i : r_i \text{ and } \Delta t_i \text{ correct}\} = n$
then $Out_V = 1$ else $\mathsf{Out_V} = 0$
$\xleftarrow{\mathsf{Out_V}}$

Figure 4.1: SPADE [Bultel et al., 2016], an anonymous TF resistant protocol built from a public key encryption scheme E , a hash function H and a group signature G.

Definition 30 (SPADE)**.** SPADE *is a distance bounding protocol, parameterised by a public key encryption scheme* E*, a dynamic group signature scheme* G*, and a hash function H which outputs $n-bit$ strings. It is defined as follows:*

$\mathsf{DB.gen}(\lambda)$ *sets the verifier keys* $(pk_V, x_V) = \mathsf{PKE.gen}(\lambda)$ *and the signature key pair* $(\mathsf{gpk}, \mathsf{msk}) = \mathsf{G.gen}(\lambda)$*. It also returns the master key* $\mathsf{MK} = (\mathsf{msk}, \mathsf{gpk}, pk_V, x_V)$ *and a verification key* $\mathsf{VK} = (x_V, \mathsf{gpk})$*, and sets the user list* UL *and the revoked-user list* RL*.*

$\mathsf{DB.join}(i, \mathsf{UL})$ *runs the algorithm* $\mathsf{G.join}_{\mathsf{msk}}(i, \mathsf{gpk}, \mathsf{UL})$ *to get* ssk_i *and constructs* $\mathsf{psk}_i = (pk_V, \mathsf{ssk}_i)$ *for the prover* i*. This algorithm also returns a value* reg_i *and adds* P_i *to* UL*.*

$\mathsf{DB.prover}(\mathbb{K}_{\mathsf{ID}})$ *is the algorithm of the prover, illustrated on Figure 4.1. The prover first picks a n bit value* NP *at random, and computes* $\sigma_p = \mathsf{G.sig}_{\mathsf{ssk_P}}(\mathrm{NP})$ *and* $e = \mathsf{PKE.enc}_{pk_V}(\mathrm{NP}, \sigma_p)$*. The prover then sends it and receives m and* NV*, which it uses to compute* $a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$*. Then, at each time critical round, it responds to the challenge* c_i *with* $r_i = (a_i \wedge \overline{c_i}) \vee ((a_i \oplus \mathrm{NP}_i \oplus m_i) \wedge c_i)$*. Finally, during a final slow phase, the prover sends* $\mathcal{T} = \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R})$*, where* C *and* R *are respectively the concatenation of the challenges and responses.*

$\mathsf{DB.verifier}(\mathbb{K}_V, \mathsf{UL}, \mathsf{RL})$ *is the algorithm of the verifier, illustrated on Figure 4.1. After receiving an initial message e, it computes* $(\mathrm{NP}, \sigma_p) = \mathsf{PKE.dec}_{x_V}(e)$*, and sends two random n bit values* NV *and m. It computes* $a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$*. Afterwards, during the n time-critical phases, he generates a random bit* c_i *from a uniform distribution, starts his clock, sends* c_i*, gets back* r_i*, stops his clock and stores the corresponding time* Δt_i*. It then receives* $\mathscr{T}$*. Finally, it verifies that (1)* $\Delta t_i \le t_{\mathsf{max}}$*, (2)* $r_i = (a_i \wedge \overline{c_i}) \vee ((a_i \oplus \mathrm{NP}_i \oplus m_i) \wedge c_i)$ *for all* $i \le n$*, and (3)* $\mathscr{T} = \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R})$*. If these conditions hold, he sends an accepting bit* $\mathsf{Out_V} = 1$*, while otherwise he sends* $\mathsf{Out_V} = 0$*.*

$\mathsf{DB.revoke}(\mathsf{ID}, \mathsf{RL}, \mathsf{UL})$ *runs* $\mathsf{G.rev_{msk}}(\mathsf{ID}, \mathsf{RL}, \mathsf{UL}, \mathsf{gpk})$.

$\mathsf{DB.open}(\mathsf{trans})$ *computes* $(\mathrm{NP}, \sigma_p) = \mathsf{PKE.dec}_{x_V}(e)$*, in which e is the first message of the transcript. Afterwards, it outputs the prover* $\mathsf{P_{ID}} = \mathsf{G.ope_{msk}}(\sigma_p, \mathrm{NP}, \mathsf{UL}, \mathsf{gpk})$.

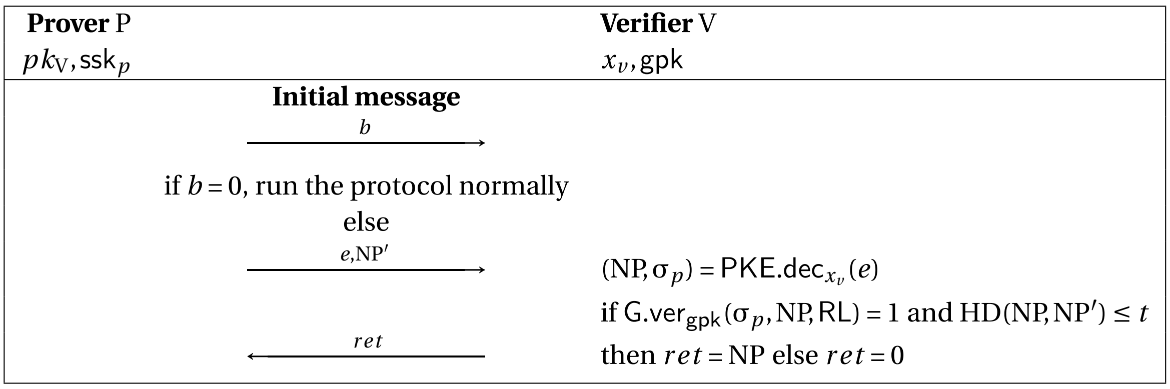
Prover P
$pk_{\mathsf{V}}, \mathsf{ssk}_p$
Verifier V
x_v, gpk
Initial message
b
if $b = 0$, run the protocol normally
else
e,NP′
$(\mathrm{NP}, \sigma_p) = \mathsf{PKE.dec}_{x_v}(e)$
if $\mathsf{G.ver_{gpk}}(\sigma_p, \mathrm{NP}, \mathsf{RL}) = 1$ and $\mathrm{HD}(\mathrm{NP}, \mathrm{NP}') \le t$
ret
then $ret = \mathrm{NP}$ else $ret = 0$

Figure 4.2: The backdoor mechanism. If the initial message is $b = 0$, the protocol is run normally. Otherwise, the verifier simply waits to receive a value e that he parses as (NP, σ_p) and a string NP'. If NP and NP' are close enough, he returns NP.

SPADE is a secure anonymous distance bounding protocol. Below, we give the security proofs for each property, in the DFKO model. In each of these proofs, in the game G0, the prover and verifier behave as defined in Definition 30.

4.5.2 Properties

In this section, we present and prove the security properties of SPADE. Remember that to avoid PRF programming issues that could result from XORing the key of the PRF with its output, we model the PRF $\mathrm{PRF_{NP}}(\cdot)$ as $\mathsf{H}(\mathrm{NP}||\cdot)$, and do our proofs in the random oracle model.

Distance Hijacking

The distance hijacking resistance of SPADE comes from the message m chosen by the verifier, which ensures that the two responses are different for half the rounds on average, so that $\mathscr{A}$ cannot efficiently send a response in advance. Additionally, due to the encryption of NP, $\mathscr{A}$ cannot know which value was chosen by a honest prover to exploit him.

Theorem 1. *If* $p_{back}(n, t)$ *is negligible,* G *is a traceable group signature and* E *is a* $\mathsf{IND\text{-}CCA2}$*-secure encryption scheme, then* SPADE *is DH-resistant.*

Proof. We first rule out the possibility for $\mathscr{A}$ to obtain the value NP used by a honest prover, by preventing the repetition of nonces, and replacing the encryption of NP by the encryption of a dummy value. After that, we can carry the proof in the resulting simplified game, which can be seen as a classical shared-secret DB protocol: the secret NP is shared offline between the prover and the verifier, and the adversary $\mathscr{A}$ does not have any more access to a ciphertext containing this secret.

In the original game G0, the prover and verifier oracles follow the rules of the protocol.

$\mathsf{Prover}(i)$ The prover first picks a n bit value NP at random, and computes $\sigma_p = \mathsf{G.sig}_{\mathsf{ssk_P}}(\mathrm{NP})$ and $e = \mathsf{PKE.enc}_{pk_\mathrm{V}}(\mathrm{NP}, \sigma_p)$. The prover then sends it and receives m and NV, which it uses to compute $a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$. Then, at each time critical round, it responds to the challenge c_i with $r_i = (a_i \wedge \overline{c_i}) \vee ((a_i \oplus \mathrm{NP}_i \oplus m_i) \wedge c_i)$. Finally, during a final slow phase, the prover sends $\mathcal{T} = \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R})$, where C and R are respectively the concatenation of the challenges and responses.

$\mathsf{Verifier}(\cdot)$ After receiving an initial message e, it computes $(\mathrm{NP}, \sigma_p) = \mathsf{PKE.dec}_{x_\mathrm{V}}(e)$, and sends two random n bit values NV and m. It computes $a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$. Afterwards, during the n time-critical phases, he generates a random bit c_i from a uniform distribution, starts his clock, sends c_i, gets back r_i, stops his clock and stores the corresponding time Δt_i. It then receives $\mathcal{T}$. Finally, it verifies that (1) $\Delta t_i \leq t_{\mathsf{max}}$, (2) $r_i = (a_i \wedge \overline{c_i}) \vee ((a_i \oplus \mathrm{NP}_i \oplus m_i) \wedge c_i)$ for all $i \leq n$, and (3) $\mathcal{T} = \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R})$. If these conditions hold, he sends an accepting bit $\mathsf{Out_V} = 1$, while otherwise he sends $\mathsf{Out_V} = 0$.

Game G1. The oracle $\mathsf{Prover}(\cdot)$ uses different NP values for each of the q_P calls. $\mathscr{A}$ loses an advantage of at most $\frac{q_\mathsf{P}^2}{2^n}$. This follows from the binomial expansion of the probability that a given value has been selected zero or once and the union bound. Thus, $|\Pr[\mathrm{G1}] - \Pr[\mathrm{G0}]| \leq \frac{q_\mathsf{P}^2}{2^n}$.

Game G2. The oracle $\mathsf{Verifier}(\cdot)$ uses different NV values for each of the q_V calls. As for the previous transition, we have $|\Pr[\mathrm{G2}] - \Pr[\mathrm{G1}]| \leq \frac{q_\mathsf{V}^2}{2^n}$.

Game G3. $\mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R}) \neq \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}', \mathrm{R})$ if $\mathrm{C}' \neq \mathrm{C}$. The probability for a collision on the hashes of different messages is, in the random oracle model, bounded by $\frac{q_{\mathsf{H}(\cdot)}^2}{2^n}$ (where $q_{\mathsf{H}(\cdot)}$ is the polynomial number of evaluations of the hash function). Hence, $|\Pr[\mathrm{G3}] - \Pr[\mathrm{G2}]| \leq \frac{q_{\mathsf{H}(\cdot)}^2}{2^n}$.

Game G4. This step is simply a preparatory transition for the next game. Instead of only NP, two value NP_0 and NP_1 are drawn at the beginning of the protocol. The message e is computed using NP_0, and the protocol is also run with NP_0: NP_1 is never used. However, the backdoor also works for NP_1: if it is queried with (e, NP_1'), where NP_1' is a bitstring close to NP_1 (according to the backdoor's threshold), then the backdoor returns NP_1. G4 behaves exactly as G3, except in the case of the failure event that $\mathscr{A}$ sends a string close to NP_1 to the backdoor, which occurs with the negligible (by hypothesis) probability $p_{back}(n,t)$. Hence, $|\Pr[\mathrm{G4}] - \Pr[\mathrm{G3}]| \leq p_{back}(n,t)$.

Game G5. The oracle $\mathsf{Prover}(\cdot)$ encrypts NP_0 in his initial message e, and NP_1 used in the rest of the protocol *by both parties.*

The oracles are modified as follows:

- $\mathsf{Prover}(i)$ sets $\mathrm{NP}_0, \mathrm{NP}_1 \xleftarrow{\$} \{0,1\}^{2\cdot n}$, $\sigma_{p_0} = \mathsf{G.sig}_{\mathsf{psk}}(\mathrm{NP}_0)$, $\sigma_{p_1} = \mathsf{G.sig}_{\mathsf{psk}}(\mathrm{NP}_1)$, and computes the message e as $\mathsf{PKE.enc}_{pk_\mathrm{V}}(\mathrm{NP}_0, \sigma_{p_0})$. It adds $(e, \mathrm{NP}_1, \sigma_{p_1})$ to a list WL. It then sends e, but uses NP_1 to compute a, the responses r_i and τ.
- $\mathsf{Verifier}(\cdot)$ acts as in G4, except for the NP computation:
  - If $(e, \mathrm{NP}_1, \cdot) \in \mathsf{WL}$, it uses NP_1.
  - If $(e, \cdot, \cdot) \notin \mathsf{WL}$, it computes $(\mathrm{NP}^*, \sigma_p^*) = \mathsf{PKE.dec}_{x_v}(e)$ and uses NP^*.

Thus, $\mathscr{A}$ loses the possibility of recovering the value of NP from the ciphertext e. However, this advantage is negligible since the PKE encryption scheme is $\mathsf{IND\text{-}CCA2}$ secure.

Let us assume that there exists an efficient adversary $\mathscr{A}$ for the game G5. Thus, an efficient distinguisher $\mathscr{B}$ can be defined for the $\mathsf{IND\text{-}CCA2}$ experiment using $\mathscr{A}$.

Initialisation $\mathscr{B}$ sets up the environment (except the PKE setting). He creates two user lists UL and RL, and a dynamic group signature scheme setup with $\mathsf{G.gen}(1^\lambda)$, obtaining a couple $(\mathsf{gpk}, \mathsf{msk})$. He adds to UL n_P provers using $\mathsf{DB.join}_{\mathsf{msk}}(i, \mathsf{UL})$.

Simulation $\mathscr{B}$ simulates a SPADE environment for $\mathscr{A}$ with the DB-oracles of G4, with the following modifications to the prover and verifier oracles. When the oracle Prover(·) is called, it sets $e = \mathsf{LREnc}_{\mathsf{b}}^{\mathsf{pk_V}}(\mathsf{NP_0}||\sigma_{\mathsf{p}_0}, \mathsf{NP_1}||\sigma_{\mathsf{p}_1})$, adds $(e, \mathrm{NP}_1, \sigma_{p_1})$ to WL and uses NP_1 during the protocol. The oracle Verifier(·) acts as in G4 if it receives e such that $(e, \mathrm{NP}_1, \sigma_{p_1}) \in \mathsf{WL}$. Otherwise, it decrypts e with the provided decryption oracle. Finally, $\mathscr{B}$ returns the result of the authentication $\mathrm{O}utV$ to the challenger, which is 1 if the verifier accepts, and 0 otherwise. Additionally, the backdoor needs to be simulated: if it is queried with strings close enough (according to the backdoor threshold) to either NP_0 or NP_1, it returns the corresponding bitstring (NP_0 or NP_1), as in G4.

If $b = 1$, $e = \mathsf{PKE.enc}_{pk_V}(\mathrm{NP}_1, \sigma_{p_1})$, $(e, \mathrm{NP}_1, \sigma_{p_1}) \in \mathsf{WL}$ and both parties use NP_1 internally. This games corresponds to the game G4. Otherwise, $e = \mathsf{PKE.enc}_{pk_V}(\mathrm{NP}_0, \sigma_{p_0})$ while the NP_1 is used during the protocol. This simulates the game G5. The simulation works because the other parts of the protocol do not leak anything about NP_1: a leaks nothing about the input to H(·) in the random oracle model, the same goes for τ, and the change made in game G3 ensures that $\mathscr{A}$ cannot learn anything about NP_1 from tampering with the challenges.

Let $\mathscr{B}_0$ denote the event that the distinguisher outputs 0, and $\mathscr{B}_1$ the event that it outputs 1. Thus, $\mathrm{P}r[\mathscr{B}_1 | b = 1] = \mathrm{P}r[\mathrm{G4}]$ and $\mathrm{P}r[\mathscr{B}_0 | b = 0] = 1 - \mathrm{P}r[\mathrm{G5}]$, since $\mathscr{B}$ returns the result of the authentication to the distinguisher. The winning probability of $\mathscr{B}$ is then $\mathrm{P}r[\mathscr{B}_1 \wedge b = 1] + \mathrm{P}r[\mathscr{B}_0 \wedge b = 0]$, which is equal to $\mathrm{P}r[\mathrm{G4}] \cdot \frac{1}{2} + (1 - \mathrm{P}r[\mathrm{G5}]) \cdot \frac{1}{2} = \frac{1}{2} + \frac{\mathrm{P}r[\mathrm{G4}] - \mathrm{P}r[\mathrm{G5}]}{2}$. Now assume that $|\mathrm{P}r[\mathrm{G4}] - \mathrm{P}r[\mathrm{G5}]|$ is non negligible. Then $\mathscr{B}$ has a non-negligible advantage on the extended IND-CCA2 game. However, the advantage of $\mathscr{B}$ cannot be more than q_{p} times the advantage on the original IND-CCA2 experiment. Thus, $|\mathrm{P}r[\mathrm{G5}] - \mathrm{P}r[\mathrm{G4}]| \leq q_{\mathsf{p}} \cdot \mathsf{Adv}_{\mathsf{PKE}}^{\mathrm{IND-CCA2}}(\lambda)$, which is negligible by hypothesis.

The final game. We now prove that the success probability of $\mathscr{A}$ in G5 is negligible.

First note that the adversary trying to perform a distance hijacking cannot learn anything useful from the initial message e sent by a honest, close-by prover, since it encrypts a nonce unrelated to the one that is used during the rest of the protocol.

Additionally, since $\mathscr{A}$ wants to be authenticated, he needs to send the initial message: due to the traceability of G, a signature by a honest prover P cannot be opened to the identity of $\mathscr{A}$. However, this initial message cannot (except with negligible probability) contain the value NP_1 used by P during the protocol. Then, during the time-critical phases, $\mathscr{A}$ can either use Prompt or Commit, *i.e.*, he can either trigger the response of P or commit to his own response. Let r_i^{P} denote the response corresponding to Prompt.

- If $\mathscr{A}$ uses Prompt, then his response is valid with probability $\frac{1}{2}$. This corresponds to the probability to have $r_i = r_i^{\mathrm{P}}$, knowing that $\mathrm{NP}_1^{\mathrm{P}} \neq \mathrm{NP}^{\mathscr{A}}$, where $\mathrm{NP}_1^{\mathrm{P}}$ is the value used by P, and $\mathrm{NP}^{\mathscr{A}}$ is the value used by $\mathscr{A}$.

- If he uses Commit, then either $a_i = a_i \oplus \mathrm{NP}_i \oplus m_i$, and he can commit to a correct response with probability 1, or it is not the case, and then he must guess the challenge to commit to the correct response. Since m is uniformly distributed and unknown to $\mathscr{A}$ at the time when he picks NP, we have $\mathrm{P}r[a_i = a_i \oplus \mathrm{NP}_i \oplus m_i] = \frac{1}{2}$. Hence, the probability to commit to the valid response is $\mathrm{P}r[a_i = a_i \oplus \mathrm{NP}_i \oplus m_i] \cdot 1 + \mathrm{P}r[a_i \neq a_i \oplus \mathrm{NP}_i \oplus m_i] \cdot \frac{1}{2} = \frac{3}{4}$.

From this, it follows that the best strategy for $\mathscr{A}$ is to respond by himself, as in a classical DF, using Commit. For n challenges, this strategy succeeds with a negligible probability: $\frac{3}{4}^n$. □

Mafia Fraud Resistance

SPADE is MF resistant. This resistance derives from the fact that NP is sent encrypted, so that it is not accessible to the adversary, and that a final authentication of the transcript, using NP, is performed. Moreover, $\mathscr{A}$ cannot forge a valid signature on his own. For these reasons, the adversary needs to either break the security of the encryption, signature or randomness of the hash function, or to guess the challenges or the responses for each of the n rounds.

Theorem 2. *If $p_{back}(n,t)$ is negligible,* E *is a* $\mathsf{IND\text{-}CCA2}$*-secure encryption scheme,* H *is a hash function in the random oracle model, and* G *is a traceable dynamic group signature scheme, then* SPADE *is MF-resistant.*

Proof. The steps of this proof are similar to the corresponding ones in the DH proof, except that in addition, we eliminate the possibility that $\mathscr{A}$ forges a valid signature.

In the following, WL denotes a list which stores the tuples $(e, \mathrm{NP}, \sigma_p)$ generated by $\mathsf{Prover}(\cdot)$. It allows the prover and the verifier oracles to exchange their shared secrets. In the original game G0, the prover and verifier oracles follow the rules of the protocol, except for the use of WL.

$\mathsf{Prover}(i)$ The prover first picks a n bit value NP at random, and computes $\sigma_p = \mathsf{G.sig}_{\mathsf{ssk_P}}(\mathrm{NP})$ and $e = \mathsf{PKE.enc}_{pk_\mathsf{V}}(\mathrm{NP}, \sigma_p)$. The prover then adds $(e, \mathrm{NP}, \sigma_p)$ to WL, sends e and receives m and NV, which it uses to compute $a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$. Then, at each time critical round, it responds to the challenge c_i with $r_i = (a_i \wedge \overline{c_i}) \vee ((a_i \oplus \mathrm{NP}_i \oplus m_i) \wedge c_i)$. Finally, during a final slow phase, the prover sends $\mathscr{T} = \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R})$, where C and R are respectively the concatenation of the challenges and responses.

$\mathsf{Verifier}(\cdot)$ After receiving an initial message e, it computes $(\mathrm{NP}, \sigma_p) = \mathsf{PKE.dec}_{x_\mathsf{V}}(e)$, and sends two random n bit values NV and m. It computes $a = \mathsf{H}(\mathrm{NP}, \mathrm{NV})$. Afterwards, during the n time-critical phases, he generates a random bit c_i from a uniform distribution, starts his clock, sends c_i, gets back r_i, stops his clock and stores the corresponding time Δt_i. It then receives $\mathscr{T}$. Finally, it verifies that (1) $\Delta t_i \leq t_{\mathsf{max}}$, (2) $r_i = (a_i \wedge \overline{c_i}) \vee ((a_i \oplus \mathrm{NP}_i \oplus m_i) \wedge c_i)$ for all $i \leq n$, and (3) $\mathscr{T} = \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R})$. If these conditions hold, he sends an accepting bit $\mathsf{Out_V} = 1$, while otherwise he sends $\mathsf{Out_V} = 0$.

Game G1. The oracle $\mathsf{Prover}(\cdot)$ uses different NP values for each of the q_p calls. $\mathscr{A}$ loses an advantage of at most $\frac{q_\mathsf{p}^2}{2^n}$. This follows from the binomial expansion of the probability that a given value has been selected zero or once and the union bound. Thus, $|Pr[\mathrm{G1}] - Pr[\mathrm{G0}]| \leq \frac{q_\mathsf{p}^2}{2^n}$.

Game G2. The oracle $\mathsf{Verifier}(\cdot)$ uses different NV values for each of the q_v calls. Thus, as for the previous transition, we have $|Pr[\mathrm{G2}] - Pr[\mathrm{G1}]| \leq \frac{q_\mathsf{v}^2}{2^n}$.

Game G3. $\mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}, \mathrm{R}) \neq \mathsf{H}(\mathrm{NP}, \mathrm{NV}, m, \mathrm{C}', \mathrm{R})$ if $\mathrm{C}' \neq \mathrm{C}$. The probability for a collision on the hashes of different messages is, in the random oracle model, bounded by $\frac{q_{\mathsf{H}(\cdot)}^2}{2^n}$ (where $q_{\mathsf{H}(\cdot)}$ is the polynomial number of evaluations of the hash function). Hence, $|Pr[\mathrm{G3}] - Pr[\mathrm{G2}]| \leq \frac{q_{\mathsf{H}(\cdot)}^2}{2^n}$.

Game G4. The oracle $\mathsf{Verifier}(\cdot)$ aborts if $\mathsf{G.ver}_{\mathsf{gpk}}(\sigma_p, \mathrm{NP}, \mathsf{RL}) = 1$, no tuple of WL contains σ_p and $\mathsf{G.ope}_{\mathsf{msk}}(\sigma_p, \mathrm{NP}, \mathsf{UL}, \mathsf{gpk}) \not\in \mathsf{CU}$. This happens if $\mathscr{A}$ produces a fresh signature σ_p on a value NP. Since G-SIG is a traceable dynamic group signature scheme, $|Pr[\mathrm{G4}] - Pr[\mathrm{G3}]| \leq \mathsf{Adv}^{Trace}_{\mathsf{G\text{-}SIG}}(\lambda)$.

Game G5. This step is simply a preparatory transition for the next game. Instead of only NP, two value NP_0 and NP_1 are drawn at the beginning of the protocol. The message e is computed using NP_0, and the protocol is also run with NP_0: NP_1 is never used. However, the backdoor also works for NP_1: if it is queried with (e, NP_1'), where NP_1' is a bitstring close to NP_1 (according to the backdoor's threshold), then the backdoor returns NP_1. G5 behaves exactly as G4, except in the case of the failure event that $\mathscr{A}$ sends a string close to NP_1 to the backdoor, which occurs with the negligible (by hypothesis) probability $p_{back}(n,t)$. Hence, $|Pr[\mathrm{G5}] - Pr[\mathrm{G4}]| \leq p_{back}(n,t)$.

Game G6. The oracle $\mathsf{Prover}(\cdot)$ encrypts NP_0 in his initial message e, and NP_1 used in the rest of the protocol *by both parties.*

The oracles are modified as follows:

- $\mathsf{Prover}(i)$ sets $\mathrm{NP}_0, \mathrm{NP}_1 \xleftarrow{\$} \{0,1\}^{2\cdot n}$, $\sigma_{p_0} = \mathsf{G.sig}_{\mathsf{psk}}(\mathrm{NP}_0)$, $\sigma_{p_1} = \mathsf{G.sig}_{\mathsf{psk}}(\mathrm{NP}_1)$, and computes the message e as $\mathsf{PKE.enc}_{pk_\mathsf{V}}(\mathrm{NP}_0, \sigma_{p_0})$. It adds $(e, \mathrm{NP}_1, \sigma_{p_1})$ to a list WL. It then sends e, but uses NP_1 to compute a, the responses r_i and τ.

- $\mathsf{Verifier}(\cdot)$ acts as in G4, except for the NP computation:
  - If $(e, \mathrm{NP}_1, \cdot) \in \mathsf{WL}$, it uses NP_1.
  - If $(e, \cdot, \cdot) \notin \mathsf{WL}$, it computes $(\mathrm{NP}^*, \sigma_p^*) = \mathsf{PKE.dec}_{x_v}(e)$ and uses NP^*.

Thus, $\mathscr{A}$ loses the possibility of recovering the value of NP from the ciphertext e. However, this advantage is negligible since the PKE encryption scheme is IND-CCA2 secure.

Let us assume that there exists an efficient adversary $\mathscr{A}$ for the game G6. Thus, an efficient distinguisher $\mathscr{B}$ can be defined for the IND-CCA2 experiment using $\mathscr{A}$.

Initialisation $\mathscr{B}$ sets up the environment (except the PKE setting). He creates two user lists UL and RL, and a dynamic group signature scheme setup with $\mathsf{G.gen}(1^\lambda)$, obtaining a couple $(\mathsf{gpk}, \mathsf{msk})$. He adds to UL n_p provers using $\mathsf{DB.join}_\mathsf{msk}(i, \mathsf{UL})$.

Simulation $\mathscr{B}$ simulates a SPADE environment for $\mathscr{A}$ with the DB-oracles of G5, with the following modifications to the prover and verifier oracles. When the oracle $\mathsf{Prover}(\cdot)$ is called, it sets $e = \mathsf{LREnc}_\mathsf{b}^{\mathsf{pk_v}}(\mathsf{NP_0}||\sigma_{\mathsf{p0}}, \mathsf{NP_1}||\sigma_{\mathsf{p1}})$, adds $(e, \mathrm{NP}_1, \sigma_{p_1})$ to WL and uses NP_1 during the protocol. The oracle $\mathsf{Verifier}(\cdot)$ acts as in G5 if it receives e such that $(e, \mathrm{NP}_1, \sigma_{p_1}) \in \mathsf{WL}$. Otherwise, it decrypts e with the provided decryption oracle. Finally, $\mathscr{B}$ returns the result of the authentication $\mathrm{O}ut\mathrm{V}$ to the challenger, which is 1 if the verifier accepts, and 0 otherwise. Additionally, the backdoor needs to be simulated: if it is queried with strings close enough (according to the backdoor threshold) to either NP_0 or NP_1, it returns the corresponding bitstring (NP_0 or NP_1), as in G5.

If $b = 1$, $e = \mathsf{PKE.enc}_{pk_\mathrm{V}}(\mathrm{NP}_1, \sigma_{p_1})$, $(e, \mathrm{NP}_1, \sigma_{p_1}) \in \mathsf{WL}$ and both parties use NP_1 internally. This games corresponds to the game G5. Otherwise, $e = \mathsf{PKE.enc}_{pk_\mathrm{V}}(\mathrm{NP}_0, \sigma_{p_0})$ while the NP_1 is used during the protocol. This simulates the game G6. The simulation works because the other parts of the protocol do not leak anything about NP_1: a leaks nothing about the input to $\mathsf{H}(\cdot)$ in the random oracle model, the same goes for τ, and the change made in game G3 ensures that $\mathscr{A}$ cannot learn anything about NP_1 from tampering with the challenges.

Let $\mathscr{B}_0$ denote the event that the distinguisher outputs 0, and $\mathscr{B}_1$ the event that it outputs 1. Thus, $\Pr[\mathscr{B}_1 | b = 1] = \Pr[\mathrm{G5}]$ and $\Pr[\mathscr{B}_0 | b = 0] = 1 - \Pr[\mathrm{G6}]$, since $\mathscr{B}$ returns the result of the authentication to the distinguisher. The winning probability of $\mathscr{B}$ is then $\Pr[\mathscr{B}_1 \wedge b = 1] + \Pr[\mathscr{B}_0 \wedge b = 0]$, which is equal to $\Pr[\mathrm{G5}] \cdot \frac{1}{2} + (1 - \Pr[\mathrm{G6}]) \cdot \frac{1}{2} = \frac{1}{2} + \frac{\Pr[\mathrm{G5}] - \Pr[\mathrm{G6}]}{2}$. Now assume that $\Pr[\mathrm{G5}] - \Pr[\mathrm{G6}]$ is non negligible. Then $\mathscr{B}$ has a non-negligible advantage on the extended IND-CCA2 game. However, the advantage of $\mathscr{B}$ cannot be more than q_p times the advantage on the original IND-CCA2 experiment. Thus, $|\Pr[\mathrm{G6}] - \Pr[\mathrm{G5}]| \leq q_\mathsf{p} \cdot \mathsf{Adv}_{\mathsf{PKE}}^{\mathrm{IND-CCA2}}(\lambda)$, which is negligible by hypothesis.

The final game. The final step is to prove that the probability to win G6 is negligible. First note that if $p_{back}(n, t)$ is negligible, the adversary has a negligible probability to extract either NP_0 or NP_1 by using the backdoor. Since $\mathscr{A}$ cannot purely relay messages without tainting the session, for each round j:

- If $\mathscr{A}$ sends c'_j to $\mathsf{Prover}(\cdot)$ before receiving c_j from $\mathsf{Verifier}(\cdot)$, he would be wrong with probability $\frac{1}{2}$.
- If $\mathscr{A}$ sends c'_j to $\mathsf{Prover}(\cdot)$ after receiving c_j from $\mathsf{Verifier}(\cdot)$, he must send r'_j before receiving r_j since pure relay is not allowed. He would be wrong with probability $\frac{1}{2}$.

In the first case, a wrongly guessed challenge invalidates the final message τ. Hence, $\mathscr{A}$ must recompute this message to succeed. He cannot do so without guessing NP, or obtaining it from the backdoor, and his success probability is no more than $max(p_{back}(n, t), \left(\frac{1}{2}\right)^n)$ (which both are negligible). In the second case, the authentication fails if $\mathscr{A}$ sends a single wrong response, so his success probability is no more than $\left(\frac{1}{2}\right)^n$.

From this sequence of games, it follows that the success probability of $\mathscr{A}$ in the MF game is negligible. □

Terrorist Fraud Resistance

SPADE is terrorist fraud resistant. For $\mathscr{A}$ to have a non negligible probability of authenticating with the help of the prover, he needs to be able to respond to both challenges for a large number of rounds. Since knowing both responses for one round allows to recover one bit of NP, intuitively, $\mathscr{A}$ should recover most of NP. He can obtain the missing bits through the backdoor, and then, send the initial message e in a new session and use NP to authenticate.

Theorem 3. *If the challenges are drawn randomly from a uniform distribution, and the backdoor threshold t is set to $\alpha \cdot n$, where α is a constant, then* SPADE *is* GameTF*-resistant.*

Remark, if there is an adversary $\mathscr{A}$ that knows all the bits of NP, then there exists an adversary $\mathscr{B}$ to which $\mathscr{A}$ is helpful, and who wins with probability 1 (*i.e.*, in this case, SPADE is GameTF-resistant).

Similarly, SPADE is trivially GameTF-resistant (in the sense that even an empty help from the prover is enough) if insecure schemes are used. If an adversary can break the encryption scheme, he can find NP encrypted in e and he can use it to authenticate himself. In addition, if an adversary can forge a signature, he can choose the nonce NP himself, sign it and use it to authenticate.

Thus, a malicious prover cannot perform any efficient TF attacks while preserving his secret key.

Proof. Let us assume there is a polynomial-time $(t, q_{\mathsf{V}}, q_{\mathsf{P}}, q_{\mathsf{obs}})$-adversary $\mathscr{A}$ that can win the TF game with a non-negligible probability with the help of his malicious prover. Then, we can construct an adversary $\mathscr{B}$ that can always perform a mafia fraud later using $\mathscr{A}$'s view, contradicting the mF resistance of SPADE. *A fortiori*, $\mathscr{A}$ can also do so.

After a winning session by $\mathscr{A}$ helped by the prover, either of the two following things can be extracted from its internal state:

- Two n-bit strings $\mathbf{r}^0$ and $\mathbf{r}^1$ representing respectively the responses to the 0-challenges and the 1-challenges.

- An algorithm A to generate these strings.

If A is *stateless* (*i.e.*, the response to a challenge does not depend on the previous ones), these two are equivalent. For simplicity, the former case is used. Hence, $\mathscr{A}$ has the strings $(\mathbf{r}^0, \mathbf{r}^1)$, representing his internal $\mathsf{view}_{\mathscr{A}}(\mathsf{sid})$. Depending on the strategy chosen by the prover, there are several possibilities:

$$\begin{array}{lrclrcl}
\text{Strategy 1}: & r_i^0 = & a_i & \text{and} & r_i^1 = & a_i \oplus (\mathrm{N_P}{}_i \oplus m_i) & \\
\text{Strategy 2}: & r_i^0 = & a_i & \text{and} & r_i^1 = & \perp \quad \text{or} & \\
 & r_i^0 = & \perp & \text{and} & r_i^1 = & a_i \oplus (\mathrm{N_P}{}_i \oplus m_i) & \\
\text{Strategy 3}: & r_i^0 = & \perp & \text{and} & r_i^1 = & \perp & \\
\text{Strategy 4}: & r_i^0 = & \overline{a_i} & \text{and} & r_i^1 = & \perp \quad \text{or} & \\
 & r_i^0 = & \perp & \text{and} & r_i^1 = & \overline{a_i \oplus (\mathrm{N_P}{}_i \oplus m_i)} & \\
\text{Strategy 5}: & r_i^0 = & \overline{a_i} & \text{and} & r_i^1 = & a_i \oplus (\mathrm{N_P}{}_i \oplus m_i) & \text{or} \\
 & r_i^0 = & a_i & \text{and} & r_i^1 = & \overline{a_i \oplus (\mathrm{N_P}{}_i \oplus m_i)} & \\
\text{Strategy 6}: & r_i^0 = & \overline{a_i} & \text{and} & r_i^1 = & \overline{a_i \oplus (\mathrm{N_P}{}_i \oplus m_i)} &
\end{array}$$

Strategies 4 to 6 correspond to the case where the prover deliberately introduces noise in the responses he gives to $\mathscr{A}$.

In the random oracle model, and under the assumptions that (1) the same values of NP or NV are never chosen twice by a cautious prover and an honest verifier, the values a_i can be seen as random. Thus, strategies 2 to 6 cannot leak any information on NP.

Les us now analyse the probability for the event w_i, that $\mathscr{A}$ to correctly respond at round i for each of these strategies. In what follows, c^{good}, c^{bad}, and c^{none} respectively denote the event that the verifier sends a challenge to which $\mathscr{A}$ knows a correct response, an incorrect one, or no response.

Strategy 1: $\mathscr{A}$ know both responses, so he passes the round with probability 1.

Strategy 2: $\mathscr{A}$ succeeds if the challenge of the verifier corresponds to the response he knows, or if he guesses the response to the second challenge. The corresponding probability is computed as $Pr[c^{good}] \cdot Pr[w_i|c^{good}] + Pr[c^{none}] \cdot Pr[w_i|c^{none}]$. This is equal to $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$.

Strategy 3: $\mathscr{A}$ succeeds if he guesses the response bit, so his success probability is $\frac{1}{2}$.

Strategy 4: $\mathscr{A}$ succeeds only if c^{none} and he guesses r_i correctly. The corresponding success probability, $Pr[c^{none}] \cdot Pr[w_i|c^{bad}]$ is $\frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$.

Strategy 5: $\mathscr{A}$ succeeds if c^{good} occurs, so $\Pr[w_i] = \frac{1}{2}$.

Strategy 6: $\mathscr{A}$ cannot succeed: $Pr[w] = 0$.

If the prover uses strategy 1 for all rounds, then $\mathscr{A}$ recovers NP, and the fraud fails. Hence, he needs to use strategies 2, 3, 4, 5 or 6 for a given number r of rounds. Strategy 6 can only be used if the protocol is set up to tolerate communication errors: in this case, the authentication will be accepted even if a number n_{err} of responses are false. However, our analysis focuses on the noiseless scenario, even though it could be generalised to include noise, so we leave strategy 6 out.

Assume that a malicious prover P^* provides to his accomplice $\mathscr{A}$ two strings (r^0, r^1) such that $\mathscr{A}$ does not know the two correct responses for r rounds. The success probability of $\mathscr{A}$ is at most $\frac{3}{4}$ for each of these rounds, so that his final success probability is at most $1^{n-r} \cdot \left(\frac{3}{4}\right)^r$

Let us suppose that the authentication of $\mathscr{A}$ succeeded with a non-negligible probability $p_{\mathscr{A}}$. Consider the adversary-verifier session sid^* for which $\mathscr{A}$ has fooled V. This has happened with probability at least $\frac{p_{\mathscr{A}}}{q_{\mathsf{V}}}$. In such a case, $\mathscr{A}$ has successfully guessed the missing answers, which have been requested. Since this happened, $(\frac{3}{4})^r$ should be greater than the non-negligible $\frac{p_{\mathscr{A}}}{q_{\mathsf{V}}}$. Hence,

$$\exists c, \forall n_c, \exists n > n_c, \left[\left(\frac{3}{4}\right)^r > \frac{n^{-c}}{q_{\mathsf{V}}} > n^{-c'} \right].$$

since $q_{\mathsf{V}} \in n^{O(1)}$. Thus, r should be in $O(\log n)$.

If an adversary $\mathscr{B}$ gets the internal $\mathsf{view}_{\mathscr{A}}(\mathsf{sid}^*)$ and has eavesdropped to all the communications involving P, $\mathscr{A}$, and V, he would get e and N'_{P} such that $\mathrm{HD}(\mathrm{N}_{\mathrm{P}}, \mathrm{N}'_{\mathrm{P}}) \in O(\log n)$, where HD denotes the Hamming Distance. Thus, $\mathscr{B}$ (as well as $\mathscr{A}$ himself) would be able to retrieve N_{P} directly through the backdoor of V and eventually authenticate on behalf of P with probability one.

□

Anonymity

SPADE is private and prover anonymous. These properties are due to the dynamic group signature scheme: the only place in which a long term value linked to the prover is used in the protocol is to compute this signature. Hence, if the verifier or an eavesdropper can link two sessions of the prover, they break the security of the group signature.

Theorem 4. *If* G-SIG *is an anonymous dynamic group signature scheme, then* SPADE *is prover anonymous.*

Proof. Assume that there is a polynomial-time adversary $\mathscr{A}$ having a non-negligible advantage $\mathsf{Adv}^{\mathrm{PA}}_{\mathscr{A},\mathrm{SPADE}}(\lambda)$ on a challenger in $\mathsf{Exp}^{\mathrm{PA}}_{\mathscr{A},\mathrm{SPADE}}(\lambda)$. $\mathscr{A}$ can be used to build an adversary $\mathscr{B}$ against the anonymity experiment on the signature scheme $\mathsf{Exp}^{\mathsf{Anon}}_{\mathscr{B},\mathsf{G\text{-}SIG}}(\lambda)$, such that $\mathsf{Adv}^{\mathsf{Anon}}_{\mathscr{B},\mathsf{G\text{-}SIG}}(\lambda) \geq \mathsf{Adv}^{\mathrm{PA}}_{\mathscr{A},\mathrm{SPADE}}(\lambda)$, contradicting the assumption that G-SIG is anonymous.

Initially, the challenger in $\mathsf{Exp}^{\mathsf{Anon}}_{\mathscr{B},\mathsf{G\text{-}SIG}}(\lambda)$ sends the key gpk and the revoked list RL to $\mathscr{B}$, which uses it to generate the experiment $\mathsf{Exp}^{\mathrm{PA}}_{\mathscr{A},\mathrm{SPADE}}(\lambda)$ for $\mathscr{A}$.

Then, $\mathscr{B}$ creates the empty list CU. Having access to G-SIG-oracles from his challenger, $\mathscr{B}$ can simulate the DB-oracles for $\mathscr{A}$ as follows:

$\overline{\mathsf{DB}}.\mathsf{Join}^h(\cdot)$: $\mathscr{A}$ creates P_{ID}. $\mathscr{B}$ relays it to $\overline{\mathsf{G}}.\mathsf{Join}^h(\cdot)$ and adds P_i to UL.

$\overline{\mathsf{DB}}.\mathsf{Join}^c(\cdot)$: $\mathscr{A}$ creates P_{ID}. $\mathscr{B}$ relays it to $\overline{\mathsf{G}}.\mathsf{Join}^c(\cdot)$, obtains the signing key $\mathsf{ssk}_{\mathsf{ID}}$,and adds P_{ID} to UL and CU. $\mathscr{A}$ gets $\mathsf{ssk}_{\mathsf{ID}}$.

$\overline{\mathsf{DB}}.\mathsf{Revoke}(\cdot)$: $\mathscr{A}$ revokes P_{ID}. $\mathscr{B}$ relays it to $\overline{\mathsf{G}}.\mathsf{Revoke}(\cdot)$, which updates RL and sends it to $\mathscr{B}$. He relays it to $\mathscr{A}$.

$\overline{\mathsf{DB}}.\mathsf{Corrupt}(\cdot)$: $\mathscr{A}$ corrupts P_{ID}. $\mathscr{B}$ relays it to $\overline{\mathsf{G}}.\mathsf{Corrupt}(\cdot)$ and gets $\mathsf{ssk}_{\mathsf{ID}}$. $\mathscr{B}$ adds P_{ID} to CU and returns $\mathsf{ssk}_{\mathsf{ID}}$ to $\mathscr{A}$.

$\overline{\mathsf{DB}}.\mathsf{Prover}(\cdot)$: $\mathscr{B}$ simulates P_{ID} as follows: $\mathscr{B}$ picks $\mathrm{NP} \xleftarrow{\$} \{0,1\}^n$, sends $(\mathsf{ID}, \mathrm{NP})$ to $\overline{\mathsf{G}}.\mathsf{Sign}(\mathsf{ID}, \mathrm{NP})$ and gets back σ_p from $\mathscr{A}$. $\mathscr{B}$ then computes $e = \mathsf{PKE.enc}_{pk_{\mathsf{V}}}(\mathrm{NP}, \sigma_p)$, sends e to $\mathscr{A}$, receives (m, NV), and runs the rest of the protocol normally.

Then, $\mathscr{A}$ picks two identities ID_0 and ID_1 and sends them to $\mathscr{B}$. If $\mathsf{ID}_0, \mathsf{ID}_1 \not\in \mathsf{CU}$, $\mathscr{B}$ sends $(\mathsf{ID}_0, \mathsf{ID}_1)$ to the challenger. In this phase, $\mathscr{B}$ simulates $\overline{\mathsf{DB}}.\mathsf{Prover}(\cdot)$ as follows. During the initialisation phase, $\mathscr{B}$ picks $\mathrm{NP} \xleftarrow{\$} \{0,1\}^n$ sends it to $\overline{\mathsf{G}}.\mathsf{Sign}_b(\cdot,\cdot)$ and receives σ_p. He then computes $e = \mathsf{PKE.enc}_{pk_{\mathsf{V}}}(\mathrm{NP}, \sigma_p)$ and sends e to $\mathscr{A}$, and runs the rest of the protocol normally.

Finally, during the guessing phase, $\mathscr{A}$ returns b' and $\mathscr{B}$ forwards b' to the challenger of the anonymity experiment on the group signature.

The experiment is perfectly simulated for $\mathscr{A}$, and consequently, $\mathscr{B}$ and $\mathscr{A}$ have the same probability of winning their experiment, and $\mathsf{Adv}^{\mathsf{Anon}}_{\mathscr{B},\mathsf{G\text{-}SIG}}(\lambda) = \mathsf{Adv}^{\mathrm{PA}}_{\mathscr{A},\mathrm{SPADE}}(\lambda)$, contradicting the assumption on G-SIG. □

4.5.3 The presence of the backdoor

The objective of the backdoor in the verifier is to discourage any prover from helping help potential accomplices. Remark that this mechanism is stateless for the verifier, as he simply has to decrypt the initial message of the protocol to retrieve the information needed to impersonate a prover.

The probabilities for key recovery or TF attacks depends on the proximity threshold t, which is a public parameter of the protocol. There is clearly a trade-off between these two probabilities, as one increases and the other one decreases in function of t. In this section, we analyse the success probability of attacks depending on t. More precisely, we bound the probability for an adversary to recover NP through the backdoor, depending on whether this adversary was helped by the prover or not. While we consider a noiseless environment in our security analysis, noise tolerance could easily be included by allowing n_{err} errors during the time-critical phases, where $n_{err} = \beta \cdot n$ for some constant β. In this case, to avoid the terrorist frauds described in 3.3.4, t should be greater than n_{err}.

The backdoor implemented in the verifier can be queried with pairs e, NP', such that $(\mathrm{NP}, \sigma_p) = \mathsf{PKE.dec}_{x_{\mathsf{V}}}(e)$. If NP' is *close* to NP, then the verifier returns NP; otherwise, it returns $\perp$.

