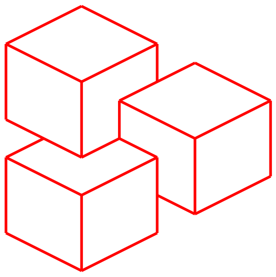
Habilitation à Diriger des Recherches

Computer Aided Security for Cryptographic Primitives, Voting protocols, and Wireless Sensor Networks



Pascal Lafourcade

6th November 2012

UNIVERSITÉ DE GRENOBLE

HABILITATION À DIRIGER DES RECHERCHES

Spécialité : **Informatique**

Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique

Présentée par

Pascal Lafourcade
préparée au sein du laboratoire **Verimag**

Computer Aided Security
For Cryptographic Primitives,
Voting protocols,
and Wireless Sensor Networks.

Habilitation à diriger des recherches soutenue publiquement le **6 Novembre 2012**, devant le jury composé de :

Dr. Gilles Barthe
IMDEA, Madrid, Espagne, Rapporteur
Pr. David Basin
ETH Zürich, Suisse, Examinateur
Pr. Hubert Comon-Lundh
LSV, ENS Cachan, France, Rapporteur
Pr. Ralf Küsters
Université de Trier, Allemagne, Rapporteur
Pr. Yassine Lakhnech
Verimag, Grenoble, France, Examinateur
Dr. David Pointcheval
LIENS, CNRS, Paris, France, Président
Pr. Peter Ryan
Université de Luxembourg, Luxembourg, Examinateur

Abstract

SECURITY is one of the main issues of modern computer science. Nowadays more and more people use a computer to perform sensitive operations like bank transfer, Internet shopping, tax payment or even to vote. Most of these users do not have any clue how the security is achieved, therefore they totally trust their applications. These applications often use cryptographic protocols which are notoriously error prone even for experts. For instance a flaw was found in the Needham-Schroeder protocol seventeen years after its publication. These errors come from several aspects:

— Proofs of security of cryptographic primitives can contain some flaws.

— Security properties are not well specified, making it difficult to formally prove them.

— Assumptions on the intruder's model might be too restrictive.

In this habilitation thesis we propose formal methods for verifying security of these three layers. First, we build Hoare logics for proving the security of cryptographic schemes like public encryption, encryption modes, Message Authentication Codes (MACs). We also study electronic voting protocols and wireless sensor networks (WSNs). In each one of these areas we first analyze the required security properties in order to propose a formal model. Then we develop adequate techniques for their verification.

Keywords: Formal verification, computational model, symbolic model, concrete security, public encryption scheme, encryption modes, MAC, homomorphic encryption, privacy, electronic voting protocol, wireless sensor networks, neigbourhood discovery, independent intruders, routing algorithms, resilience.

Résumé

La sécurité est une des préoccupations principales de l'informatique moderne. De plus en plus de personnes utilisent un ordinateur pour des opérations sensibles comme pour des transferts bancaires, des achats sur internet, le payement des impôts ou même pour voter. La plupart de ces utilisateurs ne savent pas comment la sécurité est assurée, par conséquence ils font totalement confiance à leurs applications. Souvent ces applications utilisent des protocoles cryptographiques qui sont sujet à erreur, comme le montre la célèbre faille de sécurité découverte sur le protocole de Needham-Schroeder dix-sept ans après sa publication. Ces erreurs proviennent de plusieurs aspects :

— Les preuves de primitives cryptographiques peuvent contenir des erreurs.

— Les propriétés de sécurité ne sont pas bien spécifiées, par conséquence, il n'est pas facile d'en faire la preuve.

— Les hypothèses faites sur le modèle de l'intrus sont trop restrictives.

Dans cette habilitation, nous présentons des méthodes formelles pour vérifier la sécurité selon ces trois aspects. Tout d'abord, nous construisons des logiques de Hoare afin de prouver la sécurité de primitives cryptographiques comme les chiffrements à clef publique, les modes de chiffrement asymétriques et les codes d'authentification de message (*Message authentication codes, MACs*). Nous étudions aussi les protocoles de votes électroniques et les réseaux de capteus sans fil (*Wireless Sensor Networks, WSNs*). Dans ces deux domaines, nous analysons les propriétés de sécurité afin de les modéliser formellement. Ensuite nous développons des techniques appropriées afin de les vérifier.

Mots Clefs: Vérification formelle, modèle symbolique, modèle calculatoire, sécurité concrète, chiffrement à clef publique, mode de chiffrement, MAC, chiffrement homomorphique, respect de la vie privée, vote électronique, réseaux de capteurs sans fil, découverte de voisinage, intrus indépendants, algorithmes de routage, résilience.

Thank You!

I would like first to thank all jury members, and more particularly the reviewers Gilles Barthe, Hubert Comon-Lundh and Ralf Küesters, for having accepting to read carefully my habilitation. I also want thank David Pointcheval to be the president of the jury of my habilitation. Moreover I would like to thank David Basin and Peter Ryan to come in Grenoble for the Defense as examinators. Finally I have a special thought for the head of our group Yassine Lakhnech for his support and collaboration at Verimag.

I would like to thank all members of Verimag and also all my collaborators for these successful years, all papers we have written together and also the good time spend doing research together. Additionally I have to thank all students that have worked with me during all these years: Marion, Jannik, Ali, Raphaël, Antoine, Martin, Vanessa, Sylvain to name only a few.

I also thank all my family for her help and also the woman I love for having shown the way. I a big thanks for Annie-Laure, Naima and Karim for their help.

« Dans la vie, les hommes sont tributaires les uns des autres.
Il y a donc toujours quelqu'un à maudire ou à remercier.»
Madeleine Ferron, Extrait de « Le chemin des dames ».

Contents

Chapter 1

Introduction

PREAMBLE: This habilitation thesis summarizes a selection of my research results obtained since my PhD thesis. This summary is not exhaustive. In order to facilitate reading, citations of my own works are denoted using a plain, numerical style, *e.g.* [1], while other works are cited using alpha-numerical style, *e.g.* [NS78]. Moreover many proofs and details are not presented in this document but can be found in associated articles or technical reports.

Contents

1.1 Context

Proving that a protocol is secure is not an easy task, even for simple and short protocols. In 1996 G.Lowe [Low96] found an attack on the famous Needham-Schroeder protocol [NS78] seventeen years after its publication. He found this flaw using his automatic verification tool Casper [Low98] based on the CSP (Communicating Sequential Processes) model checker FDR [Hoa85, Sch96, RSG$^+$00]. The protocol proposed by Needham and Schroeder was proven secure by using a formalism called BAN logic using the following intruder model: only one single execution of the protocol. But the flaw proposed uses two parallel executions of the protocol and assumes the so-called *perfect encryption hypothesis*. More precisely it means that the only way to decrypt a cipher text is to know the inverse key. This hypothesis abstracts the cryptography in order to detect "logical flaw" due to possible interleavings of different protocol executions. This model is called *symbolic* by contrast to the *computational* approach proposed by the cryptographers where intruders are represented by polynomial probabilistic Turing Machines and messages are bits. In the symbolic model:

— messages are represented by terms built over a signature,

— the perfect encryption hypothesis is assumed,

— the intruder controls the network,

— the intruder has a limited capabilities, usually represented by a deduction system, first introduced by Dolev and Yao in [DY81].

The discovery of the "logical" attack by G. Lowe shows that even experts can miss some flaws even on small protocols (only three messages exchanged) and often underestimate the complexity of analyzing the security of such protocols. It also indicates that automatic analysis is crucial for assessing the security of cryptographic protocols, because humans can not reasonably verify their correctness. Then formal verification methods are an ideal tool to analyze protocols and avoid bugs. Hence symbolic and automatic verification of cryptographic protocols is one of the active topics in security. Over the past two decades many automatic tools based on several different formal techniques have been designed for cryptographic protocols, *e.g* [ABB$^+$05, Bla01, Bor01, BLP03, CE02, Cre08, Low98, Mea96, MMS97, Ros95, SBP01]. One can find a survey of formal approaches for proving security protocols in [BCM11].

On the other the hand, security proofs of cryptographic primitives are usually manually done by their authors. Unfortunately mistakes can appear as it was the case with the famous case of OAEP [BR94, Sho02, FOPS04a, BGLB11a]. In order to detect some mistakes, one possible technique is to prove in the symbolic world the security of a scheme and to establish soundness results as it has been done after the pioneer work of Abadi et al [AR00], see [CKW11] for a survey of these techniques. However there is another solution which consists in directly reasoning in the computational model using logical ingredients such as predicates, deduction rules, Hoare logic.

Such cryptographic primitives are used to develop secure applications in an hostile environment. With Internet democratization, many countries might want to use electronic voting protocols for their elections. There already exist several e-voting protocols, but voter fear is real. Issues include preserving confidentiality of vote, preventing double voting, ensuring that all votes are counted etc. Many of these properties are achieved by cryptographic operators like homomorphic encryption *e.g.* J.Benaloh [Ben87]. A formal analysis of these properties should augment the confidence of citizens in such protocols and avoid bugs in applications.

Another application is the security in wireless networks. Cellular phones are now more common than wired ones, the number of mobile connections to Wireless Local Area Networks (WLANs) is increasing, and wireless devices are now used everyday at homes, companies and administrations. Moreover wireless devices have become so small and cheap that they can be used in sensor networking applications such as environmental or building monitoring. These devices communicate by relaying packets of other devices across multiple wireless links (hops). This new communication technology changes usual intruder models used in verification of cryptographic protocols. For instance: agents can move and still be connected; nodes have a limited battery and computation power; intercepting a message over the network becomes easy; an intruder cannot control the whole network.

1.2 Contributions

All works of this habilitation thesis aim to formally model security properties, to verify security of protocols and to prove schemes under well identified assumptions. All my contributions have been part of the following projects: ANR SCALP, AVOTÉ, SFINCS, ARESA2 and PROSE, project Minalogic SHIVA and project UJF MSTIC TERRA.

Security mistakes can come from several aspects:

— Proofs of security of cryptographic primitives can contain some flaws.

— Security properties are not well specified, making it difficult to formally prove them.

— Assumptions on intruder's model might be too restrictive.

In this habilitation thesis we propose formal methods for verifying security of these three axis. First, we look at the security of cryptographic schemes. Then we analyze electronic voting protocols and wireless sensors networks (WSNs).

1.2.1 — Proving Cryptographic Primitives using Logic

Several cryptographic primitives thought to be proven secure by their authors have been broken several year latter, due to errors in the original proofs. For instance it took 5 years to break the Merkle-Hellman cryptosystem [MH06], 10 years to break the Chor-Rivest cryptosystem [CR88, Jr.91, Vau98]. Proving the security of cryptographic primitives is not an easy task and is error prone.

For verifying such primitives, we analyze a cryptographic primitive as a successive application of simple actions forming a small program. We model security properties by a set of invariants. Then we prove a set of Hoare logic rules for each simple action in order to preserve or propagate invariants through the program. If the application of our rules on a program leads us to a set of invariants modeling our security property then the primitive is secure. In the first part of this habilitation thesis, we propose three logics in order to verify some public encryption schemes, symmetric encryption modes, and MACs [5, 18, 6, 17].

1.2.2 — Electronic Voting Protocols

In this area, many different formal models and definitions of the security properties have been proposed such as [CG96, DKR09, DKR10, MN06, BHM08, SRKK10, KRS10, JCJ05, JCJ02, SC10, SC11, KT09, UMQ10, LJP10]. They have been used successfully to discover bugs in systems like Helios [SC11]. However, since the structure, setting and basic design of voting protocols can be quite different depending on the primitives used, many of these definitions are tailored to fit a specific (sub-)group of protocols. Sometimes a protocol can be proved secure in one model, but not in another. This hinders the objective comparisons of protocols. We refine privacy notions, which allow us to compare the security level of existing protocols. We obtain a precise formal taxonomy of different privacy notions [11] and are able to compare security levels of existing protocols. Moreover we give a definition of privacy notion for voting protocols which allow weighted votes. In this framework, we also prove that it is equivalent to consider one or several malicious voters [10]. For obtaining this we prove a result of process decomposition unicity in applied π-calculus [14]. We also use the applied π-calculus to formalize the vote-independence: a protocol ensures this property if a vote cannot "copy" the vote of another voter. In [11], we show the relation with existing notions and give an example of protocol which does not satisfy this property.

Finally, in [15], we discover that Benaloh's homomorphic encryption [Ben87] is ambiguous: for certain parameters decryption of a cipher text gives two valid plaintexts. We illustrate the impact of such problem on several applications such as an electronic voting protocol. We also propose a modified version of this scheme and prove its correctness.

1.2.3 — Wireless Sensor Networks

During the last years several formal verification works [SSBC09a, ACD10, DDS10, CDD12] have been proposed in this area. One of the main issues for WSNs is that each node must discover or rediscover which nodes are within its communication range. This is called *neighborhood discovery*. In 2006, C. Meadows et al [MPP$^+$06] propose a first formal analysis of neighborhood discovery protocol without mobility. In [20] we give a model inspired from [SSBC09a] in which we consider mobile agents and intruders. We also provide a formal definition of the (k)-neighborhood.

Recently [ACD10], the authors propose a formal verification method for verifying routing protocols. They prove that the route built by a protocol corresponds to all visited nodes. They give an extension of the Dolev-Yao intruder model where all nodes share collected information. If we assume deployed sensors in the nature, it becomes really difficult and expensive for an intruder to collect in several points exchanged messages and to easily exchange all collected information. We consider independent intruders that do not know each other and do not share any knowledge. In this intruder model, the classical assumption of monotonicity of the intruder knowledge is no longer true. We propose a decision algorithm for constraint systems associated to this weaker intruder model [Kas12].

Another challenging problem in WSNs is to develop efficient and resilient routing algorithms. By resilient we mean that if a node disappear due to an environmental event (fire, rain etc.), or to an attack by an intruder, or simply by reaching the battery limits, then the routing mechanism should continue to work. As a consequence a deterministic routing algorithms cannot be resilient to topology changes. On the other hand random walk is really resilient but is not efficient. We propose two categories of probabilistic routing algorithms using tabu lists in order to improve random walks. Our first idea is to share the information of already visited nodes. We include into each message the list of already visited nodes. Then each node routes the messages randomly over their neighbors excluding nodes of the tabu list. This generates an overload in the network and is not secure in presence of an intruder. In order to avoid these drawbacks we propose a first family of algorithms which store routing information locally in each node. This allows them to be more efficient than random walks by avoiding links which are identified as a cycle. Moreover in order to be more resilient to the intruders we propose a last routing algorithm which uses cryptographic primitives and acknowledgment mechanism.

1.3 Content of the Thesis

Before presenting the plan of this habilitation thesis, I explain the process followed to obtain these works. I started my research with my Master thesis [22] in Artificial Intelligence on "Solving logical conflicts, using decision methods for sequencing" at University Paul Sabatier (Toulouse). I continue in PhD [23] on "Verification of cryptographic protocols in presence of algebraic properties" at ENS Cachan (LSV). In my PhD, I studied homomorphic inscriptions, which have the following property: $\{a\}_k * \{b\}_k = \{a + b\}_k$. I proposed an intruder model which takes into account this property and allows to find attacks on protocols proven secure in the classical Dolev-Yao intruder model. Then during my post-doctoral internship at the ETH Zürich, I started to look at efficiency of verification tools and at the neighborhood property in WSNs [25]. Then I naturally continue to study WSNs at Verimag, by proposing a model for verifying (k)-neighborhood property for mobile node, another model for independent intruders, and some protocols and analysis of routing algorithms. I also continue to study homomorphic encryptions by developing models for electronic voting protocols in the symbolic model. Finally I have started a new topic research with the verification of cryptographic primitives in the computational model.

1.3.1 — Outline

This document is organized in three independent parts.

Part I In this first part, we propose three Hoare Logic to verify security of asymmetric encryption in Chapter 2, prove the IND-CPA security of symmetric encryption mode in Chapter 3 and prove the security of Message Authentication Codes (MACs) in Chapter 4.

Part II In Chapter 5, we revisited Benaloh's homomorphic encryption [CB87][15]. In Chapter 6, we propose a formal taxonomy of Privacy in voting protocols [11]. Finally in Chapter 7, we formally model Privacy notions for protocols supported weighted vote in the applied π-calculus [10, 14].

Part III In Chapter 8, we present our formal model for secure (k)-neighborhood discovery protocols. In Chapter 9, we develop a formal constraint system in order to analyze routing protocols in presence of independent intruders. In Chapter 10, we propose a class of probabilistic routing algorithms which store information in the messages. We also make a theoretical analysis of such protocols. Finally in Chapter 11, we give another class of probabilistic routing algorithms which store local information in each node. We also shows that such algorithms are resilient to several types of attacks.

1.3.2 — Collaborations

The results presented in this habilitation thesis have been mainly obtained in collaboration with the researchers listed below:

— Mohammed ALNUAIMI
— Karine ALTISEN
— Judicaël COURANT
— Cas J. F. CREMERS
— Marion DAUBIGNARD
— Stéphane DEVISMES
— Jannik DREIER
— Cristian ENE
— Laurent FOUSSE
— Martin GAGNÉ
— Antoine GERBAUD
— Nacira GHOUALMI
— Raphaël JAMET
— Ali KASSEM
— Noudjoud KAHYA
— Yassine LAKHNECH
— Philippe NADEAU
— Clément PONSONNET
— Reihaneh SAFAVI-NAINI
— Vanessa TERRADE
— Sylvain VIGIER

In particular, this habilitation thesis describes some results obtained during the supervision of the following PhDs:

— Marion Daubignard, she obtained her PhD about formalization of concrete security proofs in January 2012 [Dau12]. We have collaborated for developing our first Hoare Logic for public encryption scheme [5, 6], presented in Chapter 2.

— Jannik Dreier, he started his PhD in September 2010 at Verimag, we have worked on formalizing privacy notions for electronic voting protocols [8, 14, 11, 10], works presented in Chapter 6 and 7.

— Raphaël Jamet, he began his PhD in September 2011 at Verimag, we worked together on the formalization of the (k)-neighborhood property in Chapter 8 and [20]. He is now studying resilient routing algorithms in WSNs, as it is presented in Chapter 11.

Other results have been obtained during the supervision of some undergraduate internships, master theses and post-doctoral internships:

— During the Master internship of Ali Kassem [Kas12], we proposed a model for verifying routing protocol in presence of independent intruders, work presented in Chapter 9.

— The work [15], presented in Chapter 5, about Benaloh's encryption scheme [CB87] was initiated during the Mohammed Alnuaimi's master internship.

— The results obtained during the Master thesis of Clément Ponsonnet, allowed us to design routing algorithms based on tabu lists which are presented in Chapter 10 and 11.

— In collaboration with Nacira Ghoualmi professor and Noudjoud Kahya PhD student at Badji Mokhtar University (Annaba, Algeria) we used SCYTHER [Cre08] for verifying IEEE802.16 standard (Wimax). This analysis shows some weaknesses that we corrected [19]. This work is not presented in this habilitation thesis.

— During the collaboration with Antoine Gerbaud, post-doctoral fellow at Verimag, we analyzed some routing algorithms using tabu lists [1, 2]. This work is described in Chapter 10.

— With Martin Gagné post-doctoral fellow we developed the Hoare logics for encryption mode and for MACs respectively presented in Chapter 3 and 4.

— During the undergraduate internships of Vanessa Terrade and Sylvain Vigier, we have extended the comparison of existing automatic verification tool done in [7] to protocols using algebraic properties [24]. These works are not presented in this habilitation thesis.

Part I

Proving Cryptographic Primitives using Hoare Logic

Proving security of cryptographic primitives is not an easy task. Cryptographers design schemes and usually provide hand proofs of their security. Unfortunately sometimes such proofs contain errors. For instance, the scheme proposed by Zheng and Seberry [ZS93] was claimed to be IND-CCA2 by the authors but an attack was found few years later [SSQ02]. Another famous example is OAEP [BR94], which was proven secure until Victor Shoup in 2000 [Sho02] gave an attack in a particular setting. Then an updated version using RSA was proposed in [FOPS04b]. All these examples clearly show that proving security of cryptographic primitives is difficult and error prone due to their intrinsic complexity. Then it is important to develop automatic formal methods for proving such schemes. In [Sho04], Shoup presents a first attempt by using games as a way to clarify and structure such proofs. Moreover Halevi has argued in [Hal05] that the design of formal verification techniques and their implementation in tools that could be used by the cryptographic community is crucial to support the trust in cryptographic proofs.

Related Work: *Using the Symbolic Model:* A possible approach is to prove the scheme in the symbolic model using formal methods and also prove a soundness result. In [AR00], Abadi and Rogaway proved the first *computational soundness result.* Such theorems typically argue that if proven in the symbolic model, a security criterion also holds in the computational model. After this work several relevant computational soundness results have been obtained, see [CKW11] for an inventory of the different flavors of results and practices.

In [DDMR07], Datta, Derek, Mitchell and Roy present a survey of the Protocol Composition Logic (PCL), which is a logic in Hoare style to prove security properties of network protocols, using symmetric and asymmetric schemes. The reasoning of PCL is not performed in the computational model directly. The computational soundness is given under the assumption that the protocols use an IND-CCA encryption scheme. The associated computational semantics are given in [BDD$^+$06]. Several extensions of PCL have been published [DDMW06, RDDM07, RDM07]. These techniques have been applied to prove security achievements of widely used protocols such as SSL/TLS, IEEE 802.11i, Kerberos V5, Diffie-Hellman based protocols and IKEv2, the revised standard key management protocol for IPSEC.

Fournet et al. [FKS11] developed a framework for modular code-based cryptographic verification. However, their approach considers interfaces for MACs. In a way, our work is complementary to theirs, as our results presented in Chapter 4, coupled with theirs, could enable a more complete verification of systems.

The Direct Approach: In [BR04], Bellare and Rogaway propose code-based proofs resembling game-playing techniques, as a first step towards enabling automatic verification of the proofs. However it does not provide a formalism for the verification.

Impagliazzo and Kapron were the first to develop a logic to reason about indistinguishability in [IK06]. It is based on a more general logic whose soundness relies on non-standard arithmetic. Later, Zhang has proposed a logic built on top of Hofmann's SLR, named computational SLR [Zha08]. His logic, as that of Impagliazzo and Kapron, allows for a proof that next-bit unpredictability implies pseudo-randomness and is used to show the correctness of a pseudo-random generator.

In [CdH06], R. Corin and Den Hartog present a Hoare-style proof system for game-based cryptographic proofs. They are able to prove semantic security of ElGamal.

In [BCT04, BT04], G. Barthe et al give machine-checked proofs of computational security using the Coq-proof assistant [BC04, Tea04] for hardness of the discrete logarithm in the generic model and security of signed ElGamal encryption against interactive attacks.

Nowak [Now07] also shows a preliminary implementation of the game-based approach in Coq. Backes, Berg, Unruh [BBU08] propose a formalization of a language for games in the Isabelle proof assistant, and provide a proof of the fundamental lemma of game-playing; however, subsequent work [BMU10] rather presents symbolic analysis with computational soundness results.

In [BDKL10], the authors introduce the Computational Indistinguishability Logic (CIL). It is a logic for proving the security of cryptographic primitives in the computational model. This framework can be extended in order to integrate results that are presented in this part.

Tools : Nowadays, there exist some automated verification frameworks for computational security proofs:

— CryptoVerif [BP06, BJST08, Bla08], a tool developed by Bruno Blanchet. It is based on observational equivalence. It uses rules to transform games into equivalent or almost equivalent ones. A large collection of cases studies is available with the tool including an automated proof of Kerberos 5 and Diffie-Hellman based constructions. Further progress have been reported in [BP10, Bla11, PB12].

— CertiCrypt [BGZB09] is a framework enabling machine-checked design and verification of code-based concrete security proofs. It is built on top of the proof assistant Coq. Case studies include existential unforgeability of the FDH signature scheme and semantic security of OAEP.

— EasyCrypt [BGHB11], an automated tool generating partial verifiable evidence of cryptographic proofs out of proof sketches. The sketches are checked using SMT solvers and automated theorem provers, and then compiled into verifiable proofs in the Certicrypt framework. Recently, in [BGLB11b], they prove the security of OAEP.

Contributions: Our aim is to develop a formal method in order to prove cryptographic primitives. We consider cryptographic primitives as small programs, usually it is just few lines of code. In the last century Tony Hoare provided mechanisms in order to prove that a program is correct, so-called *Hoare logic.* Our approach consists in the three following steps:

1. Modeling properties that studied primitives have to ensure by proposing some invariants.
2. Defining the language which represent the small set of commands used to construct a primitives at the good abstract level.
3. Proving Logics rules, that allows us to generate and propagate invariants according to

the commands executed in the program.

Once the language fixed, the invariants developed and the rules of the Hoare Logic established, we can verify a primitive. This approach is sound, because all our rules are proved. It means that if – after modeling a primitive in the specific language and applying the rules for each command – we obtain the desired invariant, this means that the primitive satisfies the properties. But if the analysis stops or does not provide desired invariant at the end, usually two situations appear: the scheme is not secure and there is an attack, or our system cannot prove this scheme. This last possibility comes from the fact that is very difficult to design a Logic which is complete. The three logics presented in this part contain enough rules to prove all secure primitives and to fail on insecure well-known schemes.

Outline: In Chapter 2, we present a Hoare Logic for verifying asymmetric encryption. In Chapter 3 we develop a framework for proving the IND-CPA security of symmetric encryption mode. Finally in Chapter 4, we propose other invariants and another Logic for proving the security of Message Authentication Codes (MACs).

Chapter 2

Public Encryption Schemes

On admet que le niveau minimum de sécurité des chiffrements à clef publique est IND-CPA. De plus de nombreux chiffrements à clef publique peuvent s'écrire sous forme de petits programmes de quelques lignes. Nous définissons une logique de Hoare afin d'analyser ces chiffrements et de prouver qu'ils sont IND-CPA [5, 6][Dau12].

Contents

C. A. R. Hoare [Hoa69] and R. Floyd [Flo67] proposed inference systems using statements of the form $\{P\}$ cmd $\{Q\}$, where P and Q are assertions and cmd is a command. Traditionally, P is called a precondition and Q a postcondition. A Hoare triple is valid if whenever the precondition is met, the postcondition is verified after the command execution. We propose a Hoare logic for capturing asymmetric encryption constructions in the random oracle model. Our framework is based on programs specified using a small fixed programming language operating. After having provided a semantics for this language we define predicates capturing properties of the encryption schemes.

2.1 Programming Language

We suppose that we have a finite set of symbols $\vec{H}$, representing a finite collection of hash functions; H is a meta-variable ranging over $\vec{H}$. Each hash function H is mapped to an integer $\ell(H)$, which is a (public) parameter of the system and represents the length of outputs of the hash function. Moreover, each element $H \in \vec{H}$ is associated to a specific list, the variable $\mathcal{T}_H$. The collection of all these lists is denoted HList. In addition, we have two special variables pk and sk which are meant to store public and secret key values; symbol $\tilde{f}$ denotes an algorithm which can be instantiated for any value of pk and then yield a trapdoor permutation f. We suppose that $\tilde{f}$ is fixed and known to the adversary, and $OW(t)$ denotes an upper-bound of the probability that an adversary succeeds in inverting f on a random argument in time at most t. Additionally, $\mathcal{K}$ denotes the key generation algorithm. Another special variable, σ, takes values in bitstrings of finite length, and denotes the variable in which the adversary can store its state. Finally, we have a finite set of variables Var to denote any other variables. We use any symbol different from those above to denote elements in Var, *e.g.* x, y, t, q. Though finite, the set Var is assumed to be large enough to contain all symbols that we introduce in programs.

Grammar: We now describe a simple imperative language with random assignment. Whereas the language allows the application of a trapdoor permutation f, it does not include the application of the inverse of such permutations. Our programming language is built according to the BNF described in Table 2.1, where:

— $x \xleftarrow{} \mathcal{U}(l)$ samples a value in $\mathcal{U}(l)$ and assigns it to x, with $\mathcal{U}$ the uniform distribution over the set of bitstrings of length l. Since we do not suppose that all the bitstrings are drawn in the same set $\{0,1\}^l$, we have to specify a length for each drawing command. In the sequel, we use l as a generic notation each time a length needs to be specified.

— $x := f(y)$ applies the trapdoor one-way function f to the value of y and assigns the result to x.

— $x := \alpha \oplus H(y)$, where α is a constant or a variable. This command first applies the random oracle H to the value v of y, *i.e.* a new hash value h is drawn whenever v has not been already hashed. As a side effect, the pair (v, h) is added to the variable $\mathcal{T}_H$ if it was not stored yet. After the hash computation, the bitwise exclusive or of α and the hash value is assigned to x.

— $x := y \oplus z$ applies the exclusive or operator to the values of y and z and assigns the result to x.

— $x := y||z$ represents the concatenation of the values of y and z.

— $\mathsf{cmd}_1; \mathsf{cmd}_2$ is the sequential composition of commands cmd_1 and cmd_2.

$$\begin{array}{lll} \text{Command} & \mathsf{cmd} ::= & x \leftarrow \mathcal{U}(l) \mid x := f(y) \mid x := \alpha \oplus H(y) \mid \\ & & x := y \oplus z \mid x := y||z \mid \mathsf{cmd}; \mathsf{cmd} \end{array}$$

Table 2.1 – Language Grammar.

We disregard the execution time needed for all these operations but the application of f as it is usually the case in concrete security proofs. We assume that command $x := f(y)$ admits an upper-bound on the time required for its execution, independent of the input, which we denote T_f. We can then define $\mathsf{T}_{\mathsf{cmd}}$ for any command in the language by summing the number of applications of f involved.

Semantics: States $m \in \mathsf{M}$ map pk, sk, σ and all variables of Var to bitstrings, and variables HList to lists of pairs of bitstrings. The semantics of our language is specified in Table 2.2. Notice that the semantic function of commands can be lifted in the usual way to a function from $\mathrm{DIST}(\mathsf{M})$ to $\mathrm{DIST}(\mathsf{M})$. That is, let $F : \mathsf{M} \to \mathrm{DIST}(\mathsf{M})$ be a function. Then, F defines a unique function $F^* : \mathrm{DIST}(\mathsf{M}) \to \mathrm{DIST}(\mathsf{M})$ such that $F^*(D) = [m \leftarrow D; m' \leftarrow F(m) : m']$. By abuse of notation we denote the semantic function of commands and their lifted counterparts by $[\![\mathsf{cmd}]\!]$: according to our semantics, commands denote functions that transform distributions on states into distributions on states.

$$\begin{array}{l}
[\![x \leftarrow \mathcal{U}(l)]\!](m) = [u \leftarrow \mathcal{U}(l) : m.[x \mapsto u]] \\
[\![x := f(y)]\!](m) = \delta(m.[x \mapsto \tilde{f}(m.pk, m.y)]) \\
[\![x := \alpha \oplus H(y)]\!](m) = \\
\left\{ \begin{array}{l} \delta(m.[x \mapsto m.\alpha \oplus \mathcal{T}_H(m.y)]) \text{ if } m.y \in \mathsf{dom}(m.\mathcal{T}_H) \\ [v \leftarrow \mathcal{U}(\ell(H)) : m.[x \mapsto m.\alpha \oplus v, \mathcal{T}_H \mapsto \mathcal{T}_H :: (m.y, v)]] \text{ if } m.y \notin \mathsf{dom}(m.\mathcal{T}_H) \end{array} \right. \\
[\![x := y \oplus z]\!](m) = \delta(m.[x \mapsto m.y \oplus m.z]) \\
[\![x := y||z]\!](m) = \delta(m.[x \mapsto m.y||m.z]) \\
[\![\mathsf{cmd}_1; \mathsf{cmd}_2]\!] = [\![\mathsf{cmd}_2]\!] \circ [\![\mathsf{cmd}_1]\!]
\end{array}$$

Table 2.2 – The semantics of the programming language

Definition 1 *Constructible Distribution Let χ be a function mapping each $H \in \vec{H}$ to a positive integer. A χ-*constructible *distribution is of the form:*

$$\left[(pk_0, sk_0) \leftarrow \mathcal{K}; m \leftarrow \mathfrak{A}^{\vec{H}}(pk_0) : m[pk \mapsto pk_0, sk \mapsto sk_0]\right]$$

where $\mathfrak{A}$ is a probabilistic algorithm with oracle access to the hash functions in $\vec{H}$, making a number of calls bounded by χ. Moreover, $\mathfrak{A}$'s queries to the hash oracles are recorded with their answers in the lists of HList in m; in other words, $\mathfrak{A}$ cannot *tamper with these lists. Finally, we require that for all hash function $H \in \vec{H}$, $\mathsf{Card}(\mathit{dom}(\mathcal{T}_H)) \leq \chi(H)$.*

The set of χ-constructible distributions is denoted by $\mathfrak{const}\mathfrak{D}(\chi)$. A distribution X is said constructible if there exists a tuple χ such that $X \in \mathfrak{const}\mathfrak{D}(\chi)$. The set of constructible distributions is denoted $\mathfrak{const}\mathfrak{D}$.