Mafia fraud adversary

A mafia fraud adversary $\mathscr{A}$ can obtain the session key NP associated to a message e by using the backdoor if he guesses a bitstring that is t-close to NP. Since the number of potential queries is 2^n and the number of strings at Hamming distance at most t of NP is simply $\sum_{k=0}^{t}(\binom{n}{k})$, the probability that the i^{th} random query is successful (let $\mathbb{Q}_i$ denote such an event) is

$$\mathrm{P}_i = \Pr[\mathbb{Q}_i | n, t] = \frac{1}{2^n - i} \cdot \sum_{k=0}^{t} \binom{n}{k}.$$

The probability to have a successful query among n_{back} (which is polynomial in n) is upper bounded by $\sum_{i=1}^{n_{back}} p_i$, due to the union bound, so we have:

Theorem 5. *The probability $p_{back}(n,t)$ for an adversary performing n_{back} queries, to recover the value* NP *related to an initial message e solely by querying the backdoor is upper bounded by*

$$n_{back} \cdot \frac{1}{2^{-n} - i} \cdot \sum_{k=0}^{t} \binom{n}{k}$$

Terrorist fraud adversary

As described in Section 4.5.2, if the protocol does not include noise tolerance, then, for each round, the malicious prover can either use strategy 1 (giving both responses), strategy 2 (giving only one response), strategy 3 (giving no response at all), strategy 4 (giving only one response which is incorrect) and strategy 5 (giving both one correct and one incorrect response). Note that the dishonest prover is not bound to using only one of these strategies: he can use a different strategy for different rounds. Let S_k denote the fact that a round corresponds to strategy k, and n_{S_k} denote the number of rounds corresponding to strategy k. Since the goal of P^* is to prevent $\mathscr{A}$ from impersonating him, we reasonably assume that he gives as little information to $\mathscr{A}$ as possible. In particular, he does not tell him the values n_{S_k}. We are interested in the case where the authentication of $\mathscr{A}$ was accepted, which implies that all his responses were correct, in the noiseless scenario. Hence, $\mathscr{A}$ knows at least one correct response for each round, and he knows that the response he sent is correct. For the second response, either (1) he does not know it, or (2) he knows a response, whether it is correct or incorrect. If the prover uses strategy 1 for more than $n-t$ rounds rounds, then $\mathscr{A}$ can recover NP directly through the backdoor, so that P^* needs to use the other strategies for $x > t$ rounds. Let $z = x - t$ be the number of bits for which strategy 1 is not used, which corresponds to the number of bits left to guess for $\mathscr{A}$. In case (1), $\mathscr{A}$ needs to guess the z missing bits, which succeeds with probability $p1 = 2^{-z}$. In case (2), $\mathscr{A}$ needs to find which responses are wrong to flip them and obtain a string that is t-bit close from NP. To do so, he can try to first flip one bit, then 2, and all the way to x. Using this strategy, he needs to try

$$\sum_{i=1}^{x} \binom{n}{x}$$

cases in the worst case, and succeeds with probability $p2 = \frac{1}{\sum_{i=1}^{x}\binom{n}{x}}$. Sine $p1 > p2$, we can say that $\mathscr{A}$ has, in the worst case, probability $p2$ of recovering NP through the backdoor, at each try, where $p2$ depends on x.

Theorem 6. *Let $z = n - n_1 - t$, where n_1 is the number of rounds for which strategy* 1 *is used, and t is the backdoor's threshold. Let n_{back} be the number of queries made to the backdoor. The probability for a terrorist fraud accomplice $\mathscr{A}$ to obtain* NP *through the backdoor is lower bounded by $\frac{1}{\sum_{i=1}^{x}\binom{n}{x}}$, and upper bounded by $n_{back} \cdot 2^{-x}$.*

4.5.4 PRF Programming Attacks

The security proofs are done in the random oracle model, using a hash function on the key concatenated with the input to model a PRF. If a real PRF was used instead, the protocol could be vulnerable to a mafia fraud by key recovery: there exists a function f, such that f is a PRF, and $\mathscr{A}$ can extract NP after interacting with a prover if f is used in SPADE. Consider the following function f: it uses a PRF g for most inputs, but returns non random outputs for some input values.

For inputs of length n:

$$f_{NP}(NV) = g_{NP}(NV)$$

For inputs of length $4 \cdot n$:

$$\begin{aligned} f_{NP}(NV, m, C, R) = & NP \text{ if } R = (g_{NP}(NV) \oplus NP \oplus m) \\ & g_{NP}(NV, m, C, R) \text{ otherwise} \end{aligned}$$

If g is a PRF, then f is a PRF: an adversary trying to distinguish f from a random function does not have access to its key NP, and can thus not query the values that trigger the non-random behaviour except with negligible probability: he cannot obtain a couple (X, R) such that $\mathrm{R} = \mathrm{X} \oplus \mathrm{NP}$ without guessing NP.

However, in SPADE, the output of the PRF is XORed with its key, which allows $\mathscr{A}$ to obtain values that he would not be able to obtain in the PRF security game.

For this reason, we chose to use a construction based on a hash function instead of the PRF. However, other options would have been possible, such as using the circular-keying security notion for PRFs defined in [Boureanu et al., 2015].

4.6 TREAD

In this section, we present our protocol TREAD.

4.6.1 Protocol Description

TREAD (Figure 4.3) is a generic construction, parameterised by an encryption scheme and a signature scheme. Depending on which ones are chosen, the properties of the resulting distance bounding protocol differ. In TREAD, as in SPADE, the value that is leaked to a Terrorist Fraud accomplice is not a long term secret, but a temporary, prover defined one, which can be replayed in a further session. TREAD relies on specific design choices. The first one is to eliminate the need for a PRF. It is now well known that PRF programming attacks are possible, and that in many cases, the prover can strongly influence the result of the PRF. Hence, to err on the safe side, we directly let the prover pick the two response vectors a and b, which are used during the distance bounding phase. To prevent distance frauds, in which the prover would pick $a = b$, a random bitstring m must be XORed with b, so that on average half the bits of the resulting response vector $b \oplus m$ are truly random and independent of a. Another strong design choice is to remove the final signature in the verification phase. The resulting protocols are thus simpler. If a prover helps an accomplice during a terrorist fraud attempt, then this accomplice learns large parts of the response vectors a and b, and can use this knowledge to pass the protocol again on his own.

In TREAD, the prover (resp. verifier) each hold two keys: an encryption (resp. decryption) key, and a signature (resp. verification) key. In a symmetric setting, where the encryption scheme is symmetric and the signature is a MAC, the encryption and decryption keys are identical, and the signature and verification keys are identical too. Public key instances are possible as well.

In TREAD, provers have two identities: a public one, which is sent in clear, and a private identity, which is sent encrypted for the verifier. This allows for finer grained management: for instance, the prover's public identity could be a group identity, while the private one would be his own. Only one of these identities needs to be non null. The protocol works as follows. First, the prover picks two random n-bit values a and b, and generates a signature on them using the signature key. Then, he encrypts these values, the signature, and his private identity with the encryption key, and sends the resulting message, along with his public identity, to the verifier. The verifier responds with a random bitstring m, and the distance bounding phase begins. For n rounds, the verifier sends a random bit c_i, and the prover responds with either a_i if $c_i = 0$, or $b_i \oplus m_i$ otherwise. After this phase, the verifier accepts the authentication of the prover if the signature in the initial massage corresponds to the identities of the prover and is valid, and the responses were correct and sent in time.

Definition 31 (TREAD)**.** *The construction* TREAD *is composed of five algorithms and parameterised by an encryption scheme* E*, a message authentication scheme* S*, as well as a definition for* $\mathsf{idprv}(\cdot)$ *and* $\mathsf{idpub}(\cdot)$.

$\mathsf{DB.gen}(1^\lambda)$ *is the algorithm run by an honest party, setting up the encryption scheme* E *and the signature scheme* S *for a security parameter* λ*. It sets the number of the time-critical phases* n*, which depends of* λ.

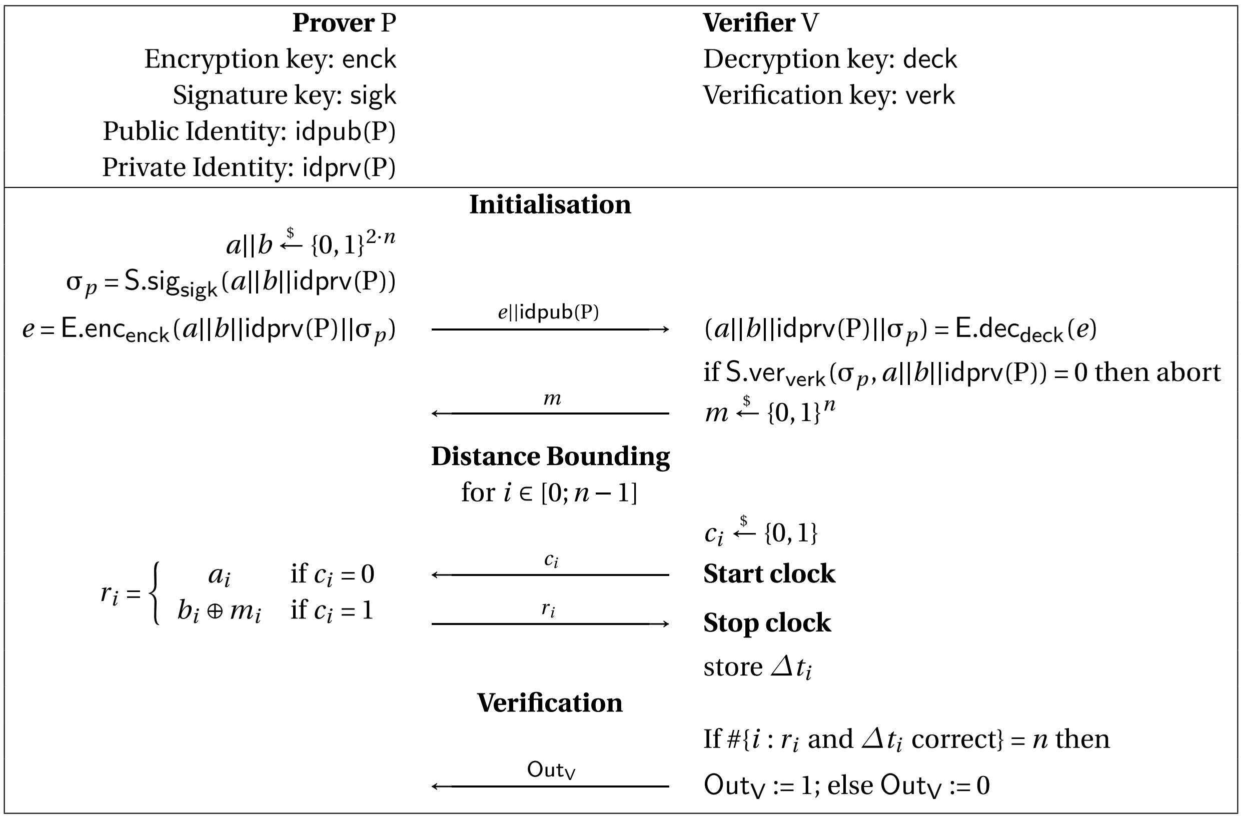
Prover P
Encryption key: enck
Signature key: sigk
Public Identity: idpub(P)
Private Identity: idprv(P)
Verifier V
Decryption key: deck
Verification key: verk
Initialisation
$a||b \xleftarrow{\$} \{0,1\}^{2\cdot n}$
$\sigma_p = \mathsf{S.sig}_{\mathsf{sigk}}(a||b||\mathsf{idprv}(P))$
$e = \mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv}(P)||\sigma_p)$
$e||\mathsf{idpub}(P)$
$(a||b||\mathsf{idprv}(P)||\sigma_p) = \mathsf{E.dec}_{\mathsf{deck}}(e)$
if $\mathsf{S.ver}_{\mathsf{verk}}(\sigma_p, a||b||\mathsf{idprv}(P)) = 0$ then abort
$m \xleftarrow{\$} \{0,1\}^n$
m
Distance Bounding
for $i \in [0; n-1]$
$c_i \xleftarrow{\$} \{0,1\}$
c_i
Start clock
$r_i = \begin{cases} a_i & \text{if } c_i = 0 \\ b_i \oplus m_i & \text{if } c_i = 1 \end{cases}$
r_i
Stop clock
store Δt_i
Verification
If $\#\{i : r_i$ and Δt_i correct$\} = n$ then
$\mathsf{Out_V}$
$\mathsf{Out_V} := 1$; else $\mathsf{Out_V} := 0$

Figure 4.3: TREAD [Avoine et al., 2017], a generic and provably secure DB construction built from an IND-CCA2-secure encryption scheme E and an EUF-CMA-secure signature scheme (or a traceable group signature) S.

$\mathsf{DB.prover}(\mathbb{K}_{\mathsf{ID}})$ *is the algorithm executed by the prover described in Figure 4.3, in which* $\mathbb{K}_{\mathsf{ID}} = (\mathsf{enck}, \mathsf{sigk})$. *The prover starts by drawing a random value $a||b$ from the uniform distribution on $\{0,1\}^{2\cdot n}$. Then, he computes a signature σ_p on it with* $\mathsf{S.sig}_{\mathsf{sigk}}(a||b||\mathsf{idprv}(P))$. *Afterwards, he generates* $e = \mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv}(P)||\sigma_p)$ *and sends* $e||\mathsf{idpub}(P)$. *Finally, during the n time-critical phases, he receives a challenge bit c_i and responds with* $r_i = (a_i \wedge \overline{c_i}) \vee ((b_i \oplus m_i) \wedge c_i)$.

$\mathsf{DB.verifier}(\mathbb{K}_{\mathsf{V}}, \mathsf{UL}, \mathsf{RL})$ *is the algorithm executed by the verifier interacting with a prover, in which* $\mathbb{K}_{\mathsf{V}} = (\mathsf{deck}, \mathsf{verk})$. *He expects an initial message e and deciphers it as* $(a||b||\mathsf{idprv}(P)||\sigma_p) = \mathsf{E.dec}_{\mathsf{deck}}(e)$. *If σ_p is invalid (i.e.,* $\mathsf{S.ver}_{\mathsf{verk}}(\sigma_p, a||b||\mathsf{idprv}(P)) = 0$*), the verifier aborts. Otherwise, he picks a random bit string m from the uniform distribution on $\{0,1\}^n$ and sends it. Afterwards, during the n time-critical phases, he generates a random bit c_i from a uniform distribution, starts his clock, sends c_i, gets back r_i, stops his clock and stores the corresponding time Δt_i. Finally, he verifies that (1) $\Delta t_i \leq t_{\mathsf{max}}$ and (2) $r_i = (a_i \wedge \overline{c_i}) \vee ((b_i \oplus m_i) \wedge c_i)$, for all $i \leq n$. If these conditions hold, he sends an accepting bit* $\mathsf{Out_V} = 1$, *while otherwise he sends* $\mathsf{Out_V} = 0$.

$\mathsf{DB.join}(\mathsf{ID}, \mathsf{UL})$ *is the algorithm to register a new prover with identifier* ID *in the list* UL. *It returns the keys* (enck, deck) *for* E *and* (sigk, verk) *for* S. *Depending on the primitives* E *and* S, deck *and* verk *may be public or private, and can sometimes be equal respectively to* enck *and* sigk.

$\mathsf{DB.revoke}(\mathsf{ID}, \mathsf{UL}, \mathsf{RL})$ *is the algorithm to revoke a legitimate prover with identifier* ID *in* UL *and transfer him to the revoked user list* RL.

4.6.2 Protocols

Our instantiations go from a computationally efficient symmetric key protocol to a prover anonymous one.

Efficient symmetric-key scheme TREAD$_{sym}$

Computational efficiency is critical for the design of DB protocols as they are usually used in efficient construction, TREAD can be instantiated with an $\mathsf{IND\text{-}CCA2}$ symmetric-key encryption scheme SKE and an $\mathsf{EUF\text{-}CMA}$ message-authentication code scheme MAC. In this case, the public identity $\mathsf{idpub}(P)$ is the identity of the prover and the private identity $\mathsf{idprv}(P)$ is set to null, *i.e.*, the $\mathsf{idprv}(P)$ field is left empty. Since SKE and MAC are symmetric, we have $\mathsf{enck} = \mathsf{deck}$ and $\mathsf{sigk} = \mathsf{verk}$. Thus, the prover and the verifier have the same *symmetric* key $\mathsf{k} = (\mathsf{enck}, \mathsf{sigk})$. In this construction, the verifiers have access to a private list UL containing all the secret keys of legitimate provers. An authority adds the prover in the private list UL or in the revocation public list RL.

Prover privacy and public-key encryption TREAD$_{priv}$. In applications such as contactless payment schemes, shared secret keys are not recommended, since a corruption of the secrets of the verifier puts the user at risk. Thus, with the emergence of NFC-enabled smartphones, public-key DB protocols are crucial.

TREAD can be instantiated with an $\mathsf{IND\text{-}CCA2}$ public-key encryption PKE and an $\mathsf{EUF\text{-}CMA}$ digital signature scheme $\mathsf{S\text{-}SIG}$. In this case, the public identity $\mathsf{idpub}(P)$ is set to $null$, and the private identity $\mathsf{idprv}(P)$ is the identity of P (or his verification key). The keys enck and deck are respectively the public and the private keys of the verifier, and sigk and verk are the (private) signature key and the (public) verification key of the prover. With such a protocol, two sessions by the same user are not linkable for an *external* eavesdropper as the only information sent about the prover's identity is encrypted with the public-key of the verifier. However, verifiers have the power to link sessions. In this construction, the verifiers have access to a public list UL containing the public keys of legitimate provers. An authority adds the provers to the public list UL or the revocation public list RL.

Prover anonymity and group signature TREAD$_{ano}$. TREAD can also be used to provide full prover anonymity even with respect to a malicious verifier. As profiling users has become a common threat, it is crucial to develop anonymity-preserving DB protocols. Both prover anonymity and revocability can be achieved by instantiating TREAD with an $\mathsf{IND\text{-}CCA2}$ public-key encryption scheme PKE and a traceable and revocable dynamic group signature scheme $\mathsf{G\text{-}SIG}$. In this case, the public identity $\mathsf{idpub}(P)$ is set to $null$, and the private identity $\mathsf{idprv}(P)$ is set to the identity of the group $\mathsf{ID_G}$. Independent groups may coexist but prover anonymity with respect to the verifier is only guaranteed up to a prover's group. The keys enck and deck are respectively the public and private keys of the verifier, sigk is the prover's signing key and verk is the public group verification key. dynamic group signature schemes allow a user to anonymously sign on behalf of a group he belongs to. Hence, the verifier can check if the prover belongs to the claimed group, but cannot identify him precisely nor link his sessions. In this scenario, the join and revoke algorithms take their full meaning.

Let $(\mathsf{gpk}, \mathsf{msk})$ be the group/master key pair of the group signature scheme $\mathsf{G\text{-}SIG}$. Then,

$\mathsf{DB.join}(\mathsf{ID}, \mathsf{gpk}, \mathsf{UL})$ *returns a prover signing key* $\mathsf{sigk_{ID}}$ *for* P_{ID}. *This algorithm also outputs a value* $\mathsf{reg_{ID}}$ *and adds* P_{ID} *to* UL.

$\mathsf{DB.revoke}(\mathsf{ID}, \mathsf{gpk}, \mathsf{RL}, \mathsf{UL})$ *computes the revocation logs* $\mathsf{rev_{ID}}$ *for* P_{ID}, *using* $\mathsf{reg_{ID}}$ *and* msk, *and moves* P_{ID} *from* UL *to* RL.

4.6.3 Properties

We now prove the security property of TREAD, as well as the privacy of TREAD$_{priv}$ and TREAD$_{ano}$. In these proofs, in the game G0, the prover and verifier behave as defined in Definition 31.

Distance Hijacking

TREAD is distance hijacking resistant: the mask m picked by the verifier ensures that the responses to the challenges 0 and 1 are different for half the rounds on average, so that $\mathcal{A}$ cannot efficiently send a response in advance. Additionally, since the initial messages e are encrypted, $\mathcal{A}$ cannot extract the values $a||b$ picked by a honest prover, and exploit him to cheat.

Theorem 7. *If* E *is a* $\mathsf{IND\text{-}CCA2}$*-secure encryption scheme, and* S *is an* $\mathsf{EUF\text{-}CMA}$ *signature or MAC scheme, or a traceable group signature scheme, then* TREAD *is DH-resistant.*

Proof. First note that if $\mathcal{A}$ uses the oracle prompt for the initial message, *i.e.*, he lets an honest prover send it and then his authentication automatically fails, as $\mathsf{idpub}(\mathrm{P})$ and/or $\mathsf{idprv}(\mathrm{P})$ do not correspond to the identity of $\mathcal{A}$.

Hence, consider the case in which $\mathcal{A}$ initiated the protocol with a message e^* (associated with a^*, b^*). Let e (and $a||b$) denote the values picked by the nearby honest prover P. For each challenge c_i, either $\mathcal{A}$ uses Prompt to let P respond or he uses Commit to respond himself before receiving c_i. In Game G0, the prover and verifier follow the specification of the protocol:

$\mathsf{DB.prover}(\mathbb{K}_{\mathsf{ID}})$ The prover starts by drawing a random value $a||b$ from the uniform distribution on $\{0,1\}^{2\cdot n}$. Then, he computes a signature σ_p on it with $\mathsf{S.sig}_{\mathsf{sigk}}(a||b||\mathsf{idprv}(\mathrm{P}))$. Afterwards, he generates $e = \mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv}(\mathrm{P})||\sigma_p)$ and sends $e||\mathsf{idpub}(\mathrm{P})$. Finally, during the n time-critical phases, he receives a challenge bit c_i and responds with $r_i = (a_i \wedge \overline{c_i}) \vee ((b_i \oplus m_i) \wedge c_i)$.

$\mathsf{DB.verifier}(\mathbb{K}_{\mathsf{V}}, \mathsf{UL}, \mathsf{RL})$ The verifier waits for an initial message e and deciphers it as $(a||b||\mathsf{idprv}(\mathrm{P})||\sigma_p) = \mathsf{E.dec}_{\mathsf{deck}}(e)$. If σ_p is invalid (*i.e.*, $\mathsf{S.ver}_{\mathsf{verk}}(\sigma_p, a||b||\mathsf{idprv}(\mathrm{P})) = 0$), the verifier aborts. Otherwise, he picks a random bit string m from the uniform distribution on $\{0,1\}^n$ and sends it. Afterwards, during the n time-critical phases, he generates a random bit c_i from a uniform distribution, starts his clock, sends c_i, gets back r_i, stops his clock and stores the corresponding time Δt_i. Finally, he verifies that (1) $\Delta t_i \leq t_{\mathsf{max}}$ and (2) $r_i = (a_i \wedge \overline{c_i}) \vee ((b_i \oplus m_i) \wedge c_i)$, for all $i \leq n$. If these conditions hold, he sends an accepting bit $\mathsf{Out_V} = 1$, while otherwise he sends $\mathsf{Out_V} = 0$.

G1**:** *In this game, no value $a||b$ is outputted more than once by the prover oracle.*

In the i^{th} session, the probability to have a collision with any of the previous $i-1$ $a||b$ values is bounded by $\frac{i}{2^{2\cdot n}}$. If $\mathcal{A}$ runs q_{p} prover sessions, the probability of a collision for a given session is bounded by $\frac{q_{\mathsf{p}}}{2^{2\cdot n}}$. From the union bound, the probability that a collision occurs at least once is bounded by $\sum_{i=0}^{q_{\mathsf{p}}} \frac{q_{\mathsf{p}}}{2^{2\cdot n}}$, which is in turn bounded by $\frac{q_{\mathsf{p}}^2}{2^{2n}}$. Thus, $|\Pr[\mathrm{G1}] - \Pr[\mathrm{G0}]| \leq \frac{q_{\mathsf{p}}^2}{2^{2n}}$, which is negligible.

G2**:** *This game aborts if σ_p was not generated by the prover oracle, and* $\mathsf{S.ver}_{\mathsf{verk}}(\sigma_p, a||b) \neq 0$.

In this game, we rule out the possibility that $\mathcal{A}$ produces a valid signature on behalf of a honest prover, which is trivially forbidden by the EUF-CMA resistance of S (or its traceability if it is a group signature). The reduction simply consists in starting EUF-CMA experiments (one for each prover) with a challenger and using queries to the corresponding signing oracle to generate the signatures of a prover. Then, if $\mathcal{A}$ sends a valid signature on behalf of one of the provers, we can return it to the challenger and win the EUF-CMA experiment. Hence, we have $|Pr[\mathrm{G2}] - Pr[\mathrm{G1}]| \leq q_{\mathsf{p}} \cdot \mathsf{Adv}_{\mathsf{S}}^{\mathrm{F}orge}(1^\lambda)$ (where $\mathrm{F}orge$ is either the EUF-CMA or traceability experiment, depending on S), which is negligible by hypothesis.

G3**:** *In this game, e is replaced by the encryption of a random string (of equal length).*

This transition is based on indistinguishability, aiming at removing any leakage of $a||b$ from e by making $a||b$ only appear during the DB phase. We prove that the probability $\epsilon = |Pr[\mathsf{G3}] - Pr[\mathsf{G2}]|$ is negligible by building a distinguisher $\mathscr{B}$ such that the advantage of $\mathscr{B}$ against the IND-CCA2 experiment is polynomial in ϵ. Hence, if ϵ is non-negligible, we reach a contradiction. By assumption, the advantage of any adversary against the IND-CCA2 experiment on E is negligible.

To build $\mathscr{B}$, we replace $\mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv(P)}||\sigma_p)$ by a string given by the IND-CCA2 challenger. Using the adversary $\mathscr{A}$, the distinguisher $\mathscr{B}$ can be built as follows.

Prover simulation: $\mathscr{B}$ generates two messages: $m_0 = (\delta||\mathsf{idprv(P)}||\mathsf{S.sig}_{\mathsf{sigk}}(\delta||\mathsf{idprv(P)}))$ and $m_1 = (a||b||\ \mathsf{S.sig}_{\mathsf{sigk}}(a||b||\mathsf{idprv(P)}))$, in which $a||b$ and δ are random binary strings of length $2 \cdot n$. Then, he sends them to the challenger to obtain c_b, the encryption of m_b (depending on a random bit b picked by the challenger before the experiment). He also adds $(c_b, a||b)$ to the list WL. Afterwards, he sends c_b as the initial message and uses $a||b$ during the challenge-response phase.

Verifier simulation: When the verifier oracle gets the initial message e, he reads the tuple $(e, a||b)$ in WL and uses the corresponding $a||b$ to verify the responses. If no such tuple exists, then he is allowed to use the decryption oracle on e (as it is not a challenge c_b). As G2 enforces that only invalid or prover generated signatures are contained in e, then either $\mathscr{A}$ loses for sending an invalid signature, or e is a new encryption for values contained in one of the challenges. In the latter case, $\mathscr{B}$ readily obtains the bit b by verifying whether the decryption of e corresponds to a m_0 or a m_1.

Return value: $\mathscr{B}$ returns $\mathsf{Out_V}$.

If $b = 1$, $\mathscr{B}$ simulates G2 (e is the encryption of $a||b$). In this case, $\mathscr{B}$ wins if $\mathsf{Out_V} = 1$. By definition, $Pr[\mathsf{Out_V} = 1]$ in G2 $= Pr[\mathsf{G2}]$. Otherwise, if $b = 0$, then $\mathscr{B}$ simulates G3 (e is the encryption of δ). In this case, $\mathscr{B}$ returns 0 if $\mathscr{A}$ loses (*i.e.*, with probability $1 - Pr[\mathsf{G3}]$). The winning probability of $\mathscr{B}$ is then $\frac{Pr[\mathsf{G2}]+1-Pr[\mathsf{G3}]}{2} = \frac{1+(Pr[\mathsf{G2}]-Pr[\mathsf{G3}])}{2}$, resulting in an advantage of $\epsilon = |Pr[\mathsf{G2}] - Pr[\mathsf{G3}]|$. It follows that any significant probability difference between the two games can be transformed into an IND-CCA2 advantage and $|Pr[\mathsf{G3}] - Pr[\mathsf{G2}]| \leq \mathsf{Adv}_{\mathsf{E}}^{\mathsf{IND\text{-}CCA2}}(\lambda)$.

The final game We now prove that the success probability of $\mathscr{A}$ in G3 is negligible. During the time-critical phases, $\mathscr{A}$ can either Prompt or Commit.

- If he uses Prompt, then his response is valid with probability $\frac{1}{2}$. This corresponds to the probability to have $a_i = a_i^*$ (respectively $b_i = b_i^*$): e (from the honest prover) is the encryption of a random string and leaks no information about $a||b$, so that the probability for $\mathscr{A}$ to chose the same values is uniform.

- If he uses Commit, then either $a_i^* = b_i^* \oplus m_i$, and he can commit to a correct response with probability 1, or $a_i^* \neq b_i^* \oplus m_i$, and then he must guess the challenge to commit to the correct response. Since m is uniformly distributed and unknown to $\mathscr{A}$ at the time when he picks $a||b$, we have $Pr[a_i^* = b_i^* \oplus m_i] = \frac{1}{2}$. Hence, the probability to commit to the valid response is $Pr[a_i^* = b_i^* \oplus m_i] \cdot 1 + Pr[a_i^* \neq b_i^* \oplus m_i] \cdot \frac{1}{2} = \frac{3}{4}$.

From this, it follows that the best strategy for $\mathscr{A}$ is to respond by himself, as in a classical DF, using Commit. For n challenges, his advantage $\mathbf{Adv}_{\mathsf{DB}}^{\mathsf{DH}}(\mathscr{A})$ is therefore upper bounded by $\left(\frac{3}{4}\right)^n$ (at most $\frac{3}{4}$ for each round), which is negligible. $\square$

Mafia Fraud

The mafia fraud resistance of TREAD comes from the fact that no adversary can efficiently obtain the values $a||b$ corresponding to an initial message e and use it to impersonate a prover, nor can he generate an authenticated message e, which would be accepted, by himself.

Theorem 8. *If* E *is an* $\mathsf{IND\text{-}CCA2}$*-secure encryption scheme, and* S *is an* $\mathsf{EUF\text{-}CMA}$ *signature or MAC scheme, or a traceable group signature scheme, then* TREAD *is MF-resistant.*

The prover and verifier oracles are simulated as defined in the protocol definition , except that after generating e, the prover adds an entry to a *witness list* WL containing $(e, a||b)$.

In Game $\mathsf{G0}$, the prover and verifier follow the specification of the protocol, except that the prover adds entries to WL:

$\mathsf{DB.prover}(\mathbb{K}_{\mathsf{ID}})$ The prover starts by drawing a random value $a||b$ from the uniform distribution on $\{0,1\}^{2\cdot n}$. Then, he computes a signature σ_p on it with $\mathsf{S.sig}_{\mathsf{sigk}}(a||b||\mathsf{idprv}(\mathrm{P}))$. Afterwards, he generates $e = \mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv}(\mathrm{P})||\sigma_p)$ and sends $e||\mathsf{idpub}(\mathrm{P})$. He also adds $(e, a||b)$ to WL. Finally, during the n time-critical phases, he receives a challenge bit c_i and responds with $r_i = (a_i \wedge \overline{c_i}) \vee ((b_i \oplus m_i) \wedge c_i)$.

$\mathsf{DB.verifier}(\mathbb{K}_{\mathsf{V}}, \mathsf{UL}, \mathsf{RL})$ The verifier waits for an initial message e and deciphers it as $(a||b||\mathsf{idprv}(\mathrm{P})||\sigma_p) = \mathsf{E.dec}_{\mathsf{deck}}(e)$. If σ_p is invalid (*i.e.*, $\mathsf{S.ver}_{\mathsf{verk}}(\sigma_p, a||b||\mathsf{idprv}(\mathrm{P})) = 0$), the verifier aborts. Otherwise, he picks a random bit string m from the uniform distribution on $\{0,1\}^n$ and sends it. Afterwards, during the n time-critical phases, he generates a random bit c_i from a uniform distribution, starts his clock, sends c_i, gets back r_i, stops his clock and stores the corresponding time Δt_i. Finally, he verifies that (1) $\Delta t_i \leq t_{\mathsf{max}}$ and (2) $r_i = (a_i \wedge \overline{c_i}) \vee ((b_i \oplus m_i) \wedge c_i)$, for all $i \leq n$. If these conditions hold, he sends an accepting bit $\mathsf{Out}_{\mathsf{V}} = 1$, while otherwise he sends $\mathsf{Out}_{\mathsf{V}} = 0$.

Proof. We start from the initial mafia fraud game $\mathsf{G0}$, and build the following sequence of games. We first rule out the possibility for $\mathscr{A}$ to extract $a||b$ from e, by forbidding $a||b$ collisions, and making e independent from $a||b$.

$\mathsf{G1}$**:** *In this game, no value* $a||b$ *is outputted more than once by the prover oracle.*

In the i^{th} session, the probability to have a collision with any of the previous $i-1$ $a||b$ values is bounded by $\frac{i}{2^{2\cdot n}}$. If $\mathscr{A}$ runs q_{p} prover sessions, the probability of a collision for a given session is bounded by $\frac{q_{\mathsf{p}}}{2^{2\cdot n}}$. From the union bound, the probability that a collision occurs at least once is bounded by $\sum_{i=0}^{q_{\mathsf{p}}} \frac{q_{\mathsf{p}}}{2^{2\cdot n}}$, which is in turn bounded by $\frac{q_{\mathsf{p}}^2}{2^{2n}}$. Thus, $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq \frac{q_{\mathsf{p}}^2}{2^{2n}}$, which is negligible.

$\mathsf{G2}$**:** *This game aborts if* σ_p *was not generated by the prover oracle, and* $\mathsf{S.ver}_{\mathsf{verk}}(\sigma_p, a||b) \neq 0$.

In this game, we rule out the possibility that $\mathscr{A}$ produces a valid signature on behalf of a honest prover, which is trivially forbidden by the $\mathsf{EUF\text{-}CMA}$ resistance of S (or its traceability if it is a group signature). The reduction simply consists in starting $\mathsf{EUF\text{-}CMA}$ experiments (one for each prover) with a challenger and using queries to the corresponding signing oracle to generate the signatures of a prover. Then, if $\mathscr{A}$ sends a valid signature on behalf of one of the provers, we can return it to the challenger and win the $\mathsf{EUF\text{-}CMA}$ experiment. Hence, we have $|Pr[\mathsf{G2}] - Pr[\mathsf{G1}]| \leq q_{\mathsf{p}} \cdot \mathsf{Adv}_{\mathsf{S}}^{\mathrm{F}orge}(1^\lambda)$ (where $\mathrm{F}orge$ is either the $\mathsf{EUF\text{-}CMA}$ or traceability experiment, depending on S), which is negligible by hypothesis.

$\mathsf{G3}$**:** *In this game,* e *is replaced by the encryption of a random string (of equal length).*

This transition is based on indistinguishability, aiming at removing any leakage of $a||b$ from e by making $a||b$ only appear during the DB phase. We prove that the probability $\epsilon = |Pr[\mathsf{G3}] - Pr[\mathsf{G2}]|$ is negligible by building a distinguisher $\mathscr{B}$ such that the advantage of $\mathscr{B}$ against the

IND-CCA2 experiment is polynomial in ϵ. Hence, if ϵ is non-negligible, we reach a contradiction. By assumption, the advantage of any adversary against the IND-CCA2 experiment on E is negligible.

To build $\mathscr{B}$, we replace $\mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv(P)}||\sigma_p)$ by a string given by the IND-CCA2 challenger. Using the adversary $\mathscr{A}$, the distinguisher $\mathscr{B}$ can be built as follows.

Prover simulation: $\mathscr{B}$ generates two messages: $m_0 = (\delta||\mathsf{idprv(P)}||\mathsf{S.sig}_{\mathsf{sigk}}(\delta||\mathsf{idprv(P)}))$ and $m_1 = (a||b||\ \mathsf{S.sig}_{\mathsf{sigk}}(a||b||\mathsf{idprv(P)}))$, in which $a||b$ and δ are random binary strings of length $2 \cdot n$. Then, he sends them to the challenger to obtain c_b, the encryption of m_b (depending on a random bit b picked by the challenger before the experiment). He also adds $(c_b, a||b)$ to the list WL. Afterwards, he sends c_b as the initial message and uses $a||b$ during the challenge-response phase.

Verifier simulation: When the verifier oracle gets the initial message e, he reads the tuple $(e, a||b)$ in WL and uses the corresponding $a||b$ to verify the responses. If no such tuple exists, then he is allowed to use the decryption oracle on e (as it is not a challenge c_b). As G2 enforces that only invalid or prover generated signatures are contained in e, then either $\mathscr{A}$ loses for sending an invalid signature, or e is a new encryption for values contained in one of the challenges. In the latter case, $\mathscr{B}$ readily obtains the bit b by verifying whether the decryption of e corresponds to a m_0 or a m_1.

Return value: $\mathscr{B}$ returns Out_V.

If $b = 1$, $\mathscr{B}$ simulates G2 (e is the encryption of $a||b$). In this case, $\mathscr{B}$ wins if $\mathsf{Out}_\mathsf{V} = 1$. By definition, $Pr[\mathsf{Out}_\mathsf{V} = 1]$ in G2 $= Pr[\mathsf{G2}]$. Otherwise, if $b = 0$, then $\mathscr{B}$ simulates G3 (e is the encryption of δ). In this case, $\mathscr{B}$ returns 0 if $\mathscr{A}$ loses (*i.e.*, with probability $1 - Pr[\mathsf{G3}]$). The winning probability of $\mathscr{B}$ is then $\frac{Pr[\mathsf{G2}]+1-Pr[\mathsf{G3}]}{2} = \frac{1+(Pr[\mathsf{G2}]-Pr[\mathsf{G3}])}{2}$, resulting in an advantage of $\epsilon = |Pr[\mathsf{G2}] - Pr[\mathsf{G3}]|$. It follows that any significant probability difference between the two games can be transformed into an IND-CCA2 advantage and $|Pr[\mathsf{G3}] - Pr[\mathsf{G2}]| \leq \mathsf{Adv}_{\mathsf{E}}^{\mathsf{IND\text{-}CCA2}}(\lambda)$.

The final game We are left to prove that $Pr[\mathsf{G3}]$ is negligible. First remark that in G3, $\mathscr{A}$ has absolutely no way to predict the value r_i for any round i (as neither a_i nor b_i appears before round i). Hence, $\mathscr{A}$ can either try to guess c_i or r_i. His success probability in the second case is $\frac{1}{2}$. In the first case, he succeeds if he guesses the challenge properly (as he can obtain the response from the prover), but also if he wrongly guesses the challenge but guesses correctly the other response. The corresponding probability is $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$ for each round. As there are n such rounds, $Pr[\mathsf{G3}] \leq \left(\frac{3}{4}\right)^n$. Note that $\mathscr{A}$ can learn a small number of bits of both a and b during a learning phase in which the prover is close: by flipping a challenge, $\mathscr{A}$ receives $r^{\overline{c_i}}$ instead of r^{c_i} and can forward it to V. If $\mathsf{Out}_\mathsf{V} = 1$ (resp. 0), then $\mathscr{A}$ deduces that $a_i = b_i \oplus m_i$ (resp. $a_i \neq b_i \oplus m_i$), and thus learns one bit of both a_i and b_i. This generalises to the case in which $\mathscr{A}$ flips k challenges: if for the k corresponding rounds, $a_i = b_i \oplus m_i$, then $\mathsf{Out}_\mathsf{V} = 1$, and otherwise, $\mathsf{Out}_\mathsf{V} = 0$. Thus, $\mathscr{A}$ learns k bits of both strings in the first case, and nothing useful otherwise (except that at least one of the k rounds has $a_i \neq b_i \oplus m_i$. Hence, his probability of obtaining k bits is $\frac{1}{2^k}$, which is negligible if k is a fraction of n. Moreover, $a||b$ is never used more than once by the prover, so this strategy can only be used once, and does not grant any significant advantage to $\mathscr{A}$. □

Terrorist Fraud

Theorem 9. *If the challenges c_i are drawn uniformly at random by the verifier,* TREAD *is* strSimTF-*resistant.*

The theorem says that there exists a simulator Sim, which, from the view of a TF accomplice $\mathscr{A}$ who successfully authenticated once, can authenticate too with a probability at least as high. In the following proof, we build this simulator. Note that this proof holds within the models, in which the messages are broadcasted.

Proof. Let sid denote the session in which $\mathscr{A}$ was authenticated, and sid' denote the session played by the simulator. The messages in sid' are denoted with an apostrophe, for instance the message NP'

The simulator Sim starts by sending his initial message e', such that $e' = e$, and receives m'. During the time-critical phases, the response vectors are the same in sid and sid', except for $m \neq m'$. However, Sim can easily translate a response from sid to a response in sid': $r'^0 = r^0$, and $r'^1 = r^1 \oplus m_i \oplus m'_i$. Let us now consider the situations of $\mathscr{A}$ and Sim right before they receive the challenge at round i. $\mathscr{A}$ has some information in his view, denoted view_i, which permits him to respond to a random challenge with a response r_i with probability p_i. On the other hand, Sim knows the complete view, including view_i. Hence, Sim has more information than $\mathscr{A}$ to produce his response r'_i, so that $p'_i \geq p_i$. The final success probability of $\mathscr{A}$ is $\mathrm{P} = \prod_{i=1}^{n} p_i$. Since $\forall i \in \{1, n\}, p'_i \geq p_i$, it follows that $\mathrm{P}' \geq \mathrm{P}$, which concludes the proof. □

Privacy

The only place where values linked to the identity of P appear is inside the message e. Hence, $\mathscr{A}$ cannot break the privacy of TREAD_{priv} if he cannot distinguish the identity contained in e. Therefore, we use the IND-CCA2 security of E to prove that TREAD_{priv} is MiM-private.

Theorem 10. *If* E *is an* IND-CCA2*-secure encryption scheme, then* TREAD_{priv} *is MiM-private.*

Proof. Assume that there is a polynomial-time adversary $\mathscr{A}$ such that $\mathsf{Adv}^{\mathsf{Priv}}_{\mathscr{A},\mathsf{TREAD}^{\mathsf{Pub}}}(\lambda)$ is non-negligible. We show how to construct an adversary $\mathscr{B}$ such that $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathscr{B},\mathsf{PKE}}(\lambda)$ is also non-negligible.

Initially, the IND-CCA2 challenger sends a key pk_{V} to $\mathscr{B}$. Then, $\mathscr{B}$ runs a modified version of $\mathsf{DB.gen}(1^\lambda)$, in which the key generation for the verifier is omitted, and sends the public parameters to $\mathscr{A}$, including pk_{V}. $\mathscr{B}$ possesses all the material to simulate the experiment, except for the private key of the verifier. Additionally, during the first phase of the IND-CCA2 experiment, $\mathscr{B}$ is allowed to use the decryption oracle $\overline{\mathsf{PKE}}.\mathsf{dec}_{\mathsf{x}}$ freely. Hence, he can perfectly simulate the environment for $\mathscr{A}$, until $\mathscr{A}$ picks two identities $(\mathsf{ID}_0, \mathsf{ID}_1)$. Then, $\mathscr{B}$ uses the $\overline{\mathsf{PKE}}.\mathsf{enc}_{\mathsf{p}}\mathsf{k}(\mathsf{LR}_{\mathsf{b}})$ oracle to simulate these two provers. $\mathscr{B}$ first initialises a list WL, and simulates the prover and verifier oracles as follows:

$\overline{\mathsf{DB}}.\mathsf{Prover}(\cdot)$**:** Picks $a||b \xleftarrow{\$} \{0,1\}^{2\cdot n}$ and computes the signatures $\sigma^0_p = \mathsf{S.sig}_{\mathsf{sigk}_{\mathsf{ID}_0}}(a||b||\mathsf{idprv}(\mathsf{P}_{\mathsf{ID}_0}))$ and $\sigma^1_p = \mathsf{S.sig}_{\mathsf{sigk}_{\mathsf{ID}_1}}(a||b||\mathsf{idprv}(\mathsf{P}_{\mathsf{ID}_1}))$. He sends the messages $m_0 = (a||b||\mathsf{idprv}(\mathsf{P}_{\mathsf{ID}_0})||\sigma^0_p)$ and $m_1 = (a||b||\mathsf{idprv}(\mathsf{P}_{\mathsf{ID}_1})||\sigma^1_p)$ to $\overline{\mathsf{PKE}}.\mathsf{enc}_{\mathsf{p}}\mathsf{k}(\mathsf{LR}_{\mathsf{b}})$ in order to obtain e, and adds (a, b, e) to WL. Finally, it sends e and receives m. The time critical phases are ran as usual, using $a||b$ and m.

$\overline{\mathsf{DB}}.\mathsf{Verifier}(\cdot)$**:** Receives e from $\mathscr{A}$. If there is no entry (a, b, e) in WL (*i.e.* if e was not generated by the $\overline{\mathsf{PKE}}.\mathsf{enc}_{\mathsf{p}}\mathsf{k}(\mathsf{LR}_{\mathsf{b}})$ oracle), then $\mathscr{B}$ is allowed to use the decryption oracle and to run the protocol normally with the deciphered value. Otherwise, it runs the protocol using the value a, b from the tuple (a, b, e) read in WL.

When $\mathscr{A}$ sends his guess b', $\mathscr{B}$ simply forwards it to the challenger.

The experiment is perfectly simulated for $\mathscr{A}$, so that $\mathscr{B}$ wins when $\mathscr{A}$ wins. Hence, we have, $\mathsf{Adv}^{\mathsf{Priv}}_{\mathscr{A},\mathsf{TREAD}^{\mathsf{Pub}}}(\lambda) = \mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathscr{B},\mathsf{PKE}}(\lambda)$, contradicting the IND-CCA2 security assumption on PKE. □

Anonymity

The anonymity of TREAD_{ano} directly depends of the anonymity of the dynamic group signature scheme, since the signature issued by P is the only value related to a long term secret that identifies him.

Theorem 11. *If* S *is an anonymous dynamic group signature scheme, then* TREAD$_{ano}$ *is prover anonymous.*

Proof. Assume that there is a polynomial-time adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{Anon}}_{\mathcal{A},\mathsf{TREAD}^{\mathrm{ANO}}}(\lambda)$ is non-negligible. We show how to build an adversary $\mathcal{B}$ such that $\mathsf{Adv}^{\mathsf{Anon}}_{\mathcal{B},\mathsf{G\text{-}SIG}}(\lambda)$ is also non negligible.

Initially, the challenger sends a key gpk and a revoked list RL to $\mathcal{B}$. Then, $\mathcal{B}$ generates a public/private key pair pk_{V}, x_v for the verifier using $\mathsf{PKE.gen}(1^\lambda)$, sends $(pk_{\mathsf{V}}, \mathsf{gpk}, \mathsf{RL})$ to $\mathcal{A}$, and creates the empty list CU of corrupted users. Having access to the G-SIG-oracles from his challenger, $\mathcal{B}$ can simulate the DB-oracles for $\mathcal{A}$ as follows:

$\overline{\mathsf{DB}}.\mathsf{Join}^h(\cdot)$: On input i, creates P_i with $\overline{\mathsf{G}}.\mathsf{Join}^h(\cdot)$, and adds P_i to UL.

$\overline{\mathsf{DB}}.\mathsf{Join}^c(\cdot)$: On input i, creates a corrupted P_i with $\overline{\mathsf{G}}.\mathsf{Join}^c(\cdot)$, adds P_i to UL and CU, and returns ssk_i.

$\overline{\mathsf{DB}}.\mathsf{Revoke}(\cdot)$: On input i, revokes P_i with $\overline{\mathsf{G}}.\mathsf{Revoke}(\cdot)$, which updates RL and returns it.

$\overline{\mathsf{DB}}.\mathsf{Corrupt}(\cdot)$: On input i, corrupts P_i with $\overline{\mathsf{G}}.\mathsf{Corrupt}(\cdot)$ and gets ssk_i. $\mathcal{B}$ adds P_i to CU, and returns ssk_i.