Indistinguishability: We define the indistinguishability between two constructible distributions.

Definition 2 *Indistinguishability Let $\mathcal{A}$ denote a (k,t)-adversary, and let X and X' be two constructible distributions in $\mathfrak{const}\mathfrak{D}(\chi)$. Let $\varepsilon : (k,\chi,t) \mapsto \varepsilon(k,\chi,t)$ be a function ranging in $[0,1]$. The advantage of $\mathcal{A}$ in distinguishing X and X' is denoted $\mathbf{Adv}(\mathcal{A},X,X')$ and defined as*

$$|\Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}(m.[sk \mapsto \lambda]) = 1] - \Pr[m \leftarrow X' : \mathcal{A}^{\vec{H}}(m.[sk \mapsto \lambda]) = 1]|$$

We say that distributions X and X' are ε-indistinguishable iff for all (k,t)-adversary $\mathcal{A}$, $\mathbf{Adv}(\mathcal{A},X,X') \leq \epsilon(k,\chi,t)$. In this case, we write $X \sim_{\varepsilon} X'$.

We stress that $\mathcal{A}$ is provided with the whole information stored in the state *but* the secret key value. In particular, it includes lists $(\mathcal{T}_H)_{H\in\vec{H}}$ of hash values computed during the construction of X or X'. As a consequence, whenever $\mathcal{A}$ queries one of its oracle H on a value appearing in list $\mathcal{T}_H$, it gets the same answer that is stored in $\mathcal{T}_H$. We emphasize that the function k bounding the number of queries to oracles in $\vec{H}$ does not take into account queries performed during the construction of X or X'.

Definition 3 *Distribution Restricted to Sets of Variables Let X be a constructible distribution, let V_1 and V_2 be sets of variables such that $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$. By $D(X,V_1,V_2)$ we denote the following distribution:*

$$D(X,V_1,V_2) = [m \leftarrow X : (m.V_1, f(m.V_2))]$$

where $m.V_1$ denotes the values of variables in V_1 in state m, and $f(m.V_2)$ denotes the point-wise application of f to the values given by state m to variables in V_2.

Definition 4 *Restricted Indistinguishability Let X and X' be two constructible distributions. X and X' are $V_1;V_2;\varepsilon$-*indistinguishable*, denoted by $X \sim_{V_1;V_2;\varepsilon} X'$, iff $D(X,V_1,V_2) \sim_{\varepsilon} D(X',V_1,V_2)$.*

We emphasize that in the above definition, V_1 and V_2 cannot contain any list $\mathcal{T}_H$, since HList and Var are disjoint. Hence, every time we use the equivalence $\sim_{V_1;V_2;\varepsilon}$, the variables $(\mathcal{T}_H)_{H\in\vec{H}}$ *are not given to the adversary.*

Replacing the value of a given variable by an independently uniformly drawn bitstring of the right length is an operation on constructible distributions that we often use. We introduce the notation $\nu x.X$ to denote distributions:

$$[v \leftarrow \mathcal{U}(l); m \leftarrow X : m.[x \mapsto v]] \text{ and } [m \leftarrow X; v \leftarrow \mathcal{U}(l) : m.[x \mapsto v]]$$

given any constructible distribution X and variable x of length l. We notice that if $X \in \mathfrak{const}\mathfrak{D}(\chi)$, then $\nu x.X$ belongs to the same set $\mathfrak{const}\mathfrak{D}(\chi)$.

2.2 Assertion Language

Our assertion language is defined by the following grammar, where ψ defines the set of atomic assertions:

$$\psi ::= \mathsf{Indis}(x; V_1; V_2; \varepsilon) \mid \mathsf{WS}(x; V_1; V_2; \varepsilon) \mid \mathsf{H}(H; e; \varepsilon)$$
$$\varphi ::= \mathsf{true} \mid \psi \mid \varphi \wedge \varphi$$

Intuitively, $\mathsf{Indis}(\nu x; V_1; V_2; \varepsilon)$ is satisfied by a distribution on configurations, if given values of variables in V_1 and images by f of values of variables in V_2, any polynomial adversary in η has probability to distinguish between the following two distributions is smaller than ε: first, the distribution resulting of computations performed using the original value of x as is in X, secondly, the distribution resulting from computations performed replacing everywhere the value of x by a random value of the same length as x. The assertion $\mathsf{WS}(x; V_1; V_2; \varepsilon)$ stands for Weak Secret and is satisfied by a distribution, if any adversary has a probability to compute the value of x smaller or equal to ε, when he is given the values of the variables in V_1 and the image by the one-way permutation of those in V_2. Lastly, $\mathsf{H}(H; e; \varepsilon)$ is satisfied when the probability that the value of e has been submitted to the hash oracle H is smaller or equal to ε. More precisely, the satisfaction relation $X \models \psi$ is defined as follows, where $\varepsilon : (k, \chi, t) \mapsto \varepsilon(k, \chi, t)$ a function ranging in $[0, 1]$:

— $X \models \mathsf{true}$.

— $X \models \mathsf{Indis}(x; V_1; V_2; \varepsilon)$ iff $X \sim_{V_1;V_2;\varepsilon} \nu x.X$

— $X \models \mathsf{WS}(x; V_1; V_2; \varepsilon)$ iff for any (k, t)-adversary $\mathcal{A}$, $\Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}((m.V_1), f(m.V_2)) = m.x] \leq \varepsilon(k, \chi, t)$.

— $X \models \mathsf{H}(H; e; \varepsilon)$ iff $\Pr[m \leftarrow X : m.e \in \mathsf{dom}(m.\mathcal{T}_H)] \leq \varepsilon(\chi)$ where $m.e$ is the evaluation of expression e in state m.

For the sake of readability, sets V_1 and V_2 appearing in the predicates are enumerated as collections of variables only separated by commas. For example, in $\mathsf{Indis}(x; y; t, z; 0)$, $V_1 = \{y\}$ and $V_2 = \{t, z\}$, and in $\mathsf{WS}(x; V, t; y; \varepsilon)$, $V_1 = V \cup \{t\}$ and $V_2 = \{y\}$.

Proposition 1 *Let cmd be an encryption using only commands described the grammar of Table 2.1 and computing a cipher in the variable out_e. Then this scheme is IND-CPA secure, if $\{\mathsf{true}\}\mathsf{cmd}\{\mathsf{Indis}(\nu\, out_e)\}$ is valid.*

Indeed, if $\{\mathsf{true}\}\mathsf{cmd}\{\mathsf{Indis}(\nu\mathrm{out}_e)\}$ holds then the encryption scheme is secure with respect to randomness of ciphertext. It is standard that randomness of ciphertext implies IND-CPA security.

2.3 Hoare Logic

Now that basic predicates and properties are set, we provide the rules that ensure their conservation or sound transformation when commands are applied. Rules fall into two categories: preservation rules that express how a property is modified after the command, and creation rules that state a property of a variable newly assigned. The rules are given along with some inequality restrictions, which are meant to be understood syntactically, *e.g.* $x \neq t, y$ means that x and t, y are not the same variable name. In all the rules we state, and as their definitions require, predicates take as arguments a variable in Var, a set V_1 included in $\mathsf{Var} \cup \{\sigma\}$ and a set V_2 is included in Var.

Generic preservation rules: Below, cmd is $x \leftarrow \mathcal{U}$ or of the form $x := e'$ with e' being either $w||y$, $w \oplus y$, $f(y)$ or $\alpha \oplus H(y)$.

Provided $x \notin V_1 \cup V_2$, and $z \neq x$:

(G1) $\{\mathsf{Indis}(z; V_1; V_2; \varepsilon)\}\ \mathsf{cmd}\ \{\mathsf{Indis}(z; V_1; V_2; \varepsilon')\}$

(G2) $\{\mathsf{WS}(z; V_1; V_2; \varepsilon)\}\ \mathsf{cmd}\ \{\mathsf{WS}(z; V_1; V_2; \varepsilon')\}$

where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa - \kappa_{\mathsf{cmd}}, t)$.

When $z \neq x$:

(G1') $\{\mathsf{Indis}(z; V_1; V_2; \varepsilon)\}\ \mathsf{cmd}\ \{\mathsf{Indis}(z; V_1, x; V_2; \varepsilon')\}$, if e' is constructible from $(V_1 \setminus \{z\}; V_2 \setminus \{z\})$,

(G2') $\{\mathsf{WS}(z; V_1; V_2; \varepsilon)\}\ \mathsf{cmd}\ \{\mathsf{WS}(z; V_1, x; V_2; \varepsilon')\}$, if e' is constructible from $(V_1; V_2)$,

where $\varepsilon'(k, \kappa, t) = \varepsilon(k + \kappa_{\mathsf{cmd}}, \kappa - \kappa_{\mathsf{cmd}}, t + \mathsf{T}_{\mathsf{cmd}}))$.

When $H' \neq H$:

(G3) $\{\mathsf{H}(H'; e[e'/x]; \varepsilon)\}\ \mathsf{cmd}\ \{\mathsf{H}(H'; e; \varepsilon')\}$, where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa - \kappa_{\mathsf{cmd}}, t)$.

(G3') $\{\mathsf{H}(H'; e; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{H}(H'; e; \varepsilon)\}$, if x does not appear in e.

Random assignment rules:

(R1) $\{\mathsf{true}\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}(x; \mathsf{Var} \cup \{\sigma\}, \emptyset; 0)\}$

(R2) $\{\mathsf{true}\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{H}(H; e; \frac{\kappa(H)}{2^l})\}$ if $x \in \mathsf{subvar}(e)$.

If $x \neq y$:

(R3) $\{\mathsf{Indis}(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}(y; V_1, x; V_2; \varepsilon)\}$

(R4) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{WS}(y; V_1, x; V_2; \varepsilon)\}$

Hash functions rules:

When $x \neq y$:

(H1) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon'')\}$, where $\varepsilon''(k, \kappa, t) = \varepsilon'(\kappa - \mathbf{1}^H) + k(H) * \varepsilon(k, \kappa - \mathbf{1}^H, t)$.

(H2) $\{\mathsf{WS}(y; V_1; V_2, y; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}(x; V_1, x; V_2, y; \varepsilon'')\}$ if $y \notin V_1$, where $\varepsilon''(k, \kappa, t) = \varepsilon'(\kappa - \mathbf{1}^H) + \varepsilon(k, \kappa - \mathbf{1}^H, t + k(H) * \mathsf{T}_f)$

When $x \neq y$, and $x \in \mathsf{subvar}(e)$:

(H3) $\{\mathsf{H}(H; y; \varepsilon)\}\ \ x := \alpha \oplus H(y)\ \ \{\mathsf{H}(H'; e; \varepsilon')\}$
where $\varepsilon'(\kappa) = \varepsilon(\kappa - \mathbf{1}^H) + \frac{\kappa(H)}{2^{\ell(H)}}$.

Provided that $x \neq y, z$:

(H4) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon_1) \wedge \mathsf{WS}(z; V_1; V_2; \varepsilon_2) \wedge \mathsf{H}(H; y; \varepsilon_3)\}\ \ x := \alpha \oplus H(y)$
$\{\mathsf{WS}(z; V_1, x; V_2; \varepsilon_4)\}$
where $\varepsilon_4(k, \kappa, t) = k(H) * \varepsilon_1(k, \kappa - \mathbf{1}^H, t) + \varepsilon_2(k, \kappa - \mathbf{1}^H, t) + \varepsilon_3(\kappa - \mathbf{1}^H)$.

Provided that $z \in \mathsf{subvar}(e) \wedge x \notin \mathsf{subvar}(e)$:

(H5) $\{\mathsf{H}(H; e; \varepsilon) \wedge \mathsf{WS}(z; y; \emptyset; \varepsilon')\}\ \ x := \alpha \oplus H(y)\ \ \{\mathsf{H}(H; e; \varepsilon'')\}$
where $\varepsilon''(\kappa) = \varepsilon(\kappa - \mathbf{1}_H) + \varepsilon'(0, \kappa - \mathbf{1}_H, 0)$.

If $x \neq y$:

(H6) $\{\mathsf{WS}(y; V_1; V_2, y; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ \ x := \alpha \oplus H(y)\ \ \{\mathsf{WS}(y; V_1, x; V_2, y; \varepsilon'')\}$
where $\varepsilon''(k, \kappa, t) = \varepsilon(k, \kappa - \mathbf{1}_H, t + (k(H) + 1) * \mathsf{T}_f) + \varepsilon'(\kappa - \mathbf{1}_H) + \varepsilon(k, \kappa - \mathbf{1}_H, t)$.

When $x \neq y, z$:

(H7) $\{\mathsf{Indis}(z; V_1, z; V_2; \varepsilon_1) \wedge \mathsf{WS}(y; V_1, z; V_2; \varepsilon_2) \wedge \mathsf{H}(H; y; \varepsilon_3)\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}(z; V_1, z, x; V_2; \varepsilon_4)\}$
where $\varepsilon_4(k, \kappa, t) = \varepsilon_1(k, \kappa - \mathbf{1}_H, t) + 2.k(H) * \varepsilon_2(k, \kappa - \mathbf{1}_H, t) + 2.\varepsilon_3(\kappa - \mathbf{1}_H)$.

One-way function rules:

If $x, y \notin V_1 \cup V_2$:

(P1) $\{\mathsf{Indis}(y; V_1; V_2, y; \varepsilon)\}\ \ x := f(y)\ \ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon)\}$

If $y \notin V_1 \cup \{x\}$:

(P2) $\{\mathsf{Indis}(y; V_1; V_2, y; \varepsilon)\}\ \ x := f(y)\ \ \{\mathsf{WS}(y; V_1, x; V_2, y; \varepsilon'\})\}$
where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa, t + \mathsf{T}_f) + OW(t)$.

If $z \neq x, y$:

(P3) $\{\mathsf{Indis}(z; V_1, z; V_2, y; \varepsilon)\}\ \ x := f(y)\ \ \{\mathsf{Indis}(z; V_1, z, x; V_2, y; \varepsilon)\}$.

If $z \neq x, y$

(P4) $\{\mathsf{WS}(z; V_1; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1; V_2, y, z; \varepsilon')\}$ $x := f(y)$ $\{\mathsf{WS}(z; V_1, x; V_2, y; \varepsilon'')\}$ where $\varepsilon''(k, \kappa, t) = \varepsilon(k, \kappa, t + \mathsf{T}_f) + \varepsilon'(k, \kappa, t + \mathsf{T}_f)$.

Exclusive or rules:

If $y \notin V_1 \cup V_2$, $y \neq x, z$:

(X1) $\{\mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon)\}$ $x := y \oplus z$ $\{\mathsf{Indis}(x; V_1, x, z; V_2; \varepsilon)\}$

(X2) $\{\mathsf{Indis}(w; V_1, y, z; V_2; \varepsilon)\}$ $x := y \oplus z$ $\{\mathsf{Indis}(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x, y, z$

(X3) $\{\mathsf{WS}(w; V_1, y, z; V_2; \varepsilon)\}$ $x := y \oplus z$ $\{\mathsf{WS}(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x$

Concatenation rules:

If $x \notin V_1 \cup V_2$:

(C1) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon)\}$ $x := y||z$ $\{\mathsf{WS}(x; V_1; V_2; \varepsilon)\}$

A dual rule applies for z.

If $y, z \notin V_1 \cup V_2 \cup \{x\}$:

(C2) $\{\mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon) \wedge \mathsf{Indis}(z; V_1, y, z; V_2; \varepsilon')\}$ $x := y||z$ $\{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon + \varepsilon')\}$,

(C3) $\{\mathsf{Indis}(w; V_1, y, z; V_2; \varepsilon)\}$ $x := y||z$ $\{\mathsf{Indis}(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x, y, z$,

(C4) $\{\mathsf{WS}(w; V_1, y, z; V_2; \varepsilon)\}$ $x := y||z$ $\{\mathsf{WS}(w; V_1, y, z, x; V_2; \varepsilon)\}$, if $w \neq x$,

Consequence and sequential composition rules:

(Csq) if $\varphi_0 \Rightarrow \varphi_1$, $\{\varphi_1\}$ cmd $\{\varphi_2\}$ and $\varphi_2 \Rightarrow \varphi_3$ then $\{\varphi_0\}$ cmd $\{\varphi_3\}$

(Seq) if $\{\varphi_0\}$ cmd_1 $\{\varphi_1\}$ and $\{\varphi_1\}$ cmd_2 $\{\varphi_2\}$, then $\{\varphi_0\}$ $\mathsf{cmd}_1; \mathsf{cmd}_2$ $\{\varphi_2\}$

(Conj) if $\{\varphi_0\}$ cmd $\{\varphi_1\}$ and $\{\varphi_0\}$ cmd $\{\varphi_1\}$, then $\{\varphi_0\}$ cmd $\{\varphi_1 \wedge \varphi_2\}$

2.4 Example of Application

We illustrate our proposition with Bellare & Rogaway's generic construction [BR93], which can be shortly described as $f(r)||(q \oplus G(r))||H(q||r)$, where q is a plaintext to be ciphered. A description of the algorithm in our framework, under the form of an oracle declaration, is the following:

$$BR(q, \mathrm{A}) : \textbf{var } \{r, g, b, c, d, s, t\};$$
$$(r \leftarrow \mathcal{U}(l); b := f(r); g := G(r); c := q \oplus g; s := q||r; d := H(s); t := b||c; \mathrm{A} := t||d)$$

We give a proof constructed using our Hoare Logic.

$V = \{q, \mathrm{A}, \sigma, r, g, b, c, d, s, t\}$:

$r \leftarrow \mathcal{U}(l)$	$\mathsf{Indis}(r; V; \emptyset; 0) \wedge \mathsf{H}(G; r; \frac{\chi_0(G)}{2^l}) \wedge \mathsf{H}(H; q\|\|r; \frac{\chi_0(H)}{2^l})$	$(R1)(R2)$
$b := f(r)$	$\mathsf{Indis}(b; V \smallsetminus \{r\}; \emptyset; 0) \wedge \mathsf{WS}(r; V \smallsetminus \{r\}; r; OW(t))$ $\wedge \mathsf{H}(G; r; \frac{\chi_0(G)}{2^l}) \wedge \mathsf{H}(H; q\|\|r; \frac{\chi_0(H)}{2^l})$	$(P1)(P2)$
$g := G(r)$	$\mathsf{Indis}(b; V \smallsetminus \{r\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(g; V \smallsetminus \{r\}; r; \varepsilon) \wedge$ $\mathsf{WS}(r; V \smallsetminus \{r\}; r; \varepsilon/2) \wedge \mathsf{H}(H; q\|\|r; \frac{\chi_0(H)}{2^l})$	$(H1)(H4)(G3)$
$c := q \oplus g$	$\mathsf{Indis}(b; V \smallsetminus \{r\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(c; V \smallsetminus \{g, r\}; r; \varepsilon) \wedge$ $\mathsf{WS}(r; V \smallsetminus \{r\}; r; \varepsilon/2) \wedge \mathsf{H}(H; q\|\|r; \frac{\chi_0(H)}{2^l})$	$(X1)(X2)(X3)(G3)$
$s := q\|\|r$	$\mathsf{Indis}(b; V \smallsetminus \{r, s\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(c; V \smallsetminus \{g, r, s\}; r; \varepsilon) \wedge$ $\mathsf{WS}(s; V \smallsetminus \{r, s\}; r; \varepsilon/2) \wedge \mathsf{H}(H; s; \frac{\chi_0(H)}{2^l})$	$(C1)(G1)(G3)$
$d := H(s)$	$\mathsf{Indis}(b; V \smallsetminus \{r, s\}; \emptyset; \varepsilon') \wedge \mathsf{Indis}(c; V \smallsetminus \{r, g, s\}; r; \varepsilon') \wedge$ $\mathsf{Indis}(d; V \smallsetminus \{r, s\}; r; \varepsilon'')$	$(H1)(H7)$
$t := b\|\|c$	$\mathsf{Indis}(t; V \smallsetminus \{b, c, r, g, s\}; \emptyset; 2.\varepsilon')$ $\mathsf{Indis}(d; V \smallsetminus \{b, c, r, s\}; r; \varepsilon'')$	$(C2)(C3)$
$\mathrm{A} := t\|\|d$	$\mathsf{Indis}(\mathrm{A}; q, \mathrm{A}, \sigma; \emptyset; 2.\varepsilon' + \varepsilon'')$	$(C2)$

where $\varepsilon(k, \chi, t) = 2.k(G)*OW(t)+2.\frac{\chi_0(G)}{2^l}$, $\varepsilon'(k, \chi, t) = (1+k(H))\varepsilon(k, \chi-\mathbf{1}_H, t)+2.\frac{\chi_0(H)}{2^l} = 2.(1+k(H))(k(G)*OW(t)+\frac{\chi_0(G)}{2^l})+2\frac{\chi_0(H)}{2^l}$, $\varepsilon''(k, \chi, t) = \frac{\chi_0(H)}{2^l}+k(H)*(k(G)*OW(t)+\frac{\chi_0(G)}{2^l})$.

2.5 Conclusion

We have developed rules for a command $x := f(y)$ where f is a one-way permutation. However, bijectivity is a strong requirement for a cryptographic primitive. In [5], we relax this assumption and provide other invariants and revisited our Hoare logic. Moreover in order to capture cases when f is not a one-way function but an injective partially trapdoor one-way function [Poi00]. Then we reformulate rules for $x := f(y)$ into rules for $z := f(x||y)$ so that they take into account the new properties of f. For instance using these extensions, we are able to prove Pointcheval's transformer [Poi00],

Chapter 3

Symmetric Encryption Modes

Utiliser un chiffrement symmétrique comme AES pour transmettre une grande qunatitié de données est beaucoup plus efficace qu'un chiffrement à clef publique. Par contre il est nécessaire d'utiliser un mode de chiffrement IND-CPA. Afin d'étudier les modes de chiffrement symmétrique, nous donnons une logique de Hoare permettant d'analyser la sécurité concréte de ces primitives. Pour vérifier les modes utilisants un compteur, nous introduisons des prédicats permettant de suivre les dépedences entres ces variables. Nous introduisons aussi la commande FOR afin de pouvoir écrire les modes de chiffrement en fonction d'un nombre d'itérations arbitraire. Il faut alors trouver l'invariant de boucle lors de l'analyse d'un programme représentant un mode de chiffremet. Pour cela nous proposons une méthode automatique de génération d'invariants pour les boucles qui fonctionne pour l'ensemble des modes testés.

Contents

Block ciphers are among the most basic building blocks in cryptography. They can be used to construct primitives as varied as message authentication codes, hash functions and, their main application, symmetric encryption. Block ciphers are deterministic, and have fixed-size input and output, so protocols, called modes of operation, are required to encrypt messages of arbitrary length. The security of these modes of operation is then proven by reduction from the security of the mode of operation to some security property of the block cipher.

While the security of early modes of operation, such as Cipher Block Chaining (CBC), Cipher Feedback mode (CFB), Output Feedback (OFB), and Counter mode (CTR), can be proven relatively easily, the same cannot always be said for more recent modes of operation. Many new modes of operation were designed in the past decade (IACBC, IAPM [Jut01], XCB [MF07], TMAC [IK03c, KI03], HCTR [CN08], HCH [CS08], EMU [HR04], EMU* [Hal04], PEP [CS06], OMAC [IK03a, IK03b], TET [Hal07], CMC [HR03], GCM [MV04], EAX [BRW04], XEX [Rog04], TAE, TCH, TBC [LRW02a, WFW05] to name only a few), which often offer security properties that early modes did not possess. However, these additional properties often come at the cost of an increased complexity of the mode of operation, which, in turn, increase the complexity of the proof of security.

Contributions: We verify the security of modes of operation using an Hoare Logic. For example we are able to verify the security of Propagating Cipher-Block Chaining (PCBC) – an encryption mode that was introduced for Kerberos version 4. We use only few predicates: one that states that the value of a variable is indistinguishable from a random value, one that states that the block cipher has never been computed at the value of a variable, one that states if a counter have been sent to the block cipher and one that keeps track of the most recent value of a counter. In order to produce tight concrete security bounds we introduce a predicate "BAD" that collect only once the probability of collision of each predicate. We also include in our grammar the command for in order to analyze modes described as program using such a loop. For sake of simplicity we give a method for generating the loop invariant only for predicates without any security parameter.

Outline: Section 3.1 introduces some notation that will be used, as well as the definitions of concepts. In Section 3.2, we define a generic encryption mode and its semantics. Then in Section 3.3, we present our assertion language, invariants. In Section 3.4, we give the set of rules that is used to propagate those invariants and prove our main result. In Section 3.5, we present our method for generating loop invariants and two examples. We conclude in Section 3.6.

3.1 Background

For a set V and an element x, we write V, x as a shorthand for $V \cup \{x\}$ and $V - x$ as a shorthand for $V \setminus \{x\}$. We say that a function $f : \mathbb{N} \to \mathbb{R}$ is negligible in η iff for any polynomial p, there exists a number η_0 such that for all $\eta \geq \eta_0$, $f(\eta) \leq \frac{1}{p(\eta)}$.

A block cipher is a family of permutations $\mathcal{E}_k : \{0,1\}^\eta \to \{0,1\}^\eta$ indexed with a key

$k \in \{0,1\}^k$. We generally omit the key used every time to simplify the notation, but it is understood that a key was selected at random and remains the same throughout.

We assume that the block cipher is a pseudo-random function. While block ciphers are really families of permutations, it is well known that pseudo-random permutations are indistinguishable from pseudo-random functions if the block size is large enough [BDJR97]. This is a standard assumption for proving the security of block-cipher-based schemes.

Definition 5 (Pseudo-Random Function) *Let $P : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a family of functions and let $\mathcal{A}$ be an algorithm that takes an oracle and returns a bit. The* prf-advantage *of $\mathcal{A}$ is defined as follows.*

$$\mathsf{Adv}^{prf}_{\mathcal{A},P} = Pr[K \xleftarrow{\$} \{0,1\}^k; \mathcal{A}^{P(K,\cdot)} = 1] - Pr[R \xleftarrow{\$} \Phi_n; \mathcal{A}^{R(\cdot)} = 1]$$

where Φ_n is the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$.

Definition 6 (Symmetric Encryption) *A* symmetric encryption scheme *is a triple of probabilistic polynomial-time algorithms $(\mathbf{K}, \mathcal{E}, \mathcal{D})$ where*
— the key generation algorithm $\mathbf{K}$ takes a security parameter η and outputs a key k

— the encryption algorithm $\mathcal{E}$ takes a key and a message m, and outputs a ciphertext c, and

— the decryption algorithm $\mathcal{D}$ takes a key and a ciphertext, and outputs a message.
These algorithms must satisfy the standard soundness requirement that for any key k generated by $\mathbf{K}$ and any message m, we have $\mathcal{D}(k, \mathcal{E}(k,m)) = m$.

We define the function LR, which takes two strings of equal lengths, as follows:

$$\mathsf{LR}(\mu_0, \mu_1, b) = \begin{cases} \mu_0 \text{ if } b = 0 \\ \mu_1 \text{ if } b = 1, \end{cases}$$

We give the formal definition of security for symmetric encryption below. In the definition, the attacking adversary has access to an oracle which, given two equal length messages, either always encrypts the first message, or always encrypts the second one. The objective of the adversary is then to determine which of the two message is encrypted by his oracle, and the encryption scheme is considered secure if no polynomial-time adversary is able to reliably distinguish between the two scenarios.

Definition 7 (LoR-CPA security [BDJR97]) *Let $\mathcal{S} = (\mathbf{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. An* LoR-CPA adversary *is a probabilistic algorithm $\mathcal{A}$ that has access to an oracle $\mathcal{E}(k, \mathsf{LR}(\cdot,\cdot,b))$, takes a security parameter η and outputs a bit b'. Consider the following experiment:*

```
Experiment $\mathbf{Exp}^{\mathsf{LoR\text{-}CPA}-b}_{\mathcal{S}}(\mathcal{A}, \eta)$:
    $k \xleftarrow{\$} \mathbf{K}(\eta)$
    $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{E}(k,\mathsf{LR}(\cdot,\cdot,b))}(\eta)$
    return $b'$
```

We define the LoR-CPA *advantage of* $\mathcal{A}$ *as follows:*

$$\mathit{Adv}_{\mathcal{S}}^{\textsf{LoR-CPA}}(\mathcal{A},\eta) = \mathit{Pr}[\mathbf{Exp}_{\mathcal{S}}^{\textsf{LoR-CPA}-1}(\mathcal{A},\eta)=1] - \mathit{Pr}[\mathbf{Exp}_{\mathcal{S}}^{\textsf{LoR-CPA}-0}(\mathcal{A},\eta)=1].$$

We say that a symmetric encryption scheme is LoR-CPA-secure *if* $\mathit{Adv}_{\mathcal{S}}^{\textsf{LoR-CPA}}(\mathcal{A},\eta)$ *is negligible for any* LoR-CPA *adversary* $\mathcal{A}$ *that runs in time polynomial in* η*.*

3.2 Programming language

We first introduce a grammar that describes the programming language used to generate encryption modes. We then formally define generic encryption modes, and present the semantics of our programming language.

3.2.1 — Grammar

We introduce our language for defining a generic encryption mode. The commands are given by the grammar of Figure 1, where:

— $x \xleftarrow{\$} \mathcal{U}(l)$ the assignment to x of a value sampled uniformly at random from $\{0,1\}^l$.

— $x := y$ denotes the assignment to x of the value of y.

— $x := y \oplus z$ denotes the assignment to x of the xor of the values of y and z.

— $x := y\|z$ denotes the assignment to x of the concatenation of the values of y and z.

— $x := y + 1$ denotes the assignment to x of the incrementation by one of the value or y. The operation is carried modulo 2^η.

— $x := \mathcal{E}(y)$ denotes the assignment to x of the value obtained by computing the function $\mathcal{E}$ to the value of y.

— $\textsf{for } k = p \textsf{ to } q \textsf{ do: } [c_k]$ denotes the successive execution of $c_p, c_{p+1}, \ldots, c_q$.

— $c_1; c_2$ is the sequential composition of c_1 and c_2.

$$\begin{array}{lll} \text{c} & ::= & x \xleftarrow{\$} \mathcal{U}(l) \mid x := y \mid x := \mathcal{E}(y) \mid x := y \oplus z \mid x := y\|z \\ & & \mid x := y + 1 \mid \textsf{for } k = p \textsf{ to } q \textsf{ do: } [c_k] \mid c_1; c_2 \end{array}$$

Figure 1 – Language grammar

To simplify our analysis, we assume that each variable is assigned a value at most once by the program. It should be clear that any program can be transformed into an equivalent program with this property.

3.2.2 — Generic Encryption Mode

We formally define a mode of encryption.

Definition 8 (Generic Encryption Mode) *A generic encryption mode M is represented by $M_{\mathcal{E}}(m_1\|\ldots\|m_n, c_n)$: **var** $\vec{x_n}$; cmd_n, where $\mathcal{E}$ is a block cipher, each m_j is a message blocks of size η according to the input length of the block cipher $\mathcal{E}$, c_n is the ciphertext output at the end of the execution of the program, cmd_n is a program built using the grammar described in Figure 1, and $\vec{x_n}$ is the set of variables used in cmd_n.*

We also assume that all programs have length polynomial in the security parameter η. We denote by Var the set containing all the variables in the program, and by **var** the set of variables in the program other than the m_i's and c_n.

In Figure 2, we present the standard encryption modes cipher block chaining ($CBC_{\mathcal{E}}$) [EMST76] and counter mode ($CTR_{\mathcal{E}}$).

```
$CBC_{\mathcal{E}}(m_1\|m_2\|\ldots\|m_n, c_n)$ :
var $i, y_1, \ldots, y_n, z_0, \ldots, z_n,$
    $c_0, \ldots, c_{n-1}$;
$z_0 \xleftarrow{\$} \mathcal{U}$; $c_0 := z_0$;
for $i = 1$ to $n$ do:
    [$y_i := z_{i-1} \oplus m_i$;
     $z_i := \mathcal{E}(y_i)$;
     $c_i = c_{i-1}\|z_i$];
```

```
$CTR_{\mathcal{E}}(m_1\|m_2\|\ldots\|m_n, c_n)$
var $i, y_1, z_1, \ldots, y_n, z_n, ctr_0, \ldots, ctr_n,$
    $c_0, \ldots, c_{n-1}$
$ctr_0 \xleftarrow{\$} \mathcal{U}$; $c_0 := ctr_0$;
for $i = 1$ to $n$ do:
    [$ctr_i := ctr_{i-1} + 1$;
     $y_i := \mathcal{E}(ctr_i)$;
     $z_i := y_i \oplus m_i$;
     $c_i := c_{i-1}\|z_i$];
```

Figure 2 – Description of $CBC_{\mathcal{E}}$ and $CTR_{\mathcal{E}}$

3.2.3 — Semantics

We view each command as a function that takes as input a configuration and outputs a distribution on configurations. A *configuration* is a tuple of the form $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$ where S is a state, σ is a bitstring, $\mathcal{E}$ is a block cipher, $\mathcal{T}$ is a set of arrays, $\mathcal{Q}$ is a set of sets, and $\mathcal{L}$ is a multiset of pairs (s, x) where x is a bitstring and v is a variable. If γ is a configuration, we write by S_γ, σ_γ, $\mathcal{E}_\gamma$, $\mathcal{T}_\gamma$, $\mathcal{Q}_\gamma$ and $\mathcal{L}_\gamma$ the state, bitstring, block cipher and sets contained in γ

A *state* is a function $S : \mathcal{V} \to \{0,1\}*$ that assigns bitstrings to variables. The domain $\mathcal{V}$ of S must contain at least all the variables Var of the program.

The bitstring σ contains information written by the algorithm that created the configuration.

The lists $\mathcal{T}$ and $\mathcal{Q}$ are used to keep track of the variables that contain values used as counters, and variables that contain values that are derived from a counter respectively. Whenever the program samples a value at random or computes the block cipher at a new point and assigns the resulting value to a variable x, a new array $\mathcal{T}_x$ with $\mathcal{T}_x[0] = x$ and, initially, $\mathcal{T}_x[i] = \bot$ for $i > 0$ is added to $\mathcal{T}$ and a new set $\mathcal{Q}_x$, initially empty, is added to $\mathcal{Q}$. For each integer $i \geq 0$, $\mathcal{T}_x[i]$ contains the first variable whose value is known to be $S(x) + i$ (or the symbol $\perp$ if no such variable is known), and $\mathcal{Q}_x$ contains the set of variables whose value is derived from the value of a variable in $\{v \in \mathsf{Var} \mid \exists i \geq 0, v = \mathcal{T}_x\}$. Counters are often used to create a one-time pad by successively computing the block cipher on each value of the counter, so computing the block cipher on variables in $\mathcal{T}$ is considered 'safe'. On the other hand, computing the block cipher on the values of the variables in $\mathcal{Q}$ is considered 'unsafe' and will invalidate the corresponding counter. We note that since we assumed that each variable is assigned a value at most once by the program, a variable will always appear at most once in $\mathcal{T}$, and the variables appearing in $\mathcal{T}$ cannot be in $\mathcal{Q}$. We denote by $\mathcal{TQ}(x)$ the set $\{v : \mathcal{T}_x[i] = v\} \cup \mathcal{Q}_x$ of all variables that are either counters based on x, or variables containing values derived from a counter based on x.

The multiset $\mathcal{L}$ contains pairs (s, x) where s is a bitstring and x is a variable. A pair (s, x) is added to $\mathcal{L}$ every time the block cipher is computed on a value s contained in a variable x - that is, every time a command of the form $y := \mathcal{E}(x)$ is executed in the program. If the computation of the block cipher occurs before the execution of the program (for example, by the algorithm that creates the configuration, see below), then a pair of the form $(s, \perp)$ is added to $\mathcal{L}$. We write $\mathcal{L}.dom$ and $\mathcal{L}.res$ to denote the multisets obtained by projecting each pair in $\mathcal{L}$ to its first element and second element respectively.

Since commands never alter σ or $\mathcal{E}$, we can, without ambiguity, write

$$(S', \mathcal{T}, \mathcal{Q}, \mathcal{L}) \overset{\$}{\leftarrow} [\![c]\!](S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$$

instead of

$$(S', \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}) \overset{\$}{\leftarrow} [\![c]\!](S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$$

Similarly, we removed σ and $\mathcal{E}$ from the configurations in Table 3.1, but it is understood that they are implied in each configuration. Let Γ denote the set of configurations and $\textsc{Dist}(\Gamma)$ the set of distributions on these configurations. The semantics is given in Table 3.1 where, $\delta(x)$ denotes the Dirac measure, i.e. $\mathsf{Pr}[x] = 1$, $S\{x \mapsto v\}$ denotes the function such that $S\{x \mapsto v\}(y) = S(y)$ for all variables y in the domain of S except for the variable x, for which we have $S\{x \mapsto v\}(x) = v$ and $g \circ f$ denotes the composition of functions f and g. We also write $\mathcal{Q}_+(x, V)$ for a variable x and a set of variables V to denote the set

$$\mathcal{Q}_+(x, V) = \{\mathcal{Q}_y \cup \{x\} : \mathcal{TQ}(y) \cap V \neq \emptyset\} \cup \{\mathcal{Q}_y : \forall v \in \mathcal{V}, \mathcal{TQ}(v) \cap V = \emptyset\}.$$

This notation is used to when we want to add the variable x to all the sets $\mathcal{Q}_y$ such that one of the variables in V is either in $\mathcal{T}_y$ or in $\mathcal{Q}_y$. This is done, for example, to update $\mathcal{Q}$ after the execution of $x := y \oplus z$: obviously, x is derived from both y and z, so we want to add x to all the sets Q_w such that $y \in \mathcal{TQ}(w)$ or $z \in \mathcal{TQ}(w)$.

Notice that the semantic function of commands can be lifted in the usual way to a function from $\textsc{Dist}(\Gamma)$ to $\textsc{Dist}(\Gamma)$. That is, let $\phi : \Gamma \to \textsc{Dist}(\Gamma)$ be a function. Then, ϕ defines a unique function $\phi^* : \textsc{Dist}(\Gamma) \to \textsc{Dist}(\Gamma)$ obtained by point-wise application of ϕ. By abuse of

$$
\begin{array}{l}
[\![x \xleftarrow{\$} \mathcal{U}(l)]\!](S,\mathcal{T},\mathcal{Q},\mathcal{L}) = [u \xleftarrow{\$} \mathcal{U}(l) : (S\{x \mapsto u\}, \mathcal{T} \cup \{\mathcal{T}_x\}, \mathcal{Q} \cup \{\mathcal{Q}_x\}, \mathcal{L})] \\
[\![x := \mathcal{E}(y)]\!](S,\mathcal{T},\mathcal{Q},\mathcal{L}) = \\
\quad \begin{cases} \delta(S\{x \mapsto \mathcal{E}(S(y))\}, \mathcal{T} \cup \{\mathcal{T}_x\}, \mathcal{Q} \cup \{\mathcal{Q}_x\}, \mathcal{L} \cup \{(S(y),y)\}) \text{if } S(y) \notin \mathcal{L}.dom \\ \delta(S\{x \mapsto \mathcal{E}(S(y))\}, \mathcal{T}, \mathcal{Q}, \mathcal{L} \cup \{(S(y),y)\}) \text{ otherwise} \end{cases} \\
[\![x := y \oplus z]\!](S,\mathcal{T},\mathcal{Q},\mathcal{L}) = \delta(S\{x \mapsto S(y) \oplus S(z)\}, \mathcal{T}, \mathcal{Q}_+(x,\{y,z\}), \mathcal{L}) \\
[\![x := y||z]\!](S,\mathcal{T},\mathcal{Q},\mathcal{L}) = \delta(S\{x \mapsto S(y)||S(z)\}, \mathcal{T}, \mathcal{Q}_+(x,\{y,z\}), \mathcal{L}) \\
[\![x := y+1]\!](S,\mathcal{T},\mathcal{Q},\mathcal{L}) = \\
\quad \begin{cases} \delta(S\{x \mapsto S(y)+1, \mathcal{T}\{\mathcal{T}_z[i+1] \mapsto x\}, \mathcal{Q}, \mathcal{L}) \text{ if } \exists z,\ y = \mathcal{T}_z[i] \wedge \mathcal{T}_z[i+1] = \perp \\ \delta(S\{x \mapsto S(y)+1, \mathcal{T}, \mathcal{Q}_+(x,\{y\}), \mathcal{L}) \text{ otherwise} \end{cases} \\
[\![\mathsf{for}\ k = p\ \mathsf{to}\ q\ \mathsf{do}\text{: } [c_k]]\!] = [\![c_q]\!] \circ [\![c_{q-1}]\!] \circ \ldots \circ [\![p_i]\!] \\
[\![c_1; c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]
\end{array}
$$

Table 3.1 – The semantics of the programming language

notation we also denote the lifted semantics by $[\![\mathsf{cmd}]\!]$.

We consider distributions of configurations that can be constructed in polynomial time. We denote their set by $\textsc{Dist}(\Gamma, \mathcal{F})$, where $\mathcal{F}$ is a family of block ciphers, and is defined as the set of distributions of the form:

$$[\mathcal{E} \xleftarrow{\$} \mathcal{F}(1^\eta); S, \sigma \xleftarrow{\$} \mathcal{A}^{M_{\mathcal{E}}(\cdot)}(1^\eta) : (S, \sigma, \mathcal{E}, \emptyset, \emptyset, \mathcal{L})]$$

where $\mathcal{A}$ is a polynomial-time algorithm with oracle access to $M_{\mathcal{E}}(\cdot)$, the encryption function corresponding to the generic encryption mode $M_{\mathcal{E}}(m_1|\ldots\|m_n, c_n)$: **var** $\vec{x_n}; \mathsf{cmd}_n$. The adversary can set the value of all the variables of S and the value of σ, and all the block cipher computations that are required to answer its calls to the oracle $M_{\mathcal{E}}(\cdot)$ are recorded in $\mathcal{L}$ as pairs of the form $(s, \perp)$. The other sets, $\mathcal{T}$ and $\mathcal{Q}$, are initially empty.

3.3 Assertion Language

Before we introduce the invariants that are the basis of our Hoare logic, we describe an important property of a configuration.

Definition 9 *We say that a configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$ is* bad *if there exists two elements (s, v) and (s', v') in $\mathcal{L}$ with $s = s'$ and at least one of v or v' is not $\perp$.*

Intuitively, a configuration becomes bad after the execution of a command of the form $x := \mathcal{E}(y)$ such that the value contained in the variable y was already in $\mathcal{L}.dom$.

In the following, for any set $V \subseteq \mathsf{Var}$ and configuration $\gamma = (S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$, we denote by $S(V)$ the multiset resulting from the application of S on each variable in V. We write $BAD(\gamma)$ to denote the event that configuration γ is bad. We also use $S(V, \ell_{\mathcal{E}})$ as a shorthand for $S(V) \cup \mathcal{L}.dom$ and Var^* as a shorthand for $\mathsf{Var} \cup \{\ell_{\mathcal{E}}\}$.

Our Hoare Logic is based on the following invariants:

$$\varphi ::= \mathsf{true} \mid \varphi \wedge \varphi \mid \psi$$
$$\psi ::= \mathsf{Bad}(\epsilon) \mid \mathsf{Indis}(x; W; \epsilon) \mid \mathsf{Icounter}(x; y; V; V'; V''; \epsilon) \mid \mathsf{C}(x; y; V; V'; \epsilon) \mid \mathsf{E}(x; \epsilon)$$

where $x, y \in \mathsf{Var}$, $V, V', V'' \subseteq \mathsf{Var}$ and $W \subseteq \mathsf{Var} \cup \{\ell_{\mathcal{E}}\}$. These invariants are properties of distribution of configurations that, intuitively, have the following meaning:

$\mathsf{Bad}(\epsilon)$: means that the probability that a configuration is bad is at most ϵ

$\mathsf{Indis}(x; W; \epsilon)$: means that the probability that a configuration is not bad, and that an algorithm can distinguish whether he is given results of computations performed using the value of x or a random value, when he is given the values $S(W)$ is at most ϵ.

$\mathsf{Icounter}(x; y; V; V'; V''; \epsilon)$: means that the probability that the conjunction of the following events is false is at most ϵ

— the configuration is not bad

— x contains the most recently computed value of a counter that started with random value y

— V contains all the variables with previous values of the counter

— V' contains all the variables not containing values that are dependent on values of the counter, but that are not previous values of the counter

— V'' contains all the variables with previous values of the counter on which the block cipher has has been computed

$\mathsf{C}(x; y; V; V'; V''; \epsilon)$: means the probability that the conjunction of the the following events is false is at most ϵ:

— the configuration is not bad

— x contains the value of a counter that started with random value y

— V contains all the variables with previous values of the counter

— V' contains all the variables not containing values that are dependent on values of the counter, but that are not previous values of the counter

— no algorithm can distinguish whether he is given results of computations performed using the value x or a random value when he is given the values $\{s : (s, v) \in \mathcal{L} \wedge v \notin V''\}$

$\mathsf{E}(x; \epsilon)$: means the probability that the image of the value of x through the block cipher has already be calculated is at most ϵ.

More formally, for each invariant ψ, we define that a distribution X satisfies ψ, denoted $X \models \psi$ as follows, where $x, y \in \mathsf{Var}$, $V, V', V'' \subseteq \mathsf{Var}$ and $W \subseteq \mathsf{Var} \cup \{\ell_{\mathcal{E}}\}$:

— $X \models \mathsf{true}$.

— $X \models \varphi \wedge \varphi'$ iff $X \models \varphi$ and $X \models \varphi'$.

— $X \models \mathsf{Bad}(\epsilon)$ iff $\mathsf{Pr}[\gamma \xleftarrow{\$} X : BAD(\gamma)] \leq \epsilon$

— $X \models \mathsf{Indis}(x; W; \epsilon)$ (where $\mathcal{L} \notin V$) iff for every polynomial-time algorithm $\mathcal{A}$,

$$\begin{aligned}&\big|\ \mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X : \mathcal{A}(S_\gamma(x), S_\gamma(W - x)) = 1 \wedge \neg BAD(\gamma)] - \\ &\mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X; u \overset{\$}{\leftarrow} \mathcal{U}(length(x)) : \mathcal{A}(u, S_\gamma(W - x)) = 1 \wedge \neg BAD(\gamma)]\ \big| \leq \epsilon.\end{aligned}$$

— $X \models \mathsf{Icounter}(x; y; V; V'; V''; \epsilon)$ iff for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$ that has non-zero probability in X and that is not bad, $x = \mathcal{T}_y[i]$ for some i, $V = \mathcal{T}(x)$, $V' = \mathcal{Q}_y$, $V'' = V \cap \mathcal{L}.res$; and for every polynomial-time algorithm $\mathcal{A}$,

$$\begin{aligned}&\big|\ \mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X : \mathcal{A}(S_\gamma(x), W) = 1 \wedge \neg BAD(\gamma)] - \\ &\mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X; u \overset{\$}{\leftarrow} \mathcal{U}(length(x)) : \mathcal{A}(u, W) = 1 \wedge \neg BAD(\gamma)]\ \big| \leq \epsilon\end{aligned}$$

where the set W is equal to $S_\gamma(\mathsf{Var} \setminus (V \cup V' \cup \{x\})) \cup \{s : (s, v) \in \mathcal{L}_\gamma \wedge v \notin V''\}$.

— $X \models \mathsf{C}(x; y; V; V'; V''; \epsilon)$ iff for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L})$ that has non-zero probability in X and that is not bad, $x = \mathcal{T}_y[i]$ for some i, $V = \mathcal{T}(x)$, $V' = \mathcal{Q}_y$, $V'' = V \cap \mathcal{L}.res$; and for every polynomial-time algorithm $\mathcal{A}$,

$$\begin{aligned}&\big|\ \mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X : \mathcal{A}(S_\gamma(x), W) = 1 \wedge \neg BAD(\gamma)] - \\ &\mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X; u \overset{\$}{\leftarrow} \mathcal{U}(length(x)) : \mathcal{A}(u, W) = 1 \wedge \neg BAD(\gamma)]\ \big| \leq \epsilon\end{aligned}$$

where the set W is equal to $\{s : (s, v) \in \mathcal{L}_\gamma \wedge v \notin V''\}$.

— $X \models \mathsf{E}(x; \epsilon)$ iff $\mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X : S_\gamma(x) \in \mathcal{L}_\gamma.dom \wedge \neg BAD(\gamma)] \leq \epsilon$.

3.3.1 — Properties of Invariants and Useful Lemmas

We present a series of result that show how each of our invariants relate with the others. We also prove a few useful lemmas that will be used repeatedly in the proofs of the rules of our Hoare logic.

We present three results that should be self-evident, since all they state is that strong invariants imply weaker ones which is a direct consequence of the definitions.

Lemma 2 *For any distribution $X \in \textsc{Dist}(\Gamma, \mathcal{F})$, any variable $x \in \mathsf{Var}$ and any sets $V' \subseteq V \subseteq \mathsf{Var}$ and $V_0, V_1, V_2 \subseteq \mathsf{Var}$,*

$$\begin{aligned}&X \models \mathit{Indis}(x; V; \epsilon) \Rightarrow X \models \mathit{Indis}(x; V'; \epsilon) \\ &X \models \mathit{Icounter}(x; y; V_0; V_1; V_2; \epsilon) \Rightarrow X \models \mathit{Indis}(x; \mathsf{Var} \setminus (V \cup V'); \epsilon).\end{aligned}$$

The following proves the intuitive observation that if the value of a variable x is indistinguishable from a random value when given the values of the variables in $V \subseteq \mathsf{Var}$, then the probability that the value of x is equal to any of the values in V is negligible.

Lemma 3 *For any distribution $X \in \textsc{Dist}(\Gamma, \mathcal{F})$, any variable $x \in \mathsf{Var}$ and any set $V \subseteq \mathsf{Var}$,*

$$X \models \mathsf{Indis}(x; V; \epsilon) \wedge X \models \mathsf{Bad}(\epsilon') \Rightarrow \mathsf{Pr}[\gamma \overset{\$}{\leftarrow} X : S_\gamma(x) \in S\gamma(V - x)] \leq \epsilon + \epsilon' + \frac{|S(V - x)|}{2^{length(x)}}.$$

As a consequence to Lemma 3, if $X \models \mathsf{Indis}(x; V, \mathcal{L}; \epsilon)$, then the probability that the value of x is either in $V - x$ or in $\mathcal{L}.dom$ and that X is bad is bounded by $\epsilon + \frac{|S(V-x)|+|\mathcal{L}.dom|}{2^{length(x)}}$. So by combining Lemma 3 with Lemma 2, we obtain the following:

Corollary 4 *For any distribution $X \in \textsc{Dist}(\Gamma, \mathcal{F})$, any variables $x \in \mathsf{Var}$ and any set $V \subseteq \mathsf{Var}$,*

$$X \models \mathsf{Indis}(x; V, \ell_{\mathcal{E}}; \epsilon) \Rightarrow X \models \mathsf{E}(x; \epsilon + \frac{|\mathcal{L}.dom|}{2^{length(x)}})$$

Using a similar reasoning, we also obtain the following:

Corollary 5 *For any distribution $X \in \textsc{Dist}(\Gamma, \mathcal{F})$, any variables $x \in \mathsf{Var}$ and any set $V \subseteq \mathsf{Var}$,*

$$X \models \mathsf{C}(x; y; V; V'; V''; \epsilon) \Rightarrow X \models \mathsf{E}(x; \epsilon + \frac{|\mathcal{L}.dom|}{2^{length(x)}})$$

3.4 Proving Semantic Security

We now link the notion of LoR-CPA security to our invariants as follows.

Proposition 6 *Let $M_{\mathcal{E}}(m_1| \dots \| m_n, c_n)$: **var** $\vec{x_n}$; cmd_n be a generic encryption mode. Then, if for any distribution $X \in \textsc{Dist}(\Gamma, \mathcal{F})$ we have $[\![\mathsf{cmd}_n]\!]X \models \mathsf{Indis}(c_n; \mathsf{Var} \setminus \textbf{var}\ ; \epsilon(q_{\mathcal{E}}, q_{\mathcal{E}})) \wedge [\![\mathsf{cmd}_n]\!]X \models \mathsf{Bad}(\epsilon'(q_{\mathcal{E}}, q_{\mathcal{E}}))$, where $q_{\mathcal{E}}$ and $q_{\mathcal{E}}$ are the number of calls to the encryption oracle made by the adversary $\mathcal{A}$ that created X and the number of computations of the block cipher required to answer all these encryption oracle queries, then for any adversary $\mathcal{B}$, the following holds:*

$$\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{M_{\mathcal{E}}}(\mathcal{B}) \leq \sum_{i=0}^{Q-1} 2 \left(\epsilon(i, q_{\mathcal{E}}^{(i)}) + \epsilon'(i, q_{\mathcal{E}}^{(i)}) \right)$$

where Q is an upper bound on the number of encryption queries made by $\mathcal{B}$, and $q_{\mathcal{E}}^{(i)}$ is an upper bound on the number of computations of the block cipher required to answer the first i encryption queries made by $\mathcal{B}$.

This proposition is a formal statement of the well-known result that an encryption scheme is secure against chosen plaintext attack if the ciphertexts are indistinguishable from random bits.

3.4.1 — Hoare Logic Rules

In Table 3.2, we present a set of rules of the form $\{\varphi\}\mathsf{cmd}\{\varphi'\}$, meaning that execution of command cmd in any distribution that satisfies φ leads to a distribution that satisfies φ'.

Using Hoare logic terminology, this means that the triple $\{\varphi\}\mathsf{cmd}\{\varphi'\}$ is valid. We group rules together according to their corresponding commands. For all the rules, we assume that $t \neq x, y, z$ unless indicated otherwise.

3.4.2 — Example

In Figure 3, we only present the three first steps of CBC. The notation (Cor) means that we use one corollary in order to transform an invariant. We denote by λ the size of $\mathcal{L}.dom$ before the execution of the program, and by η the length of all the strings used in the program.

$\mathcal{E}_{CBC}(m_1|m_2|m_3, c_3)$
var $c_0, y_1, y_2, y_3;$

	$\{\mathsf{Bad}(0)\}$	(init)
$c_0 \overset{\$}{\leftarrow} \mathcal{U};$	$\{\mathsf{Indis}(c_0; \mathsf{Var}^*; 0) \wedge \mathsf{Bad}(0)\}$	(R1)(G1)
$y_1 := c_0 \oplus m_1;$	$\{\mathsf{Indis}(c_0; \mathsf{Var}^* - y_1; 0) \wedge \mathsf{Indis}(y_1; \mathsf{Var}^* - c_0; 0)$	(X1)(G2)
	$\wedge\ \mathsf{E}(y_1; \frac{\lambda}{2^\eta}) \wedge \mathsf{Bad}(0)\}$	(Cor)(G1)
$z_1 := \mathcal{E}(y_1);$	$\{\mathsf{Indis}(c_0; \mathsf{Var} - y_1; 0) \wedge \mathsf{Indis}(z_1; \mathsf{Var}^*; 0)$	(B2)(B3)
	$\wedge\ \mathsf{Bad}(\frac{\lambda}{2^\eta})\}$	(B1)
$c_1 := c_0 \| z_1$	$\{\mathsf{Indis}(c_1; \mathsf{Var} - c_0 - z_1 - y_1; 0) \wedge \mathsf{Indis}(z_1; \mathsf{Var}^* - c_1; 0)$	(C1)(G2)
	$\wedge\ \mathsf{Bad}(\frac{\lambda}{2^\eta})\}$	(G1)
$y_2 := z_1 \oplus m_2;$	$\{\mathsf{Indis}(c_1; \mathsf{Var} - c_0 - z_1 - y_1; 0) \wedge \mathsf{Indis}(y_2; \mathsf{Var}^* - z_1)$	(G2)(X1)
	$\wedge\ \mathsf{E}(y_2; \frac{\lambda+1}{2^\eta}) \wedge\ \mathsf{Bad}(\frac{\lambda}{2^\eta})\}$	(Cor)(G1)
$z_2 := \mathcal{E}(y_2);$	$\{\mathsf{Indis}(c_1; \mathsf{Var} - c_0 - z_1 - y_1; 0) \wedge \mathsf{Indis}(z_2; \mathsf{Var}^*; 0)$	(B2)(B3)
	$\wedge\ \mathsf{Bad}(\frac{2\lambda+1}{2^\eta})\}$	(G1)
$c_2 := c_1 \| z_2$	$\{\mathsf{Indis}(c_2; \mathsf{Var} \setminus \{c_0, c_1, z_1, z_2, y_1\}; 0) \wedge \mathsf{Indis}(z_2; \mathsf{Var}^*; 0)$	(C1)(G2)
	$\wedge\ \mathsf{Bad}(\frac{2\lambda+1}{2^\eta})\}$	(G1)
$y_3 := z_2 \oplus m_3;$	$\{\mathsf{Indis}(c_2; \mathsf{Var} \setminus \{c_0, c_1, z_1, z_2, y_1\}; 0) \wedge \mathsf{Indis}(y_3; \mathsf{Var}^* - z_2)$	(G2)(X1)
	$\wedge\ \mathsf{E}(y_3; \frac{\lambda+2}{2^e ta}) \wedge \mathsf{Bad}(\frac{2\lambda+1}{2^\eta})\}$	(Cor)(G1)
$z_3 := \mathcal{E}(y_3);$	$\{\mathsf{Indis}(c_2; \mathsf{Var} \setminus \{c_0, c_1, z_1, z_2, y_1\}; 0) \wedge \mathsf{Indis}(z_3; \mathsf{Var}^*; 0)$	(B2)(B3)
	$\wedge\ \mathsf{Bad}(\frac{3\lambda+3}{2^\eta})\}$	(B1)
$c_3 := c_2 \| z_3$	$\{\mathsf{Indis}(c_3; \mathsf{Var} \setminus \{c_0, c_1, c_2, z_1, z_2, z_3, y_1\}; 0) \wedge \mathsf{Bad}(\frac{3\lambda+3}{2^\eta})\}$	(C1)

Figure 3 – Analysis of CBC encryption mode

We easily see, by extrapolating the example, that after processing n blocks, we would have that $[\![prg]\!]X \models \mathsf{Indis}(c_n; \mathsf{Var} \setminus \textbf{var}\ ; 0)$ and $[\![prg]\!]X \models \mathsf{Bad}(\sum_{j=0}^{n-1} \frac{\lambda+j}{2^\eta})$.