$\overline{\mathsf{DB}}.\mathsf{Prover}(\cdot)$: $\mathcal{B}$ simulates P_i as follows: $\mathcal{B}$ picks $a||b \xleftarrow{\$} \{0,1\}^{2\cdot n}$ and sends $(i, a||b)$ to his oracle $\overline{\mathsf{G}}.\mathsf{Sign}(\cdot,\cdot)$ to obtain $\sigma_p = \mathsf{G.sig}_{\mathsf{ssk}_i}(a||b)$. He computes $e = \mathsf{PKE.enc}_{pk_{\mathsf{V}}}(a||b||\sigma_p)$, and runs the rest of the protocol normally.

$\overline{\mathsf{DB}}.\mathsf{Verifier}(\cdot)$: the verifier can be simulated as per the protocol describes it: $\mathcal{B}$ knows the secret key of the verifier, and the verification key gpk.

After interacting with these oracles, $\mathcal{A}$ sends $(\mathsf{ID}_0, \mathsf{ID}_1)$ to $\mathcal{B}$. If ID_0 or $\mathsf{ID}_1 \in \mathsf{CU}$, $\mathcal{B}$ aborts the experiment. Otherwise, $\mathcal{B}$ sends $(\mathsf{ID}_0, \mathsf{ID}_1)$ to the challenger. Then, $\mathcal{B}$ modifies his simulation for the $\mathsf{Corrupt}(\cdot)$ and $\mathsf{Revoke}(\cdot)$ oracles by returning $\perp$ when they are called on input ID_0 or ID_1. Afterwards, $\mathcal{B}$ simulates the challenge oracle $\overline{\mathsf{DB}}.\mathsf{Prover}_b$ for $\mathrm{P}_{\mathsf{ID}_b}$ as follows: $\mathcal{B}$ picks $a||b \xleftarrow{\$} \{0,1\}^{2\cdot n}$, sends $(a||b)$ to his oracle $\overline{\mathsf{G}}.\mathsf{Sign}_b(\cdot,\cdot)$ to get the signature $\sigma_p = \mathsf{G.sig}_{\mathsf{ssk}_i}(a||b)$, and sends $e = \mathsf{PKE.enc}_{pk_{\mathsf{V}}}(a||b||\sigma_p)$. It then runs the rest of the protocol normally.

Finally, $\mathcal{A}$ sends b' to $\mathcal{B}$, who simply forwards it to the challenger.

The experiment is perfectly simulated for $\mathcal{A}$, so that $\mathcal{B}$ wins when $\mathcal{A}$ wins.

Thus, $\mathsf{Adv}^{\mathsf{Anon}}_{\mathcal{B},\mathsf{G\text{-}SIG}}(\lambda) = \mathsf{Adv}^{\mathsf{Anon}}_{\mathcal{A},\mathsf{TREAD}^{\mathsf{Ano}}}(\lambda)$, contradicting the assumption on G-SIG. □

4.6.4 Recent attacks against SPADE and TREAD

Recently, two articles [Mauw et al., 2018, Debant et al., 2018] independently proposed formal methods to verify the security of distance bounding protocols. Formal methods use a mathematical model of the target protocol and its expected security properties, and provide them to an automatic prover, which verifies whether the security properties are satisfied. Both articles use well known provers: the first one uses the Tamarin [Meier et al., 2013], while the second one uses Cryptoverif [Blanchet, 2007].

Both of these articles exhibited vulnerabilities against SPADE and TREAD. Specifically, both found distance hijacking and mafia frauds in our protocol. These attacks do not contradict our security proofs: they require multiple verifiers, some of which are dishonest, while the DFKO model considers a single, honest verifier. More specifically, the adversaries have public/private key pairs, just like the verifiers, and the provers do not check whether the public key corresponds to a legitimate verifier, thus making the adversary effectively capable of impersonating a verifier. However, when the adversary can control verifiers, then the basic idea behind both our protocols falls: our protocols are secure if the temporary secrets held within the initial encrypted message e cannot be read by an adversary. If the adversary is capable of decrypting it as a verifier, then he can extract the temporary secrets it contains and their signature, and therefore defeat the protocol.

These attacks work as follows:

Mafia Fraud

In both SPADE and TREAD, an initial message e contains the encryption of signed the session keys, NP for SPADE, and (a,b) for TREAD. There exists a mafia fraud against SPADE, and the two versions of TREAD for which e is encrypted with a public key: TREAD$_{priv}$ and TREAD$_{ano}$. To perform a mafia fraud, $\mathscr{A}$ starts a session with a prover, obtains e, and extracts the session keys and the corresponding signature from it. With these values, he can start a session with the verifier, and authenticate as P. On the other hand, TREAD$_{sym}$ is secure, because $\mathscr{A}$ does not know the encryption key x used by P to communicate with V. Hence, $\mathscr{A}$ can request a message e ciphered with a key shared between P and $\mathscr{A}$, and extract the session key and its signature from it, but he cannot encrypt them with x to make the verifier accept.

Distance Hijacking

SPADE and all versions of TREAD are vulnerable to a distance hijacking attacks if $\mathscr{A}$ can corrupt verifiers. The attack scenario is the following: the distant $\mathscr{A}$ starts a session as a verifier with a prover P located close to the verifier, and receives e, from which it extracts the session keys as for the mafia fraud. He then starts a session with V, in which he sends the same session keys, but signed with his own signature keys. He then lets P run the rest of the protocol. The responses of P are correct, since P uses the same session keys as $\mathscr{A}$, so that $\mathscr{A}$ is effectively accepted even though he is far away.

Discussion

These attacks underline an interesting problem, which is the one of multiple and possibly malicious verifiers, which was overlooked by the existing formal security models. We propose hints on how to handle these attacks.

In the article presenting SPADE [Bultel et al., 2016], we briefly mentioned this scenario, and stated that adding the identity of the verifier inside the initial signed message would prevent mafia frauds. Indeed, in this case, $\mathscr{A}$ becomes unable to use the signature produced by P with another verifier V: V would notice that the signed message contains the identity of $\mathscr{A}$ instead of his own. An inconvenient of this method is that it lowers the terrorist fraud resistance: a terrorist fraud accomplice is only capable of impersonating P with one verifier, instead of all of them.

While including the identity of V in the signed message does prevent the mafia frauds, it does nothing against the distance hijacking attacks. These attacks might be solved by changing the responses r_i to $r_i^* = r_i \oplus \mathsf{H}(e_i)$, the hash of the initial encrypted message e. Intuitively, the idea is that if $\mathscr{A}$ changes the signature contained in e, then he needs to reencrypt it to produce a message e', which should be different from e, except with negligible probability. The hash would make sure that the resulting string has the correct length n, and that the responses expected from P differ from the responses expected from $\mathscr{A}$ on half the rounds on average. Hence, the responses from P would not match with the responses expected by the verifier, which would reject the authentication.

4.7 Conclusion

In this chapter, we presented two construction, SPADE and TREAD, which both grant terrorist fraud resistance by forcing the prover to leak a session key to his accomplice, which is a shift from traditional protocols which require the leaked key to be a long term one. SPADE only comes in one anonymous version, while TREAD has three versions, each providing a different trade-off in terms of efficiency and privacy: TREAD$_{sym}$ is efficient but grants no privacy, TREAD$_{priv}$ is private against eavesdropper but not anonymous, and TREAD$_{ano}$ is fully anonymous with regards to tracking attempts by the verifier. SPADE can be seen as a preliminary version of TREAD: it uses a similar idea for ensuring both anonymity and terrorist fraud resistance, but TREAD is more simple and generic. Additionally, TREAD does not require a backdoor in order to achieve provable terrorist fraud resistance. Both constructions are provably resistant to all threats against distance bounding protocols in the DFKO model. On the other hand, there exist attacks outside of the security model, as shown in [Mauw et al., 2018, Debant et al., 2018], which consider multiple and malicious verifiers. While these particular attacks can be mitigated by modifying the protocols, we show in the next section that this is not always the case. Indeed, some specific hardware can allow for attacks that are outside of the security models. In particular, we show that the terrorist fraud resistance notion as we try to achieve it can never be reached.

Chapter 5

Hardware Based Terrorist Frauds

Contents

Achieving terrorist fraud resistance in the security models is difficult, but possible, as shown in the previous chapter. On the other hand, a paper [Ahmadi and Safavi-Naini, 2018] presented terrorist fraud attacks that circumvent the security model by using specific hardware, namely directional antennas. We proposed a more generic terrorist fraud [Boureanu et al., 2018], the Tamper-Proof Device Attack, which uses tamper-proof devices. In this chapter, we review these results, and present an impossibility result. A protocol is considered terrorist fraud resistant if, when the help of a malicious prover P^* permits his accomplice $\mathscr{A}$ to authenticate, then this help also gives $\mathscr{A}$ significant to authenticate later. We prove that this definition cannot be satisfied. On the other hand, we define a new notion, One-Step Terrorist Fraud (OSTF), in which a protocol is considered TF resistant if no help from P^* can permit $\mathscr{A}$ to authenticate.

5.1 Introduction

Terrorist fraud resistance is a long standing issue in the distance bounding community. There is a long history of protocols that claimed terrorist fraud resistance, but were broken in one way or another, *e.g.*, [Bay et al., 2013]. Similarly, there are a wide range of definitions of what a terrorist fraud is, some more exotic than others. This research problem led to designing protocols that were intentionally weaker, for instance due to the presence of a backdoor as in SPADE (Chapter 4) or [Fischlin and Onete, 2013a]. However, in the past few years, the community successfully exhibited designs that were proven to be terrorist fraud resistant, such as [Fischlin and Onete, 2013a, Vaudenay, 2015b, Boureanu et al., 2015]. While this solved a difficult research problem, the relief did not last very long. Ahmadhi $et al.$ [Ahmadi and Safavi-Naini, 2018] exhibited an attack which, by using directional antennas, can defeat the terrorist fraud resistance of SPADE and TREAD. This attack is very interesting, because it bypasses the security proofs of our protocols: the security models that are used in distance bounding assume that the messages are broadcasted, but these attacks, through the use of directional antennas, allow to send a message so that it is received by one party only. It therefore allows a malicious prover to send the initial message of SPADE and TREAD directly to the verifier, so that $\mathscr{A}$ cannot see it, nor replay it in a further session. Since the terrorist fraud resistance of SPADE and TREAD depend on $\mathscr{A}$ knowing this initial message, if directional antennas can be used, these protocols are not terrorist fraud resistant. However, this result does not apply to non anonymous protocols.

Independently, at the same time, in [Boureanu et al., 2018], we proposed a different kind of hardware based attacks. Our main result is that, if the provers can be cloned (*i.e.*, they are not tamper-proof and do not use physically uncloneable functions), then terrorist fraud resistance as it is usually defined cannot be achieved. We prove that claim by exhibiting a simple, generic terrorist fraud: the prover builds a tamper-proof device that is capable of authenticating, and programs it to erase all its memory after one successful authentication. He then gives this device to his accomplice. This very simple attack defeats the claim for terrorist fraud resistance of all protocols in the literature in the white-box model, except for one [Igier and Vaudenay, 2016], which achieves a different type of terrorist fraud resistance. We define this type of terrorist fraud resistance as OSTF-resistance, and show that this definition is sufficient.

In this chapter, we first present the attacks by Ahmadi $et al.$, and we then present our generic terrorist fraud, and the new notion of OSTF.

5.2 Directional Terrorist Frauds

5.2.1 Attacks

In [Ahmadi and Safavi-Naini, 2018], Ahmadi $et al.$ introduced the notion of Directional TF (DTF). In a directional TF, the prover uses a directional antenna pointed at the verifier to send some of the messages, as represented on Figure. 5.2.1. Let us call such messages *directional messages*. If the accomplice $\mathscr{A}$ is located near the verifier, but out of the range of the directional antenna, then he cannot see these messages. In SPADE and TREAD, if $\mathscr{A}$ cannot read the initial message of the protocol, then he cannot impersonate the prover in a latter session, even if he knows both response vectors. The DTF against SPADE is the following:

- P^* generates $e = \mathsf{PKE.enc}_{pk_\mathrm{V}}(\mathrm{NP}, \sigma_p)$ as defined in the protocol, and sends it to V with the directional antenna
- P^* sends NP to $\mathscr{A}$
- $\mathscr{A}$ receives NV and m from the verifier, and computes $a = \mathrm{PRF}_{\mathrm{NP}}(\mathrm{NV})$
- $\mathscr{A}$ responds to the challenges, using a, NP, and m
- $\mathscr{A}$ sends the final message $\mathbb{T} = \mathrm{PRF}_{\mathrm{NP}}(\mathrm{NV}, m, \mathrm{C}, \mathrm{R})$
- $\mathscr{A}$ is accepted, but he cannot authenticate again later, since he does not know the message e linked to NP, nor can he produce a valid signature on NP on his own.

We now describe the attack against TREAD:

- P^* generates $e = \mathsf{E.enc}_{\mathsf{enck}}(a||b||\mathsf{idprv}(\mathrm{P})||\sigma_p)$ as defined in the protocol, and sends $e, \mathsf{idpub}(\mathrm{P})$ it to V with the directional antenna
- P^* sends (a, b) to $\mathscr{A}$
- $\mathscr{A}$ receives m from the verifier
- $\mathscr{A}$ responds to the challenges, using a, b and m
- $\mathscr{A}$ is accepted, but he cannot authenticate again later, since he does not know the message e linked to (a, b), nor can he produce a valid signature on (a, b) on his own.

Additionally, the authors of [Ahmadi and Safavi-Naini, 2018], exhibit a similar attack on their protocol PDB [Ahmadi and Safavi-Naini, 2014].

We now discuss the relevance of these attacks.

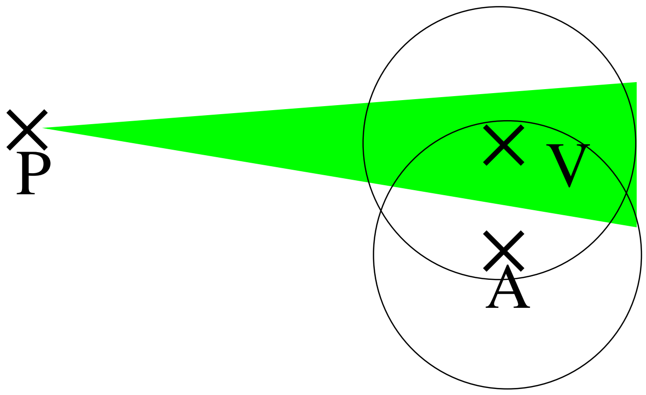

Figure 5.1: The position of the prover, adversary and verifier in a directional terrorist fraud. The green ray represents the propagation of the message sent by the directional antenna: the adversary cannot see it, but the verifier can.

5.2.2 Discussion

Directional terrorist frauds are an interesting form of attacks. They apply to SPADE and TREAD, which both came with security proofs, which raises questions about the relevance of the proofs. These attacks circumvent formal DB models, which consider that all messages are broadcasted, and that the adversary can always access them, as a Dolev-Yao adversary [Dolev and Yao, 1981]. However, this assumption does not hold if specific hardware (directional antennas) is used. This is a strong hint that hardware based attack can sometimes get around the models, which should be adapted accordingly.

The DTF attack model however has a limitation. It requires the adversary to be located outside of the range of the antenna, otherwise he would be able to see the directed messages and the attack would fail. Hence, P^* must either have a way of verifying the location of $\mathscr{A}$, for instance by staying in view range during the attack or using a video camera to observe $\mathscr{A}$, or $\mathscr{A}$ must behave exactly according to the instructions of P^*. The first scenario might not be feasible: even if $\mathscr{A}$ is located at the position expected by the prover, he can place receiving devices on the path between P^* and V beforehand, or even hide one behind V. Actually, $\mathscr{A}$ may even use an accomplice, who would be located between P^* and V, and record the initial message, for instance with a smartphone in his pocket. There are so many ways to hide a receiver that we cannot safely assume that P^* will notice it if $\mathscr{A}$ cheats. This limits the scope of application of this attack in the scenario where P^* does not trust $\mathscr{A}$. We are therefore left with the second option: the prover needs to trust his accomplice not to trick him by eavesdropping the directional messages. This leads to an interesting new research direction. It seems that the implicit assumption that is usually made is that $\mathscr{A}$ will follow some predefined algorithm decided by P^* during the attack. In other words, it is generally considered that P^* and $\mathscr{A}$ are colluding, as one single entity, but the case in which $\mathscr{A}$ has an adversarial behaviour is left on the side. For instance, in the session in which $\mathscr{A}$ is helped, he could forward incorrect nonces to the prover in order to gain some information. While this does not seem to grant a big advantage in usual protocols, for at least 3 protocols (SPADE, TREAD, PDB), an accomplice can gain some advantage by deviating from the instructions given by P^*. It would be interesting to distinguish *trusted* TF, in which $\mathscr{A}$ faithfully follows an algorithm given by P^*, from *untrusted* TF, in which the accomplice tries to cheat.

However, the results we present in the next section indicate that, for at least one attack that we present, either the accomplice cannot be helped, or there exists a help from which he cannot authenticate again later, regardless of whether he follows the instructions of P^* or not.

5.3 Impossibility Results for Terrorist Fraud Resistance

In this section, we demonstrate that terrorist fraud resistance cannot be achieved the way it is usually seeked. We prove that, if $\mathscr{A}$ authenticates thanks to a non-void help of P^*, there always exists a strategy that prevents $\mathscr{A}$ to authenticate again later. We present a generic terrorist fraud, which works for all protocols for which the prover can be cloned by its holder. Intuitively, the prover can be cloned if the user has access to its secret material, and the protocol does not use physically uncloneable functions (PUFs). Informally, a PUF is a function that maps challenges to responses, but cannot be cloned. Our attack uses a clone of the prover in a tamper-proof device that erases its memory after being used. We discuss its applicability, distinguishing white-box provers (with full control of the algorithm), black-box provers (for which the secret material is hidden), and the special case of provers which can physically not be cloned. White-box provers can apply this attack in a straightforward way, black-box provers cannot, and in some protocols using PUFs, a modified version of this attack can sometimes be applied. We also prove black-box provers and protocols using PUFs, a different kind of terrorist fraud resistance can be achieved, and define the corresponding notion OSTF.

5.3.1 The tamper-proof Device Attack

Preliminaries

We assume that the provers can either be *white-box* (WB) or *black-box* (BB). Provers are operated by a person, referred to as their holder. The holder of a white-box prover knows its secret material, and can implement any algorithm of his choice on the device. Instead, black-box provers are completely tamper-proof, and only execute their predefined algorithm, without the holder being able to modify it or to extract the secret material. Stated differently, the holder of a black-box prover can only interacts with it by sending it messages that correspond to those expected during the execution of the distance bounding protocol. In this sense, the holder of a black-box device can only interact with his device the same way an adversary would.

White-box provers are typically employed in the distance-bounding literature, such as in the formal models [Dürholz et al., 2011, Boureanu et al., 2015], and articles presenting new protocols or terrorist frauds on existing protocols [Boureanu et al., 2012, Boureanu and Vaudenay, 2014]. On the other hand, in industry, for instance in the banking protocol EMV [MasterCard, 2017], the provers are assumed to be tamper-proof devices, and therefore, black-box.

Since it is a commonplace in the industry to have the security rely on tamper-proof hardware, we believe that it is a safe assumption that tamper-proof devices, from which the secret material cannot be extracted with reasonable effort, can be built.

Another assumption that we make is that no verification is performed besides the distance bounding protocol in itself. Specifically, the identity of the holder of the device is not checked, and the device performing the protocol can be counterfeited. This assumption is clearly made in the distance bounding literature: for instance, in the case of a mafia fraud, the device presented in front of the verifier is not the legitimate prover, and the person holding it is not is legitimate owner. If additional identity checks were performed, mafia fraud would simply not exist.

A Generic Terrorist Fraud for White-Box Provers

In typical proofs for terrorist fraud resistance of distance bounding protocols, the holder of the proving device knows the secret material and algorithm ran by the device. Additionally, only two protocols [Igier and Vaudenay, 2016, Kleber et al., 2015] include a mechanism (PUF) that prevents a prover from being copied by its holder. Therefore, in all distance bounding protocols of the literature, except for the 2 that use PUFs, the prover can clone his device. The case of black box provers, which cannot be cloned due to the prover not knowing their secret material, will be treated later.

When the device can be cloned, a malicious prover P^* can build a tamper-proof device that runs exactly as the proving device k times, after which it self-destructs. In the usual definitions for

TF, k is one, but some definitions require $\mathscr{A}$ to be helped several times, which is why we mention the possibility of having k executions.

Our attack is as follows: P^* builds a disposable, k-times clone of his prover in a tamper-proof device. He then gives this to his accomplice $\mathscr{A}$. The accomplice presents the tamper-proof clone to the verifier and successfully authenticates. The device is tamper-proof, so that $\mathscr{A}$ learns nothing more than from observing an honest execution of the protocol. Finally, the device erases its memory after k successful authentications, so $\mathscr{A}$ cannot use it to perform latter authentications that are not agreed to by the prover. This self-destruction equates in fact to a program wiping all data (secret and otherwise) and software on the device. This is a generic terrorist fraud. The prover permits $\mathscr{A}$ to authenticate, without giving him any sensitive information.

Discussion

Offline help Usually, terrorist-frauds are performed online: the prover helps his accomplice during one specific session, at a given time. In contrast, our generic attack permits the accomplice get the disposable device offline, and to use it at a time of his choice. However, this limitation can be overcome by including a secure, remote activation and deactivation mechanism on the disposable clone, in order to make it unusable except at a time chosen by the prover. By doing so, the prover could delegate his authentication rights while keeping full control on his credentials and the actions of his accomplice. Actually, this would even remove the need for a self destruction mechanism, and the prover could perform a form of permanent terrorist fraud.

Provable Terrorist Fraud Resistance Even though our attack corresponds to the idea of a terrorist fraud, it is still possible to prove the terrorist fraud resistance of some protocols in the security models. For instance, consider the SimTF definition of terrorist-fraud. The SimTF definition compares the probability p_A for $\mathscr{A}$ to authenticate with the help of P^*, and the probability p_S for a simulator to authenticate afterwards, using the view of $\mathscr{A}$. If $p_A - p_S$ is negligible, then the protocol is SimTF-resistant. Now consider a protocol which is trivially broken: the verifier sends a challenge C, and the prover responds with $C \oplus x$, where x is his secret key. By eavesdropping one session, an adversary learns x and can impersonate the prover. This protocol is SimTF-resistant: $p_A = 1$, regardless of the help of P^*, and $p_S = 1$ as well, since x is in the view of $\mathscr{A}$. Note that such a protocol would also be considered TF resistant according to the Collusion Fraud definition of [Boureanu et al., 2015], in which p_A and p_S are respectively replaced by γ and γ'. However, in this case, terrorist fraud resistance is irrelevant, as $\mathscr{A}$ can authenticate even if the help from P^* is void. We elaborate on this in the next section.

5.3.2 Another Definition for Terrorist Fraud Resistance

As mentioned in the previous section, some protocols are considered terrorist fraud resistant because they are not mafia fraud resistant. We believe that this should not be the case, as the help of P^* is irrelevant if $\mathscr{A}$ can authenticate on his own. Hence, mafia fraud vulnerable protocols should not be considered terrorist fraud resistant.

Our generic attack applies to all the protocols in which the prover can be cloned. On the other hand, some protocols resist it. For instance, consider a protocol with only one time critical round, in which the response to a challenge C is PUF(C), and C is n bit long, with n depending on the security parameter. Since we consider polynomially bounded adversaries, and the number of possible challenge response pairs is exponential in the security parameter, the prover can simply not give his accomplice enough data to be able to respond with a good probability. Moreover, if the accomplice forwarded C to the far away prover, then he would not receive the response early enough to respond within the time-bound. Hence, in such a protocol, the prover has no physical way of giving his accomplice any non-negligible advantage to succeed.

It is a different form of resistance than the one that is usually achieved. Terrorist fraud resistance is usually modelled as a two-step game, one step in which $\mathscr{A}$ is helped, and one step in which $\mathscr{A}$ is on his own. Here, it becomes a one-step game: the protocol is resistant if the help of P^* brings no significant advantage to $\mathscr{A}$. We therefore propose a new definition for terrorist fraud resistance, which simply states that a protocol is terrorist fraud resistant if the success probability of $\mathscr{A}$ with the help of P^*, denoted $p_{\mathscr{A}}$, is negligibly close to the success probability of the best MF adversary, which is itself negligible. We call this definition `One Step Terrorist Fraud` (OSTF).

OSTF **Resistance**

In some cases, the prover can simply not provide any helpful information for an accomplice to authenticate: regardless of the help provided by P^*, the success probability of the accomplice is negligible.

We define this form of resistance as OSTF resistance. A protocol is OSTF-*resistant* if whatever help is given by a malicious prover to an accomplice does not allow the accomplice to perform significantly better than if the prover was honest. In this attack model, the prover is not forbidden to give his secret key as in a regular terrorist-fraud. More formally, OSTF resistance is as follows.

Definition 32. `One Step Terrorist Fraud`**.** *Let DB be a MF resistant distance bounding protocol, let* λ *be a security parameter, and let* $\mathsf{Adv}_{DB}^{\mathsf{MF}}(\lambda)$ *denote the success probability of the best MF adversary against the protocol DB.*

A `One Step Terrorist Fraud` adversary *is a pair of any PPT algorithms* P^* *and* $\mathscr{A}$*, such that* P^* *is located far-away from the designated verifier, and* $\mathscr{A}$ *is at an arbitrary position.*

Let $p_{\mathscr{A}}$ *denote the probability of an accomplice* $\mathscr{A}$ *to pass the protocol with the help of a malicious prover* P^**, where* $(\mathrm{P}^*, \mathscr{A})$ *is a* `One Step Terrorist Fraud` *adversary.*

The distance bounding protocol DB is `One Step Terrorist Fraud` resistant *if, DB is MF resistant, and, for all PPT algorithms* P^* *and* $\mathscr{A}$*,* $|p_{\mathscr{A}} - \mathsf{Adv}_{DB}^{\mathsf{MF}}(\lambda)| \leq negl(\lambda)$.

We now prove that definition is sufficient, by proving that, if there exists $(\mathrm{P}^*, \mathscr{A})$ such that $p_{\mathscr{A}}$ is non negligible, then the protocol is vulnerable to a form of tamper-proof device attack, even if the device is not fully cloneable. We first prove that a tamper proof device that authenticates with the same probability as $\mathscr{A}$ can always be built, and then that there exists such a device that leaks no more information than a honest prover. We then conclude that, if P^* can give $\mathscr{A}$ a non negligible probability of authenticating, then there exists a tamper-proof device attack.

Theorem 12. *Let DB be a MF resistant distance bounding protocol, and* $p_{\mathscr{A}}$ *denote the success probability of an adversary* $\mathscr{A}$ *helped by a prover* P^**. If there exists a couple of terrorist fraud adversaries* $(\mathrm{P}^*, \mathscr{A})$ *such that* $p_{\mathscr{A}}$ *is non negligible, then DB is vulnerable to a tamper-proof device attack.*

Proof. If there exists a pair $(\mathrm{P}^*, \mathscr{A})$ such that $\mathscr{A}$ authenticates with a non negligible probability $p_{\mathscr{A}}$, then P^* can implement the algorithm ran by $\mathscr{A}$ on a tamper-proof device $\mathrm{T}_{\mathscr{A}}$, which authenticates with the same probability $p_{\mathscr{A}}$, and instruct $\mathscr{A}$ to present $\mathrm{T}_{\mathscr{A}}$ to the verifier. We are now left to prove that $\mathscr{A}$ does not learn enough information to authenticate again later from the execution of $\mathrm{T}_{\mathscr{A}}$. This would be the case if, for instance, P^* sent some secret material to $\mathscr{A}$ in clear. To prevent this kind of leaks, $\mathrm{T}_{\mathscr{A}}$ can be built in such a way that all communications with P^* are sent over a secure channel, and that only the messages that are part of the protocol ran with V are sent in clear. The delay introduced by the encryption for the secure channel does not invalidate $\mathrm{T}_{\mathscr{A}}$: time-critical exchanges can not be relayed in time, be it encrypted or not, due to the time measurement, and messages independent from the current challenge can be computed and encrypted in advance. With this technique, $\mathscr{A}$ only sees encrypted messages, and the transcript of a correct execution of the protocol. By hypothesis, DB is MF-resistant, so that eavesdropping honest sessions does not grant $\mathscr{A}$ any advantage. Therefore, if there exists a pair $(\mathrm{P}^*, \mathscr{A})$ such that $\mathscr{A}$ authenticates with a non negligible probability, then the protocol is vulnerable to a tamper-proof device attack. □

We now proceed to proving that protocols with black-box provers are all OSTF-resistant, and that some PUF-based protocols can be too.

5.3.3 Black-box Provers and PUFs

In this section, we study the known cases in which the provers are not cloneable. The first one is black-box provers, as this model is commonly used in the industry, and the second one is PUF-based protocols. To the best of our knowledge, no other strategies preventing the cloning of provers was proposed in any protocol of the literature.

Black-box Provers

Theorem 13. *Let DB be a MF-resistant distance bounding protocol. If the provers are black-box, then DB is* OSTF*-resistant.*

Proof. Consider a distance bounding protocol DB, in which the provers are black-box. If the prover is black-box, then it cannot be cloned: the holder does not know the secret material of the device. By definition, the holder of a black-box prover has exactly the same capabilities as an adversary: They can only interact with the device through the API of the protocol. From this, it follows that whatever help the holder gives to the adversary can be obtained by the adversary on his own, by interacting with the device in the same way the holder would.

Consequently, it holds that $p_{\mathscr{A}} = \mathsf{Adv}^{\mathsf{MF}}_{\mathsf{DB}}(\lambda)$, and DB is OSTF-resistant. □

PUF-based protocols

We now study the case of protocols which use PUFs, in which the devices are, by definition of a PUF, not cloneable. A PUF if a function that maps n_{in} bit challenges to n_{out} bit responses, and such that it is not emulable (*i.e.*, their output cannot be predicted without querying them), and a response to a challenge C gives negligible information on the response to another challenge C′. It is embedded in a hardware device. The typical use case for a PUF is to query it to obtain its responses to a list of challenges at the time of initialisation, and then using these challenges for authentication later. PUFs also have a public key equivalent, which does not require knowing a list of challenge/response pairs for authentication: SIMPL (Simulation Possible but Laborious): their output for a given challenge can be computed without the hardware device, but significantly slower than with it. By definition, the PUF embedded in a prover device cannot be cloned, and if a SIMPL is used, it cannot be cloned accurately either, as the computation time will be longer.

Whilst protocols without PUFs are the great majority on the DB protocols in existence, there are two PUF-based protocols in the literature: [Igier and Vaudenay, 2016, Kleber et al., 2015]. In these protocols, the response to the challenges is computed as the output of a PUF on some challenge, which is the concatenation of a long pre challenge and the challenges from the previous and current rounds.

The protocol of Kleber [Kleber et al., 2015] is vulnerable to a terrorist fraud, which was exhibited in [Igier and Vaudenay, 2016], and the one of Vaudenay [Igier and Vaudenay, 2016] claims to be TF resistant, under some assumptions on the communication complexity. However, the TF resistance achieved by the latter is stronger than usual: it is not that if P^* helps $\mathscr{A}$, then $\mathscr{A}$ can authenticate alone. It is that P^* can simply not help $\mathscr{A}$ by providing him with the necessary responses in time, *i.e.*, there is no strategy for P^* to help $\mathscr{A}$ authenticate. This corresponds to OSTF-resistance.

5.4 Conclusion

Usually, protocols claiming to attain terrorist fraud resistance do it by attempting to force the malicious prover to give his (long term or temporary) credentials to his accomplice. While this works in the usual formal models, it becomes completely irrelevant if the prover is allowed to use specific hardware countermeasure for his attack. By using directional antennas, for instance, a malicious prover can bypass the terrorist fraud resistance mechanism of some anonymous protocols (SPADE and TREAD). By building tamper-proof copies of his proving device, and making them usable only a given number of times, a malicious prover can defeat the terrorist fraud resistance of all protocols which permit the cloning of provers. If the provers are black-box, *i.e.* their holder does not have access to their secret material, then the prover cannot give his accomplice any more than if the accomplice had queried the device himself. Finally, for PUF-based protocols, either the prover is unable to help his accomplice, or a terrorist fraud attack exists. Hence, we define OSTF, a notion that imposes no restriction on what the prover is allowed to leak, and does not depend on the accomplice learning something. A protocol is OSTF-resistant if, regardless of the help provided by a malicious prover, $\mathscr{A}$ cannot authenticate successfully.

Part II

Using Constraint Programming for Cryptanalysis

Chapter 6

Using Constraint Programming for Cryptanalysis

Contents

The security of contactless communication protocols depends, of course, of the protocols themselves, but it is also dependent of the cryptographic primitives that are used to build them. Studying the security of these primitives is a difficult and tedious task, so that the community studies solutions to automate it [Biryukov and Nikolic, 2010, Mouha et al., 2012, Sun et al., 2014, Derbez and Fouque, 2016]. We applied Constraint Programming (CP) to automatic cryptanalysis. Constraint programming is a paradigm in which the user states the problem to be solved in a declarative manner, and leaves the resolution to a solver. As a benchmark to evaluate the efficiency of constraint programming, we chose a specific cryptographic problem: the search for optimal related key differential characteristics on block ciphers. In this chapter, we present the general methods that we used for modelling, while in the following chapters, we present concrete case studies and results.

6.1 Introduction

Contactless communication protocols, including distance bounding protocols, rely heavily on the security of the cryptographic building blocks that compose them. For instance, the security proofs for SPADE and TREAD (Chapter 4) only hold if secure encryption schemes are used. Contactless applications are usually run on devices with low computational power and energy, so that there is an incentive in minimising the number of operations they have to perform. Symmetric schemes are typically faster than asymmetric ones [Tripathi and Agrawal, 2014], so they are preferred for contactless applications. For instance, most distance bounding protocols [Brelurut et al., 2016] use symmetric schemes.

In contactless protocols, the most widely used cryptographic primitives are encryption schemes, PRFs, hash functions, and MACs (as defined in Chapter 2). Encryption is typically performed using a block cipher. A block cipher is a symmetric encryption scheme which encrypts messages (or blocks) of fixed length. Once a secret key is fixed, the block cipher defines a permutation on the message space. It is said to be secure, or to behave as a good pseudorandom permutation (PRP), if it is difficult to distinguish this permutation from a random one without being given the secret key. The other primitives used in contactless protocols can be built from a block cipher:

Hash Functions There exist several techniques to build a hash function from a block cipher, such as the Merkle-Damgård construction [Merkle, 1979]. In this construction, successive calls to a block cipher are done, in which the plaintexts are blocks of the message to be hashed, and the keys used in the encryptions are a function of the previous encryptions.

PRF A secure block cipher can directly be used as a PRF (see Theorem 1 of [Lucks, 2000]), but more advanced constructions can be used for more security, such as the PRF $Sum^2(K,K',M) = \mathsf{E.enc}_K(M) \oplus \mathsf{E.dec}_{K'}(M)$, where E is a block cipher, and K and K' are random keys [Lucks, 2000].

MAC To build a MAC from a block cipher, one can either use a specialised construction such as PMAC [Black and Rogaway, 2002], or build a hash function from the block cipher, and then use HMAC, defined as $\mathrm{HMAC}_K(M) = \mathsf{H}((K \oplus opad) || \mathsf{H}(K \oplus ipad || M))$, where $\mathsf{H}(\cdot)$ is a hash function, and $opad$ and $ipad$ are constants [Bellare et al., 1996].

Embedded devices are typically restricted by their memory. For this reason, one could imagine implementing a block cipher on such a device, and deriving the other primitives from it. However, in this case, several keys are sometimes used: for instance, the PRF Sum^2 uses two different keys, and the Merkle-Damgård construction uses several keys which are partly dependent from each other. This multiple-key use case is not covered by the classical pseudo random security notion, which only considers one fixed key. This motivates the analysis of the security of block ciphers in the related key model, in which the attacker has access to encryptions under several keys that have a relation of his choice. In the related key model, the most studied kind of attacks is related key differential cryptanalysis. Related-key differential cryptanalysis is the adaptation of differential cryptanalysis [Biham and Shamir, 1991] to the case where multiple keys are used. Differential cryptanalysis evaluates whether it is possible to distinguish the cipher from a PRP (or to recover the secret key used to encrypt messages) within a reasonable number of trials, by considering plaintext pairs (X,X') and studying the propagation of the input difference $\delta X = X \oplus X'$ between X and X' while going through the ciphering process. In other words, differential cryptanalysis exploits the fact that the probability of observing output differences given input differences is not uniformly distributed. Today, differential cryptanalysis is public knowledge, and block ciphers such as the AES have proven security bounds against differential attacks. Hence, Biham proposed to consider related-key differential attacks [Biham, 1993], in which the attacker can inject differences not only between the plaintexts X and X' but also between the keys K and K' (even though the secret key K stays unknown from the attacker). To mount related-key differential attacks, the cryptanalysts must find optimal related-key differentials, *i.e.*, input and output differences that maximise the probability of observing the output difference given the input difference. The propagation of differences through the cipher is non deterministic, so that several difference propagation paths, also called related key differential characteristics, can result in the same input and output difference. Computing the probability of a differential, due to the high number of paths that can lead to it, is difficult. However, the probability of the best differential characteristic gives a lower bound on the probability of the best related key differential attack.

Finding optimal related key differential characteristics is a highly combinatorial problem, which is usually solved using custom algorithms [Fouque et al., 2013, Biryukov and Nikolic, 2010]. Implementing such algorithms is a tedious and time consuming task, which can be error prone. On the other hand, the optimisation research community has automatic tools designed specifically to tackle combinatorial problems, and be easy to use. These tools include, among others, Mixed Integer Linear Programming (MILP), SAT solvers, or Constraint Programming (CP). In these declarative frameworks, the solving algorithm is decoupled from the problem to solve: the user describes the problem in a given formalism, and provides the resulting model to a solver, which performs the resolution. This approach has several advantages over the design of custom search tools: it is significantly easier to just describe the problem than to implement a search tool from scratch, and the solvers can be much more efficient than custom tools, since they are the result of years of improvements. Moreover, there is a strong incentive for the developers of solvers to improve them even more and make them faster, in particular with the solvers competitions that take place every year. The resolution algorithms used in these solvers are therefore highly optimised.

MILP and SAT models are used more and more widely for cryptanalysis, as they save cryptanalysts the trouble of implementing a search algorithm from scratch. While the running time is not a crucial feature for cryptanalysts, it still needs to be reasonable: during the design of a cipher, this analysis may have to be repeated several times, so that a short running time is an interesting feature. The first application of a declarative framework for the cryptanalysis of block ciphers was trying to recover the secret key used in a DES encryption with a SAT solver, in 1999 [Massacci, 1999]. Since then, SAT was mostly applied to the cryptanalysis of hash functions [Mironov and Zhang, 2006], while MILP was used for ciphers. A notable example is the differential and linear cryptanalysis of the stream cipher Enocoro-128v2 by Mouha $et al.$, using the MILP paradigm [Mouha et al., 2012]. Other authors performed cryptanalysis on block ciphers using MILP, such as [Wu and Wang, 2011], who used it to perform differential cryptanalysis against several block ciphers (CAST256, SMS4, Clefia, Misty, Skipjack, Mars, Fourcell), or [Sun et al., 2014], in which MILP is used to find related key differential characteristics on SIMON, PRESENT, LBLOCK and DESL. One well known difficulty when evaluating block ciphers with MILP is that MILP is designed to solve linear equations, while block ciphers include non linear part. In particular, a component used to introduced non linearity, the substitution boxes (SBoxes), is difficult to model in MILP, since describing it in a linear way requires a huge number of equations. When the SBoxes operate on 4 bits, it is still reasonably easy to model them with MILP, but larger SBoxes have been known to be more difficult. A new representation introduced in [Abdelkhalek et al., 2017] made the modelling of 8 bit SBoxes possible. More recently, MILP was used to search for impossible differentials, *i.e.*, differentials which occur with probability 0, in [Sasaki and Todo, 2017].

While MILP and SAT are quite popular in the cryptography community, CP received very little attention. The small representation of CP for cryptanalysis in the literature is surprising: MILP is limited to solving constraints described as linear inequalities, and SAT is limited to Boolean formulas, while the CP formalism generalises the two former methods and imposes no restrictions on the constraints that can be expressed, which makes it easier to use. In addition, there exists methods to convert a CP problem to a MILP or SAT problem automatically. In particular, MILP (such as Gurobi [Gurobi Optimization, 2016]) and SAT (for instance Picat_Sat) solvers can interpret CP models written in the MiniZinc [Nethercote et al., 2007] constraint programming language. However, to the best of our knowledge, apart from our work, CP was only used for two cryptography-related tasks. It was used to automate the generation of SBoxes with optimal properties [Ramamoorthy et al., 2011]. In this work, a CP model with constraint enforcing several design criterion is proposed, and this model generates all the optimal SBoxes within a few seconds. CP was also used to perform side-channel cryptanalysis on AES in two works. The first one [Bonneau, 2006] considers the information obtained by cache look-ups that occur during AES operations, and builds a CP models relating trace data to key bits, in order to recover the key. The second one [Liu et al., 2017] uses CP to perform Tolerant Algebraic Side-Channel Analysis [Oren et al., 2010] against AES.

We explored the capability of CP to tackle cryptanalysis problems, and in particular, the problem of finding optimal related-key differential characteristics for block ciphers.

In this chapter, we explain how to model the search and enumeration of optimal related-key differential characteristics with constraint programming, and define the notations that we use in the following chapters. We first recall the definition of block ciphers in Section 6.2.1, and then we introduce some notions about related key differential cryptanalysis in Section 6.2. Then, after recalling what constraint programming (Section 6.3) is, we give the general technique we used for modelling the problem of finding optimal related key differential characteristics in Section 6.4.

6.2 Related Key Differential Cryptanalysis

In this section, we first recall the definition of block ciphers and their security, and then present related key differential cryptanalysis.

6.2.1 Block Cipher Security

Two famous examples of block ciphers are DES (Data Encryption Standard), which was the encryption standard between 1977 and 2000, and AES [Daemen and Rijmen, 2002] which is the actual standard since 2001.

Block ciphers are symmetric encryption schemes that encrypt blocks of fixed length. More formally, a block cipher is a function $\mathrm{E} : \mathbb{X} \times \mathbb{K} \to \mathbb{X}$, where $\mathbb{X} = \{0,1\}^n$ is a message space, and $\mathbb{K} = \{0,1\}^l$ is a key space. Given a binary block $\mathrm{X} \in \mathbb{X}$ (called plaintext) of length n and a binary key $\mathrm{K} \in \mathbb{K}$ of length l, outputs a binary ciphertext $\mathsf{E.enc}_{\mathrm{K}}(\mathrm{X}) \in \mathbb{X}$ of length n such that $\mathrm{X} = \mathsf{E.dec}_{\mathrm{K}}(\mathsf{E.enc}_{\mathrm{K}}(\mathrm{X}))$.

Most of today's block ciphers have an iterated structure: They apply a round function f r times so that $\mathsf{E.enc}_{\mathrm{K}}(\mathrm{X}) = \mathrm{X}_r$ with $\mathrm{X}_0 = \mathrm{X}$ and $\mathrm{X}_{i+1} = f(\mathrm{X}_i, \mathrm{K}_{i+1})$ for all $i \in [0; r-1]$.

The ciphers we studied in this thesis follow the Substitution Permutation Network (SPN) structure. The round function of the block ciphers following this structure is composed of a linear permutation layer, as well as a non-linear substitution layer. This non linear layer is implemented as SBoxes. A SBox is a substitution table, which maps bitstrings of length m to bitstrings of length w. The cipher that we studied use bijective SBoxes, so that $m = w$, and they are word oriented: they group bits as words, which are either nibbles ($m = 4$) or bytes ($m = 8$).

In the single-key scenario, a block cipher is said to be secure if it behaves as a good pseudorandom permutation (PRP). In other words, it is secure if it is difficult to distinguish it from a random permutation. Let $Perm(\mathbb{D})$ denote the set of all permutations over a domain $\mathbb{D}$. The two following definitions are inspired from [Bellare and Kohno, 2003]. In these definitions, concrete (as opposed to asymptotic) security notions are used, as in a block ciphers, the key size l is fixed. Hence, security is considered with respect to adversaries having practical resources, in terms of running time and oracle queries.

Definition 33 (PRP Experiment for a Block Cipher)**.** *Let* E *be a block cipher, and* $\mathsf{Adv}_{\mathsf{E}}^{\mathrm{PRP}}(l)$ *denote the advantage of an adversary against the PRP experiment. A block cipher is PRP secure if* $\mathsf{Adv}_{\mathsf{E}}^{\mathrm{PRP}}(l)$ *is small, for any adversary limited to a practical number of queries. This advantage is defined as* $Pr\left[\mathrm{K} \xleftarrow{\$} \mathbb{K} : \mathscr{A}^{\mathsf{E.enc}_{\mathrm{K}}(\cdot)} = 1\right] - Pr\left[g \xleftarrow{\$} Perm(\mathbb{M}) : \mathscr{A}^{g(\cdot)} = 1\right]$.

The PRP experiment can also be defined in the related key model [Bellare and Kohno, 2003]. The model defined in [Bellare and Kohno, 2003] allows the adversary to select among several relations between the keys, but since our work focuses on XOR differences, we adopt a simpler, though less generic, definition. Let $\mathsf{E.enc}_{\mathrm{RK},(\delta\mathrm{K},\mathrm{K})}(\mathrm{X})$ be an oracle that encrypts a message X with the key $\mathrm{K} \oplus \delta\mathrm{K}$. Let $Perm(\mathbb{K}, \mathbb{M})$ denote the set of block ciphers with key space $\mathbb{K}$ and message space $\mathbb{M}$.

Definition 34 (PRP-RKA Experiment for a Block Cipher)**.** *Let* E *be a block cipher, and* $\mathsf{Adv}_{\mathsf{E}}^{\mathrm{PRP-RKA}}(l)$ *denote the advantage of an adversary against the PRP-RKA experiment. A block cipher is PRP-RKA secure if* $\mathsf{Adv}_{\mathsf{E}}^{\mathrm{PRP-RKA}}(l)$ *is small, for any adversary limited to a practical number of queries. This advantage is defined as* $Pr\left[\mathrm{K} \xleftarrow{\$} \mathbb{K} : \mathscr{A}^{\mathsf{E.enc}_{\mathrm{RK},(\cdot,\mathrm{K})}(\cdot)} = 1\right] - Pr\left[\mathrm{K} \xleftarrow{\$} \mathbb{K}, g \xleftarrow{\$} Perm(\mathbb{K}, \mathbb{M}) : \mathscr{A}^{g(\mathrm{K},\cdot)} = 1\right]$.

6.2.2 Related Key Differential Cryptanalysis

In a related key differential attack, the cryptanalyst has access to $\mathsf{E.enc}_{\mathrm{RK},(\delta\mathrm{K},\mathrm{K})}(\mathrm{X})$, as defined in Definition 34. He can query the encryption of any message X of his choice, under both the secret key K and $\mathrm{K}' = \mathrm{K} \oplus \delta\mathrm{K}$, for a δK of his choice. His goal is to distinguish E from a random keyed permutation. Note that recovering the key permits $\mathscr{A}$ to trivially distinguish, by comparing $\mathsf{E.enc}_{\mathrm{K}}(\mathrm{X})$ with the output of the oracle for the encryption of X, *i.e*, $\mathsf{E.enc}_{\mathrm{RK},(0,\mathrm{K})}(\mathrm{X})$. When trying to recover the key, the cryptanalysis is considered successful if he can do it more efficiently than by exhaustive search. Exhaustive search is the strategy of requesting the encryption of a message X with the key K, and then computing $\mathsf{E.enc}_k(\mathrm{X}) : k \in \mathbb{K}$ for each of the 2^l candidate keys in $\mathbb{K}$, and checking which ciphertext matches. In general, it is not possible to mount successful cryptanalysis on the full version of a block cipher, and cryptanalysts attack round reduced versions.