Let n_i be the number of blocks in the i^{th} encryption query of the adversary and λ_i be the size of $\mathcal{L}.dom$ just before processing the i^{th} encryption query of the adversary, we apply Proposition 6 and we get the security bound of CBC:

Generic Preservation	
(G1)	$\{\mathsf{Indis}(t; V; \epsilon)\}$ cmd $\{\mathsf{Indis}(t; V; \epsilon)\}$ if $x \notin V$ unless x is constructible from $V - t$, even if $t = y$ or $t = z$
(G2)	$\{\mathsf{Icounter}(t; w; V; V'; V''; \epsilon)\}$ cmd $\{\mathsf{Icounter}(t; w; V; V'; V''; \epsilon)\}$ if $y, z \notin V \cup V'$
(G3)	$\{\mathsf{Icounter}(t; w; V; V'; V''; \epsilon)\}$ cmd $\{\mathsf{Icounter}(t; w; V; V', x; V''; \epsilon)\}$ if $y \in V \cup V'$ or $z \in V \cup V'$, even if $t = y$ or $t = z$
(G4)	$\{\mathsf{C}(t; w; V; V; V''; \epsilon)\}$ cmd $\{\mathsf{C}(t; w; V; V'; V''; \epsilon)\}$ if $x \notin V$, even if $t = y$ or $t = z$
Random Assignment	
(R1)	$\{true\}$ $x \overset{\$}{\leftarrow} \mathcal{U}$ $\{\mathsf{Indis}(x; 0) \wedge \mathsf{Icounter}(x; \{x\}; \emptyset; 0) \wedge \mathsf{C}(x; x; \{x\}; \emptyset; \emptyset; 0)\}$
(R2)	$\{\mathsf{Indis}(t; V; \epsilon)\}$ $x \overset{\$}{\leftarrow} \mathcal{U}$ $\{\mathsf{Indis}(t; V, x; \epsilon)\}$
(R3)	$\{\mathsf{C}(\mathcal{E})t; w; V; \epsilon\}$ $x \overset{\$}{\leftarrow} \mathcal{U}$ $\{\mathsf{C}(\mathcal{E})t; w; V, x; \epsilon + 2^{-length(t)}\}$
Assignment	
(A1)	$\{\mathsf{Indis}(y; V; \epsilon)\}$ $x := y$ $\{\mathsf{Indis}(x; V; \epsilon)\}$ provided $y \notin V$
Xor Operation	
(X1)	$\{\mathsf{Indis}(y; V, y, z; \epsilon)\}$ $x := y \oplus z$ $\{\mathsf{Indis}(x; V, x, z; \epsilon)\}$ if $y \neq z$ and $y \notin V$
Concatenation	
(C1)	$\{\mathsf{Indis}(y; V, y, z; \epsilon_1) \wedge \mathsf{Indis}(z; V, y, z; \epsilon_2)\}$ $x := y \Vert z$ $\{\mathsf{Indis}(x; V, x; \epsilon_1 + \epsilon_2)\}$ if $y, z \notin V$
Block Cipher	
(B1)	$\{\mathsf{E}(y; \epsilon) \wedge \mathsf{Bad}(\epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{Bad}(\epsilon + \epsilon')\}$
(B2)	$\{\mathsf{E}(y; \epsilon)\}$ $x := \mathcal{E}(y)$ $\{\mathsf{Indis}(x; \mathsf{Var}^*; 0) \wedge \mathsf{Icounter}(x; x; \{x\}; \emptyset; \emptyset; 0)\}$
(B3)	$\{\mathsf{E}(y; \epsilon) \wedge \mathsf{Indis}(t; V; \epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{Indis}(t; V, x; \epsilon')\}$ provided $x, \mathcal{L} \notin V$
(B4)	$\{\mathsf{E}(y; \epsilon) \wedge \mathsf{Indis}(t; V, y, \mathcal{L}; \epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{Indis}(t; V, x, y, \mathcal{L}; \epsilon')\}$ provided $x \notin V$
(B5)	$\{\mathsf{C}(y; z; V; V'; V''; \epsilon) \wedge \mathsf{Icounter}(t; w; V; V'; V''; \epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{Icounter}(t; w; V; V'; V''; \epsilon')\}$ provided $y \notin V'$ and $z \neq w$
(B6)	$\{\mathsf{C}(y; z; V; V'; V''; \epsilon) \wedge \mathsf{Icounter}(t; z; V; V'; V''; \epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{Icounter}(t; z; V; V'; V'', y; \epsilon')\}$ provided $y \notin V'$
(B7)	$\{\mathsf{C}(y; z; V; V'; V''; \epsilon) \wedge \mathsf{C}(t; w; V; V'; V''; \epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{C}(t; w; V; V'; V''; \epsilon')\}$ provided $y \notin V'$ and $z \neq w$
(B8)	$\{\mathsf{C}(y; z; V; V'; V''; \epsilon) \wedge \mathsf{C}(t; z; V; V'; V''; \epsilon')\}$ $x := \mathcal{E}(y)$ $\{\mathsf{C}(t; z; V; V'; V'', y; \epsilon')\}$ provided $y \notin V'$
Increment	
(I1)	$\{\mathsf{Icounter}(y; z; V; V'; V''; \epsilon)\}$ $x := y + 1$ $\{\mathsf{Icounter}(x; z; V, x; V'; V''; \epsilon) \wedge \mathsf{C}(x; z; V, x; V'; V''; \epsilon)\}$
(I2)	$\{\mathsf{Indis}(y; V; \epsilon)\}$ $x := y + 1$ $\{\mathsf{Indis}(x; V; \epsilon)\}$ if $y \notin V$
For Loop	
(F1)	$\{\psi(p-1)\}$ $\mathsf{for}\ k = p\ \mathsf{to}\ q\ \mathsf{do}$: $[c_k]$ $\{\psi(q)\}$ provided $\{\psi(k-1)\}$ c_k $\{\psi(k)\}$ for $p \leq k \leq q$
General Rules	
(Csq)	if $\phi_1 \Rightarrow \phi_2$, $\phi_3 \Rightarrow \phi_4$ and $\{\phi_2\}\mathsf{cmd}\{\phi_3\}$, then $\{\phi_1\}\mathsf{cmd}\{\phi_4\}$
(Seq)	if $\{\phi_1\}\mathsf{cmd}_1\{\phi_2\}$ and $\{\phi_2\}\mathsf{cmd}_2\{\phi_3\}$, then $\{\phi_1\}\mathsf{cmd}_1; \mathsf{cmd}_2\{\phi_3\}$
(Conj)	if $\{\phi_1\}\mathsf{cmd}\{\phi_2\}$ and $\{\phi_3\}\mathsf{cmd}\{\phi_4\}$, then $\{\phi_1 \wedge \phi_3\}\mathsf{cmd}\{\phi_2 \wedge \phi_4\}$

Table 3.2 – Hoare Logic Rules

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{M_{\mathcal{E}}}(\mathcal{B}) &\leq \sum_{i=0}^{Q-1} 2\left(\epsilon(i, q^{(i)}_{\mathcal{E}}) + \epsilon'(i, q^{(i)}_{\mathcal{E}})\right) \\
&\leq 2\sum_{i=0}^{Q-1}\left(0 + \sum_{j=0}^{n_{i+1}-1} \frac{\lambda_i + j}{2^\eta}\right) \\
&\leq 2\sum_{i=0}^{Q-1}\left(\sum_{j=0}^{n_{i+1}-1} \frac{(\sum_{k=1}^{k=i} n_k) + j}{2^\eta}\right) \\
&\leq 2 \sum_{i=0}^{(\sum_{i=1}^{Q} n_i)-1} \frac{i}{2^\eta} \\
&\leq 2 \sum_{i=0}^{Q_{\mathcal{E}}-1} \frac{i}{2^\eta} \\
&\leq \frac{Q_{\mathcal{E}}(Q_{\mathcal{E}}-1)}{2^\eta}
\end{aligned}$$

3.5 Loops Analysis

In order to keep simple our presentation we do not consider security parameters in this section. Two methods could be used to verify that $\{true\}$ cmd_n $\{\mathsf{Indis}(c_n; \mathsf{Var} \setminus \mathbf{var}\ )\}$ is valid. The first method consists in starting at the beginning of cmd_n and, at each simple command, applying every possible rule until we reach the end, and see if $\mathsf{Indis}(c_n; \mathsf{Var} \setminus \mathbf{var}\ )$. The second method consists in starting at the end of cmd_n with the invariant we need there ($\mathsf{Indis}(c_n; \mathsf{Var} \setminus \mathbf{var}\ )$) and, at each command, determining which precondition before the command in order to have the predicate we know we need afterwards.

The first method tends to be more straightforward, but may require to keep track of many invariants that are not necessary to obtain the final result. The second method lets us keep only those invariants that we need, but sometimes, several choices of preconditions to obtain the postconditions we need are possible, so each 'branch' resulting from each possibility must be explored, which could require a very large (in the worst case, exponential) number of branches being explored. Here, we prefer the second method because superfluous branches tend to die off quickly.

The code of a general encryption mode will generally contain a for loop. It is therefore important to be able to find predicates $\psi(i)$ that will enable us to apply the rule (F1) from our logic. We present two heuristic that can be used to discover such an invariant, and we show how to apply them on $CBC_{\mathcal{E}}$ and on a modified version of $CTR_{\mathcal{E}}$. The first heuristic

When we hit a command textsffor $k = p$ to q do:" $[c_k]$, we express the predicate that needs to hold at the end of the loop in the form $\varphi(k)$. We note that, usually, we expect that the loop will go from 1 to n, and the predicate that will need to hold at the end is $\mathsf{Indis}(c_n; \mathsf{Var} \setminus \mathbf{var}\ )$.

In that case, $\varphi(k) = \mathsf{Indis}(c_k; \mathsf{Var} \setminus \mathbf{var}\ )$.

Then, we go backwards through the instructions c_k of the loop to find a predicate $\psi(k-1)$ such that $\{\psi(k-1)\}\ c_k\ \{\varphi(k)\}$ holds. If $\varphi(k) \wedge \psi(k) = \varphi(k)$, then we have found a predicate such that $\{\varphi(k-1)\}\ c_k\ \{\varphi(k)\}$ holds, and we can apply rule (F1). Otherwise, we repeat this process with $\varphi'(k) = \varphi(k) \wedge \psi(k)$ until we find a stable predicate.

Example We show how to apply this heuristic in the case of $CBC_{\mathcal{E}}$, whose description can be found in Figure 2. As mentioned before, the predicate that we need at the end of the loop is $\varphi(n) = \mathsf{Indis}(c_n; \mathsf{Var} \setminus \mathbf{var}\ )$. Going backwards one first time through the loop, we find that in order to have $\varphi(k)$ hold at the end of the code of the loop, the following must hold at the beginning:

$$\begin{aligned}\psi(k-1) &= \mathsf{Indis}(c_{k-1}; \mathsf{Var} \setminus \mathbf{var}\ ) \wedge \mathsf{Indis}(z_{k-1}; \{m_k, \mathcal{L}_{\mathcal{E}}\}) \\ &= \varphi(k-1) \wedge \mathsf{Indis}(z_{k-1}; \{m_k, \mathcal{L}_{\mathcal{E}}\})\end{aligned}$$

Clearly, $\varphi(k) \wedge \psi(k) \neq \varphi(k)$, so we set $\varphi'(k) = \varphi(k) \wedge \psi(k)$. Repeating this process again with $\varphi'(k)$ as our desired end invariant, we find that, with $\{\varphi'(k-1)\}\ c_k\ \{\varphi'(k)\}$ holds (where c_k denotes the commands inside the loop in Figure 2). Therefore, we can apply rule (F1) with $\varphi'(k)$ as our stable invariant. □

We find that this simple heuristic is sufficient for finding loop invariants for all our examples, provided that the program is written using a single loop.

However, it is easy to construct examples for which the heuristic will fail. For example, Figure 4 shows an alternative description of $CTR_{\mathcal{E}}$ that makes the heuristic fail. In cases such as these, we use a second heuristic based on widening in abstract interpretation. If, after a number n of iteration of the first heuristic, we have not found a stable predicate, we decide that the heuristic has failed (the choice of the number n is artibrary, we find that $n = 2$ is sufficient for all our examples). Then, we go back to the starting predicate $\varphi(k)$ and we go backwards through the instructions c_k of the loop to find $\psi_1(k)$ and $\psi_2(k)$ such that $\{\psi_1(k-1)\}\ c_k$ $\{\varphi(k)\}$ and $\{\psi_2(k-1)\}\ c_k\ \{\varphi(k) \wedge \psi_1(k)\}$ hold. By analyzing the predicates $\varphi(k)$, $\psi_1(k)$ and $\psi_2(k)$, we identify the predicate $\gamma(k)$ such that $\gamma(k+1)$ appears in $\psi_1(k)$ and $\gamma(k+2)$ appears in $\psi_2(k)$. We then start over with the first heuristic using $\varphi'(k) = \varphi(k) \wedge \bigwedge_{j=k+1}^{q} \gamma(j)$ as the invariant that needs to hold at the end of the loop.

Example We show how the second heuristic is used to find a stable predicate for the second loop in $CTR'_{\mathcal{E}}$ (see Figure 4). As before, the predicate that we need at the end of the loop is $\varphi(n) = \mathsf{Indis}(c_n; \mathsf{Var} \setminus \mathbf{var}\ )$. Going backwards one first time through the loop, we find that in order to have $\varphi(k)$ hold at the end of the code of the loop, the following must hold at the beginning:

$$\psi_1(k-1) = \mathsf{Indis}(c_{k-1}; (\mathsf{Var} \setminus \mathbf{var}\ ), y_k) \wedge \mathsf{Indis}(y_k; (\mathsf{Var} \setminus \mathbf{var}\ ), c_{k-1})$$

```
$CTR'_{\mathcal{E}}(m_1\|m_2\|\dots\|m_n, c_n)$
var $i, y_1, z_1, \dots, y_n, z_n, ctr_0, \dots, ctr_n, c_0, \dots, c_{n-1}$
$ctr_0 \xleftarrow{\$} \mathcal{U}$; $c_0 := ctr_0$;
for $i = 1$ to $n$ do:
    $[ctr_i := ctr_{i-1} + 1;\ y_i := \mathcal{E}(ctr_i)]$;
for $i = 1$ to $n$ do:
    $[z_i := y_i \oplus m_i;\ c_i := c_{i-1}\|z_i]$;
```

Figure 4 – Alternative description of $CTR_{\mathcal{E}}$

Repeating this process with $\varphi(k) \wedge \psi_1(k)$, we obtain

$$\begin{aligned}\psi_2(k-1) =& \mathsf{Indis}(c_{k-1}; (\mathsf{Var} \setminus \mathbf{var}\ ), y_{k+1}, y_k) \wedge \mathsf{Indis}(y_{k+1}; (\mathsf{Var} \setminus \mathbf{var}\ ), c_k) \\ & \wedge \mathsf{Indis}(y_k; (\mathsf{Var} \setminus \mathbf{var}\ ), c_{k-1})\end{aligned}$$

Analyzing $\psi_1(k)$ and $\psi_2(k)$, we find the following $\gamma(l)$

$$\gamma(l) = \mathsf{Indis}(c_k; (\mathsf{Var} \setminus \mathbf{var}\ ), \{y_i\}_{i=k+1}^{l}) \wedge \mathsf{Indis}(y_l; (\mathsf{Var} \setminus \mathbf{var}\ ), c_{l-1})$$

We then start over with the first heuristic with the following invariant

$$\begin{aligned}\varphi(k)' =& \mathsf{Indis}(c_k; \mathsf{Var} \setminus \mathbf{var}\ ) \wedge \\ & \bigwedge_{j=k+1}^{q} \Big(\mathsf{Indis}(c_k; (\mathsf{Var} \setminus \mathbf{var}\ ), \{y_i\}_{i=k+1}^{j}) \wedge \mathsf{Indis}(y_j; (\mathsf{Var} \setminus \mathbf{var}\ ), c_{j-1})\Big) \\ =& \mathsf{Indis}(c_k; (\mathsf{Var} \setminus \mathbf{var}\ ), \{y_i\}_{i=k+1}^{q}) \wedge \bigwedge_{j=k+1}^{q} \mathsf{Indis}(y_j; (\mathsf{Var} \setminus \mathbf{var}\ ), c_{j-1})\end{aligned}$$

and we find that this invariant is a stable predicate. □

3.6 Conclusion

We propose a Hoare logic for the verification of symmetric encryption modes in concrete security model. Our language is expressive enough in order to take into account FOR loops. We also propose a method for generating automatically loop's invariants.

Chapter 4

Message Authentication Codes

Rappelons que les MACs (*Message authentication codes*) sont des primitives de sécurité assurant l'intégrité et l'authentification des messages. Ils sont aussi utilisés dans la construction de schémas plus complexes, comme par exemple les chiffrement à base d'identité (*identity-based encryption*). Les MACs sont souvent composés de deux phases. Une première phase qui produit un "hâché", puis une seconde phase qui rend le MACs non prédictible par un adversaire. Nous proposons une logique de Hoare vérifiant que la prmière partie est bien une fonction de hâchage quasi-universelle (*almost-universal hash function*) utilisant des fonctions de hâchage et des chiffrements symmétriques par blocs. Enfin nous donnons une liste d'options possibles pour finir d'analyser les dernières lignes des MACs. Nous avons dévelopé un prototype en Ocaml permettant de prouver la sécurité de nombreux MACs parmi DMAC, ECBC, FCBC, XCBC, PMAC et HMAC.

Contents

Messages Authentication Codes (MACs) are among the most common primitives in symmetric key cryptography. They ensure the integrity and provenance of a message, and they can be used, in conjunction with chosen-plaintext secure encryption, to obtain chosen-ciphertext secure encryption. The study of the security of MACs is, of course, not a new field. Bellare et al. [BKR94] were the first to prove the security of CBC-MAC for fixed-length inputs. Following this work, a myriad of new MACs secure for variable-length inputs were proposed ([BCK96, BHK$^+$99, BR00, BR02, PR00]).

Contributions: We present a method for proving the security of MACs based on block ciphers and hash functions. To prove the security of MACs, we first break the MAC algorithms into two parts: a 'front-end', whose work is to compress long input messages into small digests, and a 'back-end', usually a mixing step, which obfuscates the output of the front-end. We present a Hoare logic to prove that the front-ends of MACs are almost-universal hash functions. We then make a list of operations which, when composed with an almost-universal hash function, yield a secure MAC.

Proving the security of MACs requires predicates that model the security of authentication. We have to consider the simultaneous execution of a program on two input messages in order to bound collision probabilities. As a result, we propose a new semantics, and define new invariants for determining equality or inequality of values. We also present a treatment of for loops, which allows us to prove the security of protocols that can take arbitrary strings as an input. Finally, we implemented our method into a tool [16] that can be used to prove the security of several well-known MACs, such as HMAC [BCK96], DMAC [PR00], ECBC, FCBC and XCBC [BR00] and PMAC [BR02].

Outline: In Section 4.1, we introduce cryptographic background. The following section introduces our grammar, semantics and assertion language. In Section 4.3, we present our Hoare logic and method for proving the security of almost-universal hash functions. In Section 4.4, we discuss our implementation of this logic and treatment of loops. In Section 4.5, we explain how to combine our Hoare logic with one of the possible back-end options in orer to obtain a secure MAC. Finally, we conclude in Section 7.4. Moreover, all proofs are given in [17]. Note that we also use the same notations and conventions as in the previous chapters.

4.1 Cryptographic Background

Definition 10 (MAC) *A* message authentication code *is a triple of polynomial-time algorithms* $(K,\ MAC, V)$*, where* $K(1^\eta)$ *takes a security parameter* 1^η *and outputs a secret key* sk*,*[1] $MAC(sk, m)$ *takes a secret key and a message* m*, and outputs a* tag*, and* $V(sk, m, tag)$ *takes a secret key, a message and a tag, and outputs a bit: 1 for a correct tag, 0 otherwise.*

Definition 11 (Unforgeability) *A MAC given by* (K, MAC, V) *is unforgeable under a chosen-message attack (UNF-CMA) if for every oracle polynomial-time algorithm* $\mathcal{A}$ *whose*

[1]The secret key sk can consist of one or several strings, depending on the MAC.

output message m^* is different from any message it sent to the MAC oracle, the following probability is negligible

$$Pr[sk \overset{\$}{\leftarrow} K(1^\eta); (m^*, tag^*) \overset{\$}{\leftarrow} \mathcal{A}^{MAC(sk,\cdot),V(sk,\cdot,\cdot)} : V(sk, m^*, tag^*) = 1]$$

A standard method for constructing MACs is to apply a pseudo-random function, or some other form of 'mixing' step, to the output of an almost-universal hash function [WC19, WC81]. Our verification technique assumes that the MAC is constructed in this way.

Definition 12 (Almost-Universal Hash) *A family of functions $\mathcal{H} = \{h_i\}_{i\in\{0,1\}^\eta}$ is a family of almost-universal hash functions if for any two strings a and b, $Pr_{h_i\in\mathcal{H}}[h_i(a) = h_i(b)]$ is negligible, where the probability is taken over the choice of h_i in $\mathcal{H}$.*

It is much easier to work with this definition because the adversary against an almost-universal hash function is non-adaptive, and must output his attempt at finding a collision in the almost-universal hash function independently from the choice of the key.

4.2 Models

4.2.1 — Grammar

We consider the language defined by the following BNF grammar.

$$\begin{array}{lll} \mathsf{cmd} & ::= & x := \mathcal{E}(y) \mid x := \mathcal{H}(y) \mid x := y \mid x := y \oplus z \mid x := y \| z \mid x := \rho^i(y) \\ & & \mid \mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}\text{: } [\mathsf{cmd}_l] \mid \mathsf{cmd}_1; \mathsf{cmd}_2 \end{array}$$

All command have the usual expected effect, we just precise that

— We omit the key used every time to simplify the notation in block cipher applications.

— $x := \rho^i(y)$ denotes the i-fold application of the function ρ to the value of y (that is, $\rho(\ldots(\rho(y)\ldots))$, where ρ is repeated i times) and assigning the result to x.

— $\mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}$: $[\mathsf{cmd}_l]$ denotes the successive execution of $\mathsf{cmd}_p, \mathsf{cmd}_{p+1}, \ldots, \mathsf{cmd}_q$.

— $c_1; c_2$ is the sequential composition of c_1 and c_2.

The function ρ is used to compute the *tweak* in *tweakable block ciphers* ([LRW02b]). The function used to compute this tweak can vary from one protocol to the next, so we only specify that it must be a public function. When a scheme uses a function ρ, the properties of the function ρ required for the proof will be added to the initial conditions of the verification procedure.

We assume that, prior to executing the MAC, the message has been padded using some unambiguous padding scheme, so that all the message blocks $m_1, \ldots, m_i$ are of equal and appropriate length for the scheme, usually the input length of the block cipher. We denote by Var the full set of variables used in the program. This set always includes input and output variables, and the special variable k that contains a secret key. The set **var** $\vec{x_i}$ contains the variables of the programs that are neither input variables, output variables or the secret key.

Definition 13 (Generic Hash Function) *A generic hash function $Hash$ on i message blocks $m_1, \ldots, m_i$ and with output c, is represented by a tuple $(\mathcal{F}_\mathcal{E}, \mathcal{F}_\mathcal{H}, Hash(m_1\| \ldots \|m_i, c)$: $\mathbf{var}\ \vec{x_i}$; $\mathsf{cmd}_i)$, where $\mathcal{F}_\mathcal{E}$ is a family of pseudorandom permutations (usually a block cipher), $\mathcal{F}_\mathcal{H}$ is a family of cryptographic hash functions, and $Hash(m_1\| \ldots \|m_i, c) : \mathbf{var}\ \vec{x_i}; \mathsf{cmd}_i$ is the code of the hash function, where the commands of cmd_i are built using the grammar described above.*

We describe in our formalism the hash function $Hash_{CBC}$, which is used in DMAC [PR00] and ECBC [BR00]:

```
$Hash_{CBC}(m_1\| \ldots \|m_n, c_n)$ : $\mathbf{var}\ i, z_2, \ldots, z_n, c_1, \ldots, c_{n-1}$;
$c_1 := \mathcal{E}(m_1)$;
for $i = 2$ to $n$ do: $[z_i := c_{i-1} \oplus m_i;\ c_i := \mathcal{E}(z_i)]$
```

Description of $Hash_{CBC'}$, $Hash_{PMAC}$ and $Hash_{HMAC}$ are explained in [17]. These are used in FCBC, XCBC [BR00], PMAC [BR02] and HMAC [BCK96].

4.2.2 — Semantics

We consider the execution of a program on two inputs simultaneously. These simultaneous executions will enable us to define predicates about the equality or inequality of intermediate values in the program.

A program takes as input a *configuration* $(S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H})$ and yields a distribution on configurations. A configuration is composed of two states S and S', a block cipher $\mathcal{E}$, a hash function $\mathcal{H}$, and two lists of pairs $\mathcal{L}_\mathcal{E}$ and $\mathcal{L}_\mathcal{H}$. The two states S and S', which are functions that take a variable as input and return a value in $\{0,1\}^* \cup \{\perp\}$ (the symbol $\perp$ indicates that no value has been assigned to the variable yet), assign values to all the variables in each of the two simultaneous executions of the program. The lists $\mathcal{L}_\mathcal{E}$ and $\mathcal{L}_\mathcal{H}$ record the values for which the functions $\mathcal{E}$ and $\mathcal{H}$ were computed respectively. These lists are common to both executions of the program. We denote by $\mathcal{L}_\mathcal{E}.dom$ and $\mathcal{L}_\mathcal{E}.res$ the lists obtained by projecting each pair in $\mathcal{L}_\mathcal{E}$ to its first and second element respectively. We define $\mathcal{L}_\mathcal{H}.dom$ and $\mathcal{L}_\mathcal{H}.res$ similarly.

Let Γ denote the set of configurations and $\textsc{Dist}(\Gamma)$ the set of distributions on configurations. The semantics is given in Table 4.1, where $\delta(x)$ denotes the Dirac measure, i.e. $\mathsf{Pr}[x] = 1$, $S\{x \mapsto v\}$ denotes the state which assigns the value v to the variable x, and behaves like S for all other variables, $\mathcal{L}_\mathcal{E} \cdot (x, y)$ denotes the addition of element (x, y) to $\mathcal{L}_\mathcal{E}$ and $\circ$ denotes function composition. The semantic function $\phi : \Gamma \rightarrow \textsc{Dist}(\Gamma)$ of commands can be lifted in the usual way to a function $\phi^* : \textsc{Dist}(\Gamma) \rightarrow \textsc{Dist}(\Gamma)$ by point-wise application of ϕ. By abuse of notation we also denote the lifted semantics by $[\![\mathsf{cmd}]\!]$.

Here, we are only interested in the distributions that can be constructed in polynomial time by an adversary having access only to the random oracle. We denote their set by $\textsc{Dist}(\Gamma, \mathcal{F}_\mathcal{E}, \mathcal{F}_\mathcal{H})$, where $\mathcal{F}_\mathcal{E}$ is a family of block ciphers, $\mathcal{F}_\mathcal{H}$ is a family of hash functions, and is defined as the set of distributions of the form:

$$
\begin{array}{l}
[\![x := \mathcal{E}(y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) = \\
\quad\left\{\begin{array}{l}
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \text{ if } (S(y), v), (S'(y), v') \in \mathcal{L}_\mathcal{E} \\
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E} \cdot (S(y), v), \mathcal{L}_\mathcal{H}) \\
\qquad\qquad \text{if } (S(y), v) \notin \mathcal{L}_\mathcal{E}, (S'(y), v') \in \mathcal{L}_\mathcal{E} \text{ and } v = \mathcal{E}(S(y)) \\
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E} \cdot (S'(y), v'), \mathcal{L}_\mathcal{H}) \\
\qquad\qquad \text{if } (S(y), v) \in \mathcal{L}_\mathcal{E}, (S'(y), v') \notin \mathcal{L}_\mathcal{E} \text{ and } v' = \mathcal{E}(S'(y)) \\
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, (\mathcal{L}_\mathcal{E} \cdot (S(y), v)) \cdot (S'(y), v'), \mathcal{L}_\mathcal{H}) \\
\qquad\qquad \text{if } (S(y), v), (S'(y), v') \notin \mathcal{L}_\mathcal{E} \text{ and } v = \mathcal{E}(S(y)), v' = \mathcal{E}(S'(y))
\end{array}\right. \\
[\![x := \mathcal{H}(y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) = \\
\quad\left\{\begin{array}{l}
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \text{ if } (S(y), v), (S'(y), v') \in \mathcal{L}_\mathcal{H} \\
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H} \cdot (S(y), v)) \\
\qquad\qquad \text{if } (S(y), v) \notin \mathcal{L}_\mathcal{H}, (S'(y), v') \in \mathcal{L}_\mathcal{H} \text{ and } v = \mathcal{H}(S(y)) \\
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H} \cdot (S'(y), v')) \\
\qquad\qquad \text{if } (S(y), v) \in \mathcal{L}_\mathcal{H}, (S'(y), v') \notin \mathcal{L}_\mathcal{H} \text{ and } v' = \mathcal{H}(S'(y)) \\
\delta(S\{x \mapsto v\}, S'\{x \mapsto v'\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, (\mathcal{L}_\mathcal{H} \cdot (S(y), v)) \cdot (S'(y), v')) \\
\qquad\qquad \text{if } (S(y), v), (S'(y), v') \notin \mathcal{L}_\mathcal{H} \text{ and } v = \mathcal{H}(S(y)), v' = \mathcal{H}(S'(y))
\end{array}\right. \\
[\![x := y \oplus z]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) = \delta(S\{x \mapsto S(y) \oplus S(z), S'\{x \mapsto S'(y) \oplus S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[\![x := y||z]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) = \delta(S\{x \mapsto S(y)||S(z), S'\{x \mapsto S'(y)||S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[\![x := \rho^i(t)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) = \delta(S\{x \mapsto \rho^i(S(y))\}, S'\{x \mapsto \rho^i(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[\![\mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}\text{: } [\mathsf{cmd}_l]]\!] = [\![\mathsf{cmd}_q]\!] \circ [\![\mathsf{cmd}_{q-1}]\!] \circ \ldots \circ [\![\mathsf{cmd}_p]\!] \\
[\![c_1; c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]
\end{array}
$$

Table 4.1 – The semantics of the programming language

$$
\begin{array}{r}
[\mathcal{E} \xleftarrow{\$} \mathcal{F}_\mathcal{E}(1^\eta); \mathcal{H} \xleftarrow{\$} \mathcal{F}_\mathcal{H}(1^\eta); u \xleftarrow{\$} \{0,1\}^\eta; (S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} \mathcal{A}^\mathcal{H}(1^\eta) : \\
(S\{k \mapsto u\}, S'\{k \mapsto u\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H})]
\end{array}
$$

where k is a variable holding a secret string needed in some MACs (among our examples, $Hash_{PMAC}$ and $Hash_{HMAC}$ need it) and $\mathcal{A}$ is a probabilistic polynomial-time algorithm with oracle access to the hash function, such that $\mathcal{L}_\mathcal{H}$ contains the list of queries made by $\mathcal{A}$ to the random oracle, and $\mathcal{L}_\mathcal{E}$ is empty since $\mathcal{A}$ does not have access the key of the block cipher.

A notational convention. It is easy to see that commands never modify $\mathcal{E}$ or $\mathcal{H}$. Therefore, we can, without ambiguity, write $(\hat{S}, \hat{S}', \mathcal{L}'_\mathcal{E}, \mathcal{L}'_\mathcal{H}) \xleftarrow{\$} [\![c]\!](S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H})$ instead of $(\hat{S}, \hat{S}', \mathcal{E}, \mathcal{H}, \mathcal{L}'_\mathcal{E}, \mathcal{L}'_\mathcal{H}) \xleftarrow{\$} [\![c]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H})$.

4.2.3 — Assertion Language

We give an intuitive description of the invariants used in our logic, note that the three last predicates are adapted version of the one used in the previous chapters to the fact that we have two paralell executions[2]:

— Empty: means that the probability that $\mathcal{L}_\mathcal{E}$ contains an element is negligible.

— Equal($\mathbf{x}, \mathbf{y}$): means that the probability that $S(x) \neq S'(y)$ is negligible.

[2]In ordert to keep this chapter self-contain we explicitely define all predicates

— $\mathsf{Unequal}(\mathbf{x}, \mathbf{y})$: means that the probability that $S(x) = S'(y)$ is negligible.

— $\mathsf{E}(\mathcal{E}; \mathbf{x}; \mathbf{V})$: means that the probability that the value of x is either in $\mathcal{L}_\mathcal{E}.\mathsf{dom}$ or in V is negligible.

— $\mathsf{H}(\mathcal{H}; \mathbf{x}; \mathbf{V};)$: means that the probability that the value of x is either in $\mathcal{L}_\mathcal{H}.\mathsf{dom}$ or in V is negligible.

— $\mathsf{Indis}(\mathbf{x}; \mathbf{V}; \mathbf{V}')$: means that no adversary has non-negligible probability to distinguish whether he is given results of computations performed using the value of x or a random value, when he is given the values of the variables in V and the values of the variables in V' from the parallel execution. In addition to variables in Var, the set V can contain special symbols $\ell_\mathcal{E}$ or $\ell_\mathcal{H}$. When the symbol $\ell_\mathcal{E}$ is present, it means that, in addition to the other variables in V, the distinguisher is also given the values in $\mathcal{L}_\mathcal{E}.dom$, similarly for $\ell_\mathcal{H}$.

In the following, for any set $V \subseteq \mathsf{Var}$, we denote by $S(V)$ the multiset resulting from the application of S on each variable in V. We also use $S(V, \ell_\mathcal{E})$ as a shorthand for $S(V) \cup \mathcal{L}_\mathcal{E}.dom$, and similarly for $S(V, \ell_\mathcal{H})$ and $S(V, \ell_\mathcal{E}, \ell_\mathcal{H})$. For a set V and a variable, we write V, x as a shorthand for $V \cup \{x\}$ and $V - x$ as a shorthand for $V \setminus \{x\}$ and we use $\mathsf{Indis}(x)$ as a shorthand for $\mathsf{Indis}(x; \mathsf{Var}, \ell_\mathcal{E}, \ell_\mathcal{H}; \mathsf{Var})$.

Our Hoare logic is based on statements from the following language.

$\varphi ::= \mathsf{true} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \psi$

$\psi ::= \mathsf{Indis}(x; V; V') \mid \mathsf{Equal}(x, y) \mid \mathsf{Unequal}(x, y) \mid \mathsf{Empty} \mid \mathsf{E}(\mathcal{E}; x; V) \mid \mathsf{H}(\mathcal{H}; x; V;)$

where $x, y \in \mathsf{Var}$ and $V, V' \subseteq \mathsf{Var}$, except for $\mathsf{Indis}(x; V; V')$ where $V \subseteq \mathsf{Var} \cup \{\ell_\mathcal{E}, \ell_\mathcal{H}\}$.

More formally, we define that a distribution X satisfies ψ, denoted $X \models \psi$ as follows:

— $X \models \mathsf{true}$, $X \models \varphi \wedge \varphi'$ iff $X \models \varphi$ and $X \models \varphi'$, $X \models \varphi \vee \varphi'$ iff $X \models \varphi$ or $X \models \varphi'$

— $X \models \mathsf{Empty}$ iff $\mathsf{Pr}[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : \mathcal{L}_\mathcal{E} \neq \emptyset]$ is negligible

— $X \models \mathsf{Equal}(x, y)$ iff $\mathsf{Pr}[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : S(x) \neq S'(y)]$ is negligible

— $X \models \mathsf{Unequal}(x, y)$ iff $\mathsf{Pr}[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : S(x) = S'(y)]$ is negligible

— $X \models \mathsf{E}(\mathcal{E}; x; V)$ iff $\mathsf{Pr}[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : \{S(x), S'(x)\} \cap (\mathcal{L}_\mathcal{E}.\mathsf{dom} \cup S(V - x) \cup S'(V - x)) \neq \emptyset]$ is negligible[3]

— $X \models \mathsf{H}(\mathcal{H}; x; V;)$ iff $\mathsf{Pr}[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : \{S(x), S'(x)\} \cap (\mathcal{L}_\mathcal{H}.\mathsf{dom} \cup S(V - x) \cup S'(V - x)) \neq \emptyset]$ is negligible

— $X \models \mathsf{Indis}(x; V; V')$ iff the two following formulas hold:

$$[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : (S(x), S(V - x) \cup S'(V'))] \sim [(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U} : (u, S(V - x) \cup S'(V'))]$$

$$[(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X : (S'(x), S'(V - x) \cup S(V'))] \sim [(S, S', \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U} : (u, S'(V - x) \cup S(V'))]$$

[3]Since the variable x is removed from the set V when taking the probability, we always have $X \models \mathsf{E}(\mathcal{E}; x; V)$ iff $X \models \mathsf{E}(\mathcal{E}; x; V, x)$. This is to remove the trivial case that $\mathsf{Pr}[\{S(x), S'(x)\} \cap (\mathcal{L}_\mathcal{E}.\mathsf{dom} \cup S(\{x\}) \cup S'(\{x\})) \neq \emptyset]$ never holds, and to simplify the notation. The same is also used for invariants $\mathsf{H}(\mathcal{H}; x; V;)$ and $\mathsf{Indis}(x; V; V')$.

Lemma 7 *The following relations are true for any sets V_1, V_2, V_3 and variables x, y with $x \neq y$*

1. $\mathsf{Indis}(x; V_1; V_2) \Rightarrow \mathsf{Indis}(x; V_3; V_4)$ *if* $V_3 \subseteq V_1$ *and* $V_4 \subseteq V_2$
2. $\mathsf{H}(\mathcal{H}; x; V; \Rightarrow)\mathsf{H}(\mathcal{H}; x; V';)$ *if* $V' \subseteq V$
3. $\mathsf{E}(\mathcal{E}; x; V) \Rightarrow \mathsf{E}(\mathcal{E}; x; V')$ *if* $V' \subseteq V$
4. $\mathsf{Indis}(x; V, \ell_{\mathcal{H}}; \emptyset) \Rightarrow \mathsf{H}(\mathcal{H}; x; V;)$
5. $\mathsf{Indis}(x; V, \ell_{\mathcal{E}}; \emptyset) \Rightarrow \mathsf{E}(\mathcal{E}; x; V)$
6. $\mathsf{Indis}(x; \emptyset; \{y\}) \Rightarrow \mathsf{Unequal}(x, y) \wedge \mathsf{Unequal}(y, x)$

The proof of this lemma is in [17]. Note that results 4, 5 and 6 are particularly helpful because the invariant Indis is much easier to propagate than the other invariants.

4.3 Proving Almost-Universal Hash with an Hoare Logic

We prove that a program is an almost-universal hash function. To do this, we require that the program be written in a way so that, on input $m_1\|\ldots\|m_n$, the program must assign values to variables $c_1, \ldots, c_n$ in such a way that the variable c_1 contains the output of the function on input m_1, the variable c_2 contains the output of the function on input $m_1\|m_2$ and so on. Under this assumption, we determine the condition under which a program computes an almost-universal hash function family.

Proposition 8 *Let the generic hash $(\mathcal{F}_{\mathcal{E}}, \mathcal{F}_{\mathcal{H}}, Hash(m_1\|\ldots\|m_n, c_n) : \textbf{var}\ \vec{x};\ \mathsf{cmd})$ describe the program to compute a hash function Hash on an n block message. Then, Hash is an almost-universal hash function if, for every positive integer n, the following holds:*

$$UNIV(n) = \left(\bigwedge_{i=1}^{n-1} \mathsf{Unequal}(c_n, c_i) \wedge \bigwedge_{i=1}^{n} \mathsf{Equal}(m_i, m_i) \right) \vee \bigwedge_{i=1}^{n} \mathsf{Unequal}(c_n, c_i)$$

The intuition for this is that if m consists of l message blocks and m' consists of $n \leq l$ message blocks, then the fact that the probability that the hashes of m_1 and m_2 are equal is negligible, when the probability is taken over the choice of the key, is captured exactly by the invariant $\mathsf{Unequal}(c_l, c_n)$. The proof of this theorem is in [17].

We prove that this predicate holds at the end of execution by propagating and generating invariants through the program using rules of our Hoare Logic. We group rules together according to their corresponding commands. The proofs of soundness of our rules are given in [17]. First we note that several rules are just revisted versions of rules described in the previous chapters, but we prefer to rewrite them in order to facilitate the read. Moreover a concrete security analysis could be done since most of the work have been done in the previous sections or are not difficut to produce[4] but in order to keep the rules more readable we omit

[4]For the ρfunction, the securiy bound have to be specify according to each given functions

it. Since the invariants $\mathsf{Equal}(m_i)$ are useful only if the whole prefix of the two messages up to the i^{th} block are equal, so that keeping track of the equality or inequality of the message blocks after that point is unnecessary. For this reason, when we design our rules, we never produce the invariants $\mathsf{Unequal}(m_i, m_i)$ even when they would be correct.

In all the rules below, unless indicated otherwise, we assume that $t \notin \{x, y, z\}$ and $x \notin \{y, z\}$. In addition, for all rules involving the invariant Indis, we assume that $\ell_{\mathcal{E}}$ and $\ell_{\mathcal{H}}$ can be among the elements in the set V.

Initialization: The following predicate holds at the beginning of execution of the program. The string k is part of the secret key sk of the MAC.

(Init) $\{\mathsf{Indis}(k; \mathsf{Var}, \ell_{\mathcal{E}}, \ell_{\mathcal{H}}; \mathsf{Var} - k) \wedge \mathsf{Equal}(k, k) \wedge \mathsf{Empty}\}$

Generic preservation rules: The following rules show how invariants are preserved by most of the commands when the invariants concern a variable other than that being operated on. For all these rules, we assume that t and t' can be y or z and cmd is either $x := \rho^i(y)$, $x := y$, $x := y\|z$, $x := y \oplus z$, $x := \mathcal{E}(y)$, or $x := \mathcal{H}(y)$.

(G1) $\{\mathsf{Equal}(t, t')\}$ cmd $\{\mathsf{Equal}(t, t')\}$

(G2) $\{\mathsf{Unequal}(t, t')\}$ cmd $\{\mathsf{Unequal}(t, t')\}$

(G3) $\{\mathsf{E}(\mathcal{E}; t; V)\}$ cmd $\{\mathsf{E}(\mathcal{E}; t; V)\}$ provided $x \notin V$ and cmd is not $x := \mathcal{E}(y)$

(G4) $\{\mathsf{H}(\mathcal{H}; t; V; \})$ cmd $\{\mathsf{H}(\mathcal{H}; t; V; \})$ provided $x \notin V$ and cmd is not $x := \mathcal{H}(y)$

(G5) $\{\mathsf{Indis}(t; V; V')\}$ cmd $\{\mathsf{Indis}(t; V; V')\}$ provided cmd is not $x := \mathcal{E}(y)$ or $x := \mathcal{H}(y)$, and $x \notin V$ unless x is constructible from $V - t$ and $x \notin V'$ unless x is constructible from $V' - t$

(G6) $\{\mathsf{Empty}\}$ cmd $\{\mathsf{Empty}\}$ provided cmd is not $x := \mathcal{E}(y)$

Function ρ: Since the details of the function ρ are not known in advance, we cannot infer many rules, other than the following, which is a simple consequence of the fact that ρ is a function.

(P1) $\{\mathsf{Equal}(y, y)\}$ $x := \rho^i(y)$ $\{\mathsf{Equal}(x, x)\}$ for any positive integer i

Assignment: Most of the rules for the assignment follow simply from the fact that the value of x is equal to the value of y.

(A1) $\{true\}$ $x := m_i$ $\{(\mathsf{Equal}(m_i, m_i) \wedge \mathsf{Equal}(x, x)) \vee \mathsf{Unequal}(x, x)\}$

(A2) $\{\mathsf{Equal}(y, y)\}$ $x := y$ $\{\mathsf{Equal}(x, x)\}$

(A3) $\{\mathsf{Unequal}(y, y)\}$ $x := y$ $\{\mathsf{Unequal}(x, x)\}$

(A4) $\{\mathsf{Indis}(y; V; V')\}$ $x := y$ $\{\mathsf{Indis}(x; V; V')\}$ provided $y \notin V$ and $x \notin V'$ unless $y \in V'$

(A5) $\{\mathsf{E}(\mathcal{E}; y; V)\}$ $x := y$ $\{\mathsf{E}(\mathcal{E}; x; V) \wedge \mathsf{E}(\mathcal{E}; y; V)\}$ provided $y \notin V$

(A6) $\{\mathsf{H}(\mathcal{H}; y; V; \})$ $x := y$ $\{\mathsf{H}(\mathcal{H}; x; V; \wedge)\mathsf{H}(\mathcal{H}; y; V; \})$ provided $y \notin V$

(A7) $\{\mathsf{E}(\mathcal{E}; t; V, y)\}$ $x := y$ $\{\mathsf{E}(\mathcal{E}; t; V, x, y)\}$

(A8) $\{\mathsf{H}(\mathcal{H}; t; V, y; \})$ $x := y$ $\{\mathsf{H}(\mathcal{H}; t; V, x, y; \})$

Concatenation: The most important rule for the concatenation is (C4), which states that the concatenation of two random strings results in a random string. Rules (C5) and (C6) state that if a string is indistinguishable from a random value given all the values in the list of queries to the block cipher (or the hash function), then clearly it cannot be a prefix of one of the strings $\mathcal{L}_{\mathcal{E}}$.

(C1) $\{\mathsf{Equal}(y,y)\}\ x := y\|m_i\ \{(\mathsf{Equal}(m_i,m_i) \wedge \mathsf{Equal}(x,x)) \vee \mathsf{Unequal}(x,x)\}$

(C2) $\{\mathsf{Equal}(y,y) \wedge \mathsf{Equal}(z,z)\}\ x := y\|z\ \{\mathsf{Equal}(x,x)\}$

(C3) $\{\mathsf{Unequal}(y,y)\}\ x := y\|z\ \{\mathsf{Unequal}(x,x)\}$

(C4) $\{\mathsf{Indis}(y;V,y,z;V') \wedge \mathsf{Indis}(z;V,y,z;V')\}\ x := y\|z\ \{\mathsf{Indis}(x;V,x;V')\}$
provided $x,y,z \notin V$, $x \notin V'$ unless $y,z \in V'$ and $y \neq z$

(C5) $\{\mathsf{Indis}(y;V,\ell_{\mathcal{E}};V)\}\ x := y\|z\ \{\mathsf{E}(\mathcal{E};x;V)\}$

(C6) $\{\mathsf{Indis}(y;V,\ell_{\mathcal{H}};V)\}\ x := y\|z\ \{\mathsf{H}(\mathcal{H};x;V;\})$

For rules (C1), (C3), (C5) and (C6), the roles of y and z, or y and m_i in the case of (C1), can be exchanged.

Xor operator: Rules (X2) is reminiscent of a one-time-pad encryption of z with a random-looking value y. The other rules are propagation of the Equal and $\mathsf{Unequal}$ predicates.

(X1) $\{\mathsf{Equal}(y,y)\}\ x := y \oplus m_i\ \{(\mathsf{Equal}(m_i,m_i) \wedge \mathsf{Equal}(x,x)) \vee \mathsf{Unequal}(x,x)\}$

(X2) $\{\mathsf{Indis}(y;V,y,z;V')\}\ x := y \oplus z\ \{\mathsf{Indis}(x;V,x,z;V')\}$
provided $y \neq z$, $y \notin V$ and $x \notin V'$ unless $y,z \in V'$

(X3) $\{\mathsf{Equal}(y,y) \wedge \mathsf{Equal}(z,z)\}\ x := y \oplus z\ \{\mathsf{Equal}(x,x)\}$

(X4) $\{\mathsf{Equal}(y,y) \wedge \mathsf{Unequal}(z,z)\}\ x := y \oplus z\ \{\mathsf{Unequal}(x,x)\}$

Due to the commutativity of the xor, the role of y and z can be exchanged in all the rules above.

Block cipher: Since block ciphers are modeled as ideal ciphers, that is, functions picked at random among all functions from $\{0,1\}^\eta$ to $\{0,1\}^\eta$, the output of the function for a point on which the block cipher has never been computed is indistinguishable from a random value. This is expressed in rules (B1) to (B3), and also used in the proof of many other rules. Since the querying of a block cipher twice at any point is undesirable, we always require the invariant E as a precondition.

(B1) $\{\mathsf{Empty}\}\ x := \mathcal{E}(m_i)\ \{(\mathsf{Equal}(m_i,m_i) \wedge \mathsf{Equal}(x,x) \wedge \mathsf{Indis}(x;\mathsf{Var},\ell_{\mathcal{E}},\ell_{\mathcal{H}};\mathsf{Var}-x)) \vee$
$(\mathsf{Unequal}(x,x) \wedge \mathsf{Indis}(x))\}$

(B2) $\{\mathsf{E}(\mathcal{E};y;\emptyset) \wedge \mathsf{Unequal}(y,y)\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(x)\}$

(B3) $\{\mathsf{E}(\mathcal{E};y;\emptyset) \wedge \mathsf{Equal}(y,y)\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(x;\mathsf{Var},\ell_{\mathcal{E}},\ell_{\mathcal{H}};\mathsf{Var}-x) \wedge \mathsf{Equal}(x,x)\}$

(B4) $\{\mathsf{E}(\mathcal{E};y;\emptyset) \wedge \mathsf{Indis}(t;V;V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t;V,x;V',x)\}$ provided $\ell_{\mathcal{E}} \notin V$, if $t = y$

(B5) $\{\mathsf{E}(\mathcal{E};y;\emptyset) \wedge \mathsf{Indis}(t;V,\ell_{\mathcal{E}},y;V',y)\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t;V,\ell_{\mathcal{E}},x,y;V',x,y)\}$

(B6) $\{\mathsf{E}(\mathcal{E};y;\emptyset) \wedge \mathsf{E}(\mathcal{E};t;V,y)\}\ x := \mathcal{E}(y)\ \{\mathsf{E}(\mathcal{E};t;V,y)\}$

We also have rules similar to (B2) to (B5), with the invariant $\mathsf{E}(\mathcal{E};y;\emptyset)$ replaced by the invariant Empty, since both imply that the value of y is not in $\mathcal{L}_{\mathcal{E}}$.

Hash Function: We note that the distinguishing adversary, described in Section 3.1, does not have access to the random oracle. This is an unusual decision, but sufficient for our purpose since our goal is only to prove inequality of strings, not their indistinguishability from random strings. As a result, the rules for the hash function are essentially the same as those for the block cipher.

(H1) $\{\mathsf{H}(\mathcal{H}; y; \emptyset; \wedge)\mathsf{Unequal}(y, y)\}\ x := \mathcal{H}(y)\ \{\mathsf{Indis}(x)\}$

(H2) $\{\mathsf{H}(\mathcal{H}; y; \emptyset; \wedge)\mathsf{Equal}(y, y)\}\ x := \mathcal{H}(y)\ \{\mathsf{Indis}(x; \mathsf{Var}, \ell_{\mathcal{H}}; \mathsf{Var} - x) \wedge \mathsf{Equal}(x, x)\}$

(H3) $\{\mathsf{H}(\mathcal{H}; y; \emptyset; \wedge)\mathsf{Indis}(t; V; V')\}\ x := \mathcal{H}(y)\ \{\mathsf{Indis}(t; V, x; V', x)\}$ provided $\ell_{\mathcal{H}} \notin V$, if $t = y$

(H4) $\{\mathsf{H}(\mathcal{H}; y; \emptyset; \wedge)\mathsf{Indis}(t; V, \ell_{\mathcal{H}}, y; V', y)\}\ x := \mathcal{H}(y)\ \{\mathsf{Indis}(t; V, \ell_{\mathcal{H}}, x, y; V', x, y)\}$

(H5) $\{\mathsf{H}(\mathcal{H}; t; V, y; \})\ x := \mathcal{H}(y)\ \{\mathsf{H}(\mathcal{H}; t; V, y; \})$

For loop: The rule for the For loop simply states that if an indexed predicate $\psi(i)$ is preserved through one iteration of the loop, then it is preserved through the entire loop, for $p \leq l \leq q$. We discuss methods for finding such a predicate in Section 4.4.

(F1) $\{\psi(p-1)\}$ $\mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}\text{: } [\mathsf{cmd}_l]$ $\{\psi(q)\}$ provided $\{\psi(l-1)\}\ c_l\ \{\psi(l)\}$

Finally, we introduce a few general rules for consequence, sequential composition, conjunction and disjunction. Let $\phi_1, \phi_2, \phi_3, \phi_4$ be any four predicates in our logic, and let cmd, cmd_1, cmd_2 be any three commands.

(Csq) if $\phi_1 \Rightarrow \phi_2$, $\phi_3 \Rightarrow \phi_4$ and $\{\phi_2\}\mathsf{cmd}\{\phi_3\}$, then $\{\phi_1\}\mathsf{cmd}\{\phi_4\}$

(Seq) if $\{\phi_1\}\mathsf{cmd}_1\{\phi_2\}$ and $\{\phi_2\}\mathsf{cmd}_2\{\phi_3\}$, then $\{\phi_1\}\mathsf{cmd}_1; \mathsf{cmd}_2\{\phi_3\}$

(Conj) if $\{\phi_1\}\mathsf{cmd}\{\phi_2\}$ and $\{\phi_3\}\mathsf{cmd}\{\phi_4\}$, then $\{\phi_1 \wedge \phi_3\}\mathsf{cmd}\{\phi_2 \wedge \phi_4\}$

(Disj) if $\{\phi_1\}\mathsf{cmd}\{\phi_2\}$ and $\{\phi_3\}\mathsf{cmd}\{\phi_4\}$, then $\{\phi_1 \vee \phi_3\}\mathsf{cmd}\{\phi_2 \vee \phi_4\}$

Theorem 9 *A generic hash $Hash(m_1\|\ldots\|m_i, c)$:* **var** $\vec{x_i}$; cmd_i *computes an almost-universal hash function if $\{init\}\mathsf{cmd}_i\{UNIV(i)\}$.*

The theorem is the consequence of Proposition 8 and of the soundness of our Hoare logic. We then say that a sequence of predicates $[\phi_0, \ldots, \phi_n]$ is a proof that a program $[\mathsf{cmd}_1, \ldots, \mathsf{cmd}_n]$ computes an almost-universal hash function if $\phi_0 = true$, $\phi_n \Rightarrow UNIV(n)$ and for all i, $1 \leq n$, $\{\phi_{i-1}\}\ \mathsf{cmd}_i\ \{\phi_i\}$ holds.

4.4 Implementation

To use our method, we start at the beginning of the program, at each command apply every possible rule and, once done, test if the invariant $UNIV(n)$ holds at the end of the program.

4.4.1 — Invariant Filter

We say that ϕ is an *invariant on* x if ϕ is either $\mathsf{Equal}(x, y)$, $\mathsf{Unequal}(x, y)$, $\mathsf{E}(\mathcal{E}; x; V)$ $\mathsf{H}(\mathcal{H}; x; V;)$ or $\mathsf{Indis}(x; V_1, V_2)$. We say that an invariant ϕ on variable x is *obsolete for program p* if x does not appear anywhere in p and if $\neg(\phi \Rightarrow \mathsf{Unequal}(c_n, c_i))$ and $\neg(\phi \Rightarrow \mathsf{Equal}(m_i, m_i))$ for any i,

$1 \leq i \leq n$. The following theorem shows that once an invariant is obsolete, it can be discarded.

Theorem 10 *If there exists a proof $[\phi_0, \dots, \phi_n]$ that a program $p = [\mathsf{cmd}_1, \dots, \mathsf{cmd}_n]$ computes an almost-universal hash function, then there also exists a proof $[\phi'_0, \dots, \phi'_n]$ that p computes an almost-universal hash function where for each i, $\phi_i \Rightarrow \phi'_i$ and each ϕ'_i does not contain any obsolete invariants for $[\mathsf{cmd}_{i+1}, \dots, \mathsf{cmd}_n]$.*

The theorem is a consequence of the fact that, in our logic, the rules for creating an invariant on x following the execution of command $x := e$ only have as preconditions invariants on the variables in e. As a result, we can always filter out obsolete invariants after processing each commands.

Also, we note that the only commands that can make an invariant $\mathsf{Equal}(m_i, m_i)$ appear are those of the form $x := e$ in which m_i appears in e. As a result, if we find that, for some integer l, the invariant $\mathsf{Equal}(m_l, m_l)$ is not present in one of the conjunction of the current predicate (after transforming the predicate in DNF form) and that the variable m_l is no longer present in the rest of the program, then there is no longer any chance that it will satisfy the conjunction with $\bigwedge_{j=1}^{n} \mathsf{Equal}(m_j, m_j)$ from $UNIV(n)$. Therefore, we can also safely filter out all other invariants of the form $\mathsf{Equal}(m_i, m_i)$ from that conjunction.

We also add a *heuristic filter* to speed up the execution of our method. We make the hypothesis that the invariant $\mathsf{Indis}(c_n; V; \{c_1, \dots, c_{n-1}\})$ will be present at the end of the program, which the case for all our examples, so that we can filter out $\mathsf{Indis}(c_i; V; V')$ if $i < n$ and c_i is no longer present in the remainder of the program. In addition to speeding up the program, filtering out these invariants greatly simplifies the construction of loop predicates discussed in the next section. If we fail to produce a proof while using the heuristic filter, we simply attempt again to find a proof without it.

4.4.2 — Finding Loop Predicates

The programs describing the almost-universal hash function usually contains for loops. It is therefore necessary to have an automatic procedure to detect the predicate $\psi(i)$ that allows us to apply rule (F1). We now show a heuristic that can be used to construct such a predicate, and illustrate how it works by applying them to $Hash_{CBC}$, described in Section 4.2.1. One could easily verify that it also works on $Hash_{CBC'}$, $Hash_{HMAC}$ and $Hash_{PMAC}$.

Once we hit a command "$\mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}$: $[\mathsf{cmd}_l]$", we express the precondition in the form $\varphi(p-1)$. The classical method for finding a stable predicate consists in processing the instructions c_l contained in the loop to find the predicate $\psi(l)$ such that $\{\varphi(l-1)\}\ c_l\ \{\phi(l)\}$. If $\varphi(l) \Rightarrow \psi(l)$, then we have found an predicate such that $\{\varphi(l-1)\}\ c_l\ \{\varphi(l)\}$ and we can apply rule $(F1)$. Otherwise, we repeat this process with $\varphi'(l) = \varphi(l) \wedge \psi(l)$ until we find a stable predicate.

Unfortunately, for certain loops, one could repeat the process infinitely and never obtain a stable predicate. If, after a certain number n of iterations of the process above, we did not find a stable invariant[5], we decide that the classical method has failed and so we need a new heuristic to construct the stable predicate. We start over with invariant $\varphi(l-1)$, and process the code of the loop once to find invariant $\psi_1(l)$ such that $\{\varphi(l-1)\}\ c_l\ \{\psi_1(l)\}$. Then, we repeat this starting with invariant $\varphi(l-1) \wedge \psi_1(l-1)$ to find invariant $\psi_2(l)$ such that $\{\varphi(l-1) \wedge \psi_1(l-1)\}\ c_l\ \{\psi_2(l)\}$. By analyzing the predicates $\varphi(l)$, $\varphi(l) \wedge \psi_1(l)$ and $\varphi(l) \wedge \psi_1(l) \wedge \psi_2(l)$, we identify the predicate $\gamma(l)$ such that $\gamma(l)$ appears in $\varphi(l)$, $\gamma(l-1)$ appears in $\psi_1(l)$ and $\gamma(l-2)$ appears in $\psi_2(l)$. We then use $\varphi'(l) = \varphi(l) \wedge \bigwedge_{j=p-1}^{j=l-1} \gamma(j)$ as our new starting predicate. In [17] we give an example of this technique.

We programmed a tool in OCaml implementing our method for proving that the front end of MACs are almost-universal hash functions. The program requires about 1000 lines of code, and can successfully produce proofs of security for all the examples discussed in this paper in less than one second on a personal workstation. Our tool is available on [16].

4.5 Proving MAC Security

We prove the security of MACs in two steps: first we show that the 'compressing' part of the MAC is an almost-universal hash function family, and then we show that the last section of the MAC, when applied to an almost-universal hash function, results in a secure MAC. The following shows how a secure MAC can be constructed from an almost-universal hash function. The proof can be found in [BCK96, BR00, BR02], so we do not repeat them here.

Proposition 11 *Let $\mathcal{F}_\mathcal{E}$ be a family of block ciphers, $\mathcal{H} = \{h_i\}_{i \in \{0,1\}^\eta}$ and $\mathcal{H}' = \{h_i\}_{i \in \{0,1\}^\eta}$ be families of almost-universal hash function and $\mathcal{G}$ be a random oracle. If $h \overset{\$}{\leftarrow} \mathcal{H}$, $h_\mathcal{E} \overset{\$}{\leftarrow} \mathcal{H}'$, $\mathcal{E} \overset{\$}{\leftarrow} \mathcal{F}_\mathcal{E}$, $\mathcal{G}$ is sampled at random from all functions with the appropriate domain and range and $k, k_1, k_2 \overset{\$}{\leftarrow} \{0,1\}^\eta$, then the following hold:*

— $MAC_1(m) = \mathcal{E}(h_i(m))$ *is a secure MAC with key* $sk = (i, \mathcal{E})$.

— $MAC_2(m) = \mathcal{G}(l \| h_i(m))$ *is a secure MAC with key* $sk = (i, k)$.

— $MAC_3(m) = \begin{cases} \mathcal{E}_1(h_i(m')) \text{ where } m' = pad(m) \text{ if } m\text{'s length is not a multiple of } \eta \\ \mathcal{E}_2(h_i(m)) \text{ if } m\text{'s length is a multiple of } \eta \end{cases}$
is a secure MAC with key $sk = (i, \mathcal{E}_1, \mathcal{E}_2)$.

— $MAC_4(m) = \begin{cases} \mathcal{E}(h_\mathcal{E}(m') \oplus k_1) \text{ where } m' = pad(m) \text{ if } m\text{'s length is not a multiple of } \eta \\ \mathcal{E}(h_\mathcal{E}(m) \oplus k_2) \text{ if } m\text{'s length is a multiple of } \eta \end{cases}$
is a secure MAC with key $sk = (\mathcal{E}, k_1, k_2)$

Using $Hash_{CBC}$ with MAC_1 and MAC_3 yield the message authentication code DMAC and ECBC respectively, using $Hash_{CBC'}$ with MAC_3 and MAC_4 yield FCBC and XCBC, combining $Hash_{PMAC}$ and MAC_4 yield a four key construction of PMAC and using $Hash_{HMAC}$ with MAC_2 yield HMAC.

[5]The choice of the number of times the process is repeated is completely arbitrary, we choose to try only two iterations since it is sufficient for all our examples.

4.6 Conclusion

We presented a Hoare logic that can be used to automatically prove the security of constructions for almost-universal hash functions based on block ciphers and compression functions modeled as random oracles. We can then obtain a secure MAC by combining with a few operations, such as those presented in Section 4.5. Our method can be used to prove the security of DMAC, ECBC, FCBC, XCBC, a two-key variant of HMAC and a four-key variant of PMAC.

Part II

Voting Protocols

Motivations: Nowadays electronic voting protocols are more and more used to organize elections. Systems like Civitas [CCM08] or Prêt à voter [CRS05, RP05] have been developed. Moreover they have been used for government elections and referendums in the United Kingdom, Estonia and Switzerland as well as municipal elections in Canada and party primary or company elections in the United States and France. The fear of a voter against possible frauds is real, *e.g*: How to ensure that nobody can vote twice? How to be sure that nobody can learn the vote of somebody or that a vote is part of the final count? In several voting protocols such properties are usually achieved using cryptographic primitives like homomorphic encryption, *e.g.* Benaloh's encryption scheme [CB87]. A formal analysis of such protocols is crucial in order to win user's confidence.

Contributions: In Chapter 5, we first look at one of the first homomorphic encryption scheme proposed by Benaloh's [CB87]. We discover that for certain key parameters the scheme is ambiguous, this means that a cipher can be decrypt into two different plaintexts. We show using an example the impact of such a flaw. Then we propose a corrected version of the scheme and prove his correctness [15].

In Chapter 6, we propose a formal taxonomy of Privacy in voting protocols. After a first attempt in [8] to model the notion of "vote independence" *i.e.* that a voter cannot copy the vote of another one, we refine existing privacy definitions according several directions using applied π-calculus:

— Communication between the attacker and the targeted voter ($SwVP$, $SwRF$, and $SwCR$).

— Vote-Independence/Corrupted Voter: The attacker can control another voter or not (Insider/Outsider) (I and O).

— Security against forced-abstention-attacks: The attacker can force abstention of a voter or not (FA and PO).

This leads us to more fine grained definitions which allow us to compare the security levels of several voting protocols which was often difficult using existing notions [11].

Finally in Chapter 7, we propose a formal modeling for Privacy notions for protocols supported weighted vote in the applied π-calculus. We show that these notions coincide with existing one for non weighted votes under some hypothesis. We also prove that under realistic assumptions one coerced voter is equivalent to several [10]. This lead us to prove a new result of decomposition unicity in the π-calculus [14].

We also mention two other works related to privacy:

— DETECTIVE: proof-of-concept implementation for medical services with electronic health records [26]. This real system shows the feasibility of a secure medical application which respects patient privacy.

— A study of an e-auction protocol [21]. We discovered a flaw in an existing protocol and proposed a corrected version, with a formal analysis of security using AVISPA [AVI].

Chapter 5

Benaloh Revisited

TROUVER un chiffrement homomorphique n'est pas facile, en 1994 Benaloh inventa est un des premiers chiffrements homomorphiques. Nous avons montré que ce chiffrement est ambigü. C'est à dire qu'un message chiffré peut se déchiffrer en deux messages différents. Nous illustrons ici l'impact de ce problème sur un exemple de système de vote utilisant ce chiffrement. L'utilisation de "mauvais" paramètres peut inverser le résultat de l'élection. Nous avons proposé une version corrigée de ce chiffrement homomorphique, prouvé que cette nouvelle version est correcte et effectué une preuve de sécurité sémantique. Enfin nous avons montré qu'étonnamment la génération de tels paramètres n'est pas si rare [15].