The first step to perform a related key differential cryptanalysis is to obtain related key differential characteristics. A related key differential characteristics c describes the propagation of an initial difference in the plaintext (δX) and in the key (δK) through the cipher, as well as the probability $p(c)$ that a given output difference $\delta\mathrm{X}_r$ is observed after r rounds of the encryption process. The cryptanalyst can expect to observe $\delta\mathrm{X}_r$ once encrypting $\frac{1}{p(c)}$ messages with the correct input difference. The number of encryptions is generally raised to $\mathrm{T} \cdot \frac{1}{p(c)}$ (where T is a constant), in order to maximise the probability of observing the correct output difference. Hence, the number of operations required to distinguish E from a random permutation depends on the probability $p(c)$.

We denote δA the *differential matrix* obtained by applying the XOR operator on two matrices and, for every row j and column k, $\delta\mathrm{A}[j][k]$ is called a *differential word* (or, depending on the context, *differential byte* or *differential nibble*). Differential words are denoted with a lower case letter, *e.g.*, δa, to avoid confusion with differential matrices. The set of all differential words, which represent the inputs, outputs and intermediary differences through the cipher, is denoted $diffWords_l$. Among these differential words, some of them pass through Sboxes, and we denote *Sboxes* this set.

The first observation is that, in a differential context, XOR operation with constants C do not need to be considered, since they are cancelled: $a \oplus \mathrm{C} \oplus a' \oplus \mathrm{C} = a \oplus a' = \delta a$

The propagation of a difference δA through a linear operation L is deterministic: for all differential matrix $\delta\mathrm{A} = \mathrm{A} \oplus \mathrm{A}'$, it holds that $\mathrm{L}(\mathrm{A}) \oplus \mathrm{L}(\mathrm{A}') = \mathrm{L}(\mathrm{A} \oplus \mathrm{A}')$, by definition. On the other hand, the propagation of a difference through a SBox S is not deterministic. Given two bitstrings a and a', $\mathrm{S}(a) \oplus \mathrm{S}(a')$ is not necessarily equal to $\mathrm{S}(a \oplus a')$.

Therefore, given an input differential words $\delta a = a \oplus a'$, we cannot deterministically compute the output difference after passing through Sboxes: we can only compute probabilities. More precisely, for every couple of differential words $(\delta a_{in}, \delta a_{out}) \in [0;n]^2$, we can compute the probability that the input difference δa_{in} becomes the output difference δa_{out}, which is the proportion of couples (a, a') such that $\delta a_{in} = a \oplus a'$ and $\delta a_{out} = \mathrm{S}(a) \oplus \mathrm{S}(a')$. More precisely, this probability is denoted $p_{\mathrm{S}}(\delta a_{out} | \delta a_{in})$ and is defined by

$$p_{\mathrm{S}}(\delta a_{out}|\delta a_{in}) = \frac{\#\{(a,a') \in [0;n]^2 \mid (a \oplus a' = \delta a_{in}) \land (\mathrm{S}(a) \oplus \mathrm{S}(a') = \delta a_{out})\}}{2^m} \tag{6.1}$$

To compute the probability of a given valuation of all differential words, we first have to check that all linear operators are satisfied by the valuation. If this is not the case, then the probability is equal to 0. Otherwise, the probability of a differential characteristic c, denoted $p(c)$, is equal to the product of the transition probabilities of all differential words that pass through SBoxes, *i.e.*, :

$$p(c) = \prod_{\delta a \in Sboxes} p_{\mathrm{S}}(\delta a_{out}|\delta a_{in}) \tag{6.2}$$

In the ciphers we studied, the SBoxes are bijective. This implies that whenever $\delta a = 0$ (*i.e*, $a = a'$), $\mathrm{S}(a) \oplus \mathrm{S}(a') = a \oplus a' = 0$. Stated differently, $p_{\mathrm{S}}(0|0) = 1$, and $p_{\mathrm{S}}(0|\delta a_{in}) = 0$ if $\delta a_{in} \neq 0$. The SBoxes which have a null input difference are *inactive*, and the SBoxes with non zero input difference are said to be *active*. Only the active SBoxes lower the probability of the differential characteristic, since inactive SBoxes are passed with probability 1. Hence, optimal differential characteristics typically have a low number of active SBoxes.

We refer the reader to [Biham and Shamir, 1991] for more details on differential characteristics.

6.3 Constraint Programming

In this Section, we briefly recall basic principles of CP and we refer the reader to [Rossi et al., 2006] for more details.

CP is used to solve Constraint Satisfaction Problems (CSPs). A CSP is defined by a triple (X, D, C) such that X is a finite set of variables, D is a function that maps every variable $x_i \in X$ to its domain $D(x_i)$ (that is, the finite set of values that may be assigned to x_i), and C is a set of constraints (that is, relations between some variables which restrict the set of values that may be assigned simultaneously to these variables).

Constraints may be defined in extension, by listing all allowed (or forbidden) tuples of the relation, or in intention, by using mathematical operators. Let us consider for example a CSP with $X = \{x_1, x_2, x_3\}$ such that $D(x_1) = D(x_2) = D(x_3) = \{0, 1\}$, and let us consider a constraint that ensures that the sum of the variables in X is different from 1. This constraint may be defined by a table constraint that enumerates all allowed tuples:

$$(x_1, x_2, x_3) \in \{(0,0,0), (0,1,1), (1,0,1), (1,1,0), (1,1,1)\}.$$

Conversely, it may be defined by enumerating all forbidden tuples:

$$(x_1, x_2, x_3) \not\in \{(1,0,0), (0,1,0), (0,0,1)\}.$$

Finally, it may be defined by using arithmetic operators: $x_1 + x_2 + x_3 \neq 1$.

Solving a CSP involves assigning values to variables such that all constraints are satisfied. More formally, an *assignment* $\mathscr{A}$ is a function which maps each variable $x_i \in X$ to a value $\mathscr{A}(x_i) \in D(x_i)$. An assignment $\mathscr{A}$ *satisfies* (resp. *violates*) a constraint $c \in C$ if the tuple defined by the values assigned to the variables of c in $\mathscr{A}$ belongs (resp. does not belong) to the relation defined by c. An assignment is *consistent* (resp. *inconsistent*) if it satisfies all the constraints (resp. violates some constraints) of the CSP. A *solution of a CSP* is a consistent assignment.

An *objective function* may be added to a CSP, thus defining a Constrained Optimisation Problem (COP). This objective function is defined on some variables of X and the goal is to find the solution that optimises (minimises or maximises) the objective function. This solution is said to be *optimal*.

CP languages provide high-level features to define CSPs and COPs in a declarative way.

Then, these problems are solved by generic constraint solvers which are usually based on a systematic exploration of the search space: Starting from an empty assignment, they incrementally extend a partial consistent assignment by choosing a non-assigned variable and a consistent value for it until either the current assignment is complete (a solution has been found) or the current assignment cannot be extended without violating constraints (the search must backtrack to a previous choice point and try another extension). To reduce the search space, this exhaustive exploration of the search space is combined with constraint propagation techniques: Each time a variable is assigned to a value, constraints are propagated to filter the domains of the variables that are not yet assigned, *i.e,*, to remove values that are not consistent with respect to the current assignment. When constraint propagation removes all values from a domain, the search must backtrack.

Let us consider for example the constraint that ensures that the sum of three variables is different from 1. Whenever two of these variables are assigned and their sum is equal to 1 (resp. 0), the propagation of this constraint removes the value 0 (resp. 1) from the domain of the third variable.

A key point to speed up the solving process of CSPs is to define the order in which variables are assigned, and the order in which values are assigned to these variables, when building the search tree. CP languages allow the user to specify this through variable and value ordering heuristics.

6.4 Modelling the Search for Optimal Related Key Differential Characteristics on Word Oriented Block Ciphers With CP

In this section, we present the strategies that we used to perform the search for optimal related-key differential characteristics.

6.4.1 Search algorithm

Finding and enumerating the optimal related key differential characteristics for a block cipher is a highly combinatorial task. It requires finding the difference propagation path, such that the plaintext difference is δX, the key difference is δK, and the difference after r rounds is δX_r. The search space is very large: typical values for the block size n and the key size l are 64, 128 or even more bits. To make the search easier, it can be decomposed into two steps[Biryukov and Nikolic, 2010, Fouque et al., 2013].

In a first step, each differential word $\delta a \in$ *diffWords* is abstracted with a Boolean variable (or differential bit) Δa such that $\Delta a = 0 \Leftrightarrow \delta a = 0$ and $\Delta a = 1 \Leftrightarrow \delta a \in [1, 2^n - 1]$. We denote the fact that Δa is a Boolean abstraction of δa with the symbol $\prec$, *i.e.*, $\Delta a \prec \delta a$. In other words, each Boolean variable assigned to 1 gives the position of a difference. The goal is to find all abstracted differential characteristics that optimise some objective function:

- an abstracted differential characteristic is an assignment of the Boolean variables associated with the differential words of *diffWords that satisfies the Boolean abstraction of the cipher operations;*
- *the objective function aims at minimising the number of differences passing through SBoxes (i.e., minimising* $\sum_{\delta a \in Sboxes} \Delta a$*).*

In a second step, for each abstracted differential characteristic, we search for actual values of the *diffWords* variables that satisfy the difference propagation rules and that maximise the probability $p(c)$ defined in Eq. (6.2). When a Boolean variable Δa is equal to 0 in the abstracted differential, there is only one possible value for δa, which is 0. However, when $\Delta a = 1$, there are $2^l - 1$ possible values for δa. Note that some abstracted differential characteristics are not valid and cannot be transformed into word solutions because abstracted differential characteristics only satisfy Boolean abstractions of the actual cipher operations. These abstracted differential characteristics are said to be *word-inconsistent*. It may also happen that the maximal probability $p(c)$ is such that it is possible to have a greater probability with a larger number v of differences that pass through SBoxes. In this case, we need to search for new abstracted differentials, with a larger number of differences.

More precisely, the complete procedure to find optimal differential characteristics is described in Algorithm 1. It first calls function *Step1-opt* to compute the minimal number v^* of differences passing through S-boxes in an abstracted differential characteristic (line 2), and it initialises v to this minimal number of differences. Then, it calls the function *Step1-enum* to compute the set T of all abstracted differential characteristics such that the number of differences passing through S-boxes is equal to v (line 6). For each abstracted differential $t \in T$, it calls function *Step2* (line 8): if t is not word-consistent, *Step2* returns $null$; otherwise, it returns the optimal differential characteristics c associated with t. If this optimal differential characteristics has a greater probability than c^*, then it updates c^* (line 9). Lines 6 to 10 are repeated with increasing values of v until a differential characteristics has been found, and it is not possible to obtain a better differential characteristics with a higher value of v.

6.4.2 Modelling the Cipher Operations

In this section, we describe the modelling tricks that we used for the operations of the cipher. The detail of the modelling are cipher dependant, and are presented in the following chapter. However, some general guidelines can be defined. Indeed, many block ciphers use XOR operations,

Algorithm 1: Computation of optimal differential characteristics

```
Input: The size $l$ of the key and the number $r$ of rounds
Output: An optimal differential characteristics $c^*$
1  begin
2      $v^* \leftarrow$ Step1-opt$(l, r)$
3      $v \leftarrow v^*$
4      $c^* \leftarrow null$
5      repeat
6          $T \leftarrow$ Step1-enum$(l, r, v)$
7          for each abstracted differential $t \in T$ do
8              $c \leftarrow$ Step2$(l, r, t)$
9              if $c \neq null$ and $(c^* = null$ or $p(c) > p(c^*))$ then $c^* \leftarrow c$;
10         $v \leftarrow v + 1$
11     until $c^* \neq null$ and $p(c^*) \geq min_p$;
12     return $c^*$
```

so we describe how we modelled them. SPN ciphers also use SBoxes, and we describe how to model them with CP. Finally, AES and other block ciphers inspired by it use an operation called MixColumn. While this operation cannot be accurately represented in Step 1, we present tricks that exploits the properties of MixColumns to filter out inconsistent solutions during Step 1.

First Step

The advantage of decomposing the problem in two steps is that it greatly reduces the search space. It additionally permits to report the study of the SBox operation to Step 2, for which the number of potential solutions is greatly filtered by Step 1. Indeed, as mentioned earlier, for bijective SBoxes, $S(a) \oplus S(b) = 0$ if $a = b$, and that $S(a) \oplus S(b) \neq 0$ otherwise, so that the output difference after a SBox is 0 if the input difference is 0, and nonzero otherwise. Hence, for the abstraction of the SBox operation in the Boolean domain is simply

$$\mathrm{SB}(\Delta a_{in}, \Delta a_{out}) \equiv \Delta a_{in} = \Delta a_{out}.$$

For this reason, the SBox operation is simply modelled as the identity operation in the first step.

On the other hand, some branching is introduced by the abstraction of the XOR operation: When the words are abstracted to bits, we lose the information of whether two words are equal. Consider the following operation: $\delta x = \delta a \oplus \delta b$. When abstracting it to the Boolean domain, we lose information about whether δa and δb are equal, and we obtain the following three cases:

- If $\Delta a = \Delta b = 0$, then $\Delta x = 0$, because $0 \oplus 0$ is zero.
- If $\Delta a \neq \Delta b$, then $\Delta x = 1$, because $0 \oplus x$, with $x \neq 0$, is always nonzero.
- If $\Delta a = \Delta b = 1$, then Δx can be either 0 (if $\delta a = \delta b$) or 1 otherwise.

This is expressed by the following constraint:

$$\mathrm{XOR}(\Delta a, \Delta b, \Delta x) \equiv \Delta a + \Delta b + \Delta c \neq 1.$$

where + is the integer addition.

Similarly, when k variables are XORed together, *i.e*, $\delta x = \bigoplus_{i \in [1,k]} \delta a_i$, we have that if all the differential bits representing the differential words δa_i are zero, then Δx is 0. If only one of these bits is nonzero, then Δx is 1. If there is more than one nonzero bit, then the result is undetermined.

$$\mathrm{XOR}(\{\Delta a_i | i \in [1,k]\}, \Delta x) \equiv \sum_{i=1}^{k} (\Delta a_i) + \Delta x \neq 1.$$

Finally, the ciphers we studied include a linear transformation called MixColumns (MC), which operates on each column of the differential matrices independently. These columns contain 4 rows for all the ciphers we studied. Let A denote a column, and MC(A) its image by the MC operation. The MC operation is chosen for its diffusion properties: For each column A, it ensures that the number of bytes that are different from zero in A and MC(A) is either 0 or greater or equal to its branch number b, *i.e.*,

$$(\sum_{j=0}^{3}(A[j] \neq 0) + (MC(A)[j] \neq 0)) \in \{0, b, b+1, \dots, 8\}.$$

If $b = 4$, the MC operation has the *quasi MDS* property, and if $b = 5$, then the MC operation has the *MDS* property, where MDS stands for Maximum Distance Separable.

MC cannot be modelled precisely at the Boolean level, as knowing where differences hold in δA is not enough to determine where they hold in $\delta MC(A)$. Indeed, the values of the words in the column are necessary to determine the positions of the zeros after MC. However, the MDS, or quasi MDS property, is modelled by constraining the number of differences in δA and $\delta MC(A)$ to be equal to 0 or greater than B.

However, the MDS or quasi MDS property of MC also holds for the XOR difference between two columns of differential bytes:

$$(\sum_{j=0}^{3}((\delta A[j] \oplus \delta B[j]) \neq 0) + ((\delta MC(A)[j] \oplus \delta MC(B)[j]) \neq 0)) \in \{0, b, b+1, \dots, 8\}.$$

Considering the branch number for the XOR of two columns of differential words permits to greatly reduce the number of byte inconsistent solutions. Indeed, we can infer equalities or differences between δA and δB from other parts of the cipher, for instance the XOR operations. More details about these inferences are given in Section 7.3.2.

Second Step

The second step is a straightforward implementation of the difference propagation rules for the cipher at word level. It includes constraints to define the Sbox operation, as well as the XOR operation, and the permutations. MixColumns is modelled by combining XOR operations. For AES and Rijndael, it also requires to implement multiplication in a finite field: the constraints corresponding to this multiplication are described in the corresponding chapter.

For XOR, $SBox$, and multiplication, we use table constraints, which extensively list the allowed tuples. A table constraint is defined with regards to a set of allowed n-tuples tuples, and a tuple of variable $t = (v_1, \dots, v_n)$. The constraint $table(t, \text{tuples})$ enforces the relation $t \in \text{tuples}$. For SBox, we use a ternary table constraint which lists all triples (X, Y, P) such that there exists two words w_1 and w_2 whose difference before and after passing through SBoxes is equal to X and Y, respectively, and such that p_S is the probability of this transformation: For all $\delta w \in$ *Sboxes*, we add the constraint

$$\begin{aligned}(\delta w_{in}, \delta w_{out}, p) \in \{(X, Y, p) \quad | \quad & \exists (w_1, w_2) \in [0, 2^l - 1] \times [0, 2^l - 1], X = w_1 \oplus w_2, \\ & Y = S(w_1) \oplus S(w_2), p = \log_2(-p_{S(\delta w_{out} | \delta w_{out})}\}.\end{aligned}$$

Using the base 2 logarithm allows us to replace the product of probabilities by a sum of integers, which is easier to model.

Similarly, for the XOR operations ($\delta a \oplus \delta b = \delta c$), we use a ternary table constraint which extensively lists all triples (A, B, C), such that $A \oplus B = C$. The corresponding tuples are called tupleXOR:

$$TX = \{(A, B, C) : (A, B, C) \in [0; 2^m - 1]^3, A \oplus B = C\},$$

where m is the word size used in the cipher (typically 4 or 8 bits).

Objective function

The same model is used to solve two problems, *i.e.*, *Step1-opt* and *Step1-enum*. The difference between these problems is in their goals.

For *Step1-opt*, the goal is to find the minimum number of differences passing through SBoxes, *i.e.*, the number of ΔB variables that are associated with differential words in $Sboxes_l$ and that are assigned to 1. Thus, we define an integer variable obj_{Step1} which is constrained to be equal to the number of differences passing through SBoxes:

$$obj_{Step1} = \sum_{\delta B \in Sboxes} \Delta B.$$

The domain of this variable is $D(obj_{Step1}) = [1, \frac{l}{6}]$. Indeed, the smallest possible value is 1 because we need to have at least one active SBox to have a differential characteristics (we forbid the obvious solution such that δX and δK only contain words set to 0, meaning that there is no difference in the initial plaintext and key). The largest possible value is $\frac{l}{6}$ because the highest probability $p_S(\delta_{out}|\delta_{in})$ to pass through the AES SBox is $2^{-6} = \frac{4}{256}$ when $\delta_{in} \neq 0$ [Daemen and Rijmen, 2002], and because we want a differential characteristics which is more efficient than the key exhaustive search (*i.e.*, the probability of which is greater than 2^{-l}). The objective function is to minimise obj_{Step1}

For *Step1-enum*, the goal is to enumerate all abstracted differentials when obj_{Step1} is assigned to a given value v, and we simply ask the solver to enumerate the solutions that satisfy the constraint.

Ordering heuristics

As the goal is to minimise the number of active SBoxes (for *Step1-opt*) or enumerate all abstracted differentials with a small number of active SBoxes (for *Step1-enum*), we define ordering heuristics as follows: first assign variables associated with bytes that pass through SBoxes (those in *Sboxes*), and first try to assign them to 0.

Tools and Experimental Setup

There exists a wide range of CP solvers, as well as SAT and MILP solvers. In preliminary experiments [Minier et al., 2014], both steps were performed with Choco [Prud'homme et al., 2016], a CP solver based on the java programming language. The problem that was studied was the search for related key differential characteristics on AES-128, and is presented in the next chapter. However, the resolution of Step 1 scaled poorly with Choco, even though the times for Step 2 were good. To remedy this, we investigated the use of different solvers, by implementing our model in the MiniZinc language [Nethercote et al., 2007]. MiniZinc is a dedicated language, which is interpreted (after compilation) by many CP solvers, as well as some SAT and MILP solvers: the whole list of compatible solvers can be found on minizinc.org. We ran experiments with Gecode [Gecode Team, 2006], Choco [Prud'homme et al., 2016], Chuffed [Chu and Stuckey, 2014], and Picat_SAT [Zhou et al., 2015]. The solver Chuffed [Chu and Stuckey, 2014] performed very well on the small instances of our problems [Gerault et al., 2016], but for larger instances, it was not sufficient. In particular, while our problem was solved efficiently for AES-128, the instances related to AES-192 were problematic. We switched to Picat_SAT after submitting our models to a competition of solvers called the MiniZinc challenge, in 2016, and noticing that it obtained particularly good results. In fact, Picat_SAT was consistently better than Chuffed, or at least as good, for solving our models. Hence, we use it to perform Step 1 in all the experiments described in the rest of this thesis. Note that Picat_SAT actually uses a SAT solver to solve CSPs: It first translates the CSP instance into a Boolean satisfiability formula, and then uses the SAT solver Lingeling [Biere, 2014] to solve it.

On the other hand, Step 2 is solved fast enough by Choco for all our instances. Moreover, it can be parallelized: each Step 1 solution can be solved in Step 2 independently. Hence, Step 2 was never a bottleneck, and we did not investigate its resolution with other solvers.

All our experiments were performed on a server with a 24-core Intel(R) Xeon(R) E5-2687Wv4 @ 3.00GHz CPU, and 768 Gb of RAM. We used version 2.1.7 of the MiniZinc suite, and Picat version 1.9#6.

When we report our results, all times are given in seconds unless otherwise specified.

6.5 Conclusion

In this chapter, we presented the problem of finding optimal related key differential characteristics on block ciphers, *i.e.*, relations between input and output differences which hold with maximal probability, and defined the associated notations. Additionally, we presented the general modelling techniques that we used, including a two-step resolution, and the use of table constraints to model the SBoxes at word level. The following chapters instantiate our techniques to two block ciphers: AES and Midori.

Chapter 7

Related Key Differential Cryptanalysis of AES with CP

Contents

The AES is the most widely used block cipher since its standardisation in 2001. In this chapter, we present the methods and results of our related key cryptanalysis on it using constraint programming. Using the methods described in the previous chapter, we solve the problem within some hours even for the hardest instances, whereas the state of the art method require several weeks. Additionally, we disprove some results that were claimed optimal in the literature.

7.1 Introduction

In 1997, a call for proposals was launched in order to replace the DES (Data Encryption Standard), which had been in use since the 1970's, and had become insecure. One of the candidates, Rijndael [Daemen and Rijmen, 2002], was modified and adopted as the AES (for Advanced Encryption Standard) in 2001. While the original in algorithm, Rijndael, several block sizes $n \in \{128, 160, 192, 224, 256\}$ could be chosen, independently of the key size $l \in \{128, 160, 192, 224, 256\}$, the AES only supports one block size (128 bits) and three key sizes: 128, 192 and 256 bits.

After almost two decades of active cryptanalysis, the community failed to find any significant threat to the security of the AES in the single-key setting. The best published single-key attacks on full-round AES [Tao and Wu, 2015] are biclique attacks. Biclique attacks on block ciphers [Bogdanov et al., 2011] permit to extend the number of rounds attacked by meet-in-the middle attacks, in which the cipher is considered as two independent subciphers. The biclique attacks on AES only give a very small advantage over exhaustive search: their complexity, for AES-128, -192 and -256, are respectively 2^{126}, $2^{189.9}$ and $2^{254.3}$. The difficulty to find attacks in the single-key model motivated the community to investigate the related-key setting, which seems more promising. There exist related-key attacks on the full-round versions of AES-192 and AES-256, and on 9 rounds (out of 10) of AES-128 [Biryukov and Nikolic, 2010].

To this day, only two papers (apart from ours) perform an automatic search for related key differential characteristics on the AES. The first one, by Biryukov $et al.$ [Biryukov and Nikolic, 2010], uses a branch and bound approach, which is a variant of Matsui's algorithm [Matsui, 1995], in order to solve the first step of the search, *i.e.*, to enumerate the optimal abstracted differential characteristics. Matsui's algorithm was initially designed to find single-key differential characteristics iteratively, starting from one-round characteristics and building up to r-rounds, using the results from the i previous iterations to bound the probability of the best characteristic for round $i+1$. With this approach, they solve *Step1-opt* (from Algorithm 1) in several days for AES-128, and several weeks for AES-192. Their give no time for AES-256. The second work [Fouque et al., 2013], by Fouque $et al.$, represents the possible differential paths as a graph, and uses a graph traversal algorithm in order to obtain the optimal path. However, for AES-128, the size of the graph is around 60 GB, so that the authors did not apply their method to AES-192 and AES-256. Building the graph takes around 30 minutes on a 12-core machine, and the optimal path for AES-128 is found within one hour after the graph is built.

We model this problem with constraint programming for the three versions of AES. We build up from a straightforward, naive model, which is very inefficient and finds a lot of word-inconsistent solutions, to a final model that includes some advanced reasoning on the properties of the operations of the AES, and which greatly reduces the number of word-inconsistent solutions, while being much faster. With this last model, coupled with a different decomposition strategy, we enumerate the optimal abstracted related key differential characteristics within less than one hour for AES-128, less than 4 hours for AES-192, and less than 2 minutes for AES-256, on a single core of an Intel(R) Xeon(R) E5-2687Wv4 @ 3.00GHz CPU. Interestingly, both [Biryukov and Nikolic, 2010] and [Fouque et al., 2013] find that the optimal 4-round related key differential characteristic for AES-128 has 13 active SBoxes, whereas we prove that the optimal one actually had only 12 active SBoxes. Similarly, our CP model finds an inconsistency in the related key differential characteristic claimed to be optimal for AES-192 in [Biryukov and Khovratovich, 2009], and obtains a better related key differential for 14 rounds of AES-256 than the one claimed to be optimal in [Biryukov and Khovratovich, 2009].

In this chapter, we first describe the AES in Section 7.2, and then our CP models for Step 1 in Section 7.3. We then present the model for Step 2 in section 7.4, and our decomposition technique in Section 7.5.

7.2 The AES

AES ciphers blocks of length $n = 128$ bits, where each block is seen as a 4×4 matrix of 8-bit words, or bytes. Given a 4×4 matrix of bytes M, we note $M[j][k]$ the byte at row $j \in [0,3]$, and column $k \in [0,3]$. The plaintext and key are transformed from strings of words to matrices by ordering them in columns, as shown on Figure 7.2.

The length of keys is $l \in \{128, 192, 256\}$ bits, and we note AES-l the AES with keys of length l. The initial key K contains KC columns. Each column is 4 bytes, or 32 bits, so that $KC = \frac{l}{32}$: there are 4 columns for AES-128, 6 for AES-192, and 8 for AES-256. The only difference when changing the length l of the key is in the KeySchedule operation. In what follows, we illustrate AES and describe our cryptanalysis models.

Like most of today's block ciphers, AES is an iterative process which is composed of r rounds. The number of rounds r depends on the key length: $r = 10$ (resp. 12 and 14) when $l = 128$ (resp. 192 and 256). An AES round has an SPN (Substitution-Permutation Network) structure and is described in Figure 7.1 for $l = 128$. Before the first round, AddRoundKey (ARK) is applied on the

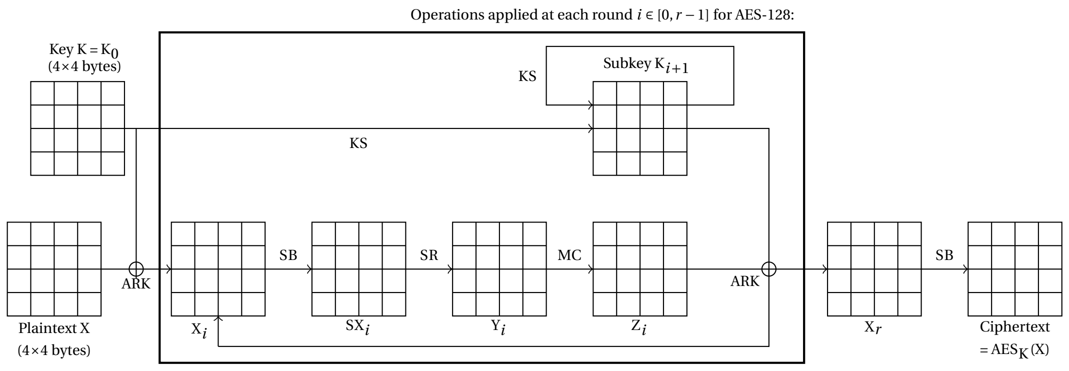

Figure 7.1: AES ciphering process for 128 bit keys. Each 4 × 4 array represents a group of 16 bytes. Before the first round, X_0 is obtained by applying ARK on the initial text X and the initial key $K = K_0$. Then, for each round $i \in [0, r-1]$, SB is applied on X_i to obtain SX_i, SR is applied on SX_i to obtain Y_i, MC is applied on Y_i to obtain Z_i, KS is applied on K_i to obtain K_{i+1}, and ARK is applied on K_{i+1} and Z_i to obtain X_{i+1}. Finally, the ciphertext is obtained by applying SB on X_r.

$$M = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

Figure 7.2: The representation of a block in the AES, where M is a matrix, and m is a string of words.

original plaintext X and the initial key $K_0 = K$ to obtain $X_0 = \text{ARK}(X, K_0)$. Then, for each round $i \in [0, r-1]$: SubBytes (SB) is applied on X_i to obtain $SX_i = S(X_i)$; ShiftRows (SR) is applied on SX_i to obtain $Y_i = SR(SX_i)$; MixColumns (MC) is applied on Y_i to obtain $Z_i = MC(Y_i)$; KeySchedule (KS) is applied on K_i to obtain $K_{i+1} = KS(K_i)$; and AddRoundKey (ARK) is applied on Z_i and K_{i+1} to obtain $X_{i+1} = \text{ARK}(Z_i, K_{i+1})$. Finally, after r rounds, the ciphertext is obtained by applying SubBytes on X_r.

Let us now describe each of these AES operations.

SubBytes (SB) SB, also called SBox, is a non-linear permutation which is applied on each byte of X_i separately, according to a look-up table $S : [0, 255] \rightarrow [0, 255]$, *i.e.*,

$$\forall i \in [0, r-1], \forall j, k \in [0, 3], SX_i[j][k] = S(X_i[j][k]).$$

ShiftRows (SR) SR rotates on the left by one (resp. two and three) byte position the second (resp. third and fourth) row of SX_i, *i.e.*, :

$$\forall i \in [0, r-1], \forall j, k \in [0; 3], Y_i[j][k] = SX_i[j][(k + j)\%4],$$

where % is the modulo operator that returns the rest of the euclidean division.

MixColumns (MC) MC multiplies each column of the input matrix Y_i by a 4 × 4 fixed matrix M:

$$\forall i \in [0, r-1], \forall j, k \in [0; 3], Z_i[j][k] = \bigoplus_{x=0}^{3} M[j][x] \cdot Y_i[x][k],$$

where · is a finite field multiplication operator. The matrix M is chosen for its good properties of diffusion (see [Daemen and Rijmen, 2002]). In particular, it has the Maximum Distance Separable (MDS) property: For each column, it ensures that the number of bytes that are different from zero at this column in Y_i and Z_i is either equal to zero or greater than four, *i.e.*,

$$\forall i \in [0, r-1], \forall k \in [0; 3], (\sum_{j=0}^{3} (Y_i[j][k] \neq 0) + (Z_i[j][k] \neq 0)) \in \{0, 5, 6, 7, 8\}.$$

The matrix M, and its inverse M^{-1} (used for decryption), are presented in Figure 7.3.

$$M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}, M^{-1} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}$$

Figure 7.3: The M matrix used for the MixColumn operation in AES encryption (left), and its inverse M^{-1}, used for decryption (right).

AddRoundKey (ARK) Before the first round, ARK performs a xor between the initial plain text X and the initial key K_0 to obtain X_0:

$$\forall j,k \in [0;3], X_0[j][k] = X[j][k] \oplus K_0[j][k].$$

Then, at each round i, ARK performs a xor between Z_i and subkey K_{i+1} to obtain X_{i+1}, *i.e.*,

$$\forall i \in [0,r-1], \forall j,k \in [0;3], X_{i+1}[j][k] = Z_i[j][k] \oplus K_{i+1}[j][k].$$

Key Schedule (KS) KS computes the subkey K_i of each round $i \in [0,r]$ from the initial key K. It is represented as a $4 \times r \times 4$ matrix EK (for Extended Key). Each column of the matrix corresponds to one column of the key. To compute the round key K_i, we simply take the corresponding columns of EK: K_0 is composed of the first 4 columns of EK, K_1 is the 4 next columns, and so on:

$$\forall i \in [0,r], \forall j,k \in [0;3], K_i[j][k] = EK[j][i \cdot 4 + k].$$

The $KC = \frac{l}{32}$ (where l is the key length) first columns of EK are the columns of the initial key. The rest of EK is computed by combining values from the previous round subkeys. The exact formula is different for AES-256 than for the other versions. We first describe the algorithm for AES-128 and AES-192, which is the same for both. The computation of the byte column $EK[j][k]$, for $j,k \in [0;3]^2$, is simply the XOR of two values from previous columns ($EK[j][k-KC]$ and $EK[j][k-1]$) if k is not a multiple of KC. Otherwise, an intermediary column $SK[j][k-1]$ is built by rotating up each byte of column $EK[j][k-1]$ by one position, applying a SB operation to the resulting column, and XORing a round constant C_i to its first row. Then, $EK[j][k-1]$ is obtained by XORing $SK[j][k]$ and $EK[j][k-KC]$.

Hence, we have,

$$\forall j \in [0;3], \forall k \in [0;KC-1] : EK[j][k] = K[j][k],$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%KC = 0 : SK[j][k-1] = S(EK[(j+1)\%4][k-1]),$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%KC = 0 : EK[j][k] = SK[j][k-1] \oplus EK[j][k-KC],$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%KC \neq 0 : EK[j][k] = EK[j][k-1] \oplus EK[j][k-KC].$$

For AES-256, an additional SBox operation is applied, without rotations, when $k\%KC$ is equal to 4 (where KC is 8):

$$\forall j \in [0;3], \forall k \in [0;KC-1] : EK[j][k] = K[j][k],$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%KC = 0 : SK[j][k-1] = S(EK[(j+1)\%4][k-1]),$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%KC = 4 : SK[j][k-1] = S(EK[j][k-1]),$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%4 = 0 : EK[j][k] = SK[j][k-1] \oplus EK[j][k-KC],$$
$$\forall j \in [0;3], \forall k \in [KC;4 \cdot (r+1)-1], k\%4 \neq 0 : EK[j][k] = EK[j][k-1] \oplus EK[j][k-KC].$$

7.3 Our CP Models

We describe three models for solving the first step of the search for optimal related key differential characteristics for the AES, and the model for Step 2. The first model, $S1_{Basic}$, is a very straightforward implementation of the AES rules. It finds many word-inconsistent solutions. The second one, $S1_{Diff}$, exploits equalities that can be infer ed from the key schedule in order to make prune inconsistent solutions, and make the search practical for AES-128. The last one, $S1_{XOR}$, exploits additional information by considering the possible combinations of variables from the key. Finally, we present a decomposition, which shifts the frontier between steps one and two, and permits to solve the problem efficiently for all versions of the AES.

7.3.1 $S1_{Basic}$

The functions *Step1-opt* and *Step1-enum* used in Algorithm 1 for computing optimal differential characteristics share the same CP model. The only difference is in the goal of the solving process: in *Step1-opt* the goal is to search for a solution that optimises a given variable called obj_{Step1}, whereas in *Step1-enum* the goal is to enumerate all abstracted differentials when the variable obj_{Step1} is assigned to a given value.

We describe a first CP model for *Step1-opt* and *Step1-enum*, called $S1_{Basic}$, which was introduced in [Minier et al., 2014] and is derived in a straightforward way from the definition of the AES operations. It associates one Boolean variable ΔB with the differential word $\delta B \in diffWords_l$: ΔB is assigned to 0 if $\delta B = 0$, and to 1 otherwise.

Variables

As defined in Section 6.2, the variables representing a related-key differential characteristic are the ones included in the *DiffWords* set. The variables that go through a SBox, among these, form the *Sboxes* set.

For readability, we include both variables representing the round subkeys K_i, and to represent the extended key EK. Even though they represent the same thing (the K_i matrices are simply the EK matrix split into 4-column blocks), having both makes the modelling easier. The key schedule algorithm is simpler to define using EK, while the ARK operations are more readable with the K_i notation.

For AES-128 and AES-192, *diffWords* contains the following differential words, representing:

- The plaintext: δX
- The round keys: $\delta K_i[j][k], \forall i \in [0;r], j,k \in [0,3]^2$
- The extended key: $\delta EK[j][k], \forall j \in [0;3], k \in [0,(r+1)\cdot 4-1]$
- The state after ARK: $\delta X_i[j][k], \forall i \in [0;r], j,k \in [0,3]^2$
- The state after SB: $\delta SX_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- The state after SR: $\delta Y_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- The state after MC: $\delta Z_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- The SK columns: $\delta SK[j][k], \forall k \in [0;4\cdot(r+1)-1], k\%KC = KC-1$

The *Sboxes* set contains:

- $\delta X_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- $\delta SK[j][k], \forall k \in [0;4\cdot r-1], k\%KC = KC-1$

For AES-256, only the δSK variables change: The *diffWords* set becomes

- The plaintext: $\delta X[j][k], \forall j,k \in [0,3]^2$
- The round keys: $\delta K_i[j][k], \forall i \in [0;r], j,k \in [0,3]^2$
- The extended key: $\delta EK[j][k], \forall j \in [0;3], k \in [0,(r+1)\cdot 4-1]$
- The state after ARK: $\delta X_i[j][k], \forall i \in [0;r], j,k \in [0,3]^2$
- The state after SB: $\delta SX_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- The state after SR: $\delta Y_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- The state after MC: $\delta Z_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- The SK columns: $\delta SK[j][k], \forall k \in [0;4\cdot(r+1)-1], k\%4=3$

and the *Sboxes* set becomes:

- $\delta X_i[j][k], \forall i \in [0;r-1], j,k \in [0,3]^2$
- $\delta SK[j][k], \forall k \in [0;4\cdot r-1], k\%4=3$

Note that the SBoxes of δK_r are not taken into account. For AES-128 and AES-256, δK_r can still be computed fully, since the last SBoxes needed to compute the whole δK_r are at round $r-1$. On the other hand, for AES-192, it might occur that a SBox needs to be crossed at round r: if $r\%3=1$, then column 1 of δK_r goes through a SBox. In this case, our model produces a *truncated* differential characteristic, where the words of the δK_r that depend on the output of a SBox crossed at round r are undetermined. Similarly, the words of δX_r that are the result of a XOR with an undetermined word of δK_r are undetermined.

Not all differential words are abstracted in Step 1. Indeed, during Step 2, the initial differential plaintext δX, the last round differential subkey δK_r (as well as the corresponding columns of EK), and the final differential ciphertext δX_r can be deterministically computed given the values of all other differential words:

- δX is obtained by XORing δX_0 and δK_0.
- δK_r is obtained by applying the key schedule algorithm. The non deterministic aspect of the SB operation can be lifted: For the bytes δB that pass through S-boxes during this last round, we deterministically choose for δSB the value that maximises $p_S(\delta SB|\delta B)$.
- δX_r is obtained by XORing δZ_{r-1} and δK_r.

Hence, $S1_{Basic}$ associates a Boolean variable ΔB with every differential byte, except for δX, δX_r and δK_r: $\delta B \in \mathit{diffWords}_l \setminus \{\delta X[j][k], \delta K_r[j][k], \delta X_r[j][k], \delta EK[j][n\cdot 4+k] : j,k \in [0,3]\}$. Each Boolean variable ΔB is assigned to 0 if $\delta B = 0$, and to 1 otherwise.

We also define an integer variable obj_{Step1} which corresponds to the number of active S-boxes. The domain of this variable is $D(obj_{Step1}) = [1, \frac{l}{6}]$. Indeed, the smallest possible value is 1 because we need to have at least one active S-box to have a differential characteristics (we forbid the obvious solution such that δX and δK only contain bytes set to 0, meaning that there is no difference in the initial plaintext and key). The largest possible value is $\frac{l}{6}$ because the highest probability $p_S(\delta_{out}|\delta_{in})$ to pass through the AES S-box is $2^{-6} = \frac{4}{256}$ when $\delta_{in} \neq 0$ [Daemen and Rijmen, 2002], and because we want a differential characteristics which is more efficient than the key exhaustive search (*i.e.*, we want a differential characteristics whose probability is greater than 2^{-l}).

The corresponding list of variables is the following. We use the $\prec$ symbol, as defined in Section 6.4.1, to denote the fact that a differential bit Δa is a Boolean abstraction of the differential word δa.

ΔK: $\forall i \in [0;r-1], j,k \in [0,3]^2, \Delta K_i[j][k] \prec \delta K_i[j][k]$

$\mathit{\Delta}$EK: $\forall j \in [0;3], k \in [0, r\cdot 4-1], \mathit{\Delta}\mathrm{EK}[j][k] \prec \delta\mathrm{EK}[j][k]$

$\mathit{\Delta}$X: $\forall i \in [0; r-1], j,k \in [0,3]^2, \mathit{\Delta}\mathrm{X}_i[j][k] \prec \delta\mathrm{X}_i[j][k]$

$\mathit{\Delta}$SB: $\forall i \in [0; r-1], j,k \in [0,3]^2, \mathit{\Delta}\mathrm{SB}_i[j][k] \prec \delta\mathrm{SB}_i[j][k]$

$\mathit{\Delta}$Y: $\forall i \in [0; r-1], j,k \in [0,3]^2, \mathit{\Delta}\mathrm{Y}_i[j][k] \prec \delta\mathrm{X}_i[j][k]$

$\mathit{\Delta}$Z: $\forall i \in [0; r-1], j,k \in [0,3]^2, \mathit{\Delta}\mathrm{Z}_i[j][k] \prec \delta\mathrm{Z}_i[j][k]$

$\mathit{\Delta}$SK: $\forall j \in [0;3], k \in [0, r\cdot 4-1], \mathit{\Delta}\mathrm{SK}[j][k] \prec \delta\mathrm{SK}[j][k]$

obj_{Step1}: $obj_{Step1} = \sum_{i \in SBoxes_l} (i)$

The constraints that model AES operations are described below.

Constraints

SubBytes As the SBox is bijective, there is an output difference if and only if there is an input difference (*i.e.*, $(\mathrm{B}\oplus\mathrm{B}' \neq 0) \Leftrightarrow (\mathrm{S}(\mathrm{B})\oplus\mathrm{S}(\mathrm{B}') \neq 0)$). As a consequence, the SubBytes operator has no effect on the presence/absence of differences during Step 1, as explained in Section 6.4.2. This is ensured by the following constraint:

$$\forall i \in [0, r-1], \forall (j,k) \in [0;3]^2 : \mathit{\Delta}\mathrm{SB}_i[j][k] = \mathit{\Delta}\mathrm{X}_i[j][k]. \qquad (\mathrm{C}_{\mathrm{SB}})$$

XOR constraint As explained in Section 6.4.2, the XOR operation cannot be modelled precisely, since the information about whether two differential words are equal is abstracted. Hence, the XOR constraint is defined as

$$\mathrm{XOR}(\mathit{\Delta}\mathrm{B}_1, \mathit{\Delta}\mathrm{B}_2, \mathit{\Delta}\mathrm{B}_3) \Leftrightarrow \mathit{\Delta}\mathrm{B}_1 + \mathit{\Delta}\mathrm{B}_2 + \mathit{\Delta}\mathrm{B}_3 \neq 1 \qquad (\mathrm{XOR})$$

AddRoundKey ARK is modelled by XOR constraints:

$$\forall i \in [1, r-1], \forall (j,k) \in [0,3]^2 : \mathrm{XOR}(\mathit{\Delta}\mathrm{Y}_{i-1}[j][k], \mathit{\Delta}\mathrm{K}_i[j][k], \mathit{\Delta}\mathrm{X}_i[j][k]). \qquad (\mathrm{C}_{\mathrm{ARK}})$$

ShiftRows SR is modelled by equality constraints that link shifted bytes:

$$\forall i \in [0, r-1], \forall j,k \in [0,3] : \mathit{\Delta}\mathrm{Y}_i[j][k] = \mathit{\Delta}\mathrm{SX}_i[j][(j+k)\%4] \qquad (\mathrm{C}_{\mathrm{SR}})$$

MixColumns MC cannot be modelled precisely at the Boolean level, as knowing where differences hold in Y_i is not enough to determine where they hold in Z_i. Indeed, the values of the bytes in the column are necessary to determine the positions of the zeros after MC. However, the MDS property is modelled by constraining the number of differences in a same column of Y_i and Z_i to be equal to 0 or greater than 4:

$$\forall i \in [0, r-2], \forall k \in [0,3] : \left(\sum_{j=0}^{3} \Delta\mathrm{Y}_i[j][k] + \mathit{\Delta}\mathrm{Z}_i[j][k]\right) \in \{0,5,6,7,8\}. \qquad (\mathrm{C}_{\mathrm{MDS1}})$$

As MC is not applied during the last round, we have

$$\forall j,k \in [0,3] : \mathit{\Delta}\mathrm{Z}_{r-1}[j][k] = \mathit{\Delta}\mathrm{Y}_{r-1}[j][k]. \qquad (\mathrm{C}_{\mathrm{MDS2}})$$

This set of constraints $\{\mathrm{C}_{\mathrm{MDS1}}, \mathrm{C}_{\mathrm{MDS2}}\}$ is denoted $\mathrm{C}_{\mathrm{MDS}}$.

KeySchedule KS is modelled by XOR constraints (combined with a rotation of the bytes of SK).

The first step is to relate the ΔK and ΔEK variables:

$$\forall j \in [0;3], \forall k \in [0;\mathrm{KC}-1] : \delta\mathrm{EK}[j][k] = \Delta\mathrm{K}[j][k]. \qquad (\mathrm{C}_{\Delta\mathrm{EK}})$$

We then need to define the ΔSK variables, with a special case when for AES-256 (KC = 8). As for the constraint $\mathrm{C_{SB}}$, the S operation does not change the value of the differential bit.

$$\left.\begin{array}{l} \forall j \in [0;3], \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], k\%\mathrm{KC}=0 : \Delta\mathrm{SK}[j][k-1] = \Delta\mathrm{EK}[(j+1)\%4][k-1], \\ \forall j \in [0;3], \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], k\%\mathrm{KC}=0 : \Delta\mathrm{SK}[j][k-1] = \Delta\mathrm{EK}[(j+1)\%4][k-1], \\ \forall j \in [0;3], \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC}=8, k\%\mathrm{KC}=4 : \Delta\mathrm{SK}[j][k-1] = \Delta\mathrm{EK}[j][k-1]. \end{array}\right\} (\mathrm{C}_{\Delta\mathrm{SK}})$$

We can now describe the key schedule constraints, which are different depending on the value of KC:

$$\left.\begin{array}{r} \forall j \in [0;3], \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC}<8, k\%\mathrm{KC}=0 : \mathrm{XOR}(\Delta\mathrm{EK}[j][k], \Delta\mathrm{SK}[j][k-1], \Delta\mathrm{EK}[j][k-\mathrm{KC}]), \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC}=8, k\%4=0 : \mathrm{XOR}(\Delta\mathrm{EK}[j][k], \Delta\mathrm{SK}[j][k-1], \Delta\mathrm{EK}[j][k-\mathrm{KC}]), \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC}<8 k\%\mathrm{KC}\neq 0 : \mathrm{XOR}(\Delta\mathrm{EK}[j][k], \Delta\mathrm{EK}[j][k-1], \Delta\mathrm{EK}[j][k-\mathrm{KC}]), \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC}=8 k\%4\neq 0 : \mathrm{XOR}(\Delta\mathrm{EK}[j][k], \Delta\mathrm{EK}[j][k-1], \Delta\mathrm{EK}[j][k-\mathrm{KC}]). \end{array}\right\} (\mathrm{C}_{simple\mathrm{KS}})$$

Performance of $\mathrm{S1}_{Basic}$**.** $\mathrm{S1}_{Basic}$ is complete in the sense that for any solution at the byte level (on δ variables), there exists a solution of $\mathrm{S1}_{Basic}$ at the Boolean level (on Δ variables). However, the experiments reported in [Gerault et al., 2016] have shown us that there is a huge number of solutions of $\mathrm{S1}_{Basic}$ which are byte inconsistent and do not correspond to solutions at the byte level. For example, for AES-128, when the number of rounds is $r = 3$, the optimal solution of *Step1-opt* has $obj_{\mathrm{S}tep1} = 3$ active SBoxes, and *Step1-enum* enumerates more than five hundred abstracted differentials with this number of active SBoxes. However, none of these abstracted differentials is byte-consistent. Actually, the optimal byte-consistent abstracted differentials has 5 active SBoxes. In this case, most of the solving time is spent at generating useless abstracted differentials which are discarded in Step 2. Hence, we had to introduce more subtle reasoning in the model to limit the number of solutions.

7.3.2 $\mathrm{S1}_{\mathrm{D}iff}$

The problem with $\mathrm{S1}_{Basic}$ is that it does not filter enough word-inconsistent solutions, so that it does not permit to solve the problem for difficult instances. To solve this problem, we developed $\mathrm{S1}_{\mathrm{D}iff}$. In $\mathrm{S1}_{\mathrm{D}iff}$, additional constraints and variables are introduced. They are used to infer equality relations between differential bytes, and these relations are used to propagate the MDS property of MixColumns at the byte level. They remove most binary solutions that cannot be transformed into byte solutions, thus speeding up the solution process.

A first weakness of $\mathrm{S1}_{Basic}$ comes from the fact that the XOR constraint between Boolean variables is a poor abstraction of the xor relation at the byte level, because whenever two XORed bytes are different from zero, we cannot know whether the result is equal to zero or not. A second weakness of $\mathrm{S1}_{Basic}$ comes from the fact that MC is only poorly approximated at the Boolean level, as we only propagate a Boolean abstraction of the MDS property. $\mathrm{S1}_{\mathrm{D}iff}$ aims at overcoming these issues.

Propagation of MDS at Byte Level

For each round $i \in [0, r-2]$ and each column $j \in [0,3]$, the MDS property of MixColumns ensures, for two columns Y and Z such that $\mathrm{Z} = \mathrm{MC(Y)}$:

$$\sum_{j=0}^{3} (\mathrm{Y}_i[j][k] \neq 0) + (\mathrm{Z}_i[j][k] \neq 0)) \in \{0,5,6,7,8\}.$$

At differential byte level, this property still holds:

$$\sum_{j=0}^{3}(\delta Y_i[j][k] \neq 0) + (\delta Z_i[j][k] \neq 0)) \in \{0,5,6,7,8\}.$$

In the first model, this property is ensured by the constraint C_{MDS}:

$$\forall i \in [0, r-1], \forall k \in [0,3] : \left(\sum_{j=0}^{3} \Delta Y_i[j][k] + \Delta Z_i[j][k]\right) \in \{0,5,6,7,8\}.$$

However, the MDS property also holds for any xor difference between two different columns in two different rounds of the differential byte model:

$$\forall i1, i2 \in [0, r-2]^2, \forall k1, k2 \in [0;3]^2 :$$
$$(\sum_{j=0}^{3}((\delta Y_{i1}[j][k1] \oplus \delta Y_{i2}[j][k2]) \neq 0) + ((\delta Z_{i1}[j][k1] \oplus \delta Z_{i2}[j][k2] \neq 0)) \in \{0,5,6,7,8\}.$$

To ensure this property, which removes most inconsistent solutions, we need to be able to verify whether two differential bytes are equal or not. To this end, we introduce additional Boolean variables: given two differential bytes δB_1 and δB_2, the Boolean variable $diff_{\delta B_1,\delta B_2}$ is equal to 1 if $\delta B_1 \neq \delta B_2$, and to 0 otherwise. We do not define a *diff* variable for every couple of differential bytes in $diffWords_l$, but restrict our attention to couples of bytes for which it is useful to know whether they are equal or not.

More precisely, we consider three separate sets of differential bytes and we only compare differential bytes that belong to a same set.

- The first set, called DK, contains bytes coming from the δK matrices.
- The second and third sets, called DY and DZ, contain bytes coming from δY and δZ matrices, respectively.

For each of these three sets, we consider a separate subset for each row $j \in [0,3]$:

- For DK, we know that every initial xor equation due to the key schedule either involves three bytes on a same row j, or it involves two bytes on a same row j and a byte that has just passed through an S-box on the next row $((j+1)\%4)$. As we cannot know if the input and output differences of S-boxes are equal or not, we can limit *diff* variables to couples of differential bytes that occur on a same row of δK matrices.
- For DY and DZ, the MDS property implies relations between columns of DY and DZ and, in Section 6.4.2, we use a generalisation of this property that xors bytes of a same row for different columns. As these xors are only performed on bytes that occur in a same row, we can limit *diff* variables to couples of differential bytes that occur on a same row of δY matrices (for DY) and δZ matrices (for DZ).

For each row $j \in [0,3]$, we define the three following sets:

$$\begin{aligned} DK_j &= \{\delta K_i[j][k] : i \in [1,r], k \in [0,3]\}, \\ DY_j &= \{\delta Y_i[j][k] : i \in [0,r-2], k \in [0,3]\}, \\ DZ_j &= \{\delta Z_i[j][k] : i \in [0,r-2], k \in [0,3]\}. \end{aligned}$$

Given these sets, we define the *diff* variables as follows: For each set $D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}$, and for each pair of differential bytes $\{\delta B_1, \delta B_2\} \in D$, we define a Boolean variable $diff_{\delta B_1,\delta B_2}$, which is equal to 1 if δB1 and δB2 are equal, and to 0 otherwise. Using these differential byte difference variables, the MDS property between different columns is ensured by the following constraint:

$$\forall i1, i2 \in [0, r-2]^2, \forall k1, k2 \in [0;3]^2 :$$
$$(\sum_{j=0}^{3}(diff_{\delta Y_{i1}[j][k1], \delta Y_{i2}[j][k2]}) \neq 0) + (diff_{\delta Y_{i1}[j][k1], \delta Y_{i2}[j][k2]} \neq 0)) \in \{0,5,6,7,8\}. \qquad (C_{DIFFMDS})$$

We now describe how these difference variables are defined.

Constraints Defining the Difference Variables

When defining the constraint $xor(\Delta A, \Delta B, \Delta C)$ (where ΔA, ΔB and ΔC are binary variables associated with differential bytes δA, δB and δC, respectively), if $\Delta A = \Delta B = 1$, then we cannot know whether ΔC is equal to 0 or 1. However, whenever $\Delta C = 0$ (resp. $\Delta C = 1$), we know for sure that the corresponding byte δC is equal to 0^8 (resp. different from 0^8), meaning that the two bytes δA and δB are equal (resp. different), *i.e.*, that $diff_{\delta A, \delta B} = 0$ (resp. $diff_{\delta A, \delta B} = 1$). The same reasoning may be done for ΔA and ΔB because $(\delta A \oplus \delta B = \delta C) \Leftrightarrow (\delta B \oplus \delta C = \delta A) \Leftrightarrow (\delta A \oplus \delta C = \delta B)$.

Since our difference variables concern differential bytes that belong to the same matrices (either δK, δY or δZ), and the key schedule is the single place where variables from the same matrix are XORed together, we introduce a different XOR constraint for the key schedule. The resulting constraint, XORKS, is defined as follows:

$$\text{XORKS}(\Delta A, \Delta B, \Delta C) \Leftrightarrow \left.\begin{array}{l} ((\Delta A + \Delta B + \Delta C \neq 1) \\ \wedge (diff_{\delta A, \delta B} = \Delta C) \\ \wedge (diff_{\delta A, \delta C} = \Delta B) \\ \wedge (diff_{\delta B, \delta C} = \Delta A)) \end{array}\right\} \text{(XORKS)}$$

The key schedule constraint is modified accordingly. When no SBox is involved, we replace XOR with XORKS. Otherwise, since we do not consider *diff* variables for the output of SBoxes, we use the regular XOR constraint. We however add an additional constraint: when computing XOR($\delta a, \delta b, \delta c$), where δb is the SBox output, δa is different from δc if δb is nonzero. Hence, the constraint $C_{simple KS}$ is replaced by a different constraint C_{KS}, defined as follows:

$$\left.\begin{array}{l} \forall j \in [0;3], \\ \forall k \in [KC; 4\cdot(r+1)-1], KC < 8, k\%KC = 0 : XOR(\Delta EK[j][k], \Delta SK[j][k-1], \Delta EK[j][k-KC]), \\ \forall k \in [KC; 4\cdot(r+1)-1], KC < 8, k\%KC = 0 : diff_{\delta EK[j][k], \delta EK[j][k-KC]} = \Delta SK[j][k-1], \\ \forall k \in [KC; 4\cdot(r+1)-1], KC = 8, k\%4 = 0 : XOR(\Delta EK[j][k], \Delta SK[j][k-1], \Delta EK[j][k-KC]), \\ \forall k \in [KC; 4\cdot(r+1)-1], KC = 8, k\%KC = 4 : diff_{\delta EK[j][k], \delta EK[j][k-KC]} = \Delta SK[j][k-1], \\ \forall k \in [KC; 4\cdot(r+1)-1], KC < 8 k\%KC \neq 0 : XORKS(\Delta EK[j][k], \Delta EK[j][k-1], \Delta EK[j][k-KC]). \\ \forall k \in [KC; 4\cdot(r+1)-1], KC = 8 k\%4 \neq 0 : XORKS(\Delta EK[j][k], \Delta EK[j][k-1], \Delta EK[j][k-KC]). \end{array}\right\} (C_{KS})$$

Constraints to ensure that difference variables define an equivalence relation.

To be consistent, our difference variables need to have both symmetry and transitivity.

Symmetry is ensured by:

$$\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \in D diff_{\delta B_1, \delta B_2} = diff_{\delta B_2, \delta B_1}. \qquad (C_{SYM})$$

Transitivity is ensured by:

$$\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2, \delta B_3\} \in D,$$
$$diff_{\delta B_1, \delta B_2} + diff_{\delta B_2, \delta B_3} + diff_{\delta B_1, \delta B_3} \neq 1. \qquad (C_{TRANS})$$

Constraints that relate difference variables with binary differential variables. If $\Delta A \neq \Delta B$, then $diff_{\delta A,\delta B} = 1$, and that if $\Delta A \neq \Delta B = 0$, then $diff_{\delta A,\delta B} = 0$. This is enforced by the following constraint:

$$\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \in D, diff_{\delta B_1,\delta B_2} + \Delta B_1 + \Delta B_2 \neq 1. \qquad (C_{DIFF})$$

Finally, as ARK defines $X_{i+1}[j][k]$ as the result of a xor between $K_i[j][k]$ and $Z_i[j][k]$, we post the constraint: $\forall i_1, i_2 \in [0, r-1], \forall j, k_1, k_2 \in [0,3]$,

$$diff_{\delta K_{i_1}[j][k_1],\delta K_{i_2}[j][k_2]} + diff_{\delta Z_{i_1}[j][k_1],\delta Z_{i_2}[j][k_2]} + \Delta X_{i_1}[j][k_1] + \Delta X_{i_2}[j][k_2] \neq 1. \qquad (C_{DIFFARK})$$

Additional equalities derived from KS

We now describe the constraints that are derived from the key schedule. We give examples of these constraints for AES-128 for the sake of simplicity.

The KeySchedule mainly performs xor operations. For instance, in AES-12, at each round i, the first column $K_i[0]$ is obtained by performing a xor between bytes of $K_{i-1}[0]$ and $K_{i-1}[3]$; for the last three columns $j \in \{1,2,3\}$, $K_i[j]$ is obtained by performing a xor between $K_{i-1}[j]$ and $K_i[j-1]$. Besides these xor operations, all bytes of $K_{i-1}[3]$ pass through the SBox before XORing them with $K_{i-1}[0]$ to obtain $K_i[0]$. Therefore, each byte of K_i, for each round $i \in [1,r]$ may be expressed as a combination of xor operations between bytes of the initial key K_0, and bytes obtained by applying the S operation on column 3 of rounds $j < i$. For example (recall that $A \oplus A = 0^8$ and $0^8 \oplus A = A$):

$$\begin{aligned} K_2[1][1] &= K_2[0][1] \oplus K_1[1][1] \\ &= K_1[0][1] \oplus S(K_1[3][2]) \oplus K_1[0][1] \oplus K_0[1][1] \\ &= S(K_1[3][2]) \oplus K_0[1][1] \end{aligned}$$

When reasoning at the differential byte levels, we have

$$\delta K_2[1][1] = \delta SK_1[3][2] \oplus \delta K_0[1][1],$$

where $\delta SK_1[3][2] = S(K_1[3][2]) \oplus S(K'_1[3][2])$. As S is a non linear operation, we cannot assume that $\delta SK_1[3][2] = S(\delta K_1[3][2])$. Therefore, $\delta SK_1[3][2]$ is a new differential byte. However, there is a finite number of such new differential bytes.

We propose to exploit the fact that each differential byte of K_i is the result of a xor between a finite set of bytes. We first use the KS rules defined in Section 7.2 to build, for each $i \in [1, r-1]$, and $j, k \in [0,3]$, the set $V(\delta K_i[j][k])$ of all differential bytes (coming either from δK_0 or from the set of new differential bytes δSK), such that:

$$\delta K_i[j][k] = \bigoplus_{\delta A \in V(i,j,k)} \delta A.$$

For example, $V(\delta K_2[1][1]) = \{\delta K_0[1][1], \delta SK_1[3][2]\}$.

Note that these sets are computed before the search and do not depend on the initial values of plaintexts and keys.

For each of these sets, we introduce a set variable which contains the corresponding binary differential variables which are equal to 1:

$$V_{inst}(\delta K_i[j][k]) = \{\Delta A \mid \delta A \in V(\delta K_i[j][k]) \wedge \Delta A = 1\}.$$

For example, in AES-128, if $\Delta K_0[1][1] = 1$ and $\Delta SK_1[3][2] = 0$, then $V_{inst}(\delta K_2[1][1]) = \{\Delta K_0[1][1]\}$.

Whenever two differential key bytes $\delta K_{i1}[j1][k1]$ and $\delta K_{i2}[j2][k2]$ have the same V_{inst} sets, then we may infer that $\delta K_{i1}[j1][k1] = \delta K_{i2}[j2][k2]$. More precisely, we define the constraint:

$$\forall j \in [0;3], \forall \{\delta B_1, \delta B_2\} \in DK_j (V_{inst}(\delta B1) = V_{inst}(\delta B2)) \Rightarrow (diff_{\delta B1,\delta B2} = 0). \qquad (C_{VSETSEQ})$$

Also, if $V_{inst}(\delta B)$ is empty (resp. contains one element), we infer that ΔB is equal to 0 (resp. a nonzero variable. This is enforced by the following constraint:

$$\forall j \in [0;3], \forall \{\delta B\} \in DK_j V_{inst}(\delta B) + \Delta B \neq 1. \qquad (C_{\text{VSETSCARD}})$$

From a practical point of view, the V_{inst} variables are not modelled with set variables, but with vectors of Boolean variables. The dimension of these vectors is equal to the number of possible elements in these sets, *i.e* one for each byte of the initial key, plus number of key bytes that go through an SBox. Each Boolean variable $V[p]$ is equal to 1 if the p^{th} element belongs to V_{inst} (*i.e.*, if the variable associated with the p^{th} element is equal to 1), and to 0 otherwise.

The set of variables used $S1_{Diff}$ are the same as the variables of $S1_{Basic}$, plus the *diff* variables, and the variables representing the V_{inst} sets.

The constraints used in $S1_{Diff}$ are C_{SB}, XOR, XORKS, C_{SR}, C_{MDS}, C_{ARK}, $C_{\Delta EK}$, $C_{\Delta SK}$, C_{KS}, $C_{DIFFMDS}$, C_{SYM}, C_{TRANS}, C_{DIFF}, $C_{DIFFARK}$, C_{VSETS}, $C_{VSETSEQ}$, and $C_{VSETSCARD}$.

Performance of $S1_{Diff}$

Compared to $S1_{Basic}$, $S1_{Diff}$ filters a lot of byte inconsistent solutions, which makes it capable of solving all the instances for AES-128 and AES-256 within 24 hours. These results are summed up in Table 7.1, and commented and compared to $S1_{XOR}$ in Section 7.3.4. In essence, $S1_{Diff}$ is capable of solving all instances but one in less than 13 hours each (at most 43359 seconds), but it fails at solving *Step1-enum* for 10 rounds of AES-192 after two weeks.

We therefore introduce another model, which reasons differently about the equalities introduced by the key schedule, and which is presented in the following section.

7.3.3 $S1_{XOR}$

In $S1_{Diff}$, we exploit the fact that the bytes of the round subkeys are computed as a combination of other bytes from the previous round subkeys. With $S1_{XOR}$, we push this reasoning further: instead of just considering the XORs that appear in the key schedule, we combine them in order to build additional XORs, results in an improved propagation of the constraints, and in a faster resolution.

Generation of Additional Xors

Every subkey differential byte $\delta K_i[j][k]$ either comes from the initial differential key δK, or is obtained by XORing two differential bytes according to the key schedule rules. Hence, the whole key schedule implies a set of $16*(r-1)$ (resp. $16*(r-1)-8$ and $16*(r-2)$) xor equations for AES-128 (resp. AES-192 and AES-256), where each of these equations involves three differential bytes. We propose to combine these initial equations to infer additional equations.

Let us consider, for example, the three equations that define $\delta K_1[0][3]$, $\delta K_2[0][2]$ and $\delta K_2[0][3]$, respectively, for AES-128:

$$\delta K_0[0][3] \oplus \delta K_1[0][2] \oplus \delta K_1[0][3] = 0 \qquad (7.1)$$

$$\delta K_1[0][2] \oplus \delta K_2[0][1] \oplus \delta K_2[0][2] = 0 \qquad (7.2)$$

$$\delta K_1[0][3] \oplus \delta K_2[0][2] \oplus \delta K_2[0][3] = 0 \qquad (7.3)$$

These equations share bytes: $\delta K_1[0][2]$ for Eq. (7.1) and (7.2), $\delta K_1[0][3]$ for Eq. (7.1) and (7.3), and $\delta K_2[0][2]$ for Eq. (7.2) and (7.3). We can combine Eq. (7.1), (7.2), and (7.3) by XORing them, and exploit the fact that $B \oplus B = 0$ for any byte B, to generate the following equation:

$$\delta K_0[0][3] \oplus \delta K_2[0][1] \oplus \delta K_2[0][3] = 0 \qquad (7.4)$$

This new equation is redundant at the byte level, as (7.1) $\wedge$ (7.2) $\wedge$ (7.3) $\Rightarrow$ (7.4). However, at the Boolean level, the propagation of the XOR constraint corresponding to Eq. (7.4) detects inconsistencies which are not detected when only considering the XOR constraints corresponding to

Eq. (7.1), (7.2), and (7.3). Let us consider, for example, the case where $\Delta K_0[0][3] = 1$, $\Delta K_2[0][1] = \Delta K_2[0][3] = 0$, and all other Boolean variables still have 0 and 1 in their domains. In this case, the propagation of XOR constraints associated with Eq. (7.1), (7.2), and (7.3) does not detect an inconsistency as only one variable is assigned for each constraint, and for the two other variables, we can always choose values such that the sum is different from 1. However, at the byte level, $\delta K_0[0][3]$ cannot be assigned to a value different from 0 when $\delta K_2[0][1] = \delta K_2[0][3] = 0$. Indeed, in this case, Eq. (7.2) and (7.3) imply:

$$\begin{aligned} & \delta K_1[0][2] \oplus \delta K_2[0][2] = \delta K_1[0][3] \oplus \delta K_2[0][2] = 0 \\ \Rightarrow \quad & \delta K_1[0][2] = \delta K_2[0][2] = \delta K_1[0][3] \\ \Rightarrow \quad & \delta K_0[0][3] \oplus \delta K_1[0][2] \oplus \delta K_1[0][3] = \delta K_0[0][3] \end{aligned}$$

As a consequence, if $\delta K_0[0][3] \neq 0$, we cannot satisfy Eq. (7.1). This inconsistency is detected when propagating the XOR constraint associated with Eq. (7.4), as it ensures that $\Delta K_0[0][3] + \Delta K_2[0][1] + \Delta K_2[0][3] \neq 1$.

Hence, we propose to combine xor equations of the key schedule to generate new equations. More precisely, given two equations $\delta B_1 \oplus \ldots \oplus \delta B_n = 0$ and $\delta B'_1 \oplus \ldots \oplus \delta B'_m = 0$ such that $\{B_1, \ldots, B_n\} \cap \{B'_1, \ldots, B'_m\} \neq \emptyset$, we generate the equation:

$$\bigoplus_{B \in \{B_1,\ldots,B_n\} \cup \{B'_1,\ldots,B'_m\} \setminus \{B_1,\ldots,B_n\} \cap \{B'_1,\ldots,B'_m\}} \delta B = 0.$$

This new equation is recursively combined with existing ones to generate other equations until no more equation can be generated.

For example, from Eq. (7.1), (7.2), and (7.3), we generate the following equations:

$$\text{From (7.1) and (7.2):} \quad \delta K_0[0][3] \oplus \delta K_1[0][3] \oplus \delta K_2[0][1] \oplus \delta K_2[0][2] = 0 \tag{7.5}$$

$$\text{From (7.1) and (7.3):} \quad \delta K_0[0][3] \oplus \delta K_1[0][2] \oplus \delta K_2[0][2] \oplus \delta K_2[0][3] = 0 \tag{7.6}$$

$$\text{From (7.2) and (7.3):} \quad \delta K_1[0][2] \oplus \delta K_1[0][3] \oplus \delta K_2[0][1] \oplus \delta K_2[0][3] = 0 \tag{7.7}$$

Then, from Eq. (7.1) and (7.7) (or, equivalently, from Eq. (7.2) and (7.6) or from Eq. (7.3) and (7.5)), we generate Eq. (7.4). As no additional equation can be generated from Eq. (7.1), (7.2), (7.3), (7.5), (7.6), (7.7), and (7.4), the process stops.

The number of additional equations that may be generated grows rapidly with respect to the number r of rounds. For example, for AES-128, when $r = 3$ (resp. $r = 4$), the total number of new equations that may be generated is 988 (resp. 16332). When further increasing r to 5, the number of new equations becomes so large that we cannot generate them within a time limit of one hour.

To avoid this combinatorial explosion, we only generate equations that involve at most four differential bytes.

Indeed, our constraints on the *diff* variables operate on at most three pairs of Boolean variables. Preliminary experiments have shown us that these strengthened XOR constraints both reduce the number of choice points and speed-up the solution process, and that further adding XOR constraints for equations that involve more than four variables (to forbid that their sum is equal to one) does not significantly reduce the number of choice points and often increases time.

For AES-128 (resp. AES-192 and AES-256) with $r = 10$ (resp. $r = 12$ and $r = 14$) rounds, the number of initial equations coming from the key schedule is 144 (resp. 168 and 192). From these initial equations, we generate 122 (resp. 168 and 144) new equations that involve three differential bytes, and 1104 (resp. 1696 and 1256) new equations that involve four differential bytes.

The time needed to generate all these equations is very small (compared to the time needed to solve Step 1): The algorithm that generates equations has been implemented in Picat, and the time spent by Picat to search for all equations of length smaller than or equal to four is always smaller than 0.1 seconds.

We note $xorEq_l$ the set of all equations (both initial and generated equations) coming from the key schedule when the key length is l.

Note that *xorEq*$_l$ actually contains all possible equations with at most four differential bytes implied by the key schedule algorithm. In other words, our procedure for generating new equations is complete, even if it does not allow the generation of intermediate equations of length greater than 4. We have proven this by exhaustively generating all possible equations with at most four differential key bytes and, for each equation that does not belong to *xorEq*$_l$, we have proven that it is not consistent with the initial set of equations of the key schedule. The whole proof, for all possible equations, is done in less than an hour for the three possible key lengths.

Constraints associated with xor equations of *xorEq*$_l$ As in $S1_{Diff}$, we use *diff* variables.

From $S1_{Diff}$, we keep the constraints ensuring an equivalence relation between the *diff* variables, defined in Section 7.3.2, as well as the constraints relating the *diff* variables to the Boolean variables representing differential bytes, defined in Section 7.3.2. However, we now also include *diff* variables for the differential bits of ΔSK. This results in a different DK2 set, defined, for each row j, as

$$DK2_j = \{\delta K_i[j][k] : i \in [1,r], k \in [0,3]\} \cup \{\delta SK[j][k] : k \in [0; r*4-1]\}$$

This extended set is used to redefine the C_{SYM}, C_{TRANS} and C_{DIFF} constraints as follows: Symmetry is ensured by:

$$\forall D \in \{DK2_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \in D\, diff_{\delta B_1,\delta B_2} = diff_{\delta B_2,\delta B_1}. \qquad (C_{sym2})$$

Transitivity is ensured by:

$$\forall D \in \{DK2_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2, \delta B_3\} \in D,$$
$$diff_{\delta B_1,\delta B_2} + diff_{\delta B_2,\delta B_3} + diff_{\delta B_1,\delta B_3} \neq 1. \qquad (C_{trans2})$$

and C_{DIFF} is redefined as

$$\forall D \in \{DK2_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \in D, diff_{\delta B_1,\delta B_2} + \Delta B_1 + \Delta B_2 \neq 1. \qquad (C_{DIFF2})$$

In $S1_{Diff}$, we only considered a XOR constraint of arity 3: XOR(ΔA, ΔB, ΔC). In $S1_{XOR}$, we also define a constraint for XOR relations of size 3, but we also build XOR relations of size 4.

The constraint related to XORs of size 3, is:

$$\forall (\delta B_1 \oplus \delta B_2 \oplus \delta B_3 = 0) \in xorEq_l,$$
$$(diff_{\delta B_1,\delta B_2} = \Delta B_3) \wedge (diff_{\delta B_1,\delta B_3} = \Delta B_2) \wedge (diff_{\delta B_2,\delta B_3} = \Delta B_1). \qquad (C_{XOR3})$$

For each equation $\delta B_1 \oplus \delta B_2 \oplus \delta B_3 = 0$ in *xorEq*$_l$, it ensures that whenever two differential bytes of $\{\delta B_1, \delta B_2, \delta B_3\}$ have different values, then the third one is different from 0 and therefore its associated Boolean variable is equal to 1. Note that this constraint combined with C_{DIFF} ensures that $\Delta B_1 + \Delta B_2 + \Delta B_3 \neq 1$. Indeed, if $\Delta B_1 = 1$ and $\Delta B_2 = \Delta B_3 = 0$, then Constraint ($C_9'$) implies that $diff_{\Delta B_2,\Delta B_3} = 0$, which is inconsistent with the fact that $diff_{\delta B_2,\delta B_3}$ must be equal to ΔB_1.

For XOR relations of size 4, we define another constraint. Let us consider an equation $\delta A \oplus \delta B \oplus \delta C \oplus \delta D = 0$ that involves four differential bytes, and let ΔA, ΔB, ΔC, and ΔD be the Boolean variables associated with δA, δB, δC, and δD, respectively. We know that whenever two differential bytes of $\{\delta A, \delta B, \delta C, \delta D\}$ are equal then the two other ones must also be equal. This is modelled by the following constraint:

$$\forall (\delta B_1 \oplus \delta B_2 \oplus \delta B_3 \oplus \delta B_4 = 0) \in xorEq_l,$$
$$(diff_{\delta B_1,\delta B_2} = diff_{\delta B_3,\delta B_4}) \wedge (diff_{\delta B_1,\delta B_3} = diff_{\delta B_2,\delta B_4}) \wedge (diff_{\delta B_1,\delta B_4} = diff_{\delta B_2,\delta B_3}). \qquad (C_{XOR4})$$

The combination of C_{DIFF} and C_{XOR4} ensures that $\Delta A + \Delta B + \Delta C + \Delta D \neq 1$. Indeed, if $\Delta A = 1$ and $\Delta B = \Delta C = \Delta D = 0$, then we have that $diff_{\Delta B,\Delta C} = 0$ and $diff_{\Delta A,\Delta C} = 1$, which is inconsistent with the fact that $diff_{\delta B,\delta C}$ must be equal to $diff_{\Delta A,\Delta C}$.

The set of variables used $S1_{XOR}$ are the same as the variables of $S1_{Basic}$, plus the *diff* variables.

The constraints used in $S1_{XOR}$ are C_{SB}, C_{XOR}, C_{XOR3}, C_{XOR4}, C_{SR}, C_{MDS}, C_{ARK}, $C_{\Delta EK}$, $C_{\Delta SK}$, $C_{DIFFMDS}$, C_{SYM}, C_{TRANS}, C_{DIFF}, and $C_{DIFFARK}$.

7.3.4 Comparison of $\mathrm{S1_{XOR}}$ with $\mathrm{S1_{Diff}}$

The main difference between $\mathrm{S1_{XOR}}$ and $\mathrm{S1_{Diff}}$ is that in $\mathrm{S1_{Diff}}$ we do not infer new xor equations from the initial equations of the key schedule, as explained in Section 7.3.3. Instead of this equation generation step, $\mathrm{S1_{Diff}}$ uses the key schedule rules to pre-compute, for each differential byte $\delta K_i[j][k]$, a set $V(\delta K_i[j][k])$ of differential bytes such that $\delta K_i[j][k]$ is equal to the result of XORing all bytes in $V(i,j,k)$.

Then, for each differential byte $\delta K_i[j][k]$, a variable $V_{inst}(\delta K_i[j][k])$ is constrained to be equal to the subset of bytes of $V(i,j,k)$ that are different from 0. These V_{inst} variables are used to infer that two differential bytes are equal when their corresponding V_{inst} variables are equal, and that $\Delta K_i[j][k]$ is equal to 0 (resp. 1) when $V_{inst}(\delta K_i[j][k])$ is empty (resp. contains only one variable).

Additionally, the set DK is extended to include the δSK variables in $\mathrm{S1_{XOR}}$.

In this section, we experimentally compare $\mathrm{S1_{XOR}}$ with $\mathrm{S1_{Diff}}$.

Experimental results All our experiments are performed according to the setting described in Section 6.4.2. While the machine used for the experiments has 24 cores, we ran these experiments on one core at a time, to avoid perturbations due to other processes and make time measurements more accurate and reproducible.

We compare $\mathrm{S1_{XOR}}$ and $\mathrm{S1_{Diff}}$ on the two problems described in Section 6.4.1, *i.e.*, *Step1-opt*, that aims at finding the minimal value of obj_{Step1}, and *Step1-enum*, that aims at enumerating all abstracted differentials when the value of obj_{Step1} is fixed to a given value v.

We consider 23 instances denoted AES-l-r where $l \in \{128, 192, 256\}$ is the key length and r is the number of rounds: $r \in [3,5]$ (resp. $[3,10]$ and $[3,14]$) when $l = 128$ (resp. 192 and 256). We do not consider values of r larger than 5 (resp. 10) when $l = 128$ (resp. 192) because for these values the maximal probability becomes smaller than 2^{-l}.

$\mathrm{S1_{XOR}}$ and $\mathrm{S1_{Diff}}$ are implemented with MiniZinc [Nethercote et al., 2007], a high-level modelling language. Many CP solvers accept MiniZinc models, and we have made experiments with Gecode [Gecode Team, 2006], Choco [Prud'homme et al., 2016], Chuffed [Chu and Stuckey, 2014], and Picat-SAT [Zhou et al., 2015]. Note that Picat-SAT actually uses a SAT solver to solve CSPs: It first translates the CSP instance into a Boolean satisfiability formula, and then uses the SAT solver Lingeling [Biere, 2014] to solve it.

We report experimental results obtained with Picat-SAT, which is the solver that has the best results.

Results for *Step1-opt* For each instance, Table 7.1 reports the optimal value v^* of Obj_{Step1} computed by *Step1-opt*. This optimal value may be different for $\mathrm{S1_{XOR}}$ and $\mathrm{S1_{Diff}}$ as they consider different abstractions of the KS xor operations. In practice, for all instances but one, both models find the same optimal value, and for this optimal value there exists at least one byte-consistent abstracted differential (see Section 7.4) so that the repeat loop (lines 6-10) of Algorithm 1 is executed only once. However, for AES-192-10, the minimal value of obj_{Step1} is equal to 27 with $\mathrm{S1_{Diff}}$ whereas it is equal to 29 with $\mathrm{S1_{XOR}}$. As a consequence, if we use $\mathrm{S1_{Diff}}$ to solve *Step1-opt*, the repeat loop of Algorithm 1 is executed three times: v is successively assigned to 27, 28, and 29, and for each of these values, we need to solve *Step1-enum* and *Step2*. When $v = 27$ (resp. 28), *Step1-enum* with $\mathrm{S1_{Diff}}$ finds 92 (resp. 1436) abstracted differentials. All these abstracted differentials are byte-inconsistent and *Step2* returns $null$ for each of them.

When $v = 29$, some abstracted differentials are byte-consistent and the repeat loop is stopped. For this instance, $\mathrm{S1_{XOR}}$ is able to infer that there is no byte-consistent abstracted differential with 27 or 28 active S-boxes, and it returns 29, which is the smallest possible value for which there are byte-consistent abstracted differentials.

When comparing CPU times needed to solve *Step1-opt* with $\mathrm{S1_{Diff}}$ and $\mathrm{S1_{XOR}}$, we note that $\mathrm{S1_{XOR}}$ is faster on all instances. $\mathrm{S1_{XOR}}$ is at most 4.9 times times as fast as $\mathrm{S1_{Diff}}$ (AES-192-9). The hardest instance, AES-192-10, is solved in less than one hour with $\mathrm{S1_{XOR}}$.

	Step1-opt					*Step1-enum*				
	$S1_{Diff}$		$S1_{XOR}$			$S1_{Diff}$		$S1_{XOR}$		
	v_1^*	t_1	v_2^*	t_2	$\frac{t_1}{t_2}$	$\#T_1$	t_3	$\#T_2$	t_4	$\frac{t_3}{t_4}$
AES-128-3	5	4	5	**3**	1,3	4	6	4	**4**	1,5
AES-128-4	12	21	12	**14**	1,5	8	74	8	**38**	1,9
AES-128-5	17	44	17	**33**	1,3	1113	32340	1113	**22869**	1,4
AES-192-3	1	3	1	**2**	1,5	15	16	15	**10**	1,6
AES-192-4	4	8	4	**5**	1,6	4	12	4	**7**	1,7
AES-192-5	5	14	5	**8**	1,8	2	13	2	**9**	1,4
AES-192-6	10	34	10	**18**	1,9	6	65	6	**45**	1,4
AES-192-7	13	72	13	**37**	1,9	4	98	4	**66**	1,5
AES-192-8	18	205	18	**73**	2,8	8	752	8	**333**	2,2
AES-192-9	24	2527	24	**520**	4,9	240	43359	240	**13524**	3,2
AES-192-10	27	3715	**29**	**3285**	1,1	27548	-	**602**	**216120**	-
AES-256-3	1	3	1	3	1	33	39	33	**29**	1,3
AES-256-4	3	8	3	**7**	1,1	14	38	14	**25**	1,5
AES-256-5	3	13	3	**8**	1,6	4	21	4	**15**	1,4
AES-256-6	5	25	5	**17**	1,5	3	29	3	**20**	1,5
AES-256-7	5	48	5	**47**	1	1	22	1	**15**	1,5
AES-256-8	10	61	10	**49**	1,2	3	76	3	**52**	1,5
AES-256-9	15	172	15	**106**	1,6	16	705	16	**430**	1,7
AES-256-10	16	236	16	**112**	2,1	4	385	4	**224**	1,7
AES-256-11	20	488	20	**286**	1,7	4	705	4	**312**	2,3
AES-256-12	20	625	20	**140**	4,5	4	1228	4	**463**	2,7
AES-256-13	24	1621	24	**822**	2	4	1910	4	**597**	3,2
AES-256-14	24	2179	24	**682**	3,2	4	1722	4	**607**	2,8

Table 7.1: Comparison of $S1_{Diff}$ and $S1_{XOR}$ for solving Step 1. For each instance, we display the results with $S1_{Diff}$ and $S1_{XOR}$ for *Step1-opt* (optimal value v_1^* (resp. v_2^*) of obj_{Step1}, and time t_1 (resp. t_2 in seconds) and *Step1-enum* (number $\#T_1$ (resp $\#T_2$) of truncated differentials when obj_{Step1} is assigned to the value v_1^* found with $S1_{XOR}$, and time t_3 (resp. t_4) in seconds). We report '-' when the time exceeds two weeks. We also give the speedup $\frac{t_1}{t_2}$ for *Step1-opt*, and $\frac{t_3}{t_4}$ for *Step1-enum*. The machine used for these experiments is the one mentioned in Section 6.4.2.

Results for *Step1-enum* In Table 7.1, we report results for solving *Step1-enum* when v is fixed to the optimal value v^* found when solving *Step1-opt* with $S1_{XOR}$. $S1_{XOR}$ is always faster than $S1_{Diff}$, and it is able to solve all instances but three in less than 607 seconds. The three most challenging instances are AES-128-5, AES-192-9, and AES-192-10, which are solved by $S1_{XOR}$ in less than 7 hours (22869 seconds), 4 hours (13524 seconds) and 60 hours (216120 seconds), respectively. $S1_{Diff}$ is able to solve AES-128-5 and AES-192-9 in less than 9 and 12 hours (32340 and 43339 seconds), respectively. However, AES-192-10 cannot be solved by $S1_{Diff}$ within two weeks. Actually, for this instance, $S1_{XOR}$ strongly reduces the number of solutions: There are 602 abstracted differentials with $S1_{XOR}$ instead of 27548 with $S1_{Diff}$[1]. For *Step1-enum*, $S1_{XOR}$ is at most 3,2 times as fast as $S1_{Diff}$ (AES-192-9 and AES-256-13).

Note that reducing the number of abstracted differentials is very important to reduce the total solving time as for each abstracted differential of Step 1, we need to search for an optimal byte solution (or prove that it is not byte-consistent, as for the abstracted differentials found with $S1_{Diff}$ but not with $S1_{XOR}$). In Section 7.5, we propose another decomposition to solve Step1 even faster.

[1]The number of solutions with $S1_{Diff}$ has been found by decomposing *Step1-enum* into two sub-steps that have been solved by Chuffed and Picat, respectively, in order to take advantage of the complementarity of the two solvers: Picat has been used to compute the number of differences per round whereas Chuffed has been used to enumerate abstracted differentials given the number of differences per round (see [Gerault et al., 2017a] for more details).

7.4 CP model for Step 2

Given an abstracted differential computed by *Step1-enum*, Step 2 aims at searching for the byte values with the highest differential probability (or proving that the abstracted differential is not byte-consistent). In this section, we describe the CP model introduced in [Gerault et al., 2016] for AES-128, extended to AES-192 and AES-256 in a straightforward way.

7.4.1 Variables

For each differential byte $\delta B \in \mathit{diffWords}_l$, we define an integer variable the domain of which depends on the value of ΔB in the abstracted differential: If $\Delta B = 0$, then $D(\delta B) = \{0\}$; otherwise, $D(\delta B) = [1, 255]$.

As mentioned in Section 6.4.2, we declare an integer variable $P_{\delta B}$ for each differential byte $\delta B \in Sboxes_l$: This variable corresponds to the base 2 logarithm of the probability $Pr(\delta B \rightarrow \delta SB)$ of obtaining the S-box output difference δSB when the S-box input difference is δB. The domain of $P_{\delta B}$ depends on the value of ΔB in the abstracted differential: If $\Delta B = 0$, then $Pr(0 \rightarrow 0) = 1$ and therefore $D(P_{\delta B}) = \{0\}$; otherwise, $Pr(\delta B \rightarrow \delta SB) \in \{\frac{2}{256}, \frac{4}{256}\}$ and $D(P_{\delta B}) = \{-7, -6\}$. These values are derived from the properties of the SBox of the AES. The constraint associated with the SubBytes operation forbids couples $(\delta B, \delta SB)$ such that $Pr(\delta B \rightarrow \delta SB) = 0$.

7.4.2 Constraints

The constraints basically follow the AES operations to relate variables, as described in Section 7.3.1 for Step 1, but consider the definition of the operations at the byte level, instead of the Boolean level.

The main difference is that the SubBytes operation, which has no effect at the Boolean level, must be modelled at the byte level. This is done thanks to a ternary table constraint which extensively lists all triples (X, Y, P) such that there exists two bytes B_1 and B_2 whose difference before and after passing through S-Boxes is equal to X and Y, respectively, and such that P is the probability of this transformation: For all $\delta B \in Sboxes_l$, we add the constraint:

$$\left.\begin{array}{l}(\delta B, \delta SB, P_{\delta B}) \in \{(X, Y, P) | \exists (B_1, B_2) \in [0,255] \times [0,255], X = B_1 \oplus B_2, \\ Y = S(B_1) \oplus S(B_2), P = \log_2(Pr(X \rightarrow Y))\}.\end{array}\right\} (C_{SBS2})$$

We also use a table constraint for the XOR of two bytes, as defined in Section 6.4.2, with tuples tupleXOR:

$$XORByte(\delta a, \delta b, \delta c) \equiv (\delta a, \delta b, \delta c) \in \mathsf{tupleXOR}. \qquad (XORByte)$$

In what follows, we assume that $C_{XORByte}$ can take an arbitrary number of arguments (the last one being the result), event though in the implementation, when a XOR of multiple variables is needed, we XOR them two by two and use temporary variables to store the intermediary results.

The MixColumn operation also needs to be modelled at byte level. MixColumn is the multiplication of each column of the state by a fixed matrix M, given in section 7.2, in Rijndael's Galois field, in which addition is replaced by XOR (see [Daemen and Rijmen, 2002] for more details). For instance, let us consider a column $C = (\delta a, \delta b, \delta c, \delta d)^t$. Then

$$MC(C) = \begin{pmatrix} (2 \cdot \delta a) \oplus (3 \cdot \delta b) \oplus \delta c \oplus \delta d \\ \delta a \oplus (2 \cdot \delta b) \oplus (3 \cdot \delta c) \oplus \delta d \\ \delta a \oplus \delta b \oplus (2 \cdot \delta c) \oplus (3 \cdot \delta d) \\ (3 \cdot \delta a) \oplus \delta b \oplus \delta c \oplus (2 \cdot \delta d) \end{pmatrix}$$

where $\cdot$ is multiplication in the a finite field called Rijndael's Galois field. We did not implement the multiplication rules, but used multiplication tables instead. AES uses multiplications by 2 and 3 for encryption, and by 9, 11, 13 and 14 for decryption. For MC, we use a table $\mathsf{tupleMUL}_{2\oplus 3}$, such that

$$\mathsf{tupleMUL}_{2\oplus 3} = \{(a, b, c) \in [0; 255]^3, c = (2 \cdot a) \oplus (3 \cdot c).$$

For its inverse, we use 4 tables such that

$$\forall j \in [0;255], \mathsf{tupleMUL}_i[j] = i \cdot j$$

where $\cdot$ is the multiplication in Rijndael's finite field, and $i \in \{9,11,13,14\}$. The multiplication tables we use are given in Appendix A.

We implement both MC and its inverse. Indeed, it permits the solver to obtain the k^{th} column of δY_i directly from the k^{th} column of δZ_i, instead of having to enumerate all possible values and applying MC in the forward direction to verify whether the affectation is correct.

The constraint for MC is defined as follows, using temporary variables t_z:

$$\left.\begin{array}{ll}\forall i \in [0;r-2], \forall k \in [0;3] & \\ & table((\delta Y_i[0][k], \delta Y_i[1][k], t_0), \mathsf{tupleMUL}_{2\oplus 3}) \\ \wedge & \mathrm{XORB}yte(t_0, \delta Y_i[2][k], \delta Y_i[3][k], \delta Z_i[0][k]) \\ \wedge & table((\delta Y_i[1][k], \delta Y_i[2][k], t_1), \mathsf{tupleMUL}_{2\oplus 3}) \\ \wedge & \mathrm{XORB}yte(t_1, \delta Y_i[0][k], \delta Y_i[3][k], \delta Z_i[1][k]) \\ \wedge & table((\delta Y_i[2][k], \delta Y_i[3][k], t_2), \mathsf{tupleMUL}_{2\oplus 3}) \\ \wedge & \mathrm{XORB}yte(t_2, \delta Y_i[0][k], \delta Y_i[1][k], \delta Z_i[2][k]) \\ \wedge & table((\delta Y_i[3][k], \delta Y_i[0][k], t_3), \mathsf{tupleMUL}_{2\oplus 3}) \\ \wedge & \mathrm{XORB}yte(t_3, \delta Y_i[1][k], \delta Y_i[1][k], \delta Z_i[3][k]) \end{array}\right\} (\mathrm{C}_{\mathrm{MCS2-1}})$$

And for the last round, where MC is skipped:

$$\forall j,k \in [0;3]^2 : \mathrm{DZ}_{r-1}[j][k] = \mathrm{DY}_{r-1}[j][k]. \qquad (\mathrm{C}_{\mathrm{MCS2-2}})$$

The $\mathrm{C}_{\mathrm{MCS2}}$ constraint is the set $\{\mathrm{C}_{\mathrm{MCS2-1}}, \mathrm{C}_{\mathrm{MCS2-2}}\}$.

The inverse of the MC operation is defined similarly: We first define a temporary variable $\mathrm{M}ul^z[j][k]$ holding the product of each word of the input column by 9, 11, 13 and 14:

$$\forall i \in [0;r-2], \forall (j,k) \in [0;3]^2, z \in \{9,11,13,14\} table((\delta Y_i[j][k], \mathrm{M}ul_i^z[j][k]), \mathsf{tupleMUL}_z). \qquad (\mathrm{INVMCS2\text{-}1})$$

We then use these variables to define the inverse operation of MC:

$$\left.\begin{array}{l}\forall i \in [0;r-2], \forall k \in [0;3] \\ \mathrm{XORB}yte(\mathrm{M}ul^{14}[0][k], \mathrm{M}ul^{11}[1][k], \mathrm{M}ul^{13}[2][k], \mathrm{M}ul^{9}[3][k]\delta Z_i[0][k]) \\ \mathrm{XORB}yte(\mathrm{M}ul^{9}[0][k], \mathrm{M}ul^{14}[1][k], \mathrm{M}ul^{11}[2][k], \mathrm{M}ul^{13}[3][k]\delta Z_i[1][k]) \\ \mathrm{XORB}yte(\mathrm{M}ul^{13}[0][k], \mathrm{M}ul^{9}[1][k], \mathrm{M}ul^{14}[2][k], \mathrm{M}ul^{11}[3][k]\delta Z_i[2][k]) \\ \mathrm{XORB}yte(\mathrm{M}ul^{11}[0][k], \mathrm{M}ul^{13}[1][k], \mathrm{M}ul^{9}[2][k], \mathrm{M}ul^{14}[3][k]\delta Z_i[3][k]) \end{array}\right\} (\mathrm{C}_{\mathrm{INVMCS2-2}})$$

The $\mathrm{C}_{\mathrm{INVMCS2}}$ constraint is the set $\{\mathrm{C}_{\mathrm{INVMCS2-1}}, \mathrm{C}_{\mathrm{INVMCS2-2}}\}$.