Contents

We discover that Benaloh's encryption scheme is ambiguous. In general the ambiguity in the ciphertexts means that for a given cleartext $m \in \mathbb{Z}_r$ the value actually computed by the decryption algorithm is

$$m' = D(E_r(m)) \equiv m \bmod r' \tag{5.1}$$

with $r' \neq r$. Depending on the implementation of the discrete logarithm used by D (naive enumeration, baby steps/giant steps, possibly combined with a divide-and-conquer strategy when r is smooth) the value m' can be any of the values defined mod r that satisfy Equation (5.1). As the discrete logarithm algorithm used is not aware of the reduction from r to r', the impact in terms of computation time should be minimal (except for a naive increasing enumeration strategy which will always finish earlier, and where we are guaranteed to get the canonical solution mod r').

Outline: We first recall original Benaloh's scheme, then explain on a small example why it is ambiguous. We analyze the consequences of using a bad y parameter produced during the key generation for one application. Then we give a corrected version of the scheme and give a semantic security proof using the Decisional Subgroup Membership Problem (DSMP). We also explain that it occurs more often than expected, before comparing with other schemes in the conclusion. All detailled proofs are given in [15].

5.1 Original Benaloh's Scheme

Benaloh's "Dense Probabilistic Encryption" [Ben94] describes a homomorphic encryption scheme with a significant improvement in terms of expansion factor compared to Goldwasser-Micali [GM82]. For the same security parameter (the size of the RSA modulus n), the ciphertext is in both cases an integer mod n, but the cleartext in Benaloh's scheme is an integer mod r for some parameter r depending on the key, whereas the cleartext in Goldwasser-Micali is only a bit. When computing the expansion factor for random keys, we found that it is most of the times close to 2 while it is $\lceil \log_2(n) \rceil$ for Goldwasser-Micali. We now recall the three steps of the original scheme given in Benaloh's paper [Ben94].

Key Generation: The public and private key are generated as follows:

— Choose a block size r and two large primes p and q such that:
  - r divides $(p-1)$.
  - r and $(p-1)/r$ are relatively prime.
  - r and $q-1$ are relatively prime.
  - $n = pq$.

— Select $y \in (\mathbb{Z}_n)^* = \{x \in \mathbb{Z}_n : \gcd(x, n) = 1\}$ such that

$$y^{\varphi/r} \neq 1 \bmod n \tag{5.2}$$

where φ denotes $(p-1)(q-1)$.

The public key is (y, r, n), and the private key is the two primes p and q.

Encryption: If m is an element in $\mathbb{Z}_r$ and u a random number in $(\mathbb{Z}_n)^*$ then we compute the randomized encryption of m using the following formula:

$$E_r(m) = \{y^m u^r \bmod n : u \in (\mathbb{Z}_n)^*\}.$$

It is easily verified that:

$$E_r(m_1) \times E_r(m_2) = E_r(m_1 + m_2).$$

Decryption: We first notice that for any m, u we have:

$$(y^m u^r)^{(p-1)(q-1)/r} = y^{m(p-1)(q-1)/r} u^{(p-1)(q-1)} = y^{m(p-1)(q-1)/r} \mod n.$$

Since $m < r$ and $y^{(p-1)(q-1)/r} \neq 1 \mod n$, Benaloh concludes that $m = 0 \bmod r$ if and only if $(y^m u^r)^{(p-1)(q-1)/r} = 1 \bmod n$. So if $z = y^m u^r \mod n$ is an encryption of m, given the secret key (p, q) we can determine whether $m = 0 \bmod r$. If r is small, we can decrypt z by doing an exhaustive search of the smallest non-negative integer m such that $(y^{-m}z \bmod n) \in E_r(0)$. By precomputing values and using the baby-step giant-step algorithm it is possible to perform the decryption in time $O(\sqrt{r})$. Finally if r is smooth we can use classical index-calculus techniques. More details about these optimization of decryption are discussed in the original paper [Ben94].

We remark that there is a balance to find between three parameters in this cryptosystem:

— ease of decryption, which requires that r is a product of small prime powers,

— a small expansion factor, defined as the ratio between the size of the ciphertexts and the size of the cleartexts. Because p and q have the same size and $r \mid p-1$, this expansion factor is at least 2,

— strength of the private key, meaning that n should be hard to factorize. In the context of the P-1 factorization method [Pol74], a large smooth factor of $p-1$ is a definite weakness.

We notice that the cryptosystem proposed by Naccache-Stern [NS98] four years after Benaloh's scheme and based on the same approach addresses this issue and does not produce ambiguous encryption.

5.2 A Small Counter-Example with Huge Consequences

5.2.1 — Simple Counter-Example

We start by picking a secret key $n = pq = 241 \times 179 = 43139$, for which we can set $r = 15$. Algorithm 1 may be used to compute the maximal suitable value of the r parameter if you start by picking p and q at random, but a smaller and smoother value may be used instead, for an easier decryption.

Algorithm 1 Compute r from p and q.

```
$r \leftarrow p - 1$;
**while** $\gcd(q-1, r) \neq 1$ **do**
    $r \leftarrow r/\gcd(r, q-1)$;
**end while**
```

We verify that $r = 15$ divides $p - 1 = 240 = 16 \times 15$, r and $(p-1)/r = 16$ are relatively prime, $r = 15 = 3 \times 5$ and $q - 1 = 178 = 2 \times 89$ are coprime. Assume we pick $y = 27$, then $\gcd(y, n) = 1$ and $y^{(p-1)(q-1)/r} = 40097 \neq 1 \bmod n$ so according to Benaloh's key generation procedure all the original conditions are satisfied.

By definition, $y^1 12^r = 24187 \bmod n$ is a valid encryption of $m_1 = 1$, while $y^6 4^r = 24187 \bmod n$ is also a valid encryption of $m_2 = 6$. In fact we can verify that with this choice of y, the true cleartext space is now $\mathbb{Z}_5$ instead of $\mathbb{Z}_{15}$ (hence the ambiguity in decryption): first notice that in $\mathbb{Z}_p$, $y^5 = 27^5 = 8 = 41^{15} = 41^r$. This means that a valid encryption of 5 is also a valid encryption of 0. For any message m, the set of encryptions of m is the same as the set of encryptions of $m + 5$, hence the collapse in message space size. The fact that the message space size does not collapse further can be checked by brute force with this small set of parameters.

5.2.2 — One Application: Receipt-free Elections

In [BT94], Benaloh and Tuinstra propose an application of homomorphic encryption for designing new receipt-free secret-ballot elections. They describe two protocols which use a homomorphic encryption scheme and verify a list of properties. They also give in the appendix of the paper a precise description of an encryption scheme which satisfies their properties.

The new voting protocol uses the fact that the encryption is homomorphic and probabilistic. If we have two candidates Nicolas and Ségolène then the system associates for instance the ballot 0 for Nicolas and the ballot 1 for Ségolène. The main idea is that the server collects the m authenticated encrypted ballots $\{v_i\}_k$ corresponding to the choices v_i of the m voters. Then the server performs the multiplication of the ciphertexts to sum the votes and decrypts the product once to obtain the result. The number obtained corresponds to the number of votes n_S for Ségolène and the difference $m - n_S$ gives the number of votes for Nicolas.

Based on the simple counter-example proposed in 5.2.1, we construct a basic application of the first protocol proposed in [BT94]. We consider only 12 voters and the final result obtained by the server is $\{11\}_k$. It means that after decryption Ségolène has 11 votes and Nicolas has 1 vote. But if instead of computing the result 11 mod 15 we are taking the result modulo 5, then we obtain a result of 11 mod 5 = 1. This time the server concludes that Nicolas obtains 11 votes and Ségolène only 1. This example clearly shows that the flaw in the parameters generation process can have important consequences. In [15], we present other examples of the impact of this flaw.

5.3 Revisited Benaloh's Scheme

Let g be a generator of the group $(\mathbb{Z}_p)^*$, and since y is coprime with n, let α be the value in $\mathbb{Z}_{p-1}$ such that $y = g^\alpha \bmod p$. We are able to prove the Theorem 12[1]

Theorem 12 *The following properties are equivalent:*

1. *α and r are coprime;*
2. *decryption works unambiguously;*
3. *for all prime factors s of r, we have $y^{(\varphi/s)} \neq 1 \bmod n$.*

Property (2) is what we expect of the scheme, while (1) is useful to analyze the proportion of invalid y's and (3) is more efficient to verify in practice than (1), especially considering that in order to decrypt efficiently the factorization of r is assumed to be known.

5.4 Semantic Security of the Corrected Scheme

In [Gjo05], Kristian Gjøsteen formulates the security of several homomorphic encryption schemes in a common setting and relates the semantic security of the schemes to a generic problem (the Decisional Subgroup Membership Problem [DSMP]) which we recall here:

Problem [DSMP] Let G be an abelian group with subgroups K, H such that $G = KH$ and $K \cap H = \{1\}$. The *Decisional Subgroup Membership Problem* is to decide whether a given $g \in G$ is in K or not.

The cryptosystems by Goldwasser-Micali, Naccache-Stern, Okamoto-Uchiyama and Paillier respectively are shown to fit in this setting, with a proper definition for G (the ciphertexts space), H (coding the cleartexts) and K (the “cloak” space used to randomize encryptions). For example for Paillier's encryption, the ciphertext space is $G = (\mathbb{Z}_{n^2})^* \simeq (\mathbb{Z}_n)^* \times \mathbb{Z}_n$, the cleartexts coding subgroup H is the subgroup of order n (generated by $g = 1+n$) and K is the set of the invertible n-th powers mod n^2. This is consistent with the probabilistic encryption function

$$E_u(m) = (1+n)^m u^n \bmod n^2.$$

It can be verified quite easily that the following choices make the corrected version of Benaloh's scheme fit in this setting:

— $G = (\mathbb{Z}_n)^*$

— H the cyclic subgroup of order r of G

— K the set of invertible r-th powers in G

[1]The detailed proof is available in [15].

— the public element y must generate H.

Using the result in [Gjo05], the semantic security of our corrected scheme is therefore equivalent to the DSMP for K, that is, being able to distinguish r-th powers modulo n.

Although several homomorphic encryption schemes are analyzed in [Gjo05], Benaloh's is not, because it not satisfy all requirement by the framework proposed in [Gjo05]. Our correction ensures that the last condition is met, otherwise y could generate a strict subgroup of the intended group H. This explains why it was not listed by Kristian Gjøsteen, and provides a semantic security proof of the corrected version.

5.5 Probability of Failure of Benaloh's Scheme

We now estimate the probability of failure in the scheme as originally described. For this we need to count the numbers y that satisfy Equation (5.2) and not property (3) of Theorem 12. We call these values of y "faulty".

Lemma 13 *Equation (5.2) is equivalent to the statement:* $r \nmid \alpha$.

Since picking $y \in (\mathbb{Z}_p)^*$ at random is the same when seen $\bmod\ p$ as picking $\alpha \in \{0, \ldots, p-2\}$ at random, we can therefore conclude that the proportion ρ of faulty y's is exactly the proportion of non-invertible numbers mod r among the non-zero mod r. So $\rho = 1 - \frac{\varphi(r)}{r-1}$. We notice that this proportion depends on r only, and it is non-zero when r is not a prime. Since decryption in Benaloh's scheme is essentially solving a discrete logarithm in the subgroup of $(\mathbb{Z}_p)^*$ of order r, the original scheme recommends to use r as a product of small primes' powers, which tends to increase ρ. In fact, denoting by (p_i) the prime divisors of r we have:

$$\begin{aligned}
\rho &= 1 - \frac{\varphi(r)}{r-1} \\
&= 1 - \frac{r}{r-1}\frac{\varphi(r)}{r} \\
&= 1 - \frac{r}{r-1}\prod_i \frac{p_i - 1}{p_i} \\
&\approx 1 - \prod_i \frac{p_i - 1}{p_i}
\end{aligned}$$

which shows that the situation where decryption is easy also increases the proportion of invalid y's when using the initial description of the encryption scheme. As a practical example, assume we pick two 512 bits primes p and q as

$$\begin{aligned}
p &= 2 \times (3 \times 5 \times 7 \times 11 \times 13) \times p' + 1 \\
p' &= 44648045054753903095484598728624196228702516885 08955\backslash \\
&\quad 503737449698209045631060122203397227538517117358 5381\backslash \\
&\quad 391469152467701810702240466022543944167995 3592 \\
q &= 10055855947456947824680518748654384595609524365 44429\backslash \\
&\quad 503329267108279132302255516023260140572362517757 0767\backslash \\
&\quad 523893639864538140315412108959927459825236754568279.
\end{aligned}$$

Then

$$\begin{aligned}
\gcd(q-1, p-1) &= 2 \\
r &= (3 \times 5 \times 7 \times 11 \times 13) \times p' \\
\rho &= 1 - \frac{r}{r-1} \times \frac{2}{3} \times \frac{4}{5} \times \frac{6}{7} \times \frac{10}{11} \times \frac{12}{13} \times \frac{p'-1}{p'} \\
\rho &> 61\%.
\end{aligned}$$

This example was constructed quite easily: first we take p' of suitable size, and multiply its value until $p = k \times p' + 1$ is prime. Then we generate random primes q of suitable size until the condition $\gcd(p-1, q-1) = 2$ is verified; it took less than a second on a current laptop using Sage [S+10].

5.6 Conclusion

We first discuss some schemes related to that of [Ben94].

— In [BT94], Benaloh and Tuinstra describe a cryptosystem which closely resembles that of [Ben94], but the conditions given on r are less strict. Let us recall briefly the parameters of the cryptosystem as described in [BT94]:
  - $r \mid p-1$ but $r^2 \nmid p-1$.
  - $r \nmid q-1$.
  - y is coprime with n and $y^{(p-1)(q-1)/r} \neq 1 \bmod n$.

  It is clear that $r^2 \nmid p-1$ is weaker than $\gcd((p-1)/r, r) = 1$, and that $r \nmid q-1$ is weaker than $\gcd(q-1, r) = 1$. Therefore any set of parameters satisfying [Ben94] are also valid parameters as defined in [BT94].

  Unfortunately the condition imposed on y is the same and still insufficient, and finding counter-examples is again a matter of picking α not coprime with r. Our theorem still stands for this cryptosystem if you replace condition (3) by the following condition:

$$\text{For all prime factors } s \text{ of } r, \text{ we have } y^{(p-1)/s} \neq 1 \bmod p. \tag{5.3}$$

— Going back in time, the scheme of Goldwasser and Micali [GM82] can be seen as a precursor of [BT94] with a fixed choice of $r = 2$. The choice of y in [GM82] as a quadratic non-residue mod n is clearly an equivalent formulation of condition (5.3).

— Before [Ben94] and [BT94], the scheme was defined by Benaloh in [Ben87], with the parameter r being a prime. In this case our condition (3) is the same as the one proposed by Benaloh, and the scheme in this thesis is indeed correct. The main difference between the different versions proposed afterwards and this one is that it is not required for r to be prime, which leads in some cases to ambiguous ciphers. This remark clearly shows that all details are important in cryptography and that the problem we discovered is subtle because even Benaloh himself did not notice it.

— Finally the scheme proposed by Naccache and Stern [NS98] is quite close to the one proposed in [Ben87] but with a parameterization of p and q. It makes decryption correct, efficient, and leaves the expansion factor as an explicit function of the desired security level with respect to methods of factoring taking advantage of this specific form of n, like the $P-1$ method [Pol74] (the expansion is essentially the added size of the big cofactors of $p-1$ and $q-1$). If we drop this requirement that $p-1$ and $q-1$ have big cofactors, their scheme becomes a corrected generalization of Benaloh's, so application writers should probably use Naccache-Stern's scheme directly. We note that a modulus size of 768 bits was considered secure at the time, a fact disproved twelve years later [KAF$^+$10] only!

We have shown that the original definition of Benaloh's homomorphic encryption does not give sufficient conditions in the choice of public key to get an unambiguous encryption scheme. We also explain on one example what can be the consequences of the use of the original Benaloh scheme. We propose a necessary and sufficient condition which fixes the scheme and prove this semantic security. Finally we show that the probability of choosing an incorrect public key is non negligible parameters where decryption is efficient. In fact, it is surprising this result was not found before, considering the number of applications built on the homomorphic property of Benaloh's scheme. This strongly suggests this scheme was rarely implemented or even worse, implementations were rarely well tested.

Chapter 6

Hierarchy of Privacy Properties

On remarque que les protocoles de vote électronique sont de plus en plus utilisés dans les démocracies modernes. Il est donc important d'identifier, de formaliser et de vérifier les propriétés qu'ils doivent assurer. Dans ce cadre nous avons étendu en π-calcul appliqué les propriétés existantes pour le respect de la vie privée (Privacy). Ceci afin de prendre en compte à la fois :

— les communications entre l'attaquant et le votant attaqué ($SwVP$, $SwRF$, et $SwCR$).

— le contôle de part l'adversaire ou non d'un autre participant au vote (I et O).

— la possibilité de forcer un votant à s'abstenir ou non (FA et PO).

Nous obtenons alors un ensemble de propriétés plus précises que les propriétés existantes, mais aussi des relations claires entres ces propriétés. Ceci nous permet de comparer les niveaux de sécurité des protocoles de la littérature [11].

Contents

6.1 Introduction

Since the existence of democracy, populations have been often consulted about critical decisions. To achieve it, a lot of voting systems have been designed. That's why since the 1980's, electronic voting schemes have attracted intrest of many researchers. While more and more protocols have been developed, the set of security properties that a protocol has to achieve has evolved. Thus, researchers keep combining existing or created cryptographic primitives to construct efficient electronic voting schemes, with respect to these security requirements. Researchers have identified numerous security properties that are required for secure voting systems and protocols. These properties can be classified into three categories: i) Correctness and robustness properties, ii) verifiability and iii) privacy properties. Typical correctness and robustness properties include:

— *Eligibility*: Only the registered voters can vote, and nobody can submit more votes than allowed (typically only one).

— *Fairness*: No preliminary results that could influence other voters' decisions are made available.

— *Robustness*: The protocol can tolerate a certain number of misbehaving voters.

Verifiability is usually split into two properties:

— *Individual Verifiability*: Each voter can check whether his vote was counted correctly.

— *Universal Verifiability*: Anybody can verify that the announced result corresponds to the sum of all votes.

Last different levels of privacy can be achieved:

— *Vote-Privacy*: The votes are kept private. This can also be modeled as an unlinkability between the voter and his vote.

— *Receipt-Freeness*: A voter cannot construct a receipt which allows him to prove to a third party that he voted for a certain candidate. This is to prevent vote-buying.

— *Coercion-Resistance*: Even when a voter interacts with a coercer during the entire voting process, the coercer cannot be sure whether he followed his instructions or actually voted for another candidate.

However the design of complex protocols to fulfill all these partly antipodal requirements [CMFP$^+$06] is notoriously difficult and error-prone. In particular in the area of privacy properties there is a great variety of models. Recent research shows that some existing definitions might be insufficient: Smyth and Cortier [SC11] pointed out that the ability to copy another voter's vote can enable attacks on privacy.

Our Contributions.

— We propose a new family of privacy notions which allows us to assess of the level of privacy provided by a voting protocol. These notions are based on formal definitions of the classical notions (Vote-Privacy, Receipt-Freeness and Coercion-Resistance) in the applied π-calculus [AF01a], with a refined protocol model and including new attacks. The resulting family gives a deep understanding of the different levels and requirements for privacy. Additionally our notions generalize [8] and deal with attacks based on vote-copying that were not captured in the model by Delaune et al. [DKR09].

— A deep understanding of privacy properties, and in particular the relationship between the different notions, is a prerequisite for the correct design of voting protocols. We provide a thorough comparison of existing and new notions.

— In [9], we analyze several case studies to illustrate our definitions [FOO92, Oka96, JCJ05, BMQR07]. To automatically analyze these protocols, we use ProSwapper [KSR10] and ProVerif [BAF08].

Related Work. Previous research on formal verification of voting protocols concerned privacy properties (privacy, receipt-freeness and coercion-resistance) [DKR09, DKR10, MN06, BHM08][8, 9], election verifiability [SRKK10, KRS10], or both [JCJ05, JCJ02].

Although the informal specifications of the properties are very general, most of the formal models and definitions in the literature are tailored to a specific type of protocols. Many protocols where in fact developed together with their own definitions (e.g. [MN06, JCJ05, JCJ02]) and analyzed by hand in the original paper.

Juels et al. [JCJ05] (which became the bases for Civitas [CCM08]) were the first to give a formal, but computational definition of coercion-resistance. It was later translated to the applied π-calculus and automated using ProVerif [BHM08]. However – as their protocol is based on voting "credentials" – credentials also appear in the definition. Their model is thus unsuitable for protocols that do not use credentials (e.g. Bingo Voting [BMQR07] or the protocol by Lee et al. [LBD$^+$04]).

More general definitions were developed by Delaune, Kremer and Ryan [DKR09, DKR10]. They express different levels of privacy as observational equivalence in the applied π-calculus [AF01a]. An attacker should not be able to distinguish one case in which the voter complies with the coercer's instructions and another in which he only pretends to do so and votes as he wishes. Unfortunately their definitions are too strong for the protocol proposed by Juels et al.: since in one case the targeted voter complies and posts only one correct ballot, and in the other he secretly posts his actual ballot and a fake one to cheat the coercer, both cases can be distinguished by counting the ballots.

Smyth and Cortier [SC10, SC11] showed that being able to copy votes can compromise privacy if the number of participants is small or a noticeable fraction of voters can be corrupted. For example in the case of three voters, the third voter can try to copy the first voter's vote and submit it as his vote. This will result in (at least) two votes for the candidate chosen by the first voter and his choice can thus be inferred from the result. They also formally analyzed ballot secrecy in Helios using an adaption of the model by Delaune, Kremer and Ryan. However in [8, 9] we demonstrate that the protocol by Lee et al. [LBD$^+$04], shown to be coercion-resistant in DKR model [DKR09], is vulnerable to a vote-copy attacks. It clearly shows DKR model is not sufficient to capture vote-independence. Moreover the DKR model can not deal with weighted votes.

Küsters and Truderung [KT09] proposed a first model independent definition of coercion-resistance for voting protocols. Their definition has to be instantiated using a concrete formal model. The exact security level can be defined with respect to certain chosen goals, and excluding explicit special cases. In contrast to our family of notions, their definition is based on traces and not bisimulations, and they only define coercion-resistance.

Langer et al. [LJP10] developed verifiability definitions and privacy notions based on (un-)linkability between a voter and his vote. Similarly to Küsters and Truderung, their definitions have to be instantiated with a concrete formal process and attacker model.

Computational definitions of receipt-freeness [CG96] and coercion-resistance [UMQ10] that can be applied to other applications than voting have also been proposed. Completely application-independent anonymity notions were proposed by Bohli and Pashalidis [BP09]. Although their definitions are very general, the application on voting protocols results in – for this context – rather unusual privacy notions (Pseudonymity etc.), compared to the classic properties such as receipt-freeness or coercion-resistance.

Outline: In the next section, we give a brief introduction of the applied π-calculus. In Section 6.3 starts by explaining informally our privacy notions and subsequently gives the formal definitions. In the next section, we discuss the relationship between the different notions and explain the hierarchy implied by the definitions. In the last section, we conclude.

6.2 Preliminaries

In the first part of this section, we introduce the applied π-calculus. We use it to model voting protocols and express privacy properties. In the second part, we define our model of a voting protocol in the applied π-calculus.

6.2.1 — The Applied π-Calculus

The applied π-calculus [AF01a] is a formal language to describe concurrent processes. The calculus consists of *names* (which typically correspond to data or channels), *variables*, and a *signature* Σ of *function symbols* which can be used to build *terms*. Functions typically include encryption and decryption (for example $\mathtt{enc}(message, key)$, $\mathtt{dec}(message, key)$), hashing, signing etc. Terms are correct (i.e. respecting arity and sorts) combinations of names and functions. To model equalities we use an equational theory E which defines a relation $=_E$. A classical example which describes the correctness of symmetric encryption is $\mathtt{dec}(\mathtt{enc}(message, key), key)$ $=_E$ *message*.

There are two types of processes in the applied π-calculus: *plain processes* and *extended processes*. Plain processes are constructed using the following grammar:

P, Q, R :=	plain processes
0	null process
$P\|Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction ("new")
$\texttt{if } M = N \texttt{ then } P \texttt{ else } Q$	conditional
$\texttt{in}(u, x).P$	message input
$\texttt{out}(u, x).P$	message output

Extended processes are plain processes or active substitutions:

A, B, C :=	active processes
P	plain process
$A\|B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{^{M}/_{x}\}$	active substitution

The substitution $\{^{M}/_{x}\}$ replaces the variable x with term M. We denote by $fv(A)$, $bv(A)$, $fn(A)$, $bn(A)$ the free variables, bound variables, free names or bound names respectively. A process is *closed* if all variables are bound or defined by an active substitution.

The *frame* $\Phi(A)$ of an extended process A is obtained when replacing all plain processes in A by 0. This frame can be seen as a representation of what is statically known to the exterior about a process. The domain $\texttt{dom}(\Phi)$ of a frame Φ is the set of variables for which Φ defines a substitution. An evaluation context $C[\_]$ denotes an extended process with a hole for an extended process.

The semantics of the calculus are given by *Structural equivalence* ($\equiv$), which is defined as the smallest equivalence relation on extended processes that is closed under application of evaluation contexts, α-conversion on names and variables such that:

PAR-0	$A\|0 \equiv A$
PAR-A	$A\|(B\|C) \equiv (A\|B)\|C$
PAR-C	$A\|B \equiv B\|A$
NEW-0	$\nu n.0 \equiv 0$
NEW-C	$\nu u.\nu v.A \equiv \nu v.\nu u.A$
NEW-PAR	$A\|\nu u.B \equiv \nu u.(A\|B)$ if $u \notin fn(A) \cup fn(b)$
REPL	$!P \equiv P\|!P$
REWRITE	$\{^{M}/_{x}\} \equiv \{^{N}/_{x}\}$ if $M =_E N$
ALIAS	$\nu x.\{^{M}/_{x}\} \equiv 0$
SUBST	$\{^{M}/_{x}\}\|A \equiv \{^{M}/_{x}\}\|A\{^{M}/_{x}\}$

and extended by *Internal reduction* ($\rightarrow$), the smallest relation on extended processes closed by structural equivalence and application of evaluation contexts such that:

COMM	$\mathtt{out}(a,x).P \mid \mathtt{in}(a,x).Q$	$\rightarrow$	$P \mid Q$
THEN	$\mathtt{if}\ M = M\ \mathtt{then}\ P\ \mathtt{else}\ Q$	$\rightarrow$	P
ELSE	$\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q$	$\rightarrow$	Q
	for any ground terms such that $M \neq_E N$		

To describe the interaction of processes with the exterior, we use labeled operational semantics ($\xrightarrow{\alpha}$) where α can be an input or an output of a channel name or a variable of base type.

IN	$\mathtt{in}(a,x).P \xrightarrow{\mathtt{in}(a,M)} P\{^{M}/_{x}\}$
OUT-ATOM	$\mathtt{out}(a,u).P \xrightarrow{\mathtt{out}(a,u)} P$
OPEN-ATOM	$\dfrac{A \xrightarrow{\mathtt{out}(a,u)} A' \qquad u \neq a}{\nu u.A \xrightarrow{\nu u.\mathtt{out}(a,u)} A'}$
SCOPE	$\dfrac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$
PAR	$\dfrac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$
STRUCT	$\dfrac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}$

Labeled transitions are not closed under the evaluation contexts. Note that a term M cannot be output directly. Instead, we have to assign M to a variable, which can then be output. This is to model that e.g. the output of $\mathtt{enc}(m,k)$ does not give the context access to m. In our definitions we will use the following equivalence and bisimilarity properties:

Definition 14 (Equivalence in a Frame) *Two terms M and N are equal in the frame ϕ, written $(M = N)\phi$, if and only if $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma = N\sigma$, and $\{\tilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\tilde{n}$ and some substitution σ.*

Definition 15 (Static Equivalence ($\approx_s$)) *Two closed frames ϕ and ψ are statically equivalent, written $\phi \approx_s \psi$, when $dom(\phi) = dom(\psi)$ and when for all terms M and N $(M = N)\phi$ if and only if $(M = N)\psi$. Two extended processes A and B are statically equivalent ($A \approx_s B$) if their frames are statically equivalent.*

The intuition behind this definition is that two processes are statically equivalent if the messages exchanged with the environment cannot be distinguished by an attacker (i.e. all operations on both sides give the same results). This idea can be extended to *labeled bisimilarity.*

Definition 16 (Labeled Bisimilarity ($\approx_l$))
Labeled bisimilarity is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that $A\ \mathcal{R}\ B$ implies

1. $A \approx_s B$,

2. *if $A \rightarrow A'$, then $B \rightarrow B'$ and $A' \mathcal{R} B'$ for some B',*
3. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq dom(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some B'.*

In this case each interaction on one side can be simulated by the other side, and the processes are statically equivalent at each step during the execution, thus an attacker cannot distinguish both sides.

To formally describe the interaction between a voter and the attacker, we use the following two definitions. The first one turns a process P into another process P^{ch} that reveals all its inputs and secret data on the channel ch. This can be seen as trying to construct a receipt.

Definition 17 (Process P^{ch} [DKR09]) *Let P be a plain process and ch be a channel name, P^{ch} is defined by:*
— $0^{ch} \hat{=} 0$,

— $(P|Q)^{ch} \hat{=} P^{ch}|Q^{ch}$,

— $(\nu n.P)^{ch} \hat{=} \nu n.\mathtt{out}(ch, n).P^{ch}$ *when n is a name of base type,*

— $(\nu n.P)^{ch} \hat{=} \nu n.P^{ch}$ *otherwise,*

— $(\mathtt{in}(u, x).P)^{ch} \hat{=} \mathtt{in}(u, x).\mathtt{out}(ch, x).P^{ch}$ *when x is a variable of base type,*

— $(\mathtt{in}(u, x).P)^{ch} \hat{=} \mathtt{in}(u, x).P^{ch}$ *otherwise,*

— $(\mathtt{out}(u, M).P)^{ch} \hat{=} \mathtt{out}(u, M).P^{ch}$,

— $(!P)^{ch} \hat{=} !P^{ch}$,

— $(\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q)^{ch} \hat{=} \mathtt{if}\ M = N\ \mathtt{then}\ P^{ch}\ \mathtt{else}\ Q^{ch}$.

In the remainder we assume that $ch \notin fn(P) \cup bn(P)$ before applying the transformation. The second definition does not only reveal the secret data, but also takes orders from an outsider before sending a message or branching, i.e. the process is under complete remote control.

Definition 18 (Process P^{c_1,c_2} [DKR09]) *Let P be a plain process and c_1, c_2 be channel names, We define P^{c_1,c_2} is defined by:*
— $0^{c_1,c_2} \hat{=} 0$,

— $(P|Q)^{c_1,c_2} \hat{=} P^{c_1,c_2}|Q^{c_1,c_2}$,

— $(\nu n.P)^{c_1,c_2} \hat{=} \nu n.\mathtt{out}(c_1, n).P^{c_1,c_2}$ *when n is a name of base type,*

— $(\nu n.P)^{c_1,c_2} \hat{=} \nu n.P^{c_1,c_2}$ *otherwise,*

— $(\mathtt{in}(u, x).P)^{c_1,c_2} \hat{=} \mathtt{in}(u, x).\mathtt{out}(c_1, x).P^{c_1,c_2}$ *when x is a variable of base type and x is a fresh variable,*

— $(\mathtt{in}(u, x).P)^{c_1,c_2} \hat{=} \mathtt{in}(u, x).P^{c_1,c_2}$ *otherwise,*

— $(\mathtt{out}(u, M).P)^{c_1,c_2} \hat{=} \mathtt{in}(c_2, x).\mathtt{out}(u, x).P^{c_1,c_2}$,

— $(!P)^{c_1,c_2} \hat{=} !P^{c_1,c_2}$,

— $(\texttt{if } M = N \texttt{ then } P \texttt{ else } Q)^{c_1,c_2} \hat{=} \texttt{in}(c_2, x).\texttt{if } x = \mathit{true} \texttt{ then } P^{c_1,c_2} \texttt{ else } Q^{c_1,c_2}$ *where and x is a fresh variable and true is a constant.*

To hide the output of a process, we use the following definition.

Definition 19 (Process $A^{\setminus out(ch,\cdot)}$ [DKR09]) *Let A be an extended process. The process $A^{\setminus out(ch,\cdot)}$ is defined by $\nu ch.(A|!in(ch, x))$.*

6.2.2 — Voting Protocol and Process

First of all, we define the notion of a *voting protocol.* Informally, a voting protocol specifies the processes executed by voters and authorities.

Definition 20 (Voting Protocol) *A voting protocol is a tuple $(V, A_1, \ldots, A_m, \tilde{n})$ where V is the process that is executed by the voter, the A_j's are the processes executed by the election authorities, and $\tilde{n}$ is a set of private channels.*

Note that the protocol only defines one process V which will be instantiated for each voter. Yet here may be several authorities, for example a registrar, a bulletin board, a mixer, a tallier, . . . In our definitions we reason about privacy using concrete instances of a voting protocol. An instance is called a *Voting Process.*

Definition 21 (Voting Process) *A voting process of a voting protocol $(V, A_1, \ldots, A_m, \tilde{n})$ is a closed plain process*

$$\nu\tilde{m}.(V\sigma_{id_1}\sigma_{f_1}\sigma_{v_1}| \ldots |V\sigma_{id_n}\sigma_{f_n}\sigma_{v_n}|A_1| \ldots |A_l)$$

where $l \leq m$, $\tilde{m}$ includes the secret channel names, $V\sigma_{id_i}\sigma_{v_i}\sigma_{f_i}$ are the processes executed by the voters where:

— *σ_{id_i} is a substitution assigning the identity to a process (this determines for example the secret keys),*

— *σ_{v_i} specifies the vote(s) and if the voter abstains,*

— *and σ_{f_i} defines the other behavior, in particular if fake votes are issued,*

and A_js are the election authorities which are required to be honest.

We notice that if an authority is not supposed to be honest, it is not modeled and left to the context, i.e. the attacker (thus $l \leq m$).

Note also that each voter runs the same process V, which is instantiated with a different σ_{id_i} (his identity), σ_{v_i} (his vote(s)) and σ_{f_i} (the fakes). If a protocol does not allow fakes, σ_{f_i} is empty.

This model allows us to reason about protocols where voters can submit fake ballots and/or several real ballots, even if only one of them is actually counted (like in the one by Juels et al. [JCJ05]). In that case σ_{v_i} determines those who are actually counted, and σ_{f_i} the others.

Note that as voting always includes a deadline, we will only consider finite processes V with a finite number of choices.

Example As a running example, we consider the following simple voting protocol.

Informal description: To construct his ballot, each voter encrypts his vote with the administrator's public key and signs it using his secret key. The resulting ballot is posted on the bulletin board. After the voting deadline is over, the administrator checks if each ballot is signed by an eligible voter. He then decrypts the correct ballots and publishes the result.

Formal description in our model: The protocol is a tuple $(V, A, \emptyset)$ where

$A \;=\; \texttt{in}(ch, (sig, vote)).$ $\texttt{if checksign}(sig, pkv) = vote$ $\texttt{then sync } 1.$ $\texttt{out}(chR, \texttt{dec}(vote, ska))$	$V \;=\; \nu r.$ $\texttt{let } evote = \texttt{enc}(v, pka, r) \texttt{ in}$ $\texttt{out}(ch, (\texttt{sign}(evote, skv), evote))$

where – by abuse of notation, this can rewritten in the "pure" calculus – `sync` 1 is a synchronization point as defined by ProSwapper [KSR10]. A substitution determining the identity of a voter would in this case assign the secret key, e.g. $\sigma_{id_k} = \{^{sk_k}/_{skv}\}$. The substitution specifying the vote as for example a vote for candidate a would be $\sigma_{v_k} = \{^{a}/_{v}\}$. As the protocol does not specify the possibility to create fakes, σ_{v_k} is the empty substitution.

To facilitate notation we denote by S and S' two contexts which are like voting processes but with holes for two and three voters respectively.

Definition 22 (S and S') *We define evaluation contexts S and S' such that*

— *S is like a voting process, but has a hole instead of three processes among $(V\sigma_{id_i}\sigma_{v_i}\sigma_{f_i})_{1\leq i\leq n}$.*

— *S' is like as voting process, but has a hole instead of two processes among $(V\sigma_{id_i}\sigma_{v_i}\sigma_{f_i})_{1\leq i\leq n}$.*

Finally, we formally define what it means for a voting process to abstain. An abstaining voter does not send any message on any channel, in particular no ballot. In the real world, this would correspond to a voter that does not even go to polling station. This is stronger than just voting for a particular "null" candidate $\perp$, which will still result in sending a ballot (a *blank* vote).

Definition 23 (Abstention) *A substitution σ_{v_i} makes a voter abstain if $V\sigma_{id_i}\sigma_{v_i} \approx_l 0$.*

Note that abstention is determined by σ_{v_i} only, so the voter abstains for any σ_{f_i}.

6.3 Hierarchy of Privacy Properties

In our approach, the attacker targets one voter (the *targeted voter*) and tries to extract information about the targeted voter's vote(s). If the attacker knows the votes of all other voters, he can infer the targeted voter's vote from the result. Thus we suppose that he is unsure about the vote(s) of one other voter (the *counterbalancing voter*).

We express privacy as an observational equivalence. Intuitively, an attacker should not be able to distinguish between an execution in which the targeted voter behaves and votes as the attacker wishes, and another execution where he only pretends to do so and votes differently. To ensure that the attacker cannot tell the difference by just comparing the result, the counterbalancing voter will compensate the different vote.

Starting from the definitions of Coercion-Resistance, Receipt-Freeness and Vote-Privacy in the literature [DKR09, BHM08, JCJ05] we propose extensions in the four following dimensions: Communication between the attacker and the targeted voter, Vote-Independence, security against forced-abstention-attacks and knowledge about the behavior of the counterbalancing voter.

1. *Communication between the attacker and the targeted voter:* We define three different levels:
   (a) In the simplest case, the attacker only observes publicly available data and communication. We call this case Vote-Privacy, denoted $SwVP$.
   (b) In the second case, the targeted voter tries to convince the attacker that he voted for a certain candidate by revealing his secret data. Yet the attacker should not be able to determine if he actually sent his real data, or a fake receipt. We call this case Receipt-Freeness, denoted $SwRF$.
   (c) In the strongest case, the voter pretends to be completely under the control of the attacker, i.e. he reveals his secret data and follows the intruder's instructions. Yet the attacker should be unable to determine if he complied with his instructions or if he only pretended to do so. We call this case Coercion-Resistance, denoted $SwCR$.

   It is easy to see that Coercion-Resistance is stronger than Receipt-Freeness, which is stronger than Vote-Privacy ($SwCR > SwRF > SwVP$).
2. *Vote-Independence/Corrupted Voter:* The attacker may control another legitimate voter (neither the targeted nor the counterbalancing voter). In that case he could be able to compromise privacy by trying to relate the corrupted voter's vote to the targeted voter's vote (e.g. by copying it) or using the corrupted voter's secret data, such as his credentials or keys. In our definitions, we distinguish two case for Eve (the attacker):
   (a) Eve is an Outsider (denoted O): The attacker is an external observer.
   (b) Eve is an Insider (denoted I): The attacker has a legitimate voter under his control.

   Intuitively, Insider is the stronger setting ($I > O$).
3. *Security against forced-abstention-attacks:* A protocol can ensure that a voter can still vote as intended, although a coercer wants him to abstain. Note that in contrast to

the literature [JCJ05, BHM08], we define this property independently from of Coercion-Resistance, as we also want to apply it in the case of Vote-Privacy. Our model expresses this by requiring the observational equivalence to hold:

(a) in any case, i.e. even if the voter is forced to abstain. We call this case security against Forced-Abstention-Attacks, denoted FA.

(b) if the targeted voter does not abstain from voting (i.e. always participates). We call this case Participation Only, denoted PO.

In this dimension security against Forced-Abstention-Attacks is a stronger property than Participation Only ($FA > PO$).

4. *Knowledge about the behavior of the counterbalancing voter*: To model different knowledge about the behavior of the counterbalancing voter, we consider two cases:

   (a) The observational equivalence holds for any behavior of this voter, i.e. any σ_{f_i}. This models an attacker that knows if the counterbalancing voter is going to post fake ballots, or a situation where there is no "noise" (=fake ballots) on the bulletin board. We call this case Any Behavior, denoted AB.

   (b) The observational equivalence holds for at least one behavior of this voter, which may additionally change, i.e. one σ_{f_i} and one $\sigma_{f'_i}$. In this case, the attacker is unsure about the number of fake ballots, i.e. there is enough noise. We call this case Exists Behavior, denoted EB.

   Any Behavior is stronger than EB ($AB > EB$).

The strongest possible property is thus $SwCR^{I,FA,AB}$, the weakest $SwVP^{O,PO,EB}$. If we leave out the parameter, we take the weakest setting as a default, i.e. $SwVP$ denotes $SwVP^{O,PO,EB}$.

6.3.1 — Definitions in the applied π-calculus

Our definition is parameterized using the following parameters (as explained above):

— *Privacy* $= \{SwCR, SwRF, SwVP\}$ ("Coercion-Resistance", "Receipt-Freeness" or "Vote-Privacy").

— *Eve* $= \{I, O\}$ ("Insider" or "Outsider").

— *Abs* $= \{FA, PO\}$ ("Security against Forced-Abstention-Attacks"or "Participation Only").

— *Behavior* $= \{AB, EB\}$ ("Any Behavior"or "Exists Behavior").

Definition 24 ($Privacy^{Eve,Abs,Behavior}$) *A protocol fulfills* $Privacy^{Eve,Abs,Behavior}$ *if for any voting process* $\mathcal{S}$ *there exists a process* V' *and for any substitution* σ_{f_A} *and* σ_{f_C}*, and any context* A *such that* $A = \nu \tilde{ch}.(\_\,|A'^{chout}))$ *where* $\tilde{ch}$ *are all unbound channels and names in* A' *and in the "hole",*

— *if Behavior is EB: there exist substitutions* σ_{f_B}, $\sigma_{f'_B}$ *and* $\sigma_{f'_A}$,

— *if Behavior is AB: for any substitutions* $\sigma_{f_B} = \sigma_{f'_B}$ *there exists a substitution* $\sigma_{f'_A}$,

such that for all votes σ_{v_A} and σ_{v_B} where $V\sigma_{v_B}$ does not make a voter abstain[1]*, one of the following holds depending on the privacy setting:*
— if Privacy is Vote-Privacy (SwVP):

$$A\left[\mathcal{S}\left[V\sigma_{id_A}\sigma_{f_A}\sigma_{v_A}|V\sigma_{id_B}\sigma_{f_B}\sigma_{v_B}|\mathcal{V}_C\right]\right] \approx_l A\left[\mathcal{S}\left[V\sigma_{id_A}\sigma_{f'_A}\sigma_{v_B}|V\sigma_{id_B}\sigma_{f'_B}\sigma_{v_A}|\mathcal{V}_C\right]\right]$$

— if Privacy is Receipt-Freeness (SwRF):

- $V'^{\backslash out(chc,\cdot)} \approx_l V\sigma_{id_A}\sigma_{f'_A}\sigma_{v_B}$
- $A\left[\mathcal{S}\left[V\sigma_{id_A}\sigma_{f_A}\sigma_{v_A}^{chc}|V\sigma_{id_B}\sigma_{f_B}\sigma_{v_B}|\mathcal{V}_C\right]\right] \approx_l A\left[\mathcal{S}\left[V'|V\sigma_{id_B}\sigma_{f'_B}\sigma_{v_A}|\mathcal{V}_C\right]\right]$

— if Privacy is Coercion-Resistance (SwCR):
For any context $C = \nu c_1.\nu c_2.(\_|P')$ with $\tilde{n} \cap fn(C) = \emptyset$ and
$\mathcal{S}\left[C\left[V\sigma_{id_A}^{c_1,c_2}\right]|V\sigma_{id_B}\sigma_{f_B}\sigma_{v_B}|\mathcal{V}_C\right] \approx_l \mathcal{S}\left[V\sigma_{id_A}\sigma_{f_A}\sigma_{v_A}^{chc}|V\sigma_{id_B}\sigma_{f_B}\sigma_{v_B}|\mathcal{V}_C\right]$
we have

- $C\left[V'\right]^{\backslash out(chc,\cdot)} \approx_l V\sigma_{id_A}\sigma_{f'_A}\sigma_{v_B}$
- $A\left[\mathcal{S}\left[C\left[V\sigma_{id_A}^{c_1,c_2}\right]|V\sigma_{id_B}\sigma_{f_B}\sigma_{v_B}|\mathcal{V}_C\right]\right] \approx_l A\left[\mathcal{S}\left[C\left[V'\right]|V\sigma_{id_B}\sigma_{f'_B}\sigma_{v_A}|\mathcal{V}_C\right]\right]$

where
— If Eve is:

- *Insider(I): $\mathcal{S} := S$ and $\mathcal{V}_C := V\sigma_{id_C}^{c_1,c_2}$*
- *Outsider (O): $\mathcal{S} := S'$ and $\mathcal{V}_C := 0$*

— If Abs is:

- *Participation Only (PO): $V\sigma_{id_A}$ does not abstain, i.e. $V\sigma_{id_A}\sigma_{f_A}\sigma_{v_A} \not\approx_l 0$.*
- *Security against Forced-Abstention-Attacks (FA), he may abstain.*

The context A represents the attacker. We chose to make the attacker's behavior explicit as some protocols (such as the one by Juels et al. [JCJ05]) require σ_{f_B} and $\sigma_{f'_B}$ to be chosen as a function of the attacker. To ensure that A does not only forward the channels (which would leave the attacker to the outside again and thus contradict our intention of choosing the processes as a function of A), we require A to bind all free names and channels inside. He will only forward the knowledge he is able to obtain during the execution of the protocol.

Note that $SwVP^{O,PO,AB}$ coincides with the definition of Vote-Privacy given by Delaune et al. [DKR09]: Two situations where two voters swap votes are bisimilar. Similarly $SwVP^{O,FA,AB}$ coincides with the definition of Vote-Independence with Passive Collaboration in the DKR-model [DKR09]: If a protocol is receipt-free, there exists a counter-strategy (V') that allows the targeted voter to fake the receipt and vote differently. Analogously, Vote-Independence in the DKR-model corresponds to $SwVP^{I,PO,AB}$, and Vote-Independence with passive Collaboration corresponds to $SwCR^{I,PO,AB}$ in our model.

[1]This condition is needed to ensure that in the case PO no voter can abstain.

Proposition 14 *For Privacy* $\in \{SwVP, SwRF, SwCR\}$ *, Abs* $\in \{FA, PO\}$ *and Behavior* $\in \{AB, EB\}$ *we have:*

1. *Any attack that works for an outsider can also be used for an insider: If a protocol respects* $Privacy^{I,Abs,Behavior}$*, then it also respects* $Privacy^{O,Abs,Behavior}$*.*
2. *If a protocol is secure against Forced-Abstention attacks, it is also secure in the "PO" case: If a protocol respects* $Privacy^{Eve,FA,Behavior}$*, it also respects* $Privacy^{Eve,PO,Behavior}$*.*
3. *If the property holds for any behavior, there exists a behavior for which it holds: If a protocol respects* $Privacy^{Eve,Abs,AB}$*, it also respects* $Privacy^{Eve,Abs,EB}$*.*
4. *Coercion-Resistance is stronger than Receipt-Freeness, which is stronger than Vote-Privacy:*
   — *If a protocol respects* $SwCR^{Eve,Abs,Behavior}$*, it also respects* $SwRF^{Eve,Abs,Behavior}$*.*
   — *If a protocol respects* $SwRF^{Eve,Abs,Behavior}$*, it also respects* $SwVP^{Eve,Abs,Behavior}$*.*

Formal proofs are given in [11]. This leads us to a clear hierarchy of privacy notions described in Figure 5. Notice that for protocols that do not use fakes, the forth dimensions "*Behavior*" collapses and we obtain a simple tower (see Figure 6).

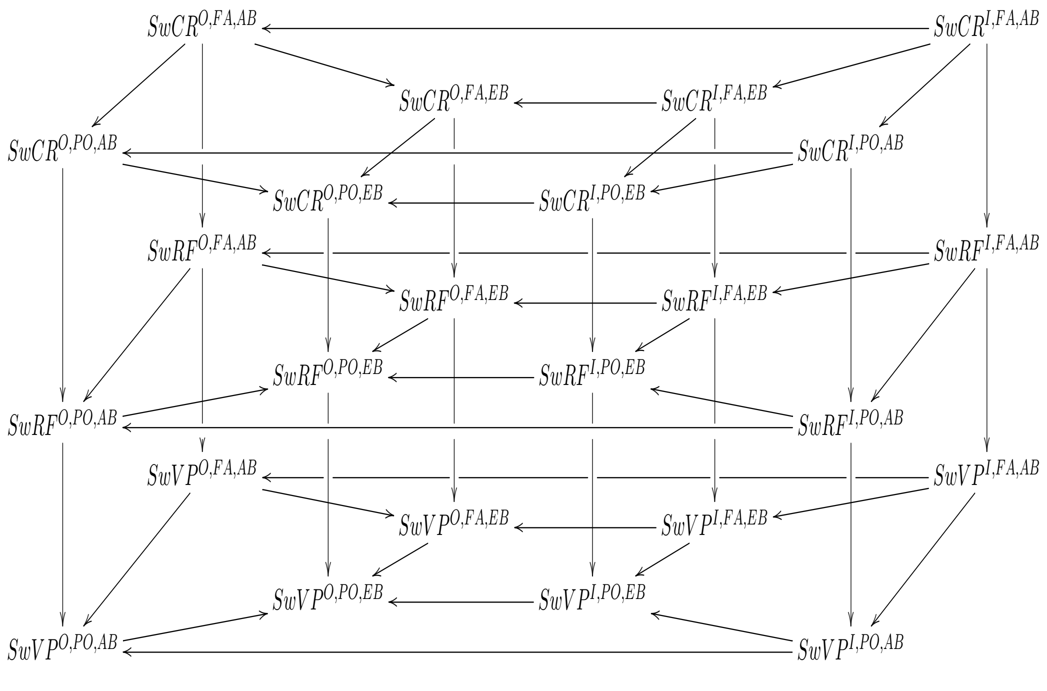

$$\begin{tikzcd}
& SwCR^{O,FA,AB} \arrow[drr] \arrow[ddl] \arrow[dddd] & & & & & & SwCR^{I,FA,AB} \arrow[llllll] \arrow[dlll] \arrow[ddl] \arrow[dddd] \\
& & & SwCR^{O,FA,EB} \arrow[ddl] \arrow[dddd] & SwCR^{I,FA,EB} \arrow[l] \arrow[ddr] \arrow[dddd] & & & \\
SwCR^{O,PO,AB} \arrow[drr] \arrow[dddd] & & & & & & SwCR^{I,PO,AB} \arrow[llllll] \arrow[dl] \arrow[dddd] & \\
& & SwCR^{O,PO,EB} \arrow[dddd] & & & SwCR^{I,PO,EB} \arrow[lll] \arrow[dddd] & & \\
& SwRF^{O,FA,AB} \arrow[drr] \arrow[ddl] \arrow[dddd] & & & & & & SwRF^{I,FA,AB} \arrow[llllll] \arrow[dlll] \arrow[ddl] \arrow[dddd] \\
& & & SwRF^{O,FA,EB} \arrow[ddl] \arrow[dddd] & SwRF^{I,FA,EB} \arrow[l] \arrow[ddr] \arrow[dddd] & & & \\
SwRF^{O,PO,AB} \arrow[drr] \arrow[dddd] & & & & & & SwRF^{I,PO,AB} \arrow[llllll] \arrow[dl] \arrow[dddd] & \\
& & SwRF^{O,PO,EB} \arrow[dddd] & & & SwRF^{I,PO,EB} \arrow[lll] \arrow[dddd] & & \\
& SwVP^{O,FA,AB} \arrow[drr] \arrow[ddl] & & & & & & SwVP^{I,FA,AB} \arrow[llllll] \arrow[dlll] \arrow[ddl] \\
& & & SwVP^{O,FA,EB} \arrow[ddl] & SwVP^{I,FA,EB} \arrow[l] \arrow[ddr] & & & \\
SwVP^{O,PO,AB} \arrow[drr] & & & & & & SwVP^{I,PO,AB} \arrow[llllll] \arrow[dl] & \\
& & SwVP^{O,PO,EB} & & & SwVP^{I,PO,EB} \arrow[lll] & &
\end{tikzcd}$$

Figure 5 – Hierarchy of privacy notions. $A \to B$ means that any protocol that respects property A also respects property B.

6.4 Conclusion

In [9], we applied our family of notions on several case studies (FOO [FOO92], Okamoto [Oka96], Juels et al. [JCJ05], Bingo Voting [BMQR07], Lee et al. [LBD+04]) chosen to show that each

$$\begin{tikzcd}
 & SwCR^{O,FA} \arrow[dl] \arrow[dd] & & SwCR^{I,FA}* \arrow[ll] \arrow[dl] \arrow[dd] \\
SwCR^{O,PO}\text{[LBD}^{+}\text{04]} \arrow[dd] & & SwCR^{I,PO}\text{[BMQR07]} \arrow[ll] \arrow[dd] & \\
 & SwRF^{O,FA} \arrow[dl] \arrow[dd] & & SwRF^{I,FA}\diamond \arrow[ll] \arrow[dl] \arrow[dd] \\
SwRF^{O,PO} \arrow[dd] & & SwRF^{I,PO}\text{[Oka96]} \arrow[ll] \arrow[dd] & \\
 & SwVP^{O,FA} \arrow[dl] & & SwVP^{I,FA}\dagger \arrow[ll] \arrow[dl] \\
SwVP^{O,PO}\bullet & & SwVP^{I,PO}\text{[FOO92]} \arrow[ll] &
\end{tikzcd}$$

Figure 6 – Collapsed hierarchy of privacy notions with examples: The simple voting protocol, our running example (●); Bingo Voting with the assumption that the attacker does not know if a voter entered the voting booth ($*$); Okamoto with a private channel to the administrator ($\diamond$); FOO with a private channel to the administrator (†); and Lee et al.

of our dimensions corresponds to a different property of real-world protocols. The results are summed up in and Table 6.1, the position of the case studies within our hierarchy is shown in Figure 6. We proposed a modular family of formal privacy notions in the applied π-calculus which allows to assess the level of privacy provided by a voting protocol.

Protocol	**Privacy Notion**	**Comments**
Juels et al. [JCJ05]	$SwCR^{I,FA,EB}$	Requires fakes to achieve coercion-resistance
Bingo Voting [BMQR07]	$SwCR^{I,PO,AB}$	Trusted voting machine, vulnerable to forced abstention
- variant	$SwCR^{I,FA,AB}$	Secure against forced abstention if the attacker is unaware of the voter entering the voting booth
Lee et al. [LBD+04]	$SwCR^{O,PO,AB}$	Trusted randomizer, vulnerable to vote-copying
Okamoto [Oka96]	$SwRF^{I,PO,AB}$	Based on trap-door commitments
- variant	$SwRF^{I,FA,AB}$	Private channel to the administrator, secure against forced abstention attacks
Fujioka et al. [FOO92]	$SwVP^{I,PO,AB}$	Based on blind signatures
- variant	$SwVP^{I,PO,AB}$	Permits multiple votes
Simple Voting Protocol	$SwVP^{O,PO,AB}$	The running example, vulnerable to vote-copying

Table 6.1 – Results of the case studies

Chapter 7

One coercer is enough

Il existe des protocoles de vote qui pondérent les votes de chaque votant par exemple en fonction de leur part dans une entreprise. Dans ce chapitre, nous modélisons en π-calcul appliqué les propriétés de respect de la vie privée (Privacy) d'absence de reçu (Receipt-Freness) et de non coercition pour des protocoles de votes pondérés. Nous exhibons les conditions rendant équivalent ces nouvelles définitions et les définition basé sur l'échange de vote. De plus, ce formalisme nous permet de considérer qu'un seul ou plusieurs votants sont corrompus. Nous prouvons que sous des hypothèses réalistes il est équivalent de considérer un seul votants compromis que plusieurs [10].

Contents

Most existing formal privacy definitions for voting protocols are based on observational equivalence between two situations where two voters swap their votes. If the votes are private, a case where Alice votes “yes” and Bob votes “no” should be indistinguishable from a case where Alice votes “no” and Bob votes “yes”. Yet this definition is unsuitable for some situations, for example in companies where votes are weighted according to the proportion of shares held by each shareholder. Consider the following example: Alice owns 50% of the stocks, and Bob and Carol each hold 25%. The cases where Alice and Bob swap votes are now easily distinguishable if Carol votes “yes” all the time, as the result of the vote is different: 75% vs. 50% vote for “yes”. Protocols supporting vote weights have been proposed, for example Eliasson and Zúquete [EZ06] developed a voting system supporting vote weights based on REVS [JZF03], which itself is based on the protocol by Fujioka et al. [FOO92].

We define a symbolic privacy notion in the Applied π-calculus [AF01a] that takes weighted votes into account. Instead of requiring two executions where voters swap votes to be bisimilar, we require two executions to be bisimilar if they publish the same result, independent of the mapping between voters and votes. We analyze the relationship of our notion to the existing swap-based ones and give precise conditions for formally proving the equivalence between them. Then, we generalize our notion to Receipt-Freeness and Coercion-Resistance for weighted votes. Using our model, we are also able to define multi-voter coercion, i.e. situations where several voters are attacked at the same time. Then we prove that under certain realistic assumptions a protocol secure against coercion of a single voter is also secure against coercion of multiple voters. This applies for Receipt-Freeness as well as Coercion-Resistance. Note that all detailled proofs and examples can be found in [10, 12, 13].

Outline: In Section 7.1, we introduce our privacy definition and show under which condition it is equivalent to the existing ones. Then, in Section 7.2, we define Single- and Multi-Voter Receipt-Freeness, analyze their relationship and prove their equivalence under certain assumptions. In Section 7.3 we define Single- and Multi-Voter Coercion-Resistance and again prove their equivalence under the same hypotheses, before concluding in Section 7.4.

7.1 Privacy Notions for Weighted Votes

Our privacy definition is based on the observation that - as the result of the vote is always published - some knowledge about the voter’s choices can always be inferred from the outcome. The classical example is the case of a unanimous vote where the contents of all votes are revealed just by the result. Yet - as already discussed in the introduction - there can also be other cases where some of the votes can be inferred from the result, in particular in the case of weighted votes. If for example Alice holds 66% of the shares and Bob 34%, both votes are always revealed when announcing the result: If one option gets 66% and the other 34%, it is clear which one was chosen by Alice or Bob. However, if we have a different distribution of the shares (e.g. 50%, 25% and 25%), some privacy is still possible as there are several situations with the same result. Thus our main idea: If two instances of a protocol give the same results, an attacker should not be able to distinguish them. Note that this includes the classic definition where votes are swapped, if this give the same result.

7.1.1 — Formal Definition

To express this formally, we need to define the result of an election. We suppose that the result is always published on a special channel res. The following definition allows us to hide all channels except for a specified channel c, which we can use for example to reason about the result on channel res.

Definition 25 **($P|_c$)** *Let $P|_c = \nu \tilde{ch}.P$ where $\tilde{ch}$ are all channels except for c, i.e. we hide all channels except for c.*

Now we can formally define our privacy notion: If two instances of a protocol give the same result, they should be bisimilar.

Definition 26 (Vote-Privacy (VP)) *A voting protocol ensures* Vote-Privacy (VP) *if for any two instances $VP_A = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^A} \mid \dots \mid V\sigma_{id_n}\sigma_{v_n^A} \mid A_1 \mid \dots \mid A_l)$ and $VP_B = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^B} \mid \dots \mid V\sigma_{id_n}\sigma_{v_n^B} \mid A_1 \mid \dots \mid A_l)$ we have*

$$VP_A|_{res} \approx_l VP_B|_{res} \Rightarrow VP_A \approx_l VP_B.$$

A simple interpretation of this definition is that everything apart from the result on channel res has to remain private. This obviously relies heavily on the notion of "result" and the modeling of the protocol. Typically the result will only contain only the sum of all votes, which corresponds to a simple and intuitive understanding of privacy.

Some protocols may leak some additional information, for example the number of ballots on the bulletin board. For instance in the protocol by Juels et al. [JCJ05] voters can post fake ballots. In this case, the above definition of the result may lead to a too restrictive privacy notion, since two situations with the same votes but a different number of fakes are required to be bisimilar. To address this issue, we can include the number of ballots in the result if we want to accept the additional leakage. This gives very fine-grained control about the level of privacy we want to model.

Note that if the link between a voter and his vote is also published as part of the result on channel res, our definition of privacy may be true although this probably does not correspond to the intuitive understanding of privacy. This is however coherent within the model since everything apart from the result is private; simply the result itself leaks too much information.

7.1.2 — Link to Existing Definitions

To establish the relationship of our definition and the existing ones, we need to formally characterize their difference. Intuitively the swap-based definition assumes that swapping two votes will not change the result. This can be formalized as follows: If two instances of the protocol with the same voters give the same result, then the votes are a permutation of each other, and vice versa. This precludes weighted votes, thus the name "Equality of Votes".

Definition 27 (Equality of Votes (EQ)) *A voting protocol respects* Equality of Votes (EQ) *if for any*

$$VP_A = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^A}|\dots|V\sigma_{id_n}\sigma_{v_n^A}|A_1|\dots|A_l)$$

and

$$VP_B = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^B}|\dots|V\sigma_{id_n}\sigma_{v_n^B}|A_1|\dots|A_l)$$

we have

$$VP_A|_{res} \approx_l VP_B|_{res} \Leftrightarrow \exists\pi : \forall i : \sigma_{v_i^B} = \sigma_{v_{\pi(i)}^A},$$

where π is a permutation.

This allows us to formally prove that our definition is equivalent to the existing ones if (EQ) holds.

Theorem 15 (Equivalence of Privacy Definitions) *If a protocol respects (EQ), then (VP) and (SwP) are equivalent.*

Intuitively, because of (EQ), two instances of a protocol can only have the same result if the votes are a permutation of each other. As any permutation can be written as a sequence of simple permutations (swaps), (SwP) is enough to generate any possible permutation, which gives (VP). Conversely, the definition of (SwP) becomes just a particular case of (VP).

It is easy to see that this condition (EQ) is necessary: If a protocol uses weighted votes (e.g. Alice 66%, Bob 34%), it may satisfy (VP), but not (SwP).

Similarly, consider the following example: In the official result announced on channel res, a pre-selected candidate always wins - this could be the case if the authorities are dishonest and want to manipulate the election outcome. If however at the same time the ballots on the bulletin board allow to calculate the result, such a protocol may ensure (SwP) – if the ballots cannot be linked to the voters –, but not (VP) because two instances with a different outcome based on the ballots will have the same "result" on res. Note that such a protocol would contradict (EQ) because we have instances where the votes are not a permutation of each other, but still give the same result.

7.2 Receipt Freeness

In this section we define receipt-freeness for weighted votes. We first consider the case where only one voter is attacked, then we define multi-voter attacks.

7.2.1 — Single-Voter Receipt-Freeness (SRF)

We combine the idea by Delaune et al. [DKR09] with our definition of Privacy: If two instances of a voting protocol give the same result, they should be bisimilar even if one voter reveals his secret data in one case or fakes it in the other.