The constraint for SR is described in the same way as for Step 1:

$$\forall i \in [0, r-1], \forall j,k \in [0,3] : \delta Y_i[j][k] = \delta SX_i[j][(j+k)\%4]. \qquad (\mathrm{C}_{\mathrm{SRS2}})$$

The constraint for ARK uses $\mathrm{XORB}ytes$:

$$\left.\begin{array}{l}\forall (j,k) \in [0,3]^2 : \mathrm{XOR}(\delta X[j][k], \delta K_0[j][k], \delta X_0[j][k]), \\ \forall i \in [1,r], \forall (j,k) \in [0,3]^2 : \mathrm{XOR}(\delta Y_{i-1}[j][k], \delta K_i[j][k], \delta X_i[j][k]). \end{array}\right\} (\mathrm{C}_{\mathrm{ARKS2}})$$

Finally, KS is defined in a very straightforward way:

$$\left.\begin{array}{l}\forall j \in [0;3], \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC} < 8, k\%\mathrm{KC} = 0 : \mathrm{XORB}yte((\delta EK[j][k], \delta SK[j][k-1], \delta EK[j][k-\mathrm{KC}]), \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC} = 8, k\%4 = 0 : \mathrm{XORB}yte(\delta EK[j][k], \delta SK[j][k-1], \delta EK[j][k-\mathrm{KC}]), \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC} < 8 k\%\mathrm{KC} \neq 0 : \mathrm{XORB}yte(\delta EK[j][k], \delta EK[j][k-1], \delta EK[j][k-\mathrm{KC}]), \\ \forall k \in [\mathrm{KC}; 4\cdot(r+1)-1], \mathrm{KC} = 8 k\%4 \neq 0 : \mathrm{XORB}yte(\delta EK[j][k], \delta EK[j][k-1], \delta EK[j][k-\mathrm{KC}]). \end{array}\right\} (\mathrm{C}_{\mathrm{KSS2}})$$

	#T	#B	p	t	$\frac{t}{\#T}$
AES-128-3	4	2	2^{-31}	10	2.5
AES-128-4	8	8	2^{-75}	40	5
AES-128-5	1113	97	2^{-105}	235086	211.2
AES-192-3	15	15	2^{-6}	15	1
AES-192-4	4	4	2^{-24}	13	3.3
AES-192-5	2	2	2^{-30}	11	5.5
AES-192-6	6	6	2^{-60}	35	5.8
AES-192-7	4	4	2^{-78}	46	11.5
AES-192-8	8	8	2^{-108}	119	14.9
AES-192-9	240	80	2^{-146}	35254	146.9
AES-192-10	602	202	2^{-176}	55310	91.9
AES-256-3	33	33	2^{-6}	26	0.8
AES-256-4	14	14	2^{-18}	25	1.8
AES-256-5	4	4	2^{-18}	12	3
AES-256-6	3	3	2^{-30}	11	3.7
AES-256-7	1	1	2^{-30}	9	8.8
AES-256-8	3	1	2^{-60}	19	6.3
AES-256-9	16	16	2^{-92}	457	28.6
AES-256-10	4	4	2^{-98}	160	40
AES-256-11	4	4	2^{-122}	178	44.5
AES-256-12	4	4	2^{-122}	237	59.3
AES-256-13	4	4	2^{-146}	244	61
AES-256-14	4	4	2^{-146}	302	75.5

Table 7.2: Results of Choco for solving Step 2. For each instance, we display: the number #T of truncated differentials, the number #B of byte-consistent truncated differentials, the maximal probability p among all byte-consistent truncated differentials, the total time t for solving Step 2 for all truncated differentials, and the average time $\frac{t}{\#T}$ for solving Step 2 for one truncated differential. Times are in seconds.

Objective function The goal is to find a byte-consistent solution with maximal differential probability. As we consider logarithms, this amount to searching for a solution that maximises the sum of all $P_{\delta B}$ variables. Hence, we introduce an integer variable obj_{Step2} which is constrained to be equal to the sum of all $P_{\delta B}$ variables:

$$obj_{Step2} = \sum_{\delta B \in Sboxes_l} P_{\delta B}$$

and we define the objective function as the maximisation of obj_{Step2}. The domain of obj_{Step2} is derived from the number of differences that pass trough S-boxes in the abstracted differentials, *i.e.*, $D(obj_{Step2}) = [-7 \cdot v, -6 \cdot v]$ where $v = \sum_{\delta B \in Sboxes_l} \Delta B$.

We sum up the constraints used in Step2: C_{SBS2}, $C_{XORByte}$, C_{MCS2}, $C_{INVMCS2}$, C_{SRS2}, C_{ARKS2}, C_{KSS2}.

Experimental evaluation Step 2 is implemented in Choco [Prud'homme et al., 2016], and we have used the *domOverWDeg* variable ordering heuristic, and the *lastConflict* search strategy that are predefined in Choco. The domOverWDeg implements the domain over weighted degree black-box search strategy [Boussemart et al., 2004], which orders variables according to a metric which considers the ratio between the size of their domain, and the number of constraints they are involved in. The lastConflict heuristic [Lecoutre et al., 2009] consists in finding the variable at the origin of an incorrect solution, in order to change the assignment of the variable which is at the root of the conflict first.

In Table 7.2, we display the CPU time needed to search for optimal differential characteristics, given the set of abstracted differentials computed by *Step1-enum*. For all instances but three (AES-128-5, AES-192-9, and AES-192-10), the time needed to find the optimal solution given one abstracted differential is smaller than 100 seconds, and the number of abstracted differentials is smaller than 20, so that the optimal solution for all abstracted differentials is found in less than 500 seconds. However, for AES-128-5 (resp. AES-192-9 and AES-192-10), the average solving time per abstracted differential is 210 (resp. 147 and 92) seconds, and there are 1236 (resp. 240 and 602) abstracted differentials so that the total solving time for all abstracted differentials exceeds 65 hours (235036 seconds) (resp. 9 hours (35254 seconds) and 15 hours (55310 seconds)). Note that for these three instances most abstracted differentials are not byte consistent: Among the 1236 (resp. 240 and 602) abstracted differentials enumerated by *Step1-enum*, only 97 (resp. 13 and 202) are byte-consistent.

7.5 A different two-step decomposition

Given an abstracted differential, Step 2 is rather quickly solved, as shown in Table 7.2. However, in some cases the number of abstracted differentials is quite large, and it becomes rather time-consuming to solve Step 2 for all abstracted differentials.

To reduce the number of abstracted differentials, we propose to shift the frontier between Steps 1 and 2. More precisely, we modify the goal of *Step1-enum*: Instead of enumerating all Boolean solutions, we restrict our attention to the variables associated with bytes that pass through S-boxes, i.e., we enumerate all assignments of Boolean variables associated with the differential bytes in $Sboxes_l$. For AES-128, this amounts to enumerating all assignments of $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ that belong to Boolean solutions (in other words, we do not enumerate the values of $\Delta K_i[j][k]$ with $k \in [0,2]$, and we do not enumerate the values of $\Delta Y_i[j][k]$ and $\Delta Z_i[j][k]$).

Step 2 is adapted to integrate the fact that abstracted differentials do not assign values to some variables. More precisely, the domains of the variables that are not assigned in an abstracted differential is $[0,255]$.

This simple reduction of the scope of abstracted differentials strongly reduces the number of different assignments (as many assignments only differ on values assigned to ΔY_i, ΔZ_i, or $\Delta K_i[j][k]$ with $k \in [0,2]$), without increasing the size of the search space to explore in Step 2. Indeed, the values of δY_i, δZ_i, and $\delta K_i[j][k]$ with $k \in [0,2]$ are deterministically inferred from the values of the variables associated with inputs and outputs of S-boxes (*i.e.*, δX_i, δSX_i, $\delta K_i[j][3]$, and $\delta SK_i[j][3]$ for AES-128).

Table 7.3 gives the results obtained with this decomposition, for shifted Step 1 and shifted Step 2. For shifted Step 1, we compare $S1_{Diff}$ and $S1_{XOR}$ for the enumeration problem. Again, $S1_{XOR}$ is faster than $S1_{Diff}$, and it is able to solve all instances but two in less than 7 minutes. The two hardest instances are AES-128-5, which is solved in 24 minutes (1409 seconds), and AES-192-10, which is solved in less than 4 hours (13558 seconds). This last instance cannot be solved with $S1_{Diff}$ within three days: We have stopped the solution process after three days and report in Table 7.3 the number of abstracted differentials enumerated within these three days.

Both $S1_{Diff}$ and $S1_{XOR}$ find the same number of abstracted differentials (#T) for all instances but one. For instance AES-192-10 there are 7 abstracted differentials with $S1_{XOR}$, and more than 40 with $S1_{Diff}$. As expected there are less abstracted differentials with the shifted decomposition than with the original one for many instances. In some cases the reduction is drastic: from 1236 (resp. 240 and 602) to 103 (resp. 3 and 7) for AES-128-5 (resp. AES-192-9 and AES-192-10). As a consequence, the time needed to enumerate all abstracted differentials is also smaller with the shifted decomposition. In some cases, the speed-up is important. For example, with $S1_{XOR}$, *Step1-enum* is solved in more than 6 (resp. 3, and 60) hours, or 22869 (resp. 13524 and 216120 seconds) with the initial decomposition for AES-128-5 (resp. AES-192-9, and AES-192-10), whereas it is solved in less than 24 (7, and 226) minutes (resp. 1409, 386 and 13558 seconds) with the shifted decomposition.

	Shifted Step 1				Shifted Step 2			Total time	
	$S1_{Diff}$		$S1_{XOR}$					*seq*	*par*
	#T	t_1	#T	t_1	#B	t_2	$\frac{t_2}{\#T}$		
AES-128-3	2	3	2	**2**	2	7	3.5	12	9
AES-128-4	1	16	1	**8**	1	13	12.6	35	35
AES-128-5	103	2651	103	**1409**	27	52313	507.9	53755	1950
AES-192-3	14	14	14	**8**	14	19	1.4	29	11
AES-192-4	2	7	2	**4**	2	7	3.5	16	13
AES-192-5	1	9	1	**4**	1	4	3.8	16	16
AES-192-6	2	27	2	**11**	2	14	7.0	43	36
AES-192-7	1	41	1	**17**	1	7	7.4	61	61
AES-192-8	1	221	1	**57**	1	8	8.2	138	138
AES-192-9	3	1720	3	**386**	3	109	36.3	1015	939
AES-192-10	≥40	-	7	**13558**	7	281	40.1	17124	16883
AES-256-3	33	34	33	**23**	33	36	1.1	62	27
AES-256-4	10	25	10	**14**	10	24	2.4	45	23
AES-256-5	4	20	4	**10**	4	15	3.8	33	22
AES-256-6	3	27	3	**12**	3	16	5.3	45	34
AES-256-7	1	21	1	**8**	1	7	7.4	62	62
AES-256-8	2	55	2	**18**	2	14	7.0	81	74
AES-256-9	4	203	4	**63**	4	69	17.3	238	186
AES-256-10	1	99	1	**41**	1	45	45.3	198	198
AES-256-11	1	320	1	**77**	1	28	27.8	391	391
AES-256-12	1	258	1	**89**	1	35	35.2	264	264
AES-256-13	1	694	1	**140**	1	46	46.0	1008	1008
AES-256-14	1	1087	1	**97**	1	35	34.8	814	814

Table 7.3: Results with the shifted decomposition. For each instance, we display: the results for the shifted Step 1 with $S1_{Diff}$ and $S1_{XOR}$ (#T = number of truncated differentials; t_1 = time spent by Picat-SAT), the results for the shifted Step 2 (#B = number of Byte-consistent truncated differentials; t_2 = time spent by Choco for all truncated differentials; $\frac{t_2}{\#T}$ = average time per truncated differential), and total time for solving the whole problem by Algorithm 1 with $S1_{XOR}$ and the new decomposition (*seq* = time when using a single core = time for solving *Step1-opt* + t_1 + t_2; *par* = time when using #T cores in parallel for solving Step 2 = time for solving *Step1-opt* + t_1 + $\frac{t_2}{\#T}$). Times are in seconds, and we report '-' when time exceeds 3 days.

For all instances but one (AES-128-5), every abstracted differential computed by *Step1-enum* is byte consistent. However, for AES-128-5, only 27 abstracted differentials, among the 103 computed by *Step1-enum*, are byte consistent.

The time needed to find the optimal differential characteristics given one abstracted differential ($\frac{t2}{\#T}$) is rather comparable to the one displayed in Table 7.2, for the initial decomposition: it is larger for 9 instances, equal for 3 instances, and smaller for 10 instances. As there are less abstracted differentials with the shifted decomposition than with the initial one, the total time for Step 2 ($t2$) is often smaller and, for the most challenging instances it is much smaller: it is larger than 65 hours (resp. 9 hours, and 15 hours) with the initial decomposition for AES-128-5 (resp. AES-192-9, and AES-192-10), whereas it is smaller than 15 hours (9 minutes, and 4 hours) with the shifted decomposition.

The last column of Table 7.3 gives the time needed to solve the whole problem as described in Algorithm 1, *i.e.*, solve *Step1-opt* with $S1_{XOR}$, then solve *Step1-enum* with $S1_{XOR}$ and the shifted Step 1, and finally solve Step 2 for each abstracted differential computed by *Step1-enum* (for all instances, the loop lines 6 to 10 is executed only once as the exit condition is satisfied when $v = v^*$). This total time is smaller than one hour for all instances but two, and 11 instances are solved in less than one minute. The two hardest instances are AES-128-5, which is solved in less than 15 hours, and AES-192-10, which is solved in less than 5 hours. However, these two instances are rather different: For AES-128-5, there are 103 abstracted differentials and most of the solving time is spent in Step 2, to compute the optimal probability given these abstracted differentials; For AES-192-10 there are 7 abstracted differentials and most of the solving time is spent to enumerate them whereas Step 2 is solved in less than 5 minutes. Note that Step 2 may be run in parallel for each abstracted differential. In this case, the solving time of AES-128-5 may be decreased to less than one hour if we solve Step 2 in parallel for the 103 abstracted differentials.

This is a clear improvement over dedicated approaches such as [Fouque et al., 2013], which cannot be extended to keys of size $l > 128$ bits, due to its memory complexity, and the approach of [Biryukov and Nikolic, 2010], which needs several weeks to solve Step 1 for AES-192.

As already pointed out in [Gerault et al., 2016], for AES-128-4, find a byte-consistent solution with $obj_{Step1} = 12$ and a probability equal to 2^{-75}. This solution is better than the solution claimed to be optimal in [Biryukov and Nikolic, 2010] and [Fouque et al., 2013]: In these papers, the authors claim that the best byte-consistent solution has $obj_{Step1} = 13$, and a probability equal to 2^{-81}. Our solution is given in Figure 7.4.

Finally, we find better solutions for AES-256. In particular, we have computed the actual optimal differential characteristics for AES-256-14, and its probability is 2^{-146}, instead of 2^{-154} for the one given in [Biryukov and Khovratovich, 2009]. Our solution is given in Figure 7.5.

Round	δX	δK_i	Pr(States)	Pr(Key)
init.	0a 00 0a 02 0a bc 06 06 0a 0c 06 06 0a bc 06 06			
$i = 0$	00 00 00 00 00 bc 00 00 00 00 00 00 00 bc 00 00	0a 00 0a 02 0a 00 06 06 0a 0c 06 06 0a 00 06 06	$2^{-6.2}$	$2^{-6.3}$
1	00 00 00 00 00 0c 00 00 00 0c 00 00 00 00 00 00	0a 0c 06 06 00 0c 00 00 0a 00 06 06 00 00 00 00	$2^{-6.2}$	–
2	00 00 00 00 00 0c 00 00 00 00 00 00 00 00 00 00	0a 0c 06 06 0a 0c 06 06 00 00 00 00 00 00 00 00	2^{-6}	–
3	00 00 00 00 00 0c 00 00 00 0c 00 00 00 0c 00 00	0a 0c 06 06 00 0c 00 00 00 0c 00 00 00 0c 00 00	$2^{-7.3}$	2^{-6}
4	0b 91 06 06 0b 91 06 06 0b 91 06 06 0b 00 06 06	0b 0c 06 06 0b 00 06 06 0b 0c 06 06 0b 00 06 06	$2^{-6.2}$	–

Table 7.4: Our 4-round related-key differential characteristic for AES-128, which occurs with probability 2^{-75}.

Round	δX	δK_i	Pr(States)	Pr(Key)
init.	ad db db 76 ad db db 76 ad db db 76 ad db db 76			
$i = 0$	69 00 00 00 00 00 00 00 69 00 00 00 00 00 00 00	c4 db db 76 ad db db 76 c4 db db 76 ad db db 76	$2^{-6.2}$	–
1	9a 00 00 00 00 00 00 00 9a 00 00 00 00 00 00 00	b5 9a 9a b5 00 00 00 00 b5 9a 9a b5 00 00 00 00	$2^{-6.2}$	–
2	69 00 00 00 69 00 00 00 00 00 00 00 00 00 00 00	c4 db db 76 69 00 00 00 ad db db 76 00 00 00 00	$2^{-6.2}$	–
3	9a 00 00 00 9a 00 00 00 00 00 00 00 00 00 00 00	b5 9a 9a b5 b5 9a 9a b5 00 00 00 00 00 00 00 00	$2^{-6.2}$	–
4	69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	c4 db db 76 ad db db 76 00 00 00 00 00 00 00 00	2^{-6}	–
5	9a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	b5 9a 9a b5 00 00 00 00 00 00 00 00 00 00 00 00	2^{-6}	–
6	69 00 00 00 69 00 00 00 69 00 00 00 69 00 00 00	c4 db db 76 69 00 00 00 69 00 00 00 69 00 00 00	$2^{-6.4}$	2^{-6}
7	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2f 9a 9a b5 2f 9a 9a b5 2f 9a 9a b5 2f 9a 9a b5	–	$2^{-7.2} \times 2^{-6.2}$
8	69 00 00 00 00 00 00 00 69 00 00 00 00 00 00 00	69 00 00 00 00 00 00 00 69 00 00 00 00 00 00 00	$2^{-6.2}$	–
9	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2f 9a 9a b5 00 00 00 00 2f 9a 9a b5 00 00 00 00	–	–
10	69 00 00 00 69 00 00 00 00 00 00 00 00 00 00 00	69 00 00 00 69 00 00 00 00 00 00 00 00 00 00 00	$2^{-6.2}$	–
11	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2f 9a 9a b5 2f 9a 9a b5 00 00 00 00 00 00 00 00	–	–
12	69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2^{-6}	–
13	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2f 9a 9a b5 00 00 00 00 00 00 00 00 00 00 00 00	–	–
14	69 00 00 00 69 00 00 00 69 00 00 00 69 00 00 00	69 00 00 00 69 00 00 00 69 00 00 00 69 00 00 00	–	–

Table 7.5: Our 14-round related-key differential characteristic for AES-256, which occurs with probability 2^{-146}.

7.6 Conclusion

In this chapter, we present 3 CP models for the search of optimal related-key differential characteristics on the AES. The first one, $S1_{Basic}$, is a straightforward implementation of the propagation rules for the differences in the AES. It finds too many inconsistent solutions to be usable. The second one, $S1_{Diff}$, introduces reasoning about equalities that can be inferred from the key schedule, as well as a constraint that propagates the MDS property of the MC operation between two columns of differential bytes. This model can solve all instances but one in less than 13 hours. However, the remaining instance cannot be solved after 2 weeks of computation. We therefore introduce $S1_{XOR}$, which introduces more advanced reasoning about the equalities in the key schedule, by considering all possible XORs that can be built between words of the key. Using this model, the instance which could not be solved with SA_{Diff}, AES-192-10, is solved in approximately 60 hours (216120 seconds). Moreover, the number of inconsistent solutions for this instance is greatly reduced, as the total number of solutions goes from 27548 for $S1_{Diff}$ to 602 for $S1_{XOR}$. We then introduce a shift in the frontier between Step 1 and Step 2: instead of enumerating the assignments of all the Boolean variables in Step 1, we only enumerates the assignments for the Boolean variables which abstract words that go through SBoxes, *i.e.*, the $SBoxes_l$ set. We adapt Step 2 accordingly. This shift greatly reduces the number of solutions found in Step 1. The most significantly changes are for AES-128-5, AES-192-9 and AES-192-10: from $S1_{XOR}$ to the shifted $S1_{XOR}$ they from 1113, 240 and 602 to 103, 3 and 7 solutions respectively. This reduction in the number of solutions permits to reduce the time needed for Step 2, from 235086 to 52313 seconds for AES-128-5, from 35254 to 109 seconds for AES-192-9, and from 55310 to 281 seconds for AES-192-10.

It would be interesting to extend this approach to Rijndael, which allows more key and block sizes than the AES. Some instances, such as the one with 256-bit keys and 256-bit blocks, are an interesting evaluation of the scale up properties of CP, due to the larger search space.

Chapter 8

Related Key Differential Cryptanalysis of Midori with CP

Contents

Midori is a lightweight block cipher with 2 versions, Midori64 and Midori128, which was introduced in [Banik et al., 2015]. It was designed for devices with low computational resources.Many distance bounding protocols are designed to be computationally efficient in order to run on such devices, so that Midori is susceptible to be used for contactless applications.

While several attack models are discussed by the authors of Midori, the authors made no claims concerning the security of Midori against related-key differential attacks. Its structure is very close to that of the AES, so that we can apply our CP methods to search for optimal related-key differential characteristics on Midori. This permits us to find full-round related-key differential characteristics for both versions of Midori, and to present related-key differential attacks for Midori64 (resp. Midori128) which require 2^{36} (resp 2^{44}) operations.

8.1 Introduction

The increasing usage of embedded devices led to a lot of research on how to adapt existing cryptographic primitives for the low power and energy constraints associated with the internet of things. Lightweight block ciphers follow this principle, and aim at providing energy efficient ways to encrypt data. The authors of Midori [Banik et al., 2015] consider the problem of minimising the

Type	Rounds	Data	Time	Reference
Midori64				
Impossible differential	10	$2^{62,4}$	$2^{80,81}$	[Chen and Wang, 2016]
Impossible differential	10	$2^{61,97}$	$2^{87,71}$	[Shahmirzadi et al., 2018]
Impossible differential	11	$2^{61,87}$	$2^{90,63}$	[Shahmirzadi et al., 2018]
Impossible differential	12	$2^{61,87}$	$2^{90.51}$	[Shahmirzadi et al., 2018]
Meet-in-the-middle	12	$2^{55.5}$	$2^{125,5}$	[Lin and Wu, 2015]
Invariant subspace[1]	full(16)	2	2^{16}	[Guo et al., 2016]
Related-key differential	14	2^{59}	2^{116}	[Dong and Shen, 2016]
Related-key differential	**full(16)**	2^{24}	2^{36}	**Section 8.6.2**
Midori128				
Impossible differential	11	2^{121}	$2^{122,75}$	[Tolba et al., 2017]
Related-key differential	**full(20)**	2^{44}	2^{44}	**Section 8.6.3**

Table 8.1: Summary of the attacks against Midori

energy cost for a lightweight block cipher. They propose a lightweight symmetric block cipher scheme called *Midori*, composed of two versions Midori64 and Midori128, which respectively cipher 64- and 128-bit message blocks, with 128-bit keys.

In this chapter, we present CP models for the search of optimal related-key differential characteristics on Midori64 and Midori128. Using our models, we give the optimal full-round related key differential characteristics for both versions. For Midori64, the probability of the best full-round related-key differential characteristic is 2^{-16}, and for Midori128, it is 2^{-40}. Using these differential characteristics, we present key recovery attacks for both versions of Midori, which require 2^{36} operations for Midori64, and 2^{44} operations for Midori128.

The existing attacks against Midori are summed up in Table 8.1.

In [Chen and Wang, 2016], the authors propose an impossible differential attack on 10 rounds of Midori64. Impossible differential attacks use differentials with probability 0 to attack the cipher. In [Shahmirzadi et al., 2018], impossible differential attacks are proposed against 10, 11 and 12 rounds of Midori64. However, the attacks on 11 and 12 rounds are performed on versions of Midori which do not include the final XOR with the whitening key. In [Lin and Wu, 2015], Li Lin and Wenling Wu describe a meet-in-the-middle attack on 12-round Midori64. In [Guo et al., 2016], the authors exhibit a class of 2^{32} weak keys, which can be distinguished with a single query. Assuming a key from this class is used, it can be recovered recovered with as little as 2^{16} operations, and a data complexity of 2^1. Finally, a related-key cryptanalysis of Midori64 is performed in [Dong and Shen, 2016]. It covers 14 rounds and has a complexity of 2^{116}, as opposed to 2^{36} for ours. This difference is due to their differential characteristics being far from optimal.

For Midori128, the only attack apart from ours, to the best of our knowledge, is an impossible differential cryptanalysis on 11 rounds [Tolba et al., 2017]. We propose a key recovery attack on the whole cipher, requiring $2^{43.7}$ encryptions.

In Section 8.2, we give a brief description of Midori. We then present our CP model for Step 1 in Section 8.3, and for Step 2 in Section 8.4. Finally, we detail our results in Section 8.5, before concluding in the last section.

[1]Note that this attack only works if a key from the weak class is used

8.2 Midori

Both versions of Midori, Midori64 and Midori128, use 128-bit keys. In both versions, the blocks are treated as 4×4 matrices of words of m bits, with $m = 4$ for Midori64 and $m = 8$ for Midori128. The encryption process consists in applying a round function that updates an internal state S, represented as shown on Figure 8.1 (where the s_i are 4-bit words for Midori64 and 8-bit words (bytes) for Midori128), for a given number of rounds R.

$$\mathrm{S} = \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}$$

Figure 8.1: Representation of the state in Midori.

For Midori64, R is equal to 16, whereas for Midori128 R is 20[2].

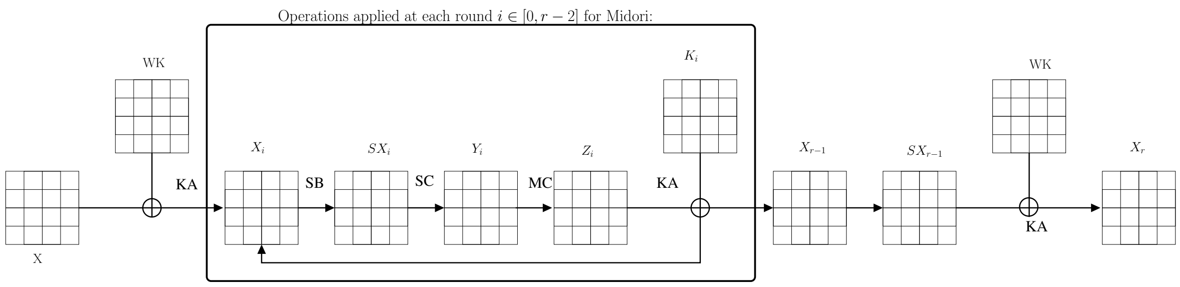

Figure 8.2: Midori ciphering process. Each 4×4 array represents a group of 16 nibbles for Midori64, and bytes for Midori128.

Midori is composed of r rounds. It uses 128-bit keys, and ciphers blocks of either 64 bits (Midori64) or 128 bits (Midori128). The number of rounds r depends on the block length: $r = 16$ (resp. 20) for Midori64 (resp. Midori128). Midori has an SPN structure and is described in Figure 8.2. Before the first round, KeyAdd (KA) is applied on the original plaintext X and the whitening key WK to obtain $\mathrm{X}_0 = \mathrm{KA}(\mathrm{X}, \mathrm{WK})$. Then, for each round $i \in [0, r-2]$: SubCell (SB) is applied on X_i to obtain $\mathrm{SX}_i = \mathrm{S}(\mathrm{X}_i)$; ShuffleCells (SC) is applied on SX_i to obtain $\mathrm{Y}_i = \mathrm{SC}(\mathrm{SX}_i)$; MixColumns (MC) is applied on Y_i to obtain $\mathrm{Z}_i = \mathrm{MC}(\mathrm{Y}_i)$; KeySchedule (KS) is applied on K_i to obtain $\mathrm{K}_i = \mathrm{KS}(\mathrm{IK}, i)$ (where IK is the initial key); and KeyAdd (KA) is applied on Z_i and K_i to obtain $\mathrm{X}_{i+1} = \mathrm{ARK}(\mathrm{Z}_i, \mathrm{K}_i)$. Finally, after $r-1$ rounds, the ciphertext is obtained by applying SubCell on X_{r-1} to obtain SB_r, and then a final KeyAdd with WK to obtain $\mathrm{X}_r = \mathrm{KA}(\mathrm{SB}_r, \mathrm{WK})$.

The round function is composed of the following consecutive operations:

SubCell (SB) A non-linear permutation, also called SBox, is applied on each word of X_i separately, according to a look-up table $\mathrm{S} : [0, 15] \to [0, 15]$ for Midori64, *i.e.*,

$$\forall i \in [0, r-1], \forall j, k \in [0, 3], \mathrm{SX}_i[j][k] = \mathrm{S}(\mathrm{X}_i[j][k]).$$

The SBox of Midori64 is given in Figure 8.3. For Midori128, 4 different SB*oxes* $\mathrm{S}_j : [0, 255] \to [0, 255]$ are used (one for each line of the state), *i.e.*,

$$\forall i \in [0, r-1], \forall j, k \in [0, 3], \mathrm{SX}_i[j][k] = \mathrm{S}_j(\mathrm{X}_i[j][k]).$$

The SBoxes of Midori128 are given in Appendix B

[2]The full specification is presented in [Banik et al., 2015].

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
SB(x)	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

Figure 8.3: The SBox of Midori64.

ShuffleCell (SC) Operates a permutation of the cells of the state. On input $(s_0, \dots, s_{15})$, it applies the following permutation:

$$P = (s_0, s_{10}, s_5, s_{15}, s_{14}, s_4, s_{11}, s_1, s_9, s_3, s_{12}, s_6, s_7, s_{13}, s_2, s_8)$$

of SX_i to obtain Y_i.

MixColumns (MC) Multiplies the state by the symmetric matrix given in Figure 8.4, thus applying a linear transformation on each column independently. It has the quasi-MDS property:

$$\forall i \in [0, r-2], \forall k \in [0;3], (\sum_{j=0}^{3} (Y_i[j][k] \neq 0) + (Z_i[j][k] \neq 0)) \in \{0,4,5,6,7,8\}.$$

and $\sum_{j=0}^{3}(Y_i[j][k]) = 0 \equiv \sum_{j=0}^{3}(Z_i[j][k]) = 0$.

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 8.4: The MixColumn matrix of Midori.

KeyAdd (KA) Before the first round, KA performs a xor between the initial plain text X and the whitening key WK to obtain X_0:

$$\forall j, k \in [0;3], X_0[j][k] = X[j][k] \oplus WK[j][k].$$

Then, at each round i, KA performs a xor between Z_i and subkey K_i to obtain X_{i+1}, *i.e.*,

$$\forall i \in [0, r-1], \forall j, k \in [0;3], X_{i+1}[j][k] = Z_i[j][k] \oplus K_{i+1}[j][k].$$

Finally, a last KA is applied to obtain the ciphertext:

$$\forall j, k \in [0;3], X_r[j][k] = SB_{r-1}[j][k] \oplus WK[j][k].$$

Key Schedule (KS) The round key derivation is very straightforard: the key K_i for each round i is obtained by XORing the initial key IK with a predefined 4×4 constant matrix α_i. For Midori64, the 128-bit key is considered as two 4×4 matrices of 4-bit nibbles IK^0 and IK^1, and WK is computed as $IK^0 \oplus IK^1$. The round key K_i is computed as $IK^{i \mod 2} \oplus \alpha_i$. For Midori128, IK is a single 4×4 bytes matrix, and WK = K. The round key K_i is then computed as $IK \oplus \alpha_i$.

8.3 CP Model for Step 1

In this section, we describe our CP models for the search of related-key differential characteristics on Midori. It is performed in two steps, as described in Chapter 6.

The model we use is similar to the $S1_{Diff}$ model (Section 7.3.2), in the sense that it includes difference variables, and a constraint on the quasi MDS property between the XOR of two differential columns. However, the key schedule contains no XOR operations, so that the $S1_{XOR}$ model would be irrelevant for Midori. Instead of using XORs to determine the equalities in the round keys, we simply use the key schedule rules: For Midori64, the differential matrices representing the round keys of the even rounds are equal, and so are the ones of the off rounds. For Midori128, the differential matrices representing the round keys are equal at all rounds. In other words, we have that $\delta K_{i1}[j][k] = \delta K_{i2}[j][k]$ if $i1\%2 = i2\%2$ for Midori 64, and $\delta K_{i1}[j][k] = \delta K_{i2}[j][k]$ for Midori128.

We first describe the variables used in our model, and then the associated constraints.

8.3.1 Variables

We first define the *diffWords* and *SBoxes* sets for Midori. For both versions of Midori, *diffWords* contains the following differential words, representing:

- The plaintext: $\delta X[j][k], \forall j \in [0;3], k \in [0,3]$
- The initial key: $\delta IK[j][k], j \in [0;3], k \in [0,3]$
- The whitening key: $\delta WK[j][k], j \in [0;3], k \in [0,3]$
- The round keys: $\delta K_i[j][k], \forall i \in [0;r-2], j \in [0;3], k \in [0, BC-1]$
- The state after KA: $\delta X_i[j][k], \forall i \in [0;r-1], j \in [0;3], k \in [0,3]$
- The state after SB: $\delta SX_i[j][k], \forall i \in [0;r-2], j \in [0;3], k \in [0,3]$
- The state after SC: $\delta Y_i[j][k], \forall i \in [0;r-2], j \in [0;3], k \in [0,3]$
- The state after MC: $\delta Z_i[j][k], \forall i \in [0;r-2], j \in [0;3], k \in [0,3]$

and the *Sboxes* set contains:

- $\delta X_i[j][k], \forall i \in [0;r-1], j \in [0,3], k \in [0, BC-1]$

As for AES, not all differential words are abstracted in Step 1. Indeed, during Step 2, the initial differential plaintext δX, as well as the whitening key WK, can be deterministically computed given the values of all other differential words. Hence, our model for Midori associates a Boolean variable ΔB with every differential byte $\delta B \in \mathit{diffBytes}_l \setminus \{\delta X[j][k], \delta WK[j][k]\}$. Each Boolean variable ΔB is assigned to 0 if $\delta B = 0$, and to 1 otherwise.

We also define an integer variable obj_{Step1} which corresponds to the number of active SBoxes. The highest transition probability through the SBoxes of both versions of Midori is 2^{-2}, so up to 64 SBoxes can be active before the probability goes below 2^{-128} (the key size being 128 bits for both versions). The domain of the obj_{Step1} is set to $D(obj_{Step1}) = [1, 64]$ accordingly.

The corresponding list of variables is the following. We use the $\prec$ symbol, as defined in Section 6.4.1, to denote the fact that a differential bit Δa is a Boolean abstraction of the differential word δa.

ΔIK: For Midori 64, the key is interpreted as two 4×4 matrices IK^0 and IK^1, so we have

$$\forall i \in [0;1], \forall j \in [0;3], k \in [0;3], \Delta IK^i[j][k] \prec \delta IK^i[j][k],$$

and for Midori128, where the key is interpreted as a single 4×4 matrix,

$$\forall j \in [0;3], k \in [0;3], \Delta IK[j][k] \prec \delta IK[j][k].$$

ΔK_i: $\forall i \in [0;r-2], j \in [0;3], k \in [0;3], \Delta K_i[j][k] \prec \delta K_i[j][k]$

ΔX: $\forall i \in [0;r-1], j \in [0;3], k \in [0;3], \Delta X_i[j][k] \prec \delta X_i[j][k]$

ΔSB: $\forall i \in [0;r-2], j \in [0;3], k \in [0;3], \Delta SB_i[j][k] \prec \delta SB_i[j][k]$

ΔY: $\forall i \in [0; r-2], j \in [0;3], k \in [0;3], \Delta Y_i[j][k] \prec \delta X_i[j][k]$

ΔZ: $\forall i \in [0; r-2], j \in [0;3], k \in [0;3], \Delta Z_i[j][k] \prec \delta Z_i[j][k]$

obj_{Step1}: $obj_{Step1} = \sum_{i \in SBoxes_l} (i)$

The DK, DY, DZ sets, used to define the difference variables, are defined as follows, for all $j \in [0;3]$:

$$\begin{array}{rcl} DK_j & = & \{\delta K_i[j][k] : i \in [0, r-2], k \in [0,3]\} \\ DY_j & = & \{\delta Y_i[j][k] : i \in [0, r-2], k \in [0,3]\} \\ DZ_j & = & \{\delta Z_i[j][k] : i \in [0, r-2], k \in [0,3]\}. \end{array}$$

Given these sets, we define the *diff* variables as follows: For each set $D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}$, and for each pair of differential bytes $\{\delta B_1, \delta B_2\} \in D$, we define a Boolean variable $diff_{\delta B_1, \delta B_2}$, which is equal to 1 if δB1 and δB2 are equal, and to 0 otherwise.

8.3.2 Constraints

In the model for Step 1, we have one constraint for each operation, and an additional constraint for the quasi MDS property. These constraints are defined as follows.

SubCell As explained in Section 6.4.2, the SBoxes do not introduce differences during Step 1, so the constraint is defined as

$$\forall i \in [0, r-2], \forall (j,k) \in [0;3]^2 : \Delta SB_i[j][k] = \Delta X_i[j][k]. \qquad (C^M_{SB})$$

XOR constraint As explained in Section 6.4.2, the XOR operation cannot be modelled precisely, since the information about whether two differential words are equal is abstracted. Hence, the XOR constraint abstracting $\delta B_3 = \delta B_1 \oplus \delta B_2$ is

$$XOR(\Delta B_1, \Delta B_2, \Delta B_3) \Leftrightarrow \Delta B_1 + \Delta B_2 + \Delta B_3 \neq 1 \qquad (C^M_{XOR})$$

We also use a XOR constraint for the XOR of 3 variables into a fourth one δB_4:

$$XOR(\Delta B_1, \Delta B_2, \Delta B_3, \Delta B_4) \Leftrightarrow \Delta B_1 + \Delta B_2 + \Delta B_3 + \Delta B_4 \neq 1 \qquad (C^M_{XOR4})$$

KeyAdd KA is modelled by XOR constraints:

$$\forall i \in [1, r-2], \forall (j,k) \in [0,3]^2 : XOR(\Delta Y_{i-1}[j][k], \Delta K_i[j][k], \Delta X_i[j][k]). \qquad (C^M_{KA})$$

ShuffleCells SC is modelled by equality constraints that link shifted words:

$$\left.\begin{array}{l}
\forall i \in [0; r-2]: \\
\Delta\mathrm{Y}[i][0][0] = \Delta\mathrm{SB}[i][0][0], \\
\Delta\mathrm{Y}[i][1][0] = \Delta\mathrm{SB}[i][2][2], \\
\Delta\mathrm{Y}[i][2][0] = \Delta\mathrm{SB}[i][1][1], \\
\Delta\mathrm{Y}[i][3][0] = \Delta\mathrm{SB}[i][3][3], \\
\Delta\mathrm{Y}[i][0][1] = \Delta\mathrm{SB}[i][2][3], \\
\Delta\mathrm{Y}[i][1][1] = \Delta\mathrm{SB}[i][0][1], \\
\Delta\mathrm{Y}[i][2][1] = \Delta\mathrm{SB}[i][3][2], \\
\Delta\mathrm{Y}[i][3][1] = \Delta\mathrm{SB}[i][1][0], \\
\Delta\mathrm{Y}[i][0][2] = \Delta\mathrm{SB}[i][1][2], \\
\Delta\mathrm{Y}[i][1][2] = \Delta\mathrm{SB}[i][3][0], \\
\Delta\mathrm{Y}[i][2][2] = \Delta\mathrm{SB}[i][0][3], \\
\Delta\mathrm{Y}[i][3][2] = \Delta\mathrm{SB}[i][2][1], \\
\Delta\mathrm{Y}[i][0][3] = \Delta\mathrm{SB}[i][3][1], \\
\Delta\mathrm{Y}[i][1][3] = \Delta\mathrm{SB}[i][1][3], \\
\Delta\mathrm{Y}[i][2][3] = \Delta\mathrm{SB}[i][2][0], \\
\Delta\mathrm{Y}[i][3][3] = \Delta\mathrm{SB}[i][0][2].
\end{array}\right\} (\mathrm{C}^{\mathrm{M}}_{\mathrm{SC}})$$

MixColumns MC cannot be modelled precisely at the Boolean level, as knowing where differences hold in Y_i is not enough to determine where they hold in Z_i. Indeed, the values of the words in the column are necessary to determine the positions of the zeros after MC. However, the quasi MDS property is modelled by constraining the number of differences in a same column of Y_i and Z_i to be equal to 0 or greater than 3:

$$\forall i \in [0, r-2], \forall k \in [0,3]: \left(\sum_{j=0}^{3} \Delta\mathrm{Y}_i[j][k] + \Delta\mathrm{Z}_i[j][k]\right) \in \{0,5,6,7,8\}. \qquad (\mathrm{C}^{\mathrm{M}}_{\mathrm{MC}})$$

Then, we model the fact that MC(0,0,0,0) = (0,0,0,0) as follows:

$$\forall i \in [0, r-2], \forall k \in [0,3]: \left(\sum_{j=0}^{3} \Delta\mathrm{Y}[i][j][k] = 0\right) \Leftrightarrow \left(\sum_{i=0}^{3} \Delta\mathrm{Z}[i][j][k] = 0\right). \qquad (\mathrm{C}^{\mathrm{M}}_{\mathrm{MC0}})$$

Finally, we can model the XORs implied in the computation of MC, since all the coefficients are 1. We implement the product of the vector ΔSB with the matrix given in Midori to get ΔZ. It is modelled as follows:

$$\left.\begin{array}{l}
\forall i \in [0; r-2], k \in [0; r-2]: \\
\mathrm{XOR}(\Delta\mathrm{Y}[i][1][k] \wedge \Delta\mathrm{Y}[i][2][k] \wedge \Delta\mathrm{Y}[i][3][k] \wedge \Delta\mathrm{Z}[i][0][k]) \\
\mathrm{XOR}(\Delta\mathrm{Y}[i][0][k] \wedge \Delta\mathrm{Y}[i][2][k] \wedge \Delta\mathrm{Y}[i][3][k] \wedge \Delta\mathrm{Z}[i][1][k]) \\
\mathrm{XOR}(\Delta\mathrm{Y}[i][0][k] \wedge \Delta\mathrm{Y}[i][1][k] \wedge \Delta\mathrm{Y}[i][3][k] \wedge \Delta\mathrm{Z}[i][2][k]) \\
\mathrm{XOR}(\Delta\mathrm{Y}[i][0][k] \wedge \Delta\mathrm{Y}[i][1][k] \wedge \Delta\mathrm{Y}[i][2][k] \wedge \Delta\mathrm{Z}[i][3][k]).
\end{array}\right\} (\mathrm{C}^{\mathrm{M}}_{\mathrm{MCXOR}})$$

KeySchedule KS depends on the version of Midori that we consider. For Midori64, we have

$$\forall i \in [0; r-1]: \Delta\mathrm{K}_i = \Delta\mathrm{K}^{i\%2}, \qquad (\mathrm{C}^{\mathrm{M}}_{\mathrm{KS}})$$

and for Midori128,

$$\forall i \in [0; r-1]: \Delta\mathrm{K}_i = \Delta\mathrm{K}. \qquad (\mathrm{C}^{\mathrm{M}}_{\mathrm{KS}})$$

Constraint relating DK **and the** $diff$ **variables** For Midori64, two differential words of the key $\delta K_{i1}[j][k]$ and $\delta K_{i2}[j][k]$, for two different rounds i_1 and i_2, are equal by definition of the key schedule if $i1\%2 = i2\%2$:

$$\forall (i_1, i_2) \in [0; r-1]^2, \forall (j,k) \in [0;3]^2, i_1\%2 = i_2\%2 : diff_{\delta K_{i_1}[j][k], \delta K_{i_2}[j][k]} = 0. \qquad (C^{M}_{DIFFK})$$

For Midori128, by definition, the differential matrices abstracting the round subkeys are equal:

$$\forall (i1, i2) \in [0; r-1]^2, \forall (j,k) \in [0;3]^2 : diff_{\delta K_{i1}[j][k], \delta K_{i2}[j][k]} = 0. \qquad (C^{M}_{DIFFK})$$

Symmetry To be consistent, our difference variables need to have symmetry:

$$\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \in D diff_{\delta B_1, \delta B_2} = diff_{\delta B_2, \delta B_1}. \qquad (C^{M}_{SYM})$$

Transitivity To be consistent, our difference variables need to have transitivity:

$$\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2, \delta B_3\} \in D,$$
$$diff_{\delta B_1, \delta B_2} + diff_{\delta B_2, \delta B_3} + diff_{\delta B_1, \delta B_3} \neq 1. \qquad (C^{M}_{TRANS})$$

Relating $diff$ **variables and** $\varDelta$ **variables** If $\varDelta A \neq \varDelta B$, then $diff_{\delta A, \delta B} = 1$, and that if $\varDelta A \neq \varDelta B = 0$, then $diff_{\delta A, \delta B} = 0$. This is enforced by the following constraint:

$$\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \in D, diff_{\delta B_1, \delta B_2} + \varDelta B_1 + \varDelta B_2 \neq 1. \qquad (C^{M}_{DIFF\varDelta})$$

Constraints relating KA **and the** $diff$ **variables** ARK defines $X_{i+1}[j][k]$ as the result of a xor between $K_i[j][k]$ and $Z_i[j][k]$, we post the constraint: $\forall i_1, i_2 \in [0, r-2], \forall j, k1, k2 \in [0,3]^3$,

$$diff_{\delta K_{i_1}[j][k_1], \delta K_{i_2}[j][k_2]} + diff_{\delta Z_{i_1}[j][k_1], \delta Z_{i_2}[j][k_2]} + \varDelta X_{i_1}[j][k_1] + \varDelta X_{i_2}[j][k_2] \neq 1. \qquad (C^{M}_{DIFFKA})$$

Quasi MDS property As in the $S1_{diff}$ model for AES, we exploit the quasi MDS property for the XOR of two differential columns by

$$\forall i1, i2 \in [0, r-2]^2, \forall k1, k2 \in [0;3]^2 :$$
$$(\sum_{j=0}^{3} (diff_{\delta Y_{i1}[j][k1], \delta Y_{i2}[j][k2]}) \neq 0) + (diff_{\delta Y_{i1}[j][k1], \delta Y_{i2}[j][k2]} \neq 0)) \in \{0,5,6,7,8\}. \qquad (C^{M}_{DIFFMDS})$$

8.4 Step 2

Given an abstracted differential computed by *Step1-enum*, Step 2 aims at searching for the word values with the highest differential probability (or proving that the abstracted differential is not word-consistent). In this section, we describe the CP model for Step 2 on Midori.

8.4.1 Variables

For each differential word $\delta B \in diffWords_l$, we define an integer variable the domain of which depends on the value of $\varDelta B$ in the abstracted differential: If $\varDelta B = 0$, then $D(\delta B) = \{0\}$; otherwise, $D(\delta B) = [1, 15]$ for Midori64, or $D(\delta B) = [1, 255]$ for Midori128.

As we look for a word-consistent solution with maximal probability, we declare an integer variable $P_{\delta B}$ for each differential word $\delta B \in Sboxes$: This variable corresponds to the base 2 logarithm of the probability $Pr(\delta B \rightarrow \delta SB)$ of obtaining the Sbox output difference δSB when the SBox input difference is δB. This probability is obtained by applying equation 6.1. The domain of $P_{\delta B}$ depends on the value of $\varDelta B$ in the abstracted differential: If $\varDelta B = 0$, then $Pr(0 \rightarrow 0) = 1$ and therefore $D(P_{\delta B}) = \{0\}$; otherwise, $Pr(\delta B \rightarrow \delta SB) \in \{2^{-3}, 2^{-2}\}$ and $D(P_{\delta B}) = \{-3, -2\}$ for Midori64, and $Pr(\delta B \rightarrow \delta SB) \in \{2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}\}$ and $D(P_{\delta B}) = \{-6, -5, -4, -3, -2\}$ for Midori128. The constraint associated with the SubCell operation forbids couples $(\delta B, \delta SB)$ such that $Pr(\delta B \rightarrow \delta SB) = 0$.

8.4.2 Constraints

The constraints basically follow the Midori operations to relate variables, as described in Section 8.3 for Step 1, but consider the definition of the operations at the word level, instead of the Boolean level.

The main difference is that the SubCell operation, which has no effect at the Boolean level, must be modelled at the word level. This is done thanks to a ternary table constraint which extensively lists all triples (X, Y, P) such that there exists two words B_1 and B_2 whose difference before and after passing through S-Boxes is equal to X and Y, respectively, and such that P is the probability of this transformation: For all $\delta B \in Sboxes$, we add the constraint

$$(\delta B, \delta SB, P_{\delta B}) \in \{(X,Y,P) | \exists (B_1, B_2) \in [0,15] \times [0,15], X = B_1 \oplus B_2,\\ Y = S(B_1) \oplus S(B_2), P = \log_2(Pr(X \rightarrow Y))\}. \qquad (C^M_{SBS2})$$

for Midori64, and

$$(\delta B, \delta SB, P_{\delta B}) \in \{(X,Y,P) | \exists (B_1, B_2) \in [0,255] \times [0,255], X = B_1 \oplus B_2,\\ Y = S(B_1) \oplus S(B_2), P = \log_2(Pr(X \rightarrow Y))\}. \qquad (C^M_{SBS2})$$

for Midori128.

We also use a table constraint for the XOR of two words, as defined in Section 6.4.2, with tuples tupleXOR.

We define:

$$XORWord(\delta a, \delta b, \delta c) \equiv (\delta a, \delta b, \delta c) \in \mathsf{tupleXOR}. \qquad (C^M_{XORWords})$$

In what follows, we assume that XOR*Words* can take an arbitrary number of arguments (the last one being the result), event though in the implementation, when a XOR of multiple variables is needed, we XOR them two by two and use temporary variables to store the intermediary results.

The MixColumn operation also needs to be modelled at word level. MixColumn is the multiplication of each column of the state by a fixed matrix M, given in section 8.2. In Rijndael's Galois field, addition is replaced by XOR. For instance, let us consider a column $C = (\delta a, \delta b, \delta c, \delta d)^t$. Then

$$MC(C) = \begin{pmatrix} \delta b \oplus \delta c \oplus \delta d \\ \delta a \delta\delta c \oplus \delta d \\ \delta a \oplus \delta b \oplus \delta d \\ \delta a \oplus \delta b \oplus \delta c \end{pmatrix}$$

We implemented both MC and its inverse. Indeed, it permits the solver to obtain the k^{th} column of δY_i directly from the k^{th} column of δZ_i, instead of having to enumerate all possible values and applying MC in the forward direction to verify whether the affectation is correct.

The MC operation is modelled as follows, using temporary variables t_z:

$$\left.\begin{array}{l} \forall i \in [0; r-2], \forall k \in [0;3]: \\ XORWord(\delta Y_i[1][k], \delta Y_i[2][k], \delta Y_i[3][k], \delta Z_i[0][k]), \\ XORWord(\delta Y_i[0][k], \delta Y_i[2][k], \delta Y_i[3][k], \delta Z_i[1][k]), \\ XORWord(\delta Y_i[0][k], \delta Y_i[1][k], \delta Y_i[3][k], \delta Z_i[2][k]), \\ XORWord(\delta Y_i[0][k], \delta Y_i[1][k], \delta Y_i[2][k], \delta Z_i[3][k]). \end{array}\right\} (C^M_{MCS2})$$

The inverse of the MC operation is modelled similarly.

$$\left.\begin{array}{l} \forall i \in [0; r-2], \forall k \in [0;3] \\ XORWord(\delta Z_i[1][k], \delta Z_i[2][k], \delta Z_i[3][k], \delta Y_i[0][k]) \\ XORWord(\delta Z_i[0][k], \delta Z_i[2][k], \delta Z_i[3][k], \delta Y_i[1][k]) \\ XORWord(\delta Z_i[0][k], \delta Z_i[1][k], \delta Z_i[3][k], \delta Y_i[2][k]) \\ XORWord(\delta Z_i[0][k], \delta Z_i[1][k], \delta Z_i[2][k], \delta Y_i[3][k]) \end{array}\right\} (C^M_{INVMCS2})$$

The constraint for SC is described in the same way as for Step 1:

$$\left.\begin{aligned}
&\forall i \in [0; r-2]:\\
&\delta Y[i][0][0] = \delta SB[i][0][0],\\
&\delta Y[i][1][0] = \delta SB[i][2][2],\\
&\delta Y[i][2][0] = \delta SB[i][1][1],\\
&\delta Y[i][3][0] = \delta SB[i][3][3],\\
&\delta Y[i][0][1] = \delta SB[i][2][3],\\
&\delta Y[i][1][1] = \delta SB[i][0][1],\\
&\delta Y[i][2][1] = \delta SB[i][3][2],\\
&\delta Y[i][3][1] = \delta SB[i][1][0],\\
&\delta Y[i][0][2] = \delta SB[i][1][2],\\
&\delta Y[i][1][2] = \delta SB[i][3][0],\\
&\delta Y[i][2][2] = \delta SB[i][0][3],\\
&\delta Y[i][3][2] = \delta SB[i][2][1],\\
&\delta Y[i][0][3] = \delta SB[i][3][1],\\
&\delta Y[i][1][3] = \delta SB[i][1][3],\\
&\delta Y[i][2][3] = \delta SB[i][2][0],\\
&\delta Y[i][3][3] = \delta SB[i][0][2].
\end{aligned}\right\} (C^{M}_{SCS2})$$

The key addition KA uses XORW$ords$:

$$\left.\begin{aligned}
&\forall (j,k) \in [0,3]^2 : XOR(\delta X[j][k], \delta WK[j][k], \delta X_0[j][k]),\\
&\forall i \in [1, r-1], \forall (j,k) \in [0,3]^2 : XOR(\delta Y_{i-1}[j][k], \delta K_i[j][k], \delta X_i[j][k]).
\end{aligned}\right\} (C^{M}_{KAS2})$$

Finally, KS is defined in a very straightforward way: for Midori64, it is

$$\forall i \in [0; r-1], \forall (j,k) \in [0,3]^2 : \delta K_i[j][k] = \delta IK^{i\%2}[j][k], \qquad (C^{M}_{KS})$$

and for Midori128,

$$\forall i \in [0; r-1], \forall (j,k) \in [0,3]^2 : \delta K_i[j][k] = \delta IK[j][k]. \qquad (C^{M}_{KS})$$

Objective function The goal is to find a byte-consistent solution with maximal differential probability. As we consider logarithms, this amount to searching for a solution that maximises the sum of all $P_{\delta B}$ variables. Hence, we introduce an integer variable obj_{Step2} which is constrained to be equal to the sum of all $P_{\delta B}$ variables:

$$obj_{Step2} = \sum_{\delta B \in Sboxes_l} P_{\delta B}$$

and we define the objective function as the maximisation of obj_{Step2}.

8.5 Results

In this section, we list the results obtained with our models. All our experiments are performed according to the setting described in Section 6.4.2, on a single core of an Intel(R) Xeon(R) E5-2687Wv4 @ 3.00GHz CPU. We consider 30 instances denoted Midori64-r (resp Midori128-r), for a number of rounds r: $r \in [3;16]$ (resp. $[3;20]$) for Midori64 (resp. Midori128).

$S1_{XOR}$ and $S1_{Diff}$ are implemented with MiniZinc [Nethercote et al., 2007].

We report the experimental results obtained with Picat_SAT.

	Step1-opt		Step1-enum		Step 2			
	v^*	t_{opt}	#T	t_{enum}	#B	$\frac{t2}{\#T}$	t_{Step2}	obj_{Step2}
Midori64-3	1	< 1	48	14	48	< 1	9	2^{-2}
Midori64-4	2	< 1	32	31	32	< 1	3	2^{-4}
Midori64-5	2	1	16	15	16	< 1	2	2^{-4}
Midori64-6	3	2	32	109	32	< 1	4	2^{-6}
Midori64-7	3	4	16	62	16	< 1	3	2^{-6}
Midori64-8	4	7	32	87	32	< 1	5	2^{-8}
Midori64-9	4	6	16	40	16	< 1	4	2^{-8}
Midori64-10	5	9	32	141	32	< 1	6	2^{-10}
Midori64-11	5	10	16	51	16	< 1	4	2^{-10}
Midori64-12	6	23	32	126	32	< 1	6	2^{-12}
Midori64-13	6	39	16	70	16	< 1	4	2^{-12}
Midori64-14	7	29	32	194	32	< 1	7	2^{-14}
Midori64-15	7	28	16	92	16	< 1	5	2^{-14}
Midori64-16	8	45	32	236	32	< 1	7	2^{-16}

Table 8.2: Our results for Midori 64. For each instance, we give the optimal number of SBoxes v^* and the time t_{opt} needed to solve the Step1-opt. We then give the results of the enumeration problem: the number of solutions #T and the corresponding search time t_{enum}. Finally, we present the results of Step 2: the number of byte consistent solutions #B, the average solving time $\frac{t2}{\#T}$, the total time t_{S2}, and the probability of the best abstracted related key differential characteristic obj_{Step2}. All times are given in seconds.

The results are given in Table 8.2 for Midori64, and in Table 8.4 for Midori128. For all instances of Midori64, the resolution, Step1-opt is performed in at most 12 seconds, Step1-enum in less than 4 minutes, and Step 2 always takes less than 1 second per solution. The instance with the longest total time, including Step1-opt, Step1-enum and Step2, is Midori64-14, and it is solved in less than 5 minutes. For each instance, we find solutions with $\lceil \frac{R}{2} \rceil$ SBoxes, and there is always an assignment to the differential nibbles that reaches the optimal probability $2^{-2 \cdot v^*}$ (where v^* is the number of active SBoxes). Hence, we obtain 32 full-round related key differential characteristics for Midori64 with 8 active SBoxes, and probability 2^{-16}. One of these characteristics is given in Table 8.3.

For Midori128, Step1-opt is performed in at most 64 seconds, Step1-enum in at most 424 seconds, and Step 2 always takes less than 3 seconds per solution. The instance with the longest total time, including Step1-opt, Step1-enum and Step2, is Midori128-20, and it is solved in less than 10 minutes. For each instance, we find solutions with R SBoxes, and there is always an assignment to the differential bytes that reaches the optimal probability $2^{-2 \cdot v^*}$. Hence, we obtain 18 full-round related key differential characteristics for Midori128 with 20 active SBoxes, and probability 2^{-40}. One of these characteristics is given in Table 8.5.

δK	0 0 0 0 0 0 0 0 2 0 2 2 0 0 0 0

δX	Value	Pr
init	0 0 0 0 0 0 0 0 2 0 2 2 0 0 0 0	
δX_0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
δX_1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
δX_2	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
δX_3	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
δX_4	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
δX_5	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
δX_6	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
δX_7	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
δX_8	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
δX_9	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
$\delta X_1 0$	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
$\delta X_1 1$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
$\delta X_1 2$	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
$\delta X_1 3$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
$\delta X_1 4$	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	$2^{-2 \cdot 1}$
$\delta X_1 5$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
$\delta X_1 6$	0 0 0 1 0 0 0 0 2 0 2 2 0 0 0 0	

Table 8.3: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}. The nibbles are given in hexadecimal notation.

	Step1-opt		Step1-enum		Step 2			
	v^*	t_{opt}	#T	t_{enum}	#B	$\frac{t2}{\#T}$	t_{Step2}	obj_{Step2}
Midori128-3	3	<1	42	21	30	< 1	16	2^{-6}
Midori128-4	4	1	18	15	18	< 1	15	2^{-8}
Midori128-5	5	3	18	21	18	< 1	17	2^{-10}
Midori128-6	6	9	18	40	18	< 2	23	2^{-12}
Midori128-7	7	9	18	46	18	< 2	21	2^{-14}
Midori128-8	8	10	18	54	18	< 2	28	2^{-16}
Midori128-9	9	15	18	87	18	< 2	24	2^{-18}
Midori128-10	10	28	18	108	18	< 3	41	2^{-20}
Midori128-11	11	21	18	127	18	< 3	37	2^{-22}
Midori128-12	12	38	18	149	18	< 2	35	2^{-24}
Midori128-13	13	75	18	174	18	< 2	34	2^{-26}
Midori128-14	14	76	18	260	18	< 3	49	2^{-28}
Midori128-15	15	82	18	146	18	< 3	47	2^{-30}
Midori128-16	16	193	18	400	18	< 3	47	2^{-32}
Midori128-17	17	177	18	413	18	< 3	47	2^{-34}
Midori128-18	18	367	18	517	18	< 4	69	2^{-36}
Midori128-19	19	194	18	557	18	< 4	57	2^{-38}
Midori128-20	20	239	18	674	18	< 4	55	2^{-40}

Table 8.4: Our results for Midori128. For each instance, we give the optimal number of SBoxes v^* and the time needed to solve the Step1-opt t_{opt}. We then give the results of the enumeration problem: the number of solutions #T and the corresponding search time t_{enum}. Finally, we present the results of Step 2: the number of byte consistent solutions #B, the average solving time $\frac{t2}{\#T}$,the total time t_{S2}, and the probability of the best abstracted related key differential characteristic obj_{Step2}. All times are given in seconds.

δK	00 02 00 00 01 01 01 00 00 00 00 00 00 00 00 00

δX	Value	Pr
init	00 00 00 00 01 01 01 00 00 00 00 00 00 00 00 00	
δX_0	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_1	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_2	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_3	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_4	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_5	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_6	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_7	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_8	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
δX_9	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 0$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 1$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 2$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 3$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 4$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 5$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 6$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 7$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 8$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_1 9$	00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$2^{-2\cdot 1}$
$\delta X_2 0$	00 03 00 00 01 01 01 00 00 00 00 00 00 00 00 00	

Table 8.5: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}. The bytes are given in hexadecimal notation.

8.6 Related Key Differential Attacks against Midori

In this section, we show how to use the related-key differential characteristics found with our models to perform a key recovery attack against both versions of Midori. We first present the method for mounting a related-key differential attack, and then apply it to Midori64 and Midori128.

8.6.1 Related-Key Differential Attacks

We remind that in a related key differential attack, we can use an oracle $\mathsf{E.enc}_{\mathrm{RK},(\delta\mathrm{IK},\mathrm{IK})}(\mathrm{X})$ that encrypts a message X with the key $\mathrm{IK} \oplus \delta\mathrm{IK}$. In order to mount a related-key differential attack, we use the related-key differential characteristic to derive the most likely input difference to the last SB operations, *i.e.*, $\delta\mathrm{X}_{r-1}$. We then obtain $\mathrm{T} \cdot p(c)^{-1}$ pairs of ciphertexts $\mathrm{C} = \mathsf{E.enc}_{\mathrm{RK},(0,\mathrm{K})}(\mathrm{X})$ and $\mathrm{C}' = \mathsf{E.enc}_{\mathrm{RK},(\delta\mathrm{IK},\mathrm{IK})}(\mathrm{X} \oplus \delta\mathrm{X})$, where $p(c)$ is the probability of the differential characteristic. The computation of T is detailed in each of our attacks. We compute $\delta\mathrm{C} = \mathrm{C} \oplus \mathrm{C}'$, and then $\delta\mathrm{SB}_{r-1} = \delta\mathrm{C} \oplus \delta\mathrm{WK}$, where $\delta\mathrm{WK}$ is known from the value of $\delta\mathrm{IK}$ in the differential characteristics. We then make the assumption that $\delta\mathrm{X}_{r-1}$ takes the value corresponding to the differential characteristic. If this actually occurs, the pair $(\mathrm{C}, \mathrm{C}')$ is called a *good pair*. When we have a good pair, we know the input differences to the SBoxes ($\delta\mathrm{X}_{r-1}$), and the output difference ($\delta\mathrm{SB}_{r-1}$). In a bijective SBox, there only a given number t of values which satisfy the relation $a \oplus a' = \delta in, \mathrm{S}(a) \oplus \mathrm{S}(a') = \delta out$. Hence, we can derive t candidate values for each word of SB_{r-1} for which $\delta\mathrm{SB}_{r-1}[j][k] \neq 0$. However, when $\delta\mathrm{SB}_{r-1}[j][k] = 0$, we cannot derive a candidate, since $\delta\mathrm{SB}_{r-1}[j][k] = 0$ only indicates that $\mathrm{SB}_{r-1}[j][k]$ is equal to $\mathrm{SB}'_{r-1}[j][k]$. Let $\mathrm{CSB}_i[j][k]$ be the set of candidates for $\mathrm{SB}_i[j][k]$. We can derive a set of candidates for $\mathrm{WK}[j][k]$ by combining the known value of $\mathrm{C}[j][k]$ (given by the oracle), and $\mathrm{CSB}_{r-1}[j][k]$: $\mathrm{CWK}[j][k] = \{x : x = \mathrm{C}[j,k] \oplus z, z \in \mathrm{CSB}_{r-1}[j][k]\}$. After repeating these operations for each of the $\mathrm{T} \cdot p(c)^{-1}$ ciphertexts, we keep the candidate for $\mathrm{WK}[j][k]$ that occurred the most often.

8.6.2 Midori64

Our key recovery attack against Midori64 works in two steps. We first use a set of 16 related-key differential characteristics to recover WK, one word at a time, in $16 \cdot 2^{20} = 2^{24}$ operations. Then, we use another set of 4 related key differential characteristics to obtain $\mathrm{K}_{14} = \mathrm{IK}^0$, in 2^{36} operations. By combining WK and K_{14}, we obtain $\mathrm{IK}^1 = \mathrm{IK}^0 \oplus \mathrm{WK}$ and deduce IK (composed of IK^0 and IK^1), for a total complexity of $2^{24} + 2^{36} \approx 2^{36}$.

Recovery of WK

Among the 32 16-round related-key differential characteristics obtained with our solvers, 16 have exactly one active word in $\delta\mathrm{X}_{15}$. These characteristics are given in Appendix C.1.1.For each of these characteristics, the active word is at a different position. Let this position be denoted $\delta\mathrm{X}_{15}[j][k]$. With each of these characteristics, we recover one word of WK, $\mathrm{WK}[j][k]$, by directly applying the method presented in Section 8.6.1. Note that we do not derive key candidates for all ciphertext pairs, as we only keep the ones for which $\delta\mathrm{SB}_{15}$ matches what is expected from the differential characteristic, except for position $\delta\mathrm{SB}_{15}[j][k]$, where any value is accepted. In Midori128, IK = WK, so that after recovering WK, we directly obtain IK.

Complexity Analysis We now determine how many ciphertext pairs we need to recover each word of WK, by computing the value T mentioned in Section 8.6.1. To determine T, we follow the approach given in [Selçuk, 2008]. It uses the signal to noise ratio S/N introduced by Biham in [Biham and Shamir, 1993]. The signal to noise ration is defined as $\mathrm{S/N} = \frac{2^k \cdot p}{\alpha \cdot \beta}$, where k is the number of key bits that we want to recover, p is the probability of the related-key differential characteristic, α is the number of key candidates suggested for each good pair, and β is the ratio of the pairs that are not discarded.

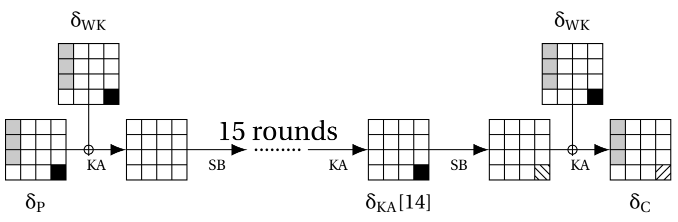

Figure 8.5: An example of a related-key differential characteristic provided by the solver.

In our attack, k is 4, since we want to recover a 4-bit nibble. We take $p = 2^{-14}$: the probability of the whole differential characteristic, including the final SB operation, is 2^{-16}, but the probability to have the correct difference in δX_{15} is 2^{-14}. The α parameter is obtained through the properties of the SBox: either two of 4 values can correspond to a given input/output difference, *i.e.*, for a given input difference δin and output difference δout,

$$\#\{a : S(a) \oplus S(a \oplus \delta in) = \delta out\} \in \{2, 4\}.$$

We want to upper bound the number of ciphertext pairs that we need, so that we take $\alpha = 4$, which minimises the signal-to-noise. For β we have $2^{-14} + 2^{-60}$, where 2^{-14} is the probability given by the solvers and 2^{-60} corresponds to the false positives, *i.e.*, pairs having the same difference pattern in δSB_{15}, with 4 bits of undetermined difference at position $\delta SB_{15}[j][k]$. Using these values, we obtain $S/N = \frac{2^k \cdot p}{\alpha \cdot \beta} = \frac{2^4 \cdot 2^{-14}}{4 \cdot (2^{-14} + 2^{-60})} = \frac{2^{-10}}{2^{-12} + 2^{-58}} \approx 4$. We denote by p_S the probability to obtain the true key. We use the equation (19) of [Selçuk, 2008], where Φ denote the density probability function of the standard normal distribution, and Φ^{-1} its inverse: $p_S = \Phi\left(\frac{\sqrt{T \cdot S/N} - \Phi^{-1}(1-2^{-k})}{\sqrt{S/N+1}}\right)$ (19). Then we can obtain p_S for given values of T, S/N and α. We want to compute which T gives us a high enough probability to recover a key word. Moreover, since we repeat the analysis 16 times (one for each word of WK), we need to have p_S^{16} sufficiently large as well. By numerical approximation we obtain $T = 20 \approx 2^{4.32}$, which gives $p_S > 0.99$, and $p_S^{16} > 0.99$. Hence, using $T \cdot p^{-1}$ plaintext pairs, we recover a key word with a probability greater than 0.99. The corresponding data complexity is then upper bounded by $16 \cdot 2 \cdot 20 \cdot 2^{14} \approx 2^{24}$ chosen plaintexts, as well as 16 related keys, one for each related-key differential characteristic used.

Recovery of K^0

For this step, we use a different set of differential characteristics, which have one active SBox in δSB_{14}, denoted $\delta SB_{14}[j][k]$. More specifically, we use 4 characteristics, and for each of them, the active SBox is moved to a different column after SC. Each of these characteristics is used to recover one different column of $K_{14} = K^0 \oplus \alpha_{14}$. These characteristics are given in Appendix C.1.2. We first cipher pairs $X, X \oplus \delta X$ with the keys $IK, IK \oplus \delta IK$ through the oracle. Using WK, previously computed, we partially decrypt C and C′. We obtain $SB_{15} = C \oplus WK$ and $SB'_{15} = C' \oplus WK \oplus \delta WK$, and X_{15} and X'_{15} by inverting the SB operation. From there we compute $\delta X_{15} = X_{15} \oplus X'_{15}$, and $\delta Z_{14} = \delta X_{15} \oplus \delta K^0$, where δK^0 is given by the differential characteristic. We then invert MC and SC to obtain δSB_{14}. Knowing δX_{14} (assuming the characteristic held), and δSB_{14}, we can compute candidates for the active differential word $\delta SB_{14}[j][k]$. Since we want to recover a whole column k of K_{14}, we want the candidates for the column k of δY, after the SC operation. Hence, we exhaustively try the possible combinations of the 3 other nibbles that are in the same column after SC for each candidate. This results in a number of operations of $2^4 \cdot 2^4 \cdot 2^4 \cdot x$, where x is the number of candidates for $SB_{14}[j][k]$. x is upper bounded by 4, so we have at most $2^4 \cdot 2^4 \cdot 2^4 \cdot 4 = 2^{14}$ combinations to try. Each of these 2^{14} combinations gives a candidate column of Z_{14}. By XORing these candidates to the corresponding column of X_{15}, which we obtained from the partial decryption, we obtain candidates for 4-nibble columns of K_{14}. Note that we do not derive key candidates for all ciphertext pairs, as we only keep the ones for which δSB_{14} matches what is expected from the differential characteristic, except for position $\delta SB_{14}[j][k]$, where any value is accepted. Once K_{14} is recovered, we obtain $IK^0 = K_{14} \oplus \alpha_{14}$.

Complexity Analysis This time we try to recover 16 bits (4 nibbles) of the key, so $k = 16$. The probability of having the correct δX_{14} is $p = 2^{-14}$, as given by the differential characteristic. The value of α is the number of candidate columns we obtain, which is 2^{14}, as detailed in the previous section. Finally, $\beta = 2^{-14} + 2^{-60}$, since we are interested in values of δSB_{14} with one arbitrary nibble difference, and zero differences in the other nibbles. Using these values, we obtain $S/N = \frac{2^k \cdot p}{\alpha \cdot \beta} = \frac{2^{16} \cdot 2^{-14}}{2^{14} \cdot (2^{-14} + 2^{-60})} = \frac{2^{-12}}{2^{-14} + 2^{-58}} \approx 4$. Then by numerical approximation we find $T = 28 \approx 2^5$, which gives $p_S > 0.99$, and $p_S^4 > 0.99$. Hence, the total number of operations is upper bounded by $4 \cdot 2 \cdot 2^5 \cdot 2^{14} \cdot 2^{14} = 2^{36}$ and the number of encryptions is upper bounded by $2 \cdot 4 \cdot 2^5 \cdot 2^{14} = 2^{22}$, and we need a total of 4 related keys.

8.6.3 Midori128

The solver finds 18 full-round differential characteristics, each of which have a different active word in δX_{15}. The probability to obtain the correct difference in δX_{15} is 2^{-38} for each characteristic. The corresponding related-key differentials are given in Appendix C.2. Hence, we directly apply the attack presented in Section 8.6.1, to recover one word of $WK = IK$, at position j, k, with each of the differential characteristics. We only keep the ciphertext pairs for which $\delta SB_{19}[j][k]$ is all zeros, except in position j, k.

Complexity evaluation: We need to use 16 related-key differential characteristics, so we want $p_S^{16} \geq 0.99$. To compute S/N, we use $k = 8$ as we recover a 8-bit word of the key for each related-key differential characteristic, and $p = 2^{-38}$. For α, we once again pick the value that minimises the signal to noise ratio, to obtain an upper bound on the number of operations, so we have $\alpha = 64$. Finally, $\beta = 2^{-38} + 2^{-120}$. With these values, we have $S/N = 4$, and need $T = 25 \approx 2^5$ to have $p_S^{16} > 0.99$. Thus, the data complexity of the attack is upper bounded by $2 \cdot 2^5 \cdot 2^{38} = 2^{44}$ encryptions, under 16 related keys, and the time complexity is 2^{44} as well.

8.7 Conclusion

In this chapter, we present our CP models to perform the search for optimal related-key differential characteristics on Midori. These models are similar to $S1_{Diff}$ (Section 7.3.2), as they include *diff* variables and the $C^M_{DIFFMDS}$ constraint, that propagates the quasi-MDS property of MC between two columns of differential words. The longest instance to solve, Midori-128-20, is solved in less than 17 minutes (239 seconds for *Step1-opt*, 674 for *Step1-enum*, and 55 for Step 2). For Midori64 (resp. 128), we find full-round related-key differential characteristics with probability 2^{-16} (resp. 2^{-40}). Using these differential characteristics, we mount key recovery attacks. We show how to recover the key in 2^{36} operations, and 2^{24} plaintexts encrypted with 20 related keys for Midori64, and 2^{44} operations and 2^{44} plaintexts encrypted with 16 related keys for Midori128.

The search for optimal related-key differential characteristics is significantly faster for Midori than for the AES. The instance that takes the most time to solve for Midori is solved within 17 minutes. The most difficult instance for AES is solved in almost 4 hours, using more advanced techniques (in particular, the different decomposition described in Section 7.5). In addition, the hardest instance of AES is on 10 rounds, whereas the hardest instance for Midori is 20 rounds. The fact that the search is faster for Midori than for AES, despite a larger number of rounds, can partly be explained by the simplicity of the MixColumns operation: all coefficients of the MixColumns matrix of Midori are zeroes or ones, so that we can model this operation more accurately during Step 1, with the constraint C^M_{MCXOR}. Another factor is the linearity of the key schedule. In the AES, every key schedule round contains SBoxes and XOR operations, which introducs branching. The key schedule of Midori in Step 1 only requires to determine three 4×4 matrices (ΔIK^0, ΔIK^1 and ΔWK) for Midori64, and one (ΔIK) for Midori128. In addition, the simple key schedule of Midori makes it more vulnerable to related-key attacks: while the hardest instance for AES has 29 active SBoxes, the hardest one for Midori only has 20.

Chapter 9

Conclusions

This thesis contains two two parts: in the first one, we treat the security of contactless communications with regards to relay attacks, while the second one deals with related key cryptanalysis with constraint programming.

In the first chapter, we present the problem of secure contactless authentication. We identify two security aspects: the security of the protocol itself, and the security of the primitives composing it, which correspond to the two parts of this thesis. In the first part, we introduce relay attacks, which permit an adversary to defeat protocols by simply relaying messages between the participants, and a countermeasure: distance bounding protocols, which deter relay attacks by measuring the time of flight of the messages. In the second part, we present the symmetric cryptographic primitives that are used for contactless protocols, and note that they can all be instantiated using a block cipher. We then present the problem of related-key cryptanalysis of block ciphers, in which an adversary has access to an oracle that ciphers messages under several different keys, which are related in a way chosen by the adversary. This form of cryptanalysis is particularly relevant when block ciphers are used to build other primitives, since this use case sometimes requires using several keys which are not necessarily independent one another.

In the second chapter, we define the notations the cryptographic primitives that we use in the rest of the thesis, as well as some mathematical tools.

The third chapter is an introduction to provable security for distance bounding. We present the general structure of distance bounding protocols, as well as some classical protocols. We then present the existing threats against distance bounding protocols: mafia fraud, distance fraud, distance hijacking and terrorist fraud. Mafia fraud is an attack in which an adversary authenticates in the presence of a far away prover, while the three other attacks concern a malicious, far away prover, authenticating from a distance. In a distance fraud, he is on his own, in a distance hijacking, he uses honest provers who are in the proximity of the verifier, and in a terrorist fraud, he is helped by an accomplice, and tries to make the accomplice succeed in the protocol, while preventing him from learning enough information to authenticate on his own later. We then present these threats in a more formal way, with the DFKO model, in which our proofs are done. Finally, we present the methodology we use for our proofs.

Chapter 4 introduces two protocols we designed: SPADE and TREAD. Both are provably secure in the DFKO model, and use the same original technique to combine terrorist fraud resistance and anonymity. Instead of forcing the prover to leak a long term secret in order to help his accomplice, these protocols have the prover leak a temporary secret. This temporary secret is meant to be used in one session only, but an adversary can start a session using the same secret. If the terrorist fraud adversary learns this secret, then he can use it again at will. In both protocols, this temporary secret chosen by the prover, and sent encrypted and authenticated to the verifier. The way it is encrypted and authenticated depends on the protocol: for SPADE, the encryption is public key and the authentication is performed with a group signature scheme, which allows a prover to sign anonymously on behalf of a group. In TREAD, we present three instances: TREAD$_{sym}$ uses a symmetric encryption scheme, and a MAC for authentication. It provides no privacy. TREAD$_{priv}$ uses a public key encryption scheme and a digital signature: it is MiM-private. Finally, TREAD$_{ano}$ uses public key encryption and a group signature, and provides anonymity against malicious verifiers. We give the security proofs of these protocols in the DFKO model. The idea of using a temporary secret during the protocol can be related to the notion of one-time secure distance bounding protocols presented in [Vaudenay, 2015a], and provides interesting options for provable security. Combining it with group signatures permits to attain anonymity, revocability and terrorist fraud resistance at the same time. Group signatures are a computationally expensive primitive, which contrasts with the use of lightweight primitives in most distance bounding protocols. However, this restriction to lightweight primitives is mostly due to the traditional use of contactless on RFID devices, which typically have low power. On the other hand, we believe that in the future, distance bounding will be applied more broadly, for instance on smartphones [Gambs et al., 2016], so that the limitation on computational power will not be as relevant.

In Chapter 5, we discuss attacks that were published against SPADE and TREAD, and which use specific hardware, namely directional antennas [Ahmadi and Safavi-Naini, 2018]. Directional antennas permit to circumvent the security model, by sending messages to one specific party instead of broadcasting these messages. Hence, in a terrorist fraud, the malicious prover can send messages to the verifier without his accomplice seeing them, which enables terrorist frauds against some anonymous protocols, in which the adversary needs to see the initial messages to be able to authenticate again later. However, the attacks of [Ahmadi and Safavi-Naini, 2018] have limited applicability, since it is difficult to make sure that the accomplice will not cheat, by placing receivers in the range of the directional antennas. We then proceed with describing our own terrorist fraud, which also uses specific hardware (temper-proof devices), and applies to all protocols of the literature in which the provers can be cloned, which concerns all protocols but 2. With this attack, we show a loophole in the models, since protocols that have been proved terrorist fraud resistant are actually vulnerable. We propose a definition for terrorist fraud, called `One Step Terrorist Fraud`. In this definition, the prover is allowed to give any information to his accomplice, even secret keys, and wins if the accomplice is able to authenticate with this help.

In the second part of the thesis, we focus on the cryptographic primitives themselves, including the lightweight block cipher Midori. In Chapter 6, we present the cryptographic problem that we study, which is related-key differential cryptanalysis of block ciphers. To perform related key differential cryptanalysis, we need to study the propagation of differences in the plaintext and in the key though the cipher, and find which input/output difference pairs are the most likely. These propagation patterns are called related-key differential characteristics. We propose to tackle this problem with constraint programming, a declarative programming paradigm, in which the problem to solve is expressed in terms of variables and constraints. From this modelling, dedicated solvers perform the resolution, so that the programmer is freed from the task of designing a solving algorithm. We present the methodology we use: the problem is decomposed into two steps. In the first step, we track the positions of the differences though the cipher, and minimise the number of nonzero differences that go through SBoxes, as they lower the overall probability of the differential characteristic. In the second step, we search for actual word values that satisfy the propagation rules of the cipher, and correspond to the results of Step 1. We also present the general methodology for solving Step 2, which mostly uses table constraints. Table constraints permit to express allowed tuples, and allow to represent the SBoxes differential distribution table. The choice of using CP over MILP or other methods is motivated by the expressivity of CP: while MILP problems must be expressed in terms of linear equations, and SAT problems as Boolean formulas, none of these limitation applies to CP. Hence, the modelling is more natural and straightforward. Moreover, by writing our models in the MiniZinc CP language, we can use different solvers, in particular the SAT solver Pica_SAT.

In the seventh chapter, we apply our techniques to the search of optimal related-key differential characteristics on the encryption standard AES. We propose 3 models for Step 1, as well as a decomposition that is different from the two step decomposition used at first. The first model, S1$_{Basic}$, is a straightforward implementation of the problem, and does not filter out enough incorrect solutions to be usable. The second model, $S1_{Diff}$, introduces reasoning about the differences between differential words that can be inferred in Step 1. It scales up to AES-192 and AES-256, but fails at solving one of the instances (AES-192-10) within the allocated time. Finally, $S1_{XOR}$ infers even more differences by combining equations from the key schedule, and solves AES-192-10 in approximately 60 hours. By shifting the frontier between steps 1 and 2, in a different decomposition of the problem, we are capable of solving AES-192-10 in less than 4 hours. Using our models, we additionally disprove incorrect results that were previously published, in papers using custom search algorithms. The results we obtain on CP indicate that it is a promising approach for cryptanalysis. On the particular example of related-key differential cryptanalysis on the AES, it performed much faster than dedicated approaches (less than four hours, instead of several weeks). We also notice that the modelling choices are determinant for the resolution speed. This limits the ability of non experts to use CP for cryptanalysis.

In Chapter 8, we extend our study to Midori, which has a structure close to the AES. We use a model derived from the $S1_{Diff}$ model we used for the AES, and solve the most difficult instances in less than 10 minutes. We find full-round related key differential characteristics with probability 2^{-16} for Midori64, and 2^{-40} for Midori128. Using these differential characteristics, we mount key recovery attacks, with complexity 2^{36} for Midori64, and 2^{44} for Midori128. Interestingly, the search for Midori is significantly faster than the search for AES. This can be explained by several factors. First, the size of the search space is smaller for Midori: Midori64 has 64-bit blocks and 128-bit keys, and Midori128 has 128-bit blocks and 128-bit keys. In contrast, AES has 128-bit blocks, and up to 256-bit keys. Moreover, the key schedule of the AES is more complex, and includes a non linear part. Additionally, the MixColumns operation is less efficient for diffusion in Midori, since it is quasi-MDS, while the MixColumns operation of AES is MDS. For these reasons, Midori is weaker than AES in the related-key setting, and the search is faster.

The security of contactless communication is a vast research topic. In this thesis, we studied two aspects: provably secure distance bounding protocols, and the related-key cryptanalysis of block ciphers. For distance bounding protocols, there are several future research avenues. It is interesting that, almost 10 years after the first formal framework [Avoine et al., 2009] was introduced, and after several different formal models appeared, new attacks, such as the ones presented in [Ahmadi and Safavi-Naini, 2018, Boureanu et al., 2018], are still frequently discovered. Hopefully, the advances in automatic verification [Debant et al., 2018, Mauw et al., 2018] for distance bounding will permit to come up with definitive models, as it already uncovered some attacks, such as distance hijacking [Cremers et al., 2012]. There is still clearly some progress to be done about the formalisation of attacks, in particular terrorist fraud. It would be interesting to confront the existing security models, to establish clearly which attacks are covered by one and not the others, and which are covered by none of them. There also needs to be a reflection about which threats are relevant and which are not, as industries do not seem to be interested in the most exotic threats, such as terrorist fraud or distance hijacking.

As for the application of CP to cryptanalysis, there are even more new research opportunities. Related-key differential cryptanalysis is just one of the many forms of cryptanalysis that could be automated. We explored some other forms of cryptanalysis in [Sun et al., 2017]. In particular, impossible differential cryptanalysis would be a good application for CP. MILP methods for finding impossible differentials exist, but they only tell that a differential is impossible. On the other hand, CP has a mechanism called explanations, which could help finding exactly where the incompatibility is in an impossible differential, therefore permitting the cryptographer to verify them by hand more easily. Finally, it would be interesting to design a declarative framework, in which the cryptanalysts describe a cipher, and an automatic tool builds an efficient CP model from the description, integrating modelling tricks such as the ones we present.

Bibliography

[Abdelkhalek et al., 2017] Abdelkhalek, A., Sasaki, Y. F., Todo, Y., Tolba, M., and Youssef, A. M. (2017). Milp modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.* 79

[Ahmadi and Safavi-Naini, 2014] Ahmadi, A. and Safavi-Naini, R. (2014). Privacy-preserving distance-bounding proof-of-knowledge. In *ICICS 2014*. 32, 44, 69

[Ahmadi and Safavi-Naini, 2018] Ahmadi, A. and Safavi-Naini, R. (2018). Directional distance-bounding identification protocols. Cryptology ePrint Archive, Report 2018/366. `https://eprint.iacr.org/2018/366`. 67, 68, 69, 130, 131

[Ahmadi et al., 2018] Ahmadi, A., Safavi-Naini, R., and Akand, M. (2018). New attacks and secure design for anonymous distance-bounding. In *ACISP 2018*. Springer. 44

[Avoine et al., 2009] Avoine, G., Bingöl, M. A., Kardas, S., Lauradoux, C., and Martin, B. (2009). A formal framework for cryptanalyzing RFID distance bounding protocols. *IACR Cryptology ePrint Archive*. 22, 31, 131

[Avoine et al., 2017] Avoine, G., Bultel, X., Gambs, S., Gérault, D., Lafourcade, P., Onete, C., and Robert, J.-M. (2017). A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *ASIACCS 2017*. ACM. 7, 8, 9, 35, 41, 44, 57

[Avoine et al., 2015] Avoine, G., Carpent, X., and Hernandez-Castro, J. (2015). Pitfalls in ultra-lightweight authentication protocol designs. 4

[Avoine and Tchamkerten, 2009] Avoine, G. and Tchamkerten, A. (2009). An efficient distance bounding RFID authentication protocol: Balancing false-acceptance rate and memory requirement. In *ISC 2009*. Springer. 24

[Banik et al., 2015] Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., and Regazzoni, F. (2015). Midori: A block cipher for low energy. In *ASIACRYPT 2015*. Springer. 8, 113, 115

[Bay et al., 2013] Bay, A., Boureanu, I., Mitrokotsa, A., Spulber, I., and Vaudenay, S. (2013). The bussard-bagga and other distance-bounding protocols under attacks. In *ICISC 2013*. Springer. 30, 44, 67

[Bellare et al., 1996] Bellare, M., Canetti, R., and Krawczyk, H. (1996). Message authentication using hash functions: The hmac construction. *RSA Laboratories' CryptoBytes*. 78

[Bellare et al., 1998] Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. (1998). Relations among notions of security for public-key encryption schemes. In *CRYPTO 1998*. Springer. 13

[Bellare et al., 1994] Bellare, M., Kilian, J., and Rogaway, P. (1994). The security of cipher block chaining. In *CRYPTO 1994*. Springer. 15

[Bellare and Kohno, 2003] Bellare, M. and Kohno, T. (2003). A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT 2003*. Springer. 80

[Bellare et al., 2003] Bellare, M., Micciancio, D., and Warinschi, B. (2003). Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT 2003*. Springer. 14

[Bellare and Namprempre, 2000] Bellare, M. and Namprempre, C. (2000). Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT 2000*. Springer. 13

[Bellare et al., 2005] Bellare, M., Shi, H., and Zhang, C. (2005). Foundations of group signatures: The case of dynamic groups. In *CT-RSA 2005*. Springer. 15

[Bengio et al., 1991] Bengio, S., Brassard, G., Desmedt, Y. G., Goutier, C., and Quisquater, J.-J. (1991). Secure implementation of identification systems. *Journal of Cryptology*, (3). 6

[Bertoni et al., 2013] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. (2013). Keccak. In Johansson, T. and Nguyen, P. Q., editors, *EUROCRYPT 2013*, pages 313–314, Berlin. 18

[Beth and Desmedt, 1991] Beth, T. and Desmedt, Y. (1991). Identification tokens — or: Solving the chess grandmaster problem. In *CRYPTO 1990*. Springer. 6

[Biere, 2014] Biere, A. (2014). Yet another local search solver and lingeling and friends entering the sat competition 2014. 86, 103

[Biham, 1993] Biham, E. (1993). New types of cryptoanalytic attacks using related keys (extended abstract). In *EUROCRYPT 1993*. Springer. 78

[Biham and Shamir, 1991] Biham, E. and Shamir, A. (1991). Differential cryptoanalysis of feal and n-hash. In *EUROCRYPT 1991*. Springer. 7, 78, 82

[Biham and Shamir, 1993] Biham, E. and Shamir, A. (1993). *Differential Cryptanalysis of the Data Encryption Standard*. Springer. 126

[Biryukov and Khovratovich, 2009] Biryukov, A. and Khovratovich, D. (2009). Related-key cryptanalysis of the full AES-192 and AES-256. In *ASIACRYPT 2009*. Springer. 90, 110

[Biryukov and Nikolic, 2010] Biryukov, A. and Nikolic, I. (2010). Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In *EUROCRYPT 2010*. Springer. 8, 77, 78, 83, 90, 110

[Black and Rogaway, 2002] Black, J. and Rogaway, P. (2002). A block-cipher mode of operation for parallelizable message authentication. In *EUROCRYPT 2002*. Springer. 78

[Blanchet, 2007] Blanchet, B. (2007). CryptoVerif: A computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied"*. 64

[Bogdanov et al., 2011] Bogdanov, A., Khovratovich, D., and Rechberger, C. (2011). Biclique cryptanalysis of the full aes. In *ASIACRYPT 2011*. Springer. 89

[Bonneau, 2006] Bonneau, J. (2006). Robust final-round cache-trace attacks against aes. jbonneau@stanford.edu 13450 received 29 Oct 2006. 79

[Boureanu et al., 2018] Boureanu, I., Gerault, D., and Lafourcade, P. (2018). Fine-grained and application-ready distance-bounding security. Cryptology ePrint Archive, Report 2018/384. `https://eprint.iacr.org/2018/384`. 9, 67, 68, 131

[Boureanu et al., 2017] Boureanu, I., Gerault, D., Lafourcade, P., and Onete, C. (2017). Breaking and fixing the HB+DB protocol. In *WISEC 2017*. ACM. 9

[Boureanu et al., 2012] Boureanu, I., Mitrokotsa, A., and Vaudenay, S. (2012). On the pseudo-random function assumption in (secure) distance-bounding protocols. In *LATINCRYPT 2012*. Springer. 27, 44, 70

[Boureanu et al., 2013] Boureanu, I., Mitrokotsa, A., and Vaudenay, S. (2013). Towards secure distance bounding. In *FSE 2013*. Springer. 26, 43

[Boureanu et al., 2015] Boureanu, I., Mitrokotsa, A., and Vaudenay, S. (2015). Practical and provably secure distance-bounding. In *ISC 2013*. Springer. 5, 7, 22, 24, 26, 27, 31, 34, 41, 56, 67, 70, 71

[Boureanu and Vaudenay, 2014] Boureanu, I. and Vaudenay, S. (2014). Optimal proximity proofs. In *Inscrypt 2014*. Springer. 27, 35, 43, 70

[Boussemart et al., 2004] Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. (2004). Boosting systematic search by weighting constraints. In *ECAI 2004*. IOS Press. 107

[Brands and Chaum, 1994] Brands, S. and Chaum, D. (1994). Distance-bounding protocols. In *EUROCRYPT 1993*. Springer. 6, 21, 25, 28, 29, 30

[Brassard et al., 1988] Brassard, G., Chaum, D., and Crépeau, C. (1988). Minimum disclosure proofs of knowledge. *Journal Computer System Sciences*. 18

[Brelurut et al., 2016] Brelurut, A., Gerault, D., and Lafourcade, P. (2016). Survey of distance bounding protocols and threats. In *FPS 2015*. Springer. 6, 7, 9, 21, 28, 77

[Bultel et al., 2017] Bultel, X., Das, M. L., Gajera, H., Gérault, D., Giraud, M., and Lafourcade, P. (2017). Verifiable private polynomial evaluation. In *PROVSEC 2017*. Springer. 10

[Bultel et al., 2016] Bultel, X., Gambs, S., Gérault, D., Lafourcade, P., Onete, C., and Robert, J.-M. (2016). A prover-anonymous and terrorist-fraud resistant distance-bounding protocol. In *WISEC 2016*. ACM. 7, 8, 9, 41, 44, 45, 46, 65

[Bussard and Bagga, 2005] Bussard, L. and Bagga, W. (2005). Distance-bounding proof of knowledge to avoid real-time attacks. In *IFIP SEC 2005*. Springer US. 29, 44

[Chen and Wang, 2016] Chen, Z. and Wang, X. (2016). Impossible differential cryptanalysis of midori. *IACR Cryptology ePrint Archive*. 114

[Chu and Stuckey, 2014] Chu, G. and Stuckey, P. J. (2014). Chuffed solver description. Available at `http://www.minizinc.org/challenge2014/description_chuffed.txt`. 86, 103

[Conway, 1976] Conway, J. (1976). *On numbers and games*. Academic Press. 5

[Cremers et al., 2012] Cremers, C., Rasmussen, K. B., Schmidt, B., and Capkun, S. (2012). Distance hijacking attacks on distance bounding protocols. In *S&P 2012*. IEEE. 25, 35, 131

[Daemen and Rijmen, 2002] Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael*. Springer. 8, 80, 86, 89, 91, 94, 105

[Danev et al., 2010] Danev, B., Luecken, H., Capkun, S., and El Defrawy, K. (2010). Attacks on physical-layer identification. In *WISEC 2010*. ACM. 6

[Debant et al., 2018] Debant, A., Delaune, S., and Wiedling, C. (2018). Proving physical proximity using symbolic models. Research report, Univ Rennes, CNRS, IRISA, France. 5, 64, 66, 131

[Denning and Sacco, 1981] Denning, D. and Sacco, G. (1981). Timestamps in key distribution protocols. 4

[Derbez and Fouque, 2016] Derbez, P. and Fouque, P.-A. (2016). Automatic search of meet-in-the-middle and impossible differential attacks. In *CRYPTO 2016*. Springer. 8, 77

[Desmedt, 1988] Desmedt, Y. (1988). Major security problems with the "unforgeable" (feife-)fiat-shamir proof of identity and how to overcome them. In *Securicom 1988*. SEDEP Paris France. 6, 24, 25

[Desmedt et al., 1987] Desmedt, Y., Goutier, C., and Bengio, S. (1987). Special uses and abuses of the fiat-shamir passport protocol. In *CRYPTO 1987*. Springer. 5

[Dolev and Yao, 1981] Dolev, D. and Yao, A. C. (1981). On the security of public key protocols. In *SFCS 1981*. IEEE. 69

[Dong and Shen, 2016] Dong, X. and Shen, Y. (2016). Cryptanalysis of reduced-round midori64 block cipher. Cryptology ePrint Archive, Report 2016/676. `https://eprint.iacr.org/2016/676`. 114

[Dürholz et al., 2011] Dürholz, U., Fischlin, M., Kasper, M., and Onete, C. (2011). A formal approach to distance-bounding rfid protocols. In *ISC 2011*. Springer. 5, 7, 8, 21, 22, 26, 31, 36, 41, 70

[Fischlin and Onete, 2013a] Fischlin, M. and Onete, C. (2013a). Terrorism in distance bounding: Modeling terrorist-fraud resistance. In *ACNS 2013*. Springer. 7, 27, 36, 67

[Fischlin and Onete, 2013b] Fischlin, M. and Onete, C. (2013b). Terrorism in distance bounding: Modeling terrorist-fraud resistance. In *ACNS 2013*. Springer. 27, 30, 32, 43

[Fouque et al., 2013] Fouque, P., Jean, J., and Peyrin, T. (2013). Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In *CRYPTO 2013*. Springer. 8, 78, 83, 90, 110

[Francillon et al., 2011] Francillon, A., Danev, B., and Capkun, S. (2011). Relay attacks on passive keyless entry and start systems in modern cars. In *NDSS 2011*. 5

[Gambs et al., 2016] Gambs, S., Lassance, C. E. R. K., and Onete, C. (2016). The not-so-distant future: Distance-bounding protocols on smartphones. In *CARDIS 2015*. Springer. 130

[Gambs et al., 2014] Gambs, S., Onete, C., and Robert, J.-M. (2014). Prover anonymous and deniable distance-bounding authentication. In *WISEC 2014*. ACM. 38, 43, 44

[Gecode Team, 2006] Gecode Team (2006). Gecode: Generic constraint development environment. Available from `http://www.gecode.org`. 86, 103

[Gérault and Lafourcade, 2016] Gérault, D. and Lafourcade, P. (2016). Related-key cryptanalysis of midori. In *INDOCRYPT 2016*. Springer. 8, 9

[Gerault et al., 2017a] Gerault, D., Lafourcade, P., Minier, M., and Solnon, C. (2017a). Combining solvers to solve a cryptanalytic problem. In *CP 2017 - Doctoral program*. 9, 104

[Gerault et al., 2016] Gerault, D., Minier, M., and Solnon, C. (2016). Constraint programming models for chosen key differential cryptanalysis. In *CP 2016*. Springer. 9, 86, 96, 105, 110

[Gerault et al., 2017b] Gerault, D., Minier, M., and Solnon, C. (2017b). Using constraint programming to solve a cryptanalytic problem. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 9

[Gildas et al., 2017] Gildas, A., Muhammed, A. B., Ioana, B., Srdjan, Č., Gerhard, H., Süleyman, K., Chong, H. K., Cédric, L., Benjamin, M., Jorge, M., et al. (2017). Security of distance- bounding: A survey. *ACM Computing Surveys*. 6, 28

[Goldreich, 2006] Goldreich, O. (2006). *Foundations of Cryptography: Volume 1*. Cambridge University Press. 12

[Goldreich et al., 1986] Goldreich, O., Goldwasser, S., and Micali, S. (1986). How to construct random functions. *J. ACM*, (4). 17

[Goldwasser et al., 1989] Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*. 18

[Goldwasser et al., 1983] Goldwasser, S., Micali, S., and Yao, A. C. (1983). Strong signature schemes. In *STOC'1983*. ACM. 15

[Gorjón and van Iterson, 2015] Gorjón, X. T. and van Iterson, P. (2015). Protecting against relay attacks forging increased distance reports. 6

[Guo et al., 2016] Guo, J., Jean, J., Nikolic, I., Qiao, K., Sasaki, Y., and Sim, S. M. (2016). Invariant subspace attack against midori64 and the resistance criteria for s-box designs. *IACR Trans. Symmetric Cryptol.*, (1). 114

[Gurobi Optimization, 2016] Gurobi Optimization, I. (2016). Gurobi optimizer reference manual. 79

[Gérault et al., 2018] Gérault, D., Lafourcade, P., Minier, M., and Solnon, C. (2018). Revisiting aes related-key differential attacks with constraint programming. *Information Processing Letters*. 9

[Hancke, 2012] Hancke, G. P. (2012). Distance-bounding for rfid: Effectiveness of 'terrorist fraud' in the presence of bit errors. *RFID-TA 2012*. 26

[Hancke and Kuhn, 2005] Hancke, G. P. and Kuhn, M. G. (2005). An rfid distance bounding protocol. In *SECURECOMM 2005*. IEEE. 23

[Hermans et al., 2013a] Hermans, J., Peeters, R., and Onete, C. (2013a). Efficient, secure, private distance bounding without key updates. In *WISEC 2013*. ACM. 33

[Hermans et al., 2013b] Hermans, J., Peeters, R., and Onete, C. (2013b). Efficient, secure, private distance bounding without key updates. In *WISEC 2013*. ACM. 43, 44

[Igier and Vaudenay, 2016] Igier, M. and Vaudenay, S. (2016). Distance bounding based on puf. In *CANS 2016*. Springer. 7, 68, 70, 73

[Juels, 2006] Juels, A. (2006). Rfid security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2). 4

[Kallenberg, 2002] Kallenberg, O. (2002). *Foundations of modern probability*. Springer, second edition. 18

[Kilinąnd Vaudenay, 2015] Kilin̜, H. and Vaudenay, S. (2015). Optimal proximity proofs revisited. *Lecture Notes in Computer Science*. 24

[Kılınç and Vaudenay, 2016] Kılınç, H. and Vaudenay, S. (2016). Efficient public-key distance bounding protocol. In *ASIACRYPT 2016*. Springer. 44

[Kim et al., 2009] Kim, C. H., Avoine, G., Koeune, F., Standaert, F.-X., and Pereira, O. (2009). The swiss-knife rfid distance bounding protocol. In *ICISC 2008*. Springer. 43, 44

[Kleber et al., 2015] Kleber, S., van der Heijden, R. W., Kopp, H., and Kargl, F. (2015). Terrorist fraud resistance of distance bounding protocols employing physical unclonable functions. In *NETSYS 2015*. Springer. 24, 70, 73

[Krumm, 2009] Krumm, J. (2009). A survey of computational location privacy. *Personal Ubiquitous Comput.*, (6). 4, 42, 43

[Krumm and Horvitz, 2004] Krumm, J. and Horvitz, E. (2004). Locadio: inferring motion and location from wi-fi signal strengths. In *MOBIQUITOUS 2004*. IEEE. 6

[Lecoutre et al., 2009] Lecoutre, C., Saïs, L., Tabary, S., and Vidal, V. (2009). Reasoning from last conflict(s) in constraint programming. *Artif. Intell.*, (18). 107

[Lin and Wu, 2015] Lin, L. and Wu, W. (2015). Meet-in-the-middle attacks on reduced-round midori-64. Cryptology ePrint Archive, Report 2015/1165. `http://eprint.iacr.org/`. 114

[Liu et al., 2017] Liu, F., Cruz, W., Ma, C., Johnson, G., and Michel, L. (2017). A tolerant algebraic side-channel attack on aes using cp. In *CP 2017*. Springer. 79

[Lucks, 2000] Lucks, S. (2000). The sum of prps is a secure prf. In *EUROCRYPT 2000*. Springer. 78

[Maes and Verbauwhede, 2010] Maes, R. and Verbauwhede, I. (2010). *Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions*. Springer. 24

[Massacci, 1999] Massacci, F. (1999). Using walk-sat and rel-sat for cryptographic key search. In *IJCAI 1999*. Morgan Kaufmann Publishers Inc. 79

[MasterCard, 2017] MasterCard (2017). Contactless paypass reader specifications v3.1,. not publically available. 70

[Matsui, 1994] Matsui, M. (1994). Linear cryptanalysis method for des cipher. In *EUROCRYPT 1993*. Springer. 7

[Matsui, 1995] Matsui, M. (1995). On correlation between the order of s-boxes and the strength of des. In *EUROCRYPT 1994*. Springer. 90

[Mauw et al., 2018] Mauw, S., Smith, Z., Toro-Pozo, J., and Trujillo-Rasua, R. (2018). Distance-bounding protocols: Verification without time and location. In *S&P 2018*. Springer. 64, 66, 131

[Meadows et al., 2007] Meadows, C., Poovendran, R., Pavlovic, D., Chang, L., and Syverson, P. (2007). Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*. Springer. 22

[Meier et al., 2013] Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The tamarin prover for the symbolic analysis of security protocols. In *CAV 2013*. Springer. 64

[Merkle, 1979] Merkle, R. C. (1979). *Secrecy, Authentication, and Public Key Systems*. PhD thesis. AAI8001972. 78

[Minier et al., 2014] Minier, M., Solnon, C., and Reboul, J. (2014). Solving a Symmetric Key Cryptographic Problem with Constraint Programming. In *MODREF 2014*. 86, 93

[Mironov and Zhang, 2006] Mironov, I. and Zhang, L. (2006). Applications of sat solvers to cryptanalysis of hash functions. In *SAT 2006*. Springer. 79

[Mouha et al., 2012] Mouha, N., Wang, Q., Gu, D., and Preneel, B. (2012). Differential and linear cryptanalysis using mixed-integer linear programming. In *ICISC 2012*. Springer. 8, 77, 79

[Needham and Schroeder, 1978] Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, (12). 4

[Nethercote et al., 2007] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). Minizinc: Towards a standard CP modelling language. In *CP 2007*. Springer. 79, 86, 103, 122

[Oren et al., 2010] Oren, Y., Kirschbaum, M., Popp, T., and Wool, A. (2010). Algebraic side-channel analysis in the presence of errors. In *CHES 2010*. Springer. 79

[Prud'homme et al., 2016] Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. 86, 103, 107

[Ramamoorthy et al., 2011] Ramamoorthy, V., Silaghi, M. C., Matsui, T., Hirayama, K., and Yokoo, M. (2011). The design of cryptographic s-boxes using csps. In *CP 2011*. Springer. 79

[Rasmussen and Capkun, 2007] Rasmussen, K. B. and Capkun, S. (2007). Implications of radio fingerprinting on the security of sensor networks. In *SecureComm 2007*. 6

[Reid et al., 2007] Reid, J., Nieto, J. M. G., Tang, T., and Senadji, B. (2007). Detecting relay attacks with timing-based protocols. In *ASIACCS2007*. ACM. 26, 29, 31

[Rossi et al., 2006] Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier. 82

[Sasaki and Todo, 2017] Sasaki, Y. and Todo, Y. (2017). New impossible differential search tool from design and cryptanalysis aspects. In *EUROCRYPT 2017*. Springer. 79

[Selçuk, 2008] Selçuk, A. A. (2008). On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, (1). 126, 127

[Shahmirzadi et al., 2018] Shahmirzadi, A. R., Azimi, S. A., Salmasizadeh, M., Mohajeri, J., and Aref, M. R. (2018). Impossible differential cryptanalysis of reduced-round midori64 block cipher. *ISeCure*, (1). 114

[Shoup, 2004] Shoup, V. (2004). Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332. `https://eprint.iacr.org/2004/332`. 33, 39

[Sun et al., 2017] Sun, S., Gerault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., and Hu, L. (2017). Analysis of aes, skinny, and others with constraint programming. *IACR Transactions on Symmetric Cryptology*, (1). 10, 131

[Sun et al., 2014] Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., and Song, L. (2014). Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, des(l) and other bit-oriented block ciphers. In *ASIACRYPT 2014*. Springer. 8, 77, 79

[Tao and Wu, 2015] Tao, B. and Wu, H. (2015). Improving the biclique cryptanalysis of aes. In *ACISP 2015*. Springer. 89

[Tolba et al., 2017] Tolba, M., Abdelkhalek, A., and Youssef, A. M. (2017). Improved multiple impossible differential cryptanalysis of midori128. *IEICE Transactions*. 114

[Tripathi and Agrawal, 2014] Tripathi, R. and Agrawal, S. (2014). Comparative study of symmetric and asymmetric cryptography techniques. 77

[Trujillo-Rasua et al., 2014] Trujillo-Rasua, R., Martin, B., and Avoine, G. (2014). Distance-bounding facing both mafia and distance frauds: Technical report. *CoRR*. 24

[Urien and Piramuthu, 2014] Urien, P. and Piramuthu, S. (2014). Elliptic curve-based rfid/nfc authentication with temperature sensor input for relay attacks. *Decis. Support Syst.* 6

[Vaudenay, 2007] Vaudenay, S. (2007). On privacy models for RFID. In *ASIACRYPT 2007*. Springer. 33

[Vaudenay, 2015a] Vaudenay, S. (2015a). Private and secure public-key distance bounding: Application to NFC payment. In *FC 2015*. Springer. 43, 44, 130

[Vaudenay, 2015b] Vaudenay, S. (2015b). Sound proof of proximity of knowledge. In *PROVSEC 2015*. Springer. 43, 67

[Vila and Rodríguez, 2015] Vila, J. and Rodríguez, R. J. (2015). Practical experiences on nfc relay attacks with android. In *Radio Frequency Identification*. Springer. 5

[Wheeler and Needham, 1995] Wheeler, D. J. and Needham, R. M. (1995). Tea, a tiny encryption algorithm. In *FSE 1994*. Springer. 7

[Wu and Wang, 2011] Wu, S. and Wang, M. (2011). Security evaluation against differential cryptanalysis for block cipher structures. *IACR Cryptology ePrint Archive*. 79

[ZDNet, 2002] ZDNet (2002). New xbox security cracked by linux fans. `http://www.zdnet.com/article/new-xbox-security-cracked-by-linux-fans`. 7

[Zhou et al., 2015] Zhou, N.-F., Kjellerstrand, H., and Fruhman, J. (2015). *Constraint Solving and Planning with Picat*. Springer. 86, 103

Appendix A

Rijndael's Finite Field Multiplication Tables

0x00	0x02	0x04	0x06	0x08	0x0a	0x0c	0x0e	0x10	0x12	0x14	0x16	0x18	0x1a	0x1c	0x1e
0x20	0x22	0x24	0x26	0x28	0x2a	0x2c	0x2e	0x30	0x32	0x34	0x36	0x38	0x3a	0x3c	0x3e
0x40	0x42	0x44	0x46	0x48	0x4a	0x4c	0x4e	0x50	0x52	0x54	0x56	0x58	0x5a	0x5c	0x5e
0x60	0x62	0x64	0x66	0x68	0x6a	0x6c	0x6e	0x70	0x72	0x74	0x76	0x78	0x7a	0x7c	0x7e
0x80	0x82	0x84	0x86	0x88	0x8a	0x8c	0x8e	0x90	0x92	0x94	0x96	0x98	0x9a	0x9c	0x9e
0xa0	0xa2	0xa4	0xa6	0xa8	0xaa	0xac	0xae	0xb0	0xb2	0xb4	0xb6	0xb8	0xba	0xbc	0xbe
0xc0	0xc2	0xc4	0xc6	0xc8	0xca	0xcc	0xce	0xd0	0xd2	0xd4	0xd6	0xd8	0xda	0xdc	0xde
0xe0	0xe2	0xe4	0xe6	0xe8	0xea	0xec	0xee	0xf0	0xf2	0xf4	0xf6	0xf8	0xfa	0xfc	0xfe
0x1b	0x19	0x1f	0x1d	0x13	0x11	0x17	0x15	0x0b	0x09	0x0f	0x0d	0x03	0x01	0x07	0x05
0x3b	0x39	0x3f	0x3d	0x33	0x31	0x37	0x35	0x2b	0x29	0x2f	0x2d	0x23	0x21	0x27	0x25
0x5b	0x59	0x5f	0x5d	0x53	0x51	0x57	0x55	0x4b	0x49	0x4f	0x4d	0x43	0x41	0x47	0x45
0x7b	0x79	0x7f	0x7d	0x73	0x71	0x77	0x75	0x6b	0x69	0x6f	0x6d	0x63	0x61	0x67	0x65
0x9b	0x99	0x9f	0x9d	0x93	0x91	0x97	0x95	0x8b	0x89	0x8f	0x8d	0x83	0x81	0x87	0x85
0xbb	0xb9	0xbf	0xbd	0xb3	0xb1	0xb7	0xb5	0xab	0xa9	0xaf	0xad	0xa3	0xa1	0xa7	0xa5
0xdb	0xd9	0xdf	0xdd	0xd3	0xd1	0xd7	0xd5	0xcb	0xc9	0xcf	0xcd	0xc3	0xc1	0xc7	0xc5
0xfb	0xf9	0xff	0xfd	0xf3	0xf1	0xf7	0xf5	0xeb	0xe9	0xef	0xed	0xe3	0xe1	0xe7	0xe5

Table A.1: Lookup table for multiplication by 2 in Rijndael's Galois field. The i^{th} element of the table contains the value $i \cdot 2$, where $\cdot$ denotes multiplication in Rijndael's Galois field.

0x00	0x03	0x06	0x05	0x0c	0x0f	0x0a	0x09	0x18	0x1b	0x1e	0x1d	0x14	0x17	0x12	0x11
0x30	0x33	0x36	0x35	0x3c	0x3f	0x3a	0x39	0x28	0x2b	0x2e	0x2d	0x24	0x27	0x22	0x21
0x60	0x63	0x66	0x65	0x6c	0x6f	0x6a	0x69	0x78	0x7b	0x7e	0x7d	0x74	0x77	0x72	0x71
0x50	0x53	0x56	0x55	0x5c	0x5f	0x5a	0x59	0x48	0x4b	0x4e	0x4d	0x44	0x47	0x42	0x41
0xc0	0xc3	0xc6	0xc5	0xcc	0xcf	0xca	0xc9	0xd8	0xdb	0xde	0xdd	0xd4	0xd7	0xd2	0xd1
0xf0	0xf3	0xf6	0xf5	0xfc	0xff	0xfa	0xf9	0xe8	0xeb	0xee	0xed	0xe4	0xe7	0xe2	0xe1
0xa0	0xa3	0xa6	0xa5	0xac	0xaf	0xaa	0xa9	0xb8	0xbb	0xbe	0xbd	0xb4	0xb7	0xb2	0xb1
0x90	0x93	0x96	0x95	0x9c	0x9f	0x9a	0x99	0x88	0x8b	0x8e	0x8d	0x84	0x87	0x82	0x81
0x9b	0x98	0x9d	0x9e	0x97	0x94	0x91	0x92	0x83	0x80	0x85	0x86	0x8f	0x8c	0x89	0x8a
0xab	0xa8	0xad	0xae	0xa7	0xa4	0xa1	0xa2	0xb3	0xb0	0xb5	0xb6	0xbf	0xbc	0xb9	0xba
0xfb	0xf8	0xfd	0xfe	0xf7	0xf4	0xf1	0xf2	0xe3	0xe0	0xe5	0xe6	0xef	0xec	0xe9	0xea
0xcb	0xc8	0xcd	0xce	0xc7	0xc4	0xc1	0xc2	0xd3	0xd0	0xd5	0xd6	0xdf	0xdc	0xd9	0xda
0x5b	0x58	0x5d	0x5e	0x57	0x54	0x51	0x52	0x43	0x40	0x45	0x46	0x4f	0x4c	0x49	0x4a
0x6b	0x68	0x6d	0x6e	0x67	0x64	0x61	0x62	0x73	0x70	0x75	0x76	0x7f	0x7c	0x79	0x7a
0x3b	0x38	0x3d	0x3e	0x37	0x34	0x31	0x32	0x23	0x20	0x25	0x26	0x2f	0x2c	0x29	0x2a
0x0b	0x08	0x0d	0x0e	0x07	0x04	0x01	0x02	0x13	0x10	0x15	0x16	0x1f	0x1c	0x19	0x1a

Table A.2: Lookup table for multiplication by 3 in Rijndael's Galois field. The i^{th} element of the table contains the value $i \cdot 3$, where $\cdot$ denotes multiplication in Rijndael's Galois field.

0x00	0x09	0x12	0x1b	0x24	0x2d	0x36	0x3f	0x48	0x41	0x5a	0x53	0x6c	0x65	0x7e	0x77
0x90	0x99	0x82	0x8b	0xb4	0xbd	0xa6	0xaf	0xd8	0xd1	0xca	0xc3	0xfc	0xf5	0xee	0xe7
0x3b	0x32	0x29	0x20	0x1f	0x16	0x0d	0x04	0x73	0x7a	0x61	0x68	0x57	0x5e	0x45	0x4c
0xab	0xa2	0xb9	0xb0	0x8f	0x86	0x9d	0x94	0xe3	0xea	0xf1	0xf8	0xc7	0xce	0xd5	0xdc
0x76	0x7f	0x64	0x6d	0x52	0x5b	0x40	0x49	0x3e	0x37	0x2c	0x25	0x1a	0x13	0x08	0x01
0xe6	0xef	0xf4	0xfd	0xc2	0xcb	0xd0	0xd9	0xae	0xa7	0xbc	0xb5	0x8a	0x83	0x98	0x91
0x4d	0x44	0x5f	0x56	0x69	0x60	0x7b	0x72	0x05	0x0c	0x17	0x1e	0x21	0x28	0x33	0x3a
0xdd	0xd4	0xcf	0xc6	0xf9	0xf0	0xeb	0xe2	0x95	0x9c	0x87	0x8e	0xb1	0xb8	0xa3	0xaa
0xec	0xe5	0xfe	0xf7	0xc8	0xc1	0xda	0xd3	0xa4	0xad	0xb6	0xbf	0x80	0x89	0x92	0x9b
0x7c	0x75	0x6e	0x67	0x58	0x51	0x4a	0x43	0x34	0x3d	0x26	0x2f	0x10	0x19	0x02	0x0b
0xd7	0xde	0xc5	0xcc	0xf3	0xfa	0xe1	0xe8	0x9f	0x96	0x8d	0x84	0xbb	0xb2	0xa9	0xa0
0x47	0x4e	0x55	0x5c	0x63	0x6a	0x71	0x78	0x0f	0x06	0x1d	0x14	0x2b	0x22	0x39	0x30
0x9a	0x93	0x88	0x81	0xbe	0xb7	0xac	0xa5	0xd2	0xdb	0xc0	0xc9	0xf6	0xff	0xe4	0xed
0x0a	0x03	0x18	0x11	0x2e	0x27	0x3c	0x35	0x42	0x4b	0x50	0x59	0x66	0x6f	0x74	0x7d
0xa1	0xa8	0xb3	0xba	0x85	0x8c	0x97	0x9e	0xe9	0xe0	0xfb	0xf2	0xcd	0xc4	0xdf	0xd6
0x31	0x38	0x23	0x2a	0x15	0x1c	0x07	0x0e	0x79	0x70	0x6b	0x62	0x5d	0x54	0x4f	0x46

Table A.3: Lookup table for multiplication by 9 in Rijndael's Galois field. The i^{th} element of the table contains the value $i \cdot 9$, where $\cdot$ denotes multiplication in Rijndael's Galois field.

0x00	0x0b	0x16	0x1d	0x2c	0x27	0x3a	0x31	0x58	0x53	0x4e	0x45	0x74	0x7f	0x62	0x69
0xb0	0xbb	0xa6	0xad	0x9c	0x97	0x8a	0x81	0xe8	0xe3	0xfe	0xf5	0xc4	0xcf	0xd2	0xd9
0x7b	0x70	0x6d	0x66	0x57	0x5c	0x41	0x4a	0x23	0x28	0x35	0x3e	0x0f	0x04	0x19	0x12
0xcb	0xc0	0xdd	0xd6	0xe7	0xec	0xf1	0xfa	0x93	0x98	0x85	0x8e	0xbf	0xb4	0xa9	0xa2
0xf6	0xfd	0xe0	0xeb	0xda	0xd1	0xcc	0xc7	0xae	0xa5	0xb8	0xb3	0x82	0x89	0x94	0x9f
0x46	0x4d	0x50	0x5b	0x6a	0x61	0x7c	0x77	0x1e	0x15	0x08	0x03	0x32	0x39	0x24	0x2f
0x8d	0x86	0x9b	0x90	0xa1	0xaa	0xb7	0xbc	0xd5	0xde	0xc3	0xc8	0xf9	0xf2	0xef	0xe4
0x3d	0x36	0x2b	0x20	0x11	0x1a	0x07	0x0c	0x65	0x6e	0x73	0x78	0x49	0x42	0x5f	0x54
0xf7	0xfc	0xe1	0xea	0xdb	0xd0	0xcd	0xc6	0xaf	0xa4	0xb9	0xb2	0x83	0x88	0x95	0x9e
0x47	0x4c	0x51	0x5a	0x6b	0x60	0x7d	0x76	0x1f	0x14	0x09	0x02	0x33	0x38	0x25	0x2e
0x8c	0x87	0x9a	0x91	0xa0	0xab	0xb6	0xbd	0xd4	0xdf	0xc2	0xc9	0xf8	0xf3	0xee	0xe5
0x3c	0x37	0x2a	0x21	0x10	0x1b	0x06	0x0d	0x64	0x6f	0x72	0x79	0x48	0x43	0x5e	0x55
0x01	0x0a	0x17	0x1c	0x2d	0x26	0x3b	0x30	0x59	0x52	0x4f	0x44	0x75	0x7e	0x63	0x68
0xb1	0xba	0xa7	0xac	0x9d	0x96	0x8b	0x80	0xe9	0xe2	0xff	0xf4	0xc5	0xce	0xd3	0xd8
0x7a	0x71	0x6c	0x67	0x56	0x5d	0x40	0x4b	0x22	0x29	0x34	0x3f	0x0e	0x05	0x18	0x13
0xca	0xc1	0xdc	0xd7	0xe6	0xed	0xf0	0xfb	0x92	0x99	0x84	0x8f	0xbe	0xb5	0xa8	0xa3

Table A.4: Lookup table for multiplication by 11 in Rijndael's Galois field. The i^{th} element of the table contains the value $i \cdot 11$, where $\cdot$ denotes multiplication in Rijndael's Galois field.

0x00	0x0d	0x1a	0x17	0x34	0x39	0x2e	0x23	0x68	0x65	0x72	0x7f	0x5c	0x51	0x46	0x4b
0xd0	0xdd	0xca	0xc7	0xe4	0xe9	0xfe	0xf3	0xb8	0xb5	0xa2	0xaf	0x8c	0x81	0x96	0x9b
0xbb	0xb6	0xa1	0xac	0x8f	0x82	0x95	0x98	0xd3	0xde	0xc9	0xc4	0xe7	0xea	0xfd	0xf0
0x6b	0x66	0x71	0x7c	0x5f	0x52	0x45	0x48	0x03	0x0e	0x19	0x14	0x37	0x3a	0x2d	0x20
0x6d	0x60	0x77	0x7a	0x59	0x54	0x43	0x4e	0x05	0x08	0x1f	0x12	0x31	0x3c	0x2b	0x26
0xbd	0xb0	0xa7	0xaa	0x89	0x84	0x93	0x9e	0xd5	0xd8	0xcf	0xc2	0xe1	0xec	0xfb	0xf6
0xd6	0xdb	0xcc	0xc1	0xe2	0xef	0xf8	0xf5	0xbe	0xb3	0xa4	0xa9	0x8a	0x87	0x90	0x9d
0x06	0x0b	0x1c	0x11	0x32	0x3f	0x28	0x25	0x6e	0x63	0x74	0x79	0x5a	0x57	0x40	0x4d
0xda	0xd7	0xc0	0xcd	0xee	0xe3	0xf4	0xf9	0xb2	0xbf	0xa8	0xa5	0x86	0x8b	0x9c	0x91
0x0a	0x07	0x10	0x1d	0x3e	0x33	0x24	0x29	0x62	0x6f	0x78	0x75	0x56	0x5b	0x4c	0x41
0x61	0x6c	0x7b	0x76	0x55	0x58	0x4f	0x42	0x09	0x04	0x13	0x1e	0x3d	0x30	0x27	0x2a
0xb1	0xbc	0xab	0xa6	0x85	0x88	0x9f	0x92	0xd9	0xd4	0xc3	0xce	0xed	0xe0	0xf7	0xfa
0xb7	0xba	0xad	0xa0	0x83	0x8e	0x99	0x94	0xdf	0xd2	0xc5	0xc8	0xeb	0xe6	0xf1	0xfc
0x67	0x6a	0x7d	0x70	0x53	0x5e	0x49	0x44	0x0f	0x02	0x15	0x18	0x3b	0x36	0x21	0x2c
0x0c	0x01	0x16	0x1b	0x38	0x35	0x22	0x2f	0x64	0x69	0x7e	0x73	0x50	0x5d	0x4a	0x47
0xdc	0xd1	0xc6	0xcb	0xe8	0xe5	0xf2	0xff	0xb4	0xb9	0xae	0xa3	0x80	0x8d	0x9a	0x97

Table A.5: Lookup table for multiplication by 13 in Rijndael's Galois field. The i^{th} element of the table contains the value $i \cdot 13$, where $\cdot$ denotes multiplication in Rijndael's Galois field.

0x00	0x0e	0x1c	0x12	0x38	0x36	0x24	0x2a	0x70	0x7e	0x6c	0x62	0x48	0x46	0x54	0x5a
0xe0	0xee	0xfc	0xf2	0xd8	0xd6	0xc4	0xca	0x90	0x9e	0x8c	0x82	0xa8	0xa6	0xb4	0xba
0xdb	0xd5	0xc7	0xc9	0xe3	0xed	0xff	0xf1	0xab	0xa5	0xb7	0xb9	0x93	0x9d	0x8f	0x81
0x3b	0x35	0x27	0x29	0x03	0x0d	0x1f	0x11	0x4b	0x45	0x57	0x59	0x73	0x7d	0x6f	0x61
0xad	0xa3	0xb1	0xbf	0x95	0x9b	0x89	0x87	0xdd	0xd3	0xc1	0xcf	0xe5	0xeb	0xf9	0xf7
0x4d	0x43	0x51	0x5f	0x75	0x7b	0x69	0x67	0x3d	0x33	0x21	0x2f	0x05	0x0b	0x19	0x17
0x76	0x78	0x6a	0x64	0x4e	0x40	0x52	0x5c	0x06	0x08	0x1a	0x14	0x3e	0x30	0x22	0x2c
0x96	0x98	0x8a	0x84	0xae	0xa0	0xb2	0xbc	0xe6	0xe8	0xfa	0xf4	0xde	0xd0	0xc2	0xcc
0x41	0x4f	0x5d	0x53	0x79	0x77	0x65	0x6b	0x31	0x3f	0x2d	0x23	0x09	0x07	0x15	0x1b
0xa1	0xaf	0xbd	0xb3	0x99	0x97	0x85	0x8b	0xd1	0xdf	0xcd	0xc3	0xe9	0xe7	0xf5	0xfb
0x9a	0x94	0x86	0x88	0xa2	0xac	0xbe	0xb0	0xea	0xe4	0xf6	0xf8	0xd2	0xdc	0xce	0xc0
0x7a	0x74	0x66	0x68	0x42	0x4c	0x5e	0x50	0x0a	0x04	0x16	0x18	0x32	0x3c	0x2e	0x20
0xec	0xe2	0xf0	0xfe	0xd4	0xda	0xc8	0xc6	0x9c	0x92	0x80	0x8e	0xa4	0xaa	0xb8	0xb6
0x0c	0x02	0x10	0x1e	0x34	0x3a	0x28	0x26	0x7c	0x72	0x60	0x6e	0x44	0x4a	0x58	0x56
0x37	0x39	0x2b	0x25	0x0f	0x01	0x13	0x1d	0x47	0x49	0x5b	0x55	0x7f	0x71	0x63	0x6d
0xd7	0xd9	0xcb	0xc5	0xef	0xe1	0xf3	0xfd	0xa7	0xa9	0xbb	0xb5	0x9f	0x91	0x83	0x8d

Table A.6: Lookup table for multiplication by 14 in Rijndael's Galois field. The i^{th} element of the table contains the value $i \cdot 14$, where $\cdot$ denotes multiplication in Rijndael's Galois field.

Appendix B

The SBoxes of Midori128

In this appendx, we give the 4 SBoxes used in Midori128.

11	10	51	50	b4	30	f4	70	59	58	19	18	fc	78	bc	38
01	00	13	12	a4	20	b6	32	0b	0a	1b	1a	ae	2a	be	3a
15	31	55	71	b5	35	f5	75	5d	79	1d	39	fd	7d	bd	3d
05	21	17	33	a5	25	b7	37	0f	2b	1f	3b	af	2f	bf	3f
4b	4a	5b	5a	ee	6a	fe	7a	49	48	41	40	ec	68	e4	60
03	02	53	52	a6	22	f6	72	09	08	43	42	ac	28	e6	62
4f	6b	5f	7b	ef	6f	ff	7f	4d	69	45	61	ed	6d	e5	65
07	23	57	73	a7	27	f7	77	0d	29	47	63	ad	2d	e7	67
95	b0	d5	f0	94	90	d4	d0	dd	f8	9d	b8	dc	d8	9c	98
85	a0	97	b2	84	80	96	92	8f	aa	9f	ba	8e	8a	9e	9a
91	b1	d1	f1	14	34	54	74	d9	f9	99	b9	5c	7c	1c	3c
81	a1	93	b3	04	24	16	36	8b	ab	9b	bb	0e	2e	1e	3e
cf	ea	df	fa	ce	ca	de	da	cd	e8	c5	e0	cc	c8	c4	c0
87	a2	d7	f2	86	82	d6	d2	8d	a8	c7	e2	8c	88	c6	c2
cb	eb	db	fb	4e	6e	5e	7e	c9	e9	c1	e1	4c	6c	44	64
83	a3	d3	f3	06	26	56	76	89	a9	c3	e3	0c	2c	46	66

Table B.1: The S_0 SBox of Midori128

88	8a	4b	cb	ac	ae	6f	ef	80	82	43	c3	94	96	57	d7
a8	aa	6b	eb	8c	8e	4f	cf	98	9a	5b	db	9c	9e	5f	df
b4	b6	77	f7	a4	a6	67	e7	90	92	53	d3	84	86	47	c7
bc	be	7f	ff	a0	a2	63	e3	b8	ba	7b	fb	b0	b2	73	f3
ca	c8	4a	0a	ee	ec	6e	2e	c2	c0	42	02	d6	d4	56	16
ea	e8	6a	2a	ce	cc	4e	0e	da	d8	5a	1a	de	dc	5e	1e
f6	f4	76	36	e6	e4	66	26	d2	d0	52	12	c6	c4	46	06
fe	fc	7e	3e	e2	e0	62	22	fa	f8	7a	3a	f2	f0	72	32
08	89	09	8b	2c	ad	2d	af	00	81	01	83	14	95	15	97
28	a9	29	ab	0c	8d	0d	8f	18	99	19	9b	1c	9d	1d	9f
34	b5	35	b7	24	a5	25	a7	10	91	11	93	04	85	05	87
3c	bd	3d	bf	20	a1	21	a3	38	b9	39	bb	30	b1	31	b3
49	c9	48	0b	6d	ed	6c	2f	41	c1	40	03	55	d5	54	17
69	e9	68	2b	4d	cd	4c	0f	59	d9	58	1b	5d	dd	5c	1f
75	f5	74	37	65	e5	64	27	51	d1	50	13	45	c5	44	07
7d	fd	7c	3f	61	e1	60	23	79	f9	78	3b	71	f1	70	33

Table B.2: The S_1 SBox of Midori128

44	c3	47	43	40	c0	c2	42	54	d3	57	53	50	d0	d2	52
3c	bb	3f	3b	38	b8	ba	3a	7c	fb	7f	7b	78	f8	fa	7a
74	f3	77	73	70	f0	f2	72	64	e3	67	63	60	e0	e2	62
34	b3	37	33	30	b0	b2	32	14	93	17	13	10	90	92	12
04	83	07	03	00	80	82	02	4c	cb	4f	4b	48	c8	ca	4a
0c	8b	0f	0b	08	88	8a	0a	5c	db	5f	5b	58	d8	da	5a
2c	ab	2f	2b	28	a8	aa	2a	6c	eb	6f	6b	68	e8	ea	6a
24	a3	27	23	20	a0	a2	22	1c	9b	1f	1b	18	98	9a	1a
45	c7	46	41	c4	c5	c6	c1	55	d7	56	51	d4	d5	d6	d1
3d	bf	3e	39	bc	bd	be	b9	7d	ff	7e	79	fc	fd	fe	f9
75	f7	76	71	f4	f5	f6	f1	65	e7	66	61	e4	e5	e6	e1
35	b7	36	31	b4	b5	b6	b1	15	97	16	11	94	95	96	91
05	87	06	01	84	85	86	81	4d	cf	4e	49	cc	cd	ce	c9
0d	8f	0e	09	8c	8d	8e	89	5d	df	5e	59	dc	dd	de	d9
2d	af	2e	29	ac	ad	ae	a9	6d	ef	6e	69	ec	ed	ee	e9
25	a7	26	21	a4	a5	a6	a1	1d	9f	1e	19	9c	9d	9e	99

Table B.3: The S_2 SBox of Midori128

22	2b	20	29	a2	ab	26	2f	4b	0b	49	09	cb	8b	4f	0f
b2	bb	34	3d	32	3b	36	3f	db	9b	5d	1d	5b	1b	5f	1f
02	43	00	41	82	c3	06	47	42	03	40	01	c2	83	46	07
92	d3	14	55	12	53	16	57	d2	93	54	15	52	13	56	17
2a	23	28	21	aa	a3	2e	27	6b	0a	69	08	eb	8a	6f	0e
ba	b3	3c	35	3a	33	3e	37	fb	9a	7d	1c	7b	1a	7f	1e
62	63	60	61	e2	e3	66	67	6a	4a	68	48	ea	ca	6e	4e
f2	f3	74	75	72	73	76	77	fa	da	7c	5c	7a	5a	7e	5e
b4	bd	24	2d	b6	bf	a6	af	dd	9d	4d	0d	df	9f	cf	8f
b0	b9	30	39	a0	a9	a4	ad	d9	99	59	19	c9	89	cd	8d
94	d5	04	45	96	d7	86	c7	d4	95	44	05	d6	97	c6	87
90	d1	10	51	80	c1	84	c5	d0	91	50	11	c0	81	c4	85
bc	b5	2c	25	be	b7	ae	a7	fd	9c	6d	0c	ff	9e	ef	8e
b8	b1	38	31	a8	a1	ac	a5	f9	98	79	18	e9	88	ed	8c
f4	f5	64	65	f6	f7	e6	e7	fc	dc	6c	4c	fe	de	ee	ce
f0	f1	70	71	e0	e1	e4	e5	f8	d8	78	58	e8	c8	ec	cc

Table B.4: The S_3 SBox of Midori128

Appendix C

Related-Key Differential Characteristics For Midori

In Chapter 8, we presented related-key key recovery attacks against Midori64 and Midori128. In this appendix, we give the corresponding related key differential characteristics.

C.1 Midori64

C.1.1 Recovery of WK

The 16 related-key differential characteristics used for the recovery of WK are the following ones.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x01 0x02 0x02	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x03 0x02 0x02	

Table C.1: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x01 0x00 0x00 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x03 0x00 0x00 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.2: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00	

Table C.3: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x00 0x02 0x01 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x00 0x02 0x03 0x00 0x00 0x00	

Table C.4: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x02 0x00 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x02 0x00 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 0x00	

Table C.5: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x01 0x01 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x01 0x01 0x00 0x00 0x00 0x00	

Table C.6: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x01 0x00 0x01 0x01 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 0x01 0x00 0x01 0x01 0x00 0x00 0x00 0x00	

Table C.7: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x02$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$

δX	Value	Pr
init	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x01$ $0x01$ $0x01$	
δX_0	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_1	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_2	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_3	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_4	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_5	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_6	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_7	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_8	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_9	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{10}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_{11}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{12}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_{13}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{14}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_{15}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{16}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x03$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x01$ $0x01$ $0x01$	

Table C.8: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	$0x00$	$0x00$	$0x01$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$

δX	Value	Pr
init	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x02$ $0x00$ $0x02$	
δX_0	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_1	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_2	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_3	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_4	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_5	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_6	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_7	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_8	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_9	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{10}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_{11}	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{12}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_{13}	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{14}	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	
δX_{15}	$0x00$ $0x00$ $0x01$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot 1}$
δX_{16}	$0x00$ $0x00$ $0x03$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x02$ $0x02$ $0x00$ $0x02$	

Table C.9: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00	

Table C.10: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x02 0x02 0x02 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x02 0x02 0x02 0x00	

Table C.11: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK																
	0x00	0x00	0x00	0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x03 0x00 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x01 0x00 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.12: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK																
	0x00	0x00	0x00	0x00	0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x02 0x02 0x00 0x02 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x02 0x02 0x00 0x02 0x00 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.13: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x01 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x03 0x02 0x02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.14: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x02

δX	Value	Pr
init	0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	$2^{-2\cdot 1}$
δX_{16}	0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03	

Table C.15: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK																
	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x02	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00	

Table C.16: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

C.1.2 Recovery of K^0

The 4 related-key differential characteristics used for the recovery of K^0 are the following ones.

δK																
	0x00	0x00	0x00	0x00	0x01	0x01	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_2	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_4	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_6	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_8	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{10}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{12}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{14}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{16}	0x00 0x02 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.17: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x00 0x02

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x00 0x02	
δX_0	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_2	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_4	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_6	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_8	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{10}	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{12}	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{14}	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{16}	0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x02 0x00 0x02	

Table C.18: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x02 0x02 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x02 0x02 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_2	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_4	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_6	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_8	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{10}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{12}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{14}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_{16}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x02 0x00 0x02 0x02 0x00 0x00 0x00 0x00	

Table C.19: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

δK	0x010x010x010x000x000x000x000x000x000x000x000x000x000x000x000x00

δX	Value	Pr
init	0x010x010x010x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_0	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_1	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_2	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_3	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_4	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_5	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_6	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_7	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_8	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_9	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_{10}	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_{11}	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_{12}	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_{13}	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_{14}	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x02	$2^{-2\cdot 1}$
δX_{15}	0x000x000x000x000x000x000x000x000x000x000x000x000x000x000x000x00	
δX_{16}	0x010x010x010x000x000x000x000x000x000x000x000x000x000x000x000x02	

Table C.20: A 16-round related-key differential characteristic for Midori64. It has probability 2^{-16}.

C.2 Midori128

The 16 related-key differential characteristics used for the recovery of WK are the following ones.

δK	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x09 0x00 0x09 0x09 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x09 0x09 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x08 0x00 0x00 0x00 0x00 0x09 0x00 0x09 0x09 0x00 0x00 0x00 0x00	

Table C.21: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x02	0x02	0x00	0x02	0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x02 0x02 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x02 0x02 0x00 0x02 0x00 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.22: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK																
	0x20	0x01	0x01	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x21 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.23: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x20	0x00	0x00	0x00	0x01	0x01	0x01	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x21 0x00 0x00 0x00 0x01 0x01 0x01 0x00	

Table C.24: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x20 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x21 0x00 0x00 0x00	

Table C.25: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x00	0x00	0x00	0x01	0x01	0x00	0x01	0x00	0x00	0x00	0x09	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x01 0x01 0x00 0x01 0x00 0x00 0x00 0x08 0x00 0x00 0x00 0x00	

Table C.26: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x09	0x09	0x09

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x09 0x09 0x09	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x08 0x00 0x00 0x00 0x00 0x00 0x09 0x09 0x09	

Table C.27: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x00	0x00	0x00	0x00	0x01	0x01	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x07	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x06 0x00	

Table C.28: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x00 0x06 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00	

Table C.29: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	$0x00$	$0x00$	$0x07$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x00$	$0x01$	$0x01$	$0x00$	$0x01$

δX	Value	Pr
init	$0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x01$ $0x01$ $0x00$ $0x01$	
δX_0	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_1	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_2	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_3	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_4	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_5	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_6	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_7	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_8	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_9	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{10}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{11}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{12}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{13}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{14}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{15}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{16}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{17}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{18}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{19}	$0x00$ $0x00$ $0x07$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$	$2^{-2\cdot1}$
δX_{20}	$0x00$ $0x00$ $0x06$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x00$ $0x01$ $0x01$ $0x00$ $0x01$	

Table C.30: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x09	0x09	0x09	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01

δX	Value																Pr
init	0x09	0x09	0x09	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	
δX_0	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_1	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_2	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_3	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_4	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_5	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_6	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_7	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_8	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_9	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{10}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{11}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{12}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{13}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{14}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{15}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{16}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{17}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{18}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{19}	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	$2^{-2\cdot 1}$
δX_{20}	0x09	0x09	0x09	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x08	

Table C.31: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x02	0x01	0x02	0x02

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x02 0x02	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x03 0x02 0x02	

Table C.32: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x01 0x00 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x01 0x00 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x01 0x00 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x06 0x00 0x00 0x00 0x00 0x00	

Table C.33: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x00	0x00	0x00	0x21	0x00	0x01	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x01 0x00 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x20 0x00 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.34: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x01 0x01 0x00 0x00 0x00 0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00	
δX_0	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x01 0x01 0x00 0x00 0x00 0x00	

Table C.35: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.

δK	0x00	0x02	0x00	0x00	0x01	0x01	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

δX	Value	Pr
init	0x00 0x00 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	
δX_0	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_1	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_2	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_3	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_4	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_5	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_6	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_7	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_8	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_9	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{10}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{11}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{12}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{13}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{14}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{15}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{16}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{17}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{18}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{19}	0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	$2^{-2\cdot 1}$
δX_{20}	0x00 0x03 0x00 0x00 0x01 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	

Table C.36: A 20-round related-key differential characteristic for Midori128. It has probability 2^{-40}.