Definition 28 (Single-Voter Receipt Freeness (SRF)) *A voting protocol ensures* Single-Voter Receipt Freeness (SRF) *if for any voting processes $VP_A = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^A} \mid \ldots \mid V\sigma_{id_n}\sigma_{v_n^A} \mid A_1 \mid \ldots \mid A_l)$, $VP_B = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^B} \mid \ldots \mid V\sigma_{id_n}\sigma_{v_n^B} \mid A_1 \mid \ldots \mid A_l)$ and any number $i \in \{1, \ldots, n\}$ there exists a process V_i' such that we have $V_i'^{\backslash out(chc_i,\cdot)} \approx_l V\sigma_{id_i}\sigma_{v_i^B}$ and*

$$VP_A|_{res} \approx_l VP_B|_{res} \Rightarrow VP_A'\left[(V\sigma_{id_i}\sigma_{v_i^A})^{chc_i}\right] \approx_l VP_B'\left[V_i'\right],$$

where VP_A' and VP_B' are like VP_A and VP_B, but with holes for the voter $V\sigma_{id_i}$.

As for (VP), our definition is equivalent to the existing one based on swapping if the protocol ensures (EQ), which is the case if it does not use weighted votes. Similarly to swap-based definitions, (SRF) is stronger than (VP). The proof is analogous to the proof in the swap-based model (see [12] for details).

7.2.2 — Multi-Voter Receipt-Freeness (MRF)

We now generalize the idea of Receipt-Freeness to the case where multiple voters are attacked. Instead of only considering one attacked voter i, we consider a set I of attacked voters. To be receipt-free, it should be possible for all attacked voters to fake the receipt. Note that we assume that there is always at least one honest voter, except for the case with only one voter.

Definition 29 (Multi-Voter Receipt Freeness (MRF)) *A voting protocol ensures* Multi-Voter Receipt Freeness (MRF) *if for any voting processes $VP_A = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^A} \mid \ldots \mid V\sigma_{id_n}\sigma_{v_n^A} \mid A_1 \mid \ldots \mid A_l)$, $VP_B = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^B} \mid \ldots \mid V\sigma_{id_n}\sigma_{v_n^B} \mid A_1 \mid \ldots \mid A_l)$ and any subset $I \subset \{1, \ldots, n\}$, $I \neq \{1, \ldots, n\}$ if $n > 1$, then there exists processes V_i' such that we have $\forall i \in I : V_i'^{\backslash out(chc,\cdot)} \approx_l V\sigma_{id_i}\sigma_{v_i^B}$ and*

$$VP_A|_{res} \approx_l VP_B|_{res} \Rightarrow VP_A'\left[\prod_{i\in I} (V\sigma_{id_i}\sigma_{v_i^A})^{chc_i}\right] \approx_l VP_B'\left[\prod_{i\in I} V_i'\right],$$

where VP_A' and VP_B' are like VP_A and VP_B, but with holes for all voters $V\sigma_{id_i}, i \in I$.

By choosing $I = \{i\}$ we obtain that (MRF) implies (SRF). Under certain conditions the converse is also true. To prove this, we define a "generalized voting process" which is like a voting process, but some voters might be under attack.

Definition 30 (Generalized Voting Process) *A* Generalized Voting Process *is a voting process VP with variables for the voter's processes that can either be a "normal" voter or a voter* *communicating with the intruder, i.e.* $VP = \nu\tilde{n}.(V_1|\dots|V_n|A_1|\dots|A_l)$ *where* $V_i \approx_l V\sigma_{id_i}\sigma_{v_i}$ *or* $V_i^{\backslash out(chc_i,\cdot)} \approx_l V\sigma_{id_i}\sigma_{v_i}$.

The next definition captures the key properties required for our proof. It expresses two modularity conditions of a voting protocol.

Definition 31 (Modularity (Mod)) *A voting protocol is* modular (Mod) *if it is* composable *and* decomposable. *A voting protocol is* composable *if for any generalized voting processes* VP_A *and* VP_B *there exists a generalized voting process* VP *such that* $VP \approx_l VP_A|VP_B$. *A voting protocol is* decomposable *if any generalized voting process* $VP = \nu\tilde{n}.(V_1|\dots|V_n|A_1|\dots|A_l)$ *can be decomposed into processes* $VP_i = \nu\tilde{n}_i.(V_i|A_1^i|\dots|A_l^i)$ *where*

$$VP \approx_l VP_1|\dots|VP_n. \tag{7.1}$$

Imagine a protocol where in order to escape coercion the voters can claim that a certain ballot on the bulletin board is their ballot, but it was actually prepared by some honest authority to allow the voters to create a fake receipt. If we suppose that this ballot exists only once no matter how many voters are attacked, it would be enough for a single voter to fake his receipt. However we cannot compose two instances with one attacked voter each, as they would use the same fake ballot which would be noticeable for the attacker. Hence the above definition also captures the fact that faking the receipt to escape coercion can be done by each voter independently.

In order to obtain this property for several examples, it is convenient to have a result of decomposition unicity in the π-calculus. We prove such result in [14].

Another property we need for our proof is *Correctness*, i.e. the fact that if in two instances the voters' choices are the same, they give the same result[1].

Definition 32 (Correctness (Cor)) *A voting protocol is* correct *if for any generalized voting processes* $VP_A = \nu\tilde{n}_A.(V_{1,A}|\dots|V_{n,A}|A_1|\dots|A_l)$ *and* $VP_B = \nu\tilde{n}_B.(V_{1,B}|\dots|V_{n,B}|A_1|\dots|A_l)$ *with for any* i *and* $X \in \{A,B\}$: $V_{i,X}^{\backslash out(chc_i,\cdot)} \approx_l V\sigma_{id_i}\sigma_{v_i}$, *we have*

$$VP_A|_{res} \approx_l VP_B|_{res} \tag{7.2}$$

It is easy to see that Correctness is implied by Equality of Votes as the identity is a permutation, hence any protocol ensuring (EQ) ensures (Cor) [12]. Putting everything together, we are able to prove the equivalence of (SRF) and (MRF).

Theorem 16 *If a protocol is modular, correct and ensures Single-Voter Receipt Freeness, it also ensures Multi-Voter Receipt Freeness.*

The main idea is that we can decompose an instance with multiple attacked voters into instances with at most one attacked voter, where we can apply the single-voter assumption,

[1]This does not entirely cover intuitive correctness as it will be fulfilled by protocols always giving the same result independently from the votes, but it will fail for a protocol announcing a random result.

and recompose the result. Note that the assumptions (Mod), (EQ) are satisfied by many well-known protocols (e.g. [BMQR07, FOO92, Oka96]), we illustrate this on an example.

Remark. We have to be careful when modeling protocols using a full PKI. If we model the PKI inside the voting process, decomposing a protocol would result in two instances using different keys, which will most probably be visible to an attacker and the bisimilarity (7.1) will not hold. A possible solution could be to externalize the PKI into a context K such that $K[VP] \approx_l K[VP_1|VP_2]$, which ensures that VP_1 and VP_2 use the same keys. This would allow us to obtain the same result for protocols such as [LBD$^+$04, WB09].

7.3 Coercion Resistance

After discussing Receipt-Freeness, we now define Coercion-Resistance. As before, we start with Single-Voter Coercion-Resistance.

7.3.1 — Single-Voter Coercion (SCR)

In this case, we combine (VP) with (SwCR): If two instances of a voting protocol give the same result, they should be bisimilar even if one voter interacts with the attacker in one case or only pretends to do so in the other case. The coercion is modeled by the context C that interacts with the voter and tries to force him to vote for a certain candidate.

Definition 33 (Single-Voter Coercion-Resistance (SCR)) *A voting protocol ensures the property of* Single-Voter Coercion-Resistance (SCR) *if for any voting processes $VP_A = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^A} \mid \dots \mid V\sigma_{id_n}\sigma_{v_n^A} \mid A_1 \mid \dots \mid A_l)$, $VP_B = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^B} \mid \dots \mid V\sigma_{id_n}\sigma_{v_n^B} \mid A_1 \mid \dots \mid A_l)$ and any number $i \in \{1, \dots, n\}$ there exists a process V_i' such that for any context C_i with $C_i = \nu c_1.\nu c_2.(\_|P_i)$ and $\tilde{n} \cap fn(C) = \emptyset$, $VP_A'\left[C_i\left[(V\sigma_{id_i}\sigma_{v_i^A})^{c_1,c_2}\right]\right] \approx_l VP_A'\left[(V\sigma_{id_i}\sigma_{v_i^A})^{chc_i}\right]$ we have $C_i\left[V_i'\right]^{\backslash out(chc,\cdot)} \approx_l V\sigma_{id_i}\sigma_{v_i^B}$ and*

$$VP_A|_{res} \approx_l VP_B|_{res} \Rightarrow VP_A'\left[C_i\left[(V\sigma_{id_i}\sigma_{v_i^A})^{c_1,c_2}\right]\right] \approx_l VP_B'\left[C_i\left[V_i'\right]\right],$$

where VP_A' and VP_B' are like VP_A and VP_B, but with a holes for the voter $V\sigma_{id_i}$.

As above, we can easily link this definition to the existing swap-based definition using (EQ): If a protocol respects (EQ), (SCR) and (SwCR) are equivalent. The proof given in [12] is similar to the (SRF) case.

7.3.2 — Multi-Voter Coercion (MCR)

We now discuss Multi-Voter Coercion-Resistance. To model the case where multiple voters are attacked, we consider the set I of attacked voters.

Definition 34 (Multi-Voter Coercion-Resistance (MCR)) *A voting protocol ensures* Multi-Voter Coercion-Resistance (MCR) *if for any voting processes* $VP_A = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^A} \mid \dots \mid V\sigma_{id_n}\sigma_{v_n^A} \mid A_1 \mid \dots \mid A_l)$*,* $VP_B = \nu\tilde{n}.(V\sigma_{id_1}\sigma_{v_1^B} \mid \dots \mid V\sigma_{id_n}\sigma_{v_n^B} \mid A_1 \mid \dots \mid A_l)$ *and any subset* $I \subset \{1,\dots,n\}$*,* $I \neq \{1,\dots,n\}$ *if* $n > 1$*, there exists processes* V_i' *such that for any contexts* $C_i, i \in I$ *with* $C_i = \nu c_1.\nu c_2.(\_|P_i)$ *and* $\tilde{n} \cap fn(C) = \emptyset$*,* $VP_A'\left[\mathop{|}\limits_{i\in I} C_i\left[(V\sigma_{id_i}\sigma_{v_i^A})^{c_1,c_2}\right]\right] \approx_l VP_A'\left[\mathop{|}\limits_{i\in I}(V\sigma_{id_i}\sigma_{v_i^A})^{chc_i}\right]$ *we have* $\forall i \in I : C_i\left[V_i'\right]^{\backslash out(chc,\cdot)} \approx_l V\sigma_{id_i}\sigma_{v_i^B}$ *and*

$$VP_A|_{res} \approx_l VP_B|_{res} \Rightarrow VP_A'\left[\mathop{|}\limits_{i\in I} C_i\left[(V\sigma_{id_i}\sigma_{v_i^A})^{c_1,c_2}\right]\right] \approx_l VP_B'\left[\mathop{|}\limits_{i\in I} C_i\left[V_i'\right]\right],$$

where VP_A' *and* VP_B' *are like* VP_A *and* VP_B*, but with holes for all voters* $V\sigma_{id_i}, i \in I$*.*

As for (MRF), (MCR) implies (SCR), and (MCR) resp. (SCR) is stronger than (MRF) resp. (SRF) (for the proofs, see the technical report [12]). We also have equivalence between (SCR) and (MCR) under the same assumptions as in the case of Receipt-Freeness using a similar proof.

7.4 Conclusion

$$\begin{tikzcd}
\text{SwCR} \arrow[r, leftrightarrow, "\text{EQ}"] \arrow[d] & \text{SCR} \arrow[r, bend left, "\text{Cor, Mod}"] \arrow[d] & \text{MCR} \arrow[l] \arrow[d] \\
\text{SwRF} \arrow[r, leftrightarrow, "\text{EQ}"] \arrow[d] & \text{SRF} \arrow[r, bend left, "\text{Cor, Mod}"] \arrow[d] & \text{MRF} \arrow[l] \\
\text{SwP} \arrow[r, leftrightarrow, "\text{EQ}"] & \text{VP} &
\end{tikzcd}$$

Figure 7 – Relations among the notions. A $\xrightarrow{C}$ B means that under the assumption C a protocol ensuring A also ensures B.

We presented an intuitive definition of privacy for voting protocols that generalizes to situations with weighted votes. We extended the definition to include Receipt-Freeness and Coercion-Resistance as well. We considered situations where only one voter is under attack, and others where multiple voters are attacked. We were able to show that - under the assumptions that votes are not weighted and correctly counted - the single voter case is equivalent to (SwP), (SwRF), (SwCR) as defined by Delaune et al. [DKR09]. Moreover, we proved that the multi-voter case is equivalent to the single-voter case if the protocol is correct (Cor) and respects a modularity condition (Mod). This condition allows us to compose and decompose protocols, which expresses the fact the different parts of the protocol are independent. Figure 7 summarizes our results. Finally, we illustrated our work by analyzing two existing protocols.

Part III

Wireless Sensor Network

More and more devices are nowadays connected using wireless communication. They use protocols that rely on the fact that each node can discover his neighbors. As it is explained in [25], discovering correct neighborhood is an important feature for Wireless Sensors Networks (WSNs). This a basic requirement for building routing algorithms, but there are often proposed without a security proof. Our first result is a formal method for proving neighborhood discovery protocols.

Moreover wireless communication changes the original intruder model proposed by Dolev-Yao, where the intruder controls all communications. In practice in a WSN with several nodes, it is not realistic that an intruder controls all communication links. Moreover if we consider that intruders are nodes then it is also not realistic that malicious nodes share all information they learned. Hence we propose a weaker intruder model where intruders are independent nodes.

Finally how can we construct efficient secure routing algorithms? A first observation shows that random walks are not efficient routing algorithms but they are unpredictable and more resilient to certain attacks than existing protocols. We propose probabilistic routing algorithms using tabu list in order to find a trade-off between security, efficiency and energy consumption.

Contributions: Our contributions can be split into two parts: the first one is the proposition of a formal trace-based model for verifying neighborhood discovery protocols and a constraint-based model for proving routing protocols in presence of independent intruders. The second contribution is the design and the analysis of some probabilistic routing algorithms using tabu lists.

Formal Models: We first propose a trace-based formal description for k-neighborhood discovery protocols. Our definition is recursive, because if two nodes are (k)-neighbor then there are necessarily nodes which are $(k-1)$-neighbor. We focus on timed protocols which enable the verification in presence of mobile nodes. In order to verify protocol, we model the following property: a protocol and two nodes of a given topology are (k)-neighborhood secure if the protocol discovers a path of length k between these two nodes without any intruder. It allows us to verify existing protocols. Finally, we construct and prove a generic secure (k)-neighborhood protocol based on any secure (1)-neighborhood protocol [20].

In our second model, we use a constraints system to verify routing protocol in the presence of independent intruders. This intruder model is weaker than the usual Dolev-Yao intruder model where he controls the whole network. Here intruders are independent nodes that do not collaborate: they even do not know each other and they do not share any knowledge. This models is different than the usual one: it corresponds to situations in which adversaries cannot communicate. Our result implies that the usual monotonicity property of constraint systems must be relaxed. It leads us to a different approach in order to solve such constraint systems modeling protocol in presence of independent intruders. We obtain an NP-time decision procedure for verifying routing protocol in presence of independent intruders.

Adaptive randomized routing protocol with tabu list: The goal of routing algorithm in WSNs is to transfer a data using distributed algorithms over a network from any node to a special node called a *sink*. We are interesting in minimizing the *hitting time* of such algorithms, it means the number of hops needed to reach the sink. One main advantage of random walks is that it is difficult to predict their behavior. The drawback is that they are not efficient for routing data. Our main idea is to improve the efficiency of random walks and to exploit their non-determinism for improving the security of routing algorithms. Because they are non predictable, an intruder will have difficulties to mount systematic attacks. We propose two randomized approaches for routing protocols using tabu lists. The first one stores the tabu list in messages sent over the network; second one each node has a tabu list per communication link:

Tabu list in the message: The main idea in this approach is to avoid to forward messages already transmitted to the same node. The structure used to prevent such event is a *tabu list* containing previous visited nodes. We call such a class of algorithms Tabu List in the Message (TLM). Of course our tabu lists are finite, then it is natural to see the impact of different update functions when the list is full. We first define a class of update rules. Then we show the impact of the update functions and the size of the list on the efficiency of such algorithms. We characterize a class of graph called *m-free graph* for which the tabu list of size m using the update function FIFO[2] is optimal. As a corollary, due to the fact that all graphs are 1-free, it proves that $FIFO$ with only one memory is also optimal [2].

Tabu list in the node: We proposed two different algorithms based on tabu list stored in node (TLN).

— Each message contains a *tabu list* of all the previous nodes visited. When a message is received by a node, it can check if the message has been already sent: then we can detect a loop. We also increase a counter associated to node corresponding to the loop. Finally we decide to route a message using a probability law which prefers link with smaller counters. Indeed we avoid to send message in a loop. This algorithm is quite efficient without any malicious node. However we propose a scenario where a strong intruder can clearly disturb our algorithm.

— As a consequence, our second algorithm aims to resist to such attacks. It uses cryptographic primitives and an acknowledgment mechanism. The main idea of our second algorithm is to detect valid route to the sink and try to route messages only by using such routes. In order to do so, we have several lists per node in order to store: i) a kind of routing table which stores acknowledgments send by the sink, ii) generated data messages sent through a link and iii) good links, i.e. links that reach the sink. Our algorithms uses a probability law which prefers nodes present in the list of good links [3].

Of course we compare our different approaches to some of existing routing protocols using a WSN simulator: SINALGO [Gro08].

[2]First In First Out

Outline: In Chapter 8, we present our formal model for secure (k)-neighborhood discovery protocols. We also give a framework which allows us to construct a secure (k)-neighborhood protocol based on any secure (1)-neighborhood protocol. In Chapter 9, we develop a formal constraint system in order to analyze routing protocols in presence of independent intruders. In Chapter 10, we propose a class of probabilistic routing algorithms which store information in the messages. We also make a theoretical analysis of such protocols. Finally in Chapter 11, we develop another class of probabilistic routing algorithms which store local information in each node. We also shows that such algorithms are resilient to several types of attacks.

Chapter 8

Formal Verification of Neighborhood Discovery

Les protocoles de communication utilisent souvent la connaissance de leur voisinage. S'assurer que la découverte du voisinage est correcte constitue un élément important de ces protocoles. Tout d'abord, nous donnons une définition formelle de la notion de (k)-voisinage entre deux nœuds. Ensuite nous proposons un système formel à base de trace permettant de vérifier la sécurité de tels protocoles. Ce système prend en compte la mobilité des nœuds et la présence d'intrus. Enfin à partir de la connaissance d'un protocole sûr de découverte de (1)-voisinage nous proposons un protocole permettant de découvrir le (k)-voisinage d'un nœud de manière sûre [20].

Contents

8.1 Introduction

Neighborhood discovery protocols are basic components in mobile wireless systems. Knowledge of node's neighbors is for instance critical for routing protocols. The goal of a neighborhood discovery protocol is to identify the nodes which are in the proximity, even in presence of intruders. Designing a secure neighborhood discovery protocol is not an easy task, as illustrated in [And01] with the famous *"MIG-in-the-middle"* attack accounted in the late 1980s. In this attack, Angolan MIG airplanes were able to be impersonate a South African unit by relaying challenge messages from South African defense to another South Africa units, which then answered the challenge for them. The Angolan MIGs could therefore bomb their targets without being attacked by the South African air defense. A secure neighborhood discovery system would have prevented such a problem, by detecting that the MIG is not actually in range for the authentication protocol, and then preventing further impersonation by enemy forces. In [PPS$^+$08], we survey the question of secure neighborhood discovery, providing definitions of neighborhood types and neighbor discovery protocol properties. We also describe different attacks against wireless networks.

We assume that two nodes are *neighbors* if they can communicate directly together, without the help of another node. As a consequence, an attack is any situation in which two nodes communicate together believing that they are neighbors when a malicious forwarder is involved.

Contributions:. Our formal model is in the continuation of the work initiated by Basin et al in [SSBC09b]. Based on the observation that the neighborhood is a physical property[1], we distinguish two layers: the physical layer representing physical characteristics of the communications (radio for instance) and the abstract layer modeling node behavior. Then we formalize the communication between nodes by a trace-based model inspired by the symbolic approach proposed by Paulson in [Pau98]. We propose a system of rules for modeling exchanges between nodes given a topology which represents the environment and position between nodes. Moreover, we notice that most of the neighborhood discovery protocols use time measurements and can work even for mobile nodes, so we add timestamps in all our events. Then we give a deduction system which models the abilities of an intruder. Using all these ingredients, we can define the neighborhood's property by generating a special event called *END* when a node concludes that he is neighbor with another node. Consequently, an attack is the situation where a node concludes that another node is his neighbor according to the protocol but indeed they are not able to communicate directly. A protocol is secure for this property if for any execution in presence of intruder, and for a given topology, there exists no attack. Then we extend it to (k)-neighborhood, *i.e.* we propose a formal definition for modeling that two nodes can communicate with k hops. examples illustrating how our Finally we propose a protocol, called *Sharek* protocol, for discovering the (k)-neighborhood of a node based on a secure (1)-neighborhood secure protocol. This protocol can be used to securely discover the

[1]We remark that is not the case for the authentication which is a property based on exchanged messages.

set of neighbors of neighbors, and further. It can help to determine the dominating sets of a wireless network, as explained in [DB97, WL99, BMSU01b].

Related works:. Neighborhood discovery and analysis of protocols are active research topics in wireless networks security. One of the first neighborhood discovery protocol was given by S. Brands and D. Chaum in [BC94]. Later other works have approached the problem differently using for instance directional antennas [VKT05, DKPR11], probabilistic protocols and challenge-response in RFID context [HK05], or specific protections against some attacks by analyzing network topology based on time of flight of messages in [SPRH09].

In [PPH08a], the authors investigate the possibility of neighborhood detection. They take into account several transmission speeds, directional emissions, localization and clock synchronization, and conclude by proving that time-based protocols cannot securely detect neighborhoods if and only if intruders can forward faster than legitimate nodes. They also consider protocols which use location data.

One of the first formal verification of neighbor discovery protocol is the work done by Meadows et al. in [MPP$^+$06]. In this paper they developed a formal methodology to prove properties of distance bounding protocols. They extend the authentication logic to reason about distance bounding property and they extend the protocol of S. Brands and D. Chaum. However, they do not consider (k)-neighborhoods, and deal only with static nodes.

Another formal approach for distance bounding protocols is introduced by Basin et al. in [SSBC09b]. Their approach also uses notions of distances, time and events-trace. They use the Isabelle/HOL proof assistant [NPW02] to check results from their model. They apply their model on three security protocols, the authentication ranging protocol [CH05], the distance bounding protocol of [BC94], which calculates distance between two nodes using communication time and finally the TESLA protocol of [PCTS02]. In their work they only consider static nodes and do not consider (k)-neighborhoods.

In [TSG10] the authors present a systematic technique for verifying that location discovery protocols satisfy local authentication whereby an entity can authenticate the physical location of a device, even in the presence of malicious adversaries. They extend the strand space theory with a metric that captures the geometric properties of time and space. They prove that several prominent location discovery protocols including GPS do not satisfy the local authentication goal and analyze a location discovery protocol that does satisfy the goal under some reasonable assumptions.

In some recent works, C. Cremers et al [CRC11] propose distance bounding protocols resistant to distance hijacking attacks. The authors in [PPH08b] and [PPH07] build a formal model based on traces. They apply their model on a protocol they introduced and on the temporal packet leash protocol [HPJ03]. However their method does not take the mobility of the nodes into account.

Outline:. In Section 8.2, we present our model. We first give the network representation, the events and traces definitions, and show how to specify a protocol and an intruder. Then in Section 8.3, we define the neighborhood property. We extend the usual notion of (1)-neighborhood to (k)-neighborhood.In Section 8.4, we propose a secure protocol to build the (k)-neighborhoods of a node, based on any secure (1)-neighborhood protocol. Finally we prove it is secure against a class of intruders, before concluding in the last section.

8.2 Timed Model

We assume the time needed to transmit a message between two nodes. As a consequence two nodes are neighbors if they can communicate directly, *i.e.* their transmission time is finite. In the topology depicted in Figure 8, if communication times are proportional to the distances, then, due to signal reflection on the wall and to the presence of a wall between A and B, we have that the transfer time between A and I plus the transfer time between I and B is strictly smaller than the transfer time between A and B. This situation shows that in some setting triangular inequality may not always hold. It is why we consider transfer time and not distances.

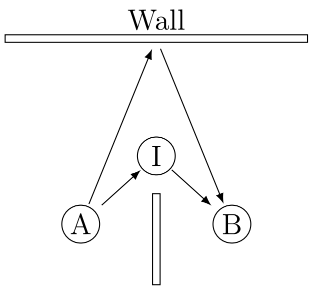


Figure 8 – Triangle inequality counter-example.

We have two kinds of node : honest nodes, which follow the protocol, and malicious nodes. We consider a simple protocol called *Ping-Pong protocol.* A node sends the message “Ping” to all his neighbors, then they answer with their name and the message “Pong”. Once the Pong is received, the protocol claims that the initiator and the responder are neighbors. A high-level description of this protocol is given in Figure 9. Of course this protocol is obviously not secure.

$$A \xrightarrow{Ping} B$$

$$A \xleftarrow{\langle B, Pong\rangle} B$$

Figure 9 – Ping-Pong protocol communications diagram

We start by formally defining characteristics of a network and nodes. Then we model behaviors of nodes using two family of events with timestamps. We explain using rules describing a given protocol how to build traces, which are sequences of events. We model by a set of rules the intruder capabilities. Finally, we explain two realistic assumptions, which allow us to take into account mobility in our approach.

8.2.1 — Networks and Nodes

Wireless Sensors Networks are composed of several nodes, that are small devices usually equipped with sensors, a battery and a radio. They use their radio for sending their measurements through the network. A network is composed of a set of nodes (identified by an unique number) and a topology, which represents communications between nodes.

Definition 35 (Network) *A network* $\mathbf{N}$ *is defined by* $(V, \mathbf{T})$ *where:*

— V *denotes the set of all node identities, which is partitioned into two sets:* $V_{\mathcal{P}}$ *denotes all the honest nodes who are following the protocol and* $V_{\mathcal{I}}$ *represents the intruder nodes which are malicious.*

— $\mathbf{T}$ *represents the matrix containing the time of communication between two nodes. More precisely,* $\mathbf{T}_{A,B}$ *denotes the time that a message takes starting from node* A *to reach node* B*. If* A *can not reach directly* B *then* $\mathbf{T}_{A,B}$ *is equal to* $+\infty$*. We also require that the communication time between a node and itself is null:* $\forall A \in V, \mathbf{T}_{A,A} = 0$.

We denote honest node identities by $A, B, C, ...$, and $I, J, ..$ for intruders. By extension, we will often refer to nodes by their identity (for example, node A).

This definition models only static networks. In order to express the possible movements of nodes, we need to take into account the changes in topology over time, and so we extend the previous definition by making the communication time depend on time. $\mathbf{N} = (V, \mathbf{T})$ becomes $\mathbf{N} = (V, \mathbf{T}(t))$, where the element of the matrix are functions with parameter t. Each element $\mathbf{T}_{A,B}(t)$ models how much time a message **emitted** by the node A at time t takes to reach node B. If needed, we use $\mathbf{T}_{A,B}(t)$ instead of $\mathbf{T}_{A,B}$.

We denote $\mathcal{N}$ the set of all possible networks, which takes into account any number of nodes, any number of intruders, and any possible evolution of the connections over time. Here are a few examples of useful network families:

— $\mathcal{N}_0$ is the family of networks where no intruder is present at all. This allows us to show that a protocol is insecure even when no intruder is present.

$$\mathcal{N}_0 = \{\mathbf{N} = (V, \mathbf{T}(t)) \in \mathcal{N} \mid V_{\mathcal{I}} = \emptyset\}.$$

— $\mathcal{N}_{sym}$ is the family of networks where message transfer times are symmetrical.

$$\mathcal{N}_{sym} = \{\mathbf{N} = (V, \mathbf{T}(t)) \in \mathcal{N} \mid \forall t, \forall A, B \in V, \mathbf{T}_{A,B}(t) = \mathbf{T}_{B,A}(t)\}.$$

— $\mathcal{N}_{fully\_connected}$ is the family of networks where each node can communicate with any other node at any time.

$$\mathcal{N}_{fully\_connected} = \{\mathbf{N} = (V, \mathbf{T}(t)) \in \mathcal{N} \mid \forall t, \forall A, B \in V, \mathbf{T}_{A,B}(t) \neq +\infty\}.$$

— $\mathcal{N}_{still}$ is the family of networks where there is no change in transfer times depending on time, or in other words a static network.

$$\mathcal{N}_{still} = \{\mathbf{N} = (V, \mathbf{T}(t)) \in \mathcal{N} \mid \forall t_1, t_2, \forall A, B \in V, \mathbf{T}_{A,B}(t_1) = \mathbf{T}_{A,B}(t_2)\}.$$

8.2.2 — Events

Neighborhood is a physical property, which is linked to communication channels. In order to isolate physical communications and abstract commands, we have two distinct layers, one of which is strictly restricted to model physical behavior, and the other corresponding to the abstract behavior of the nodes, which correspond to computations performed by a node. In order to model communication between nodes and behaviors of a node on these two layers we define the following events:

— $send_\phi(A, m, t)$: A transmits m at time t using the physical layer.

— $recv_\phi(A, m, t)$: A receives m at time t using the physical layer.

— $send_\alpha(A, m, t)$: A orders the transmission of m at time t using the logical layer.

— $recv_\alpha(A, m, t)$: A received and processed m at time t using the logical layer.

Physical events are annotated by ϕ and abstract events are annotated by α. In order to construct meaningful sequences of events, nodes need to be able to know when a message was sent or received at the physical layer. It is why each event has a timestamp. In the Ping-Pong protocol the first action performed by A is modeled by the event $send_\alpha(A, Ping, t_0)$ and $send_\phi(A, Ping, t_1)$, where t_0 is the time when the node A transmit to the radio the message $Ping$ and t_1 is the time when the radio of the node A emits the message. We introduce another event $END(N_1, ..., N_k, t_{prop}, t)$, which models that a protocol ends well at time t, and concludes that nodes $N_1, ..., N_k$ are neighbors at time t_{prop}.

8.2.3 — Rules and Traces

We build traces which are sequence of events. We use three sets of rules which model the protocol, the communications between nodes and the intruder.

§ 8.2.3.1. Building traces from rules. Our approach is based on the trace-based modeling presented by Paulson for cryptographic protocols in [Pau98]. The rules represents a step in the construction of a trace and has the following form:

$$(R)\frac{H_1 \dots H_n}{C}$$

where R is the name of the rule, $H_1 \ldots, H_n$ are the hypothesis that have to be satisfied in order to produce the conclusion C.

A trace is a set of events which models the communication between nodes during the execution of a protocol in a given network and a given intruder behavior. We use traces as sets of events, not sequences as in Paulson's model, since we have timestamps for each event, the order is implicit. We denote by $S_{\mathbf{N},\mathcal{I},\mathcal{P}}$ the set of all traces, *i.e.* possible executions, built by successive applications of rules of our system.

The behavior of a node which respects a protocol is modeled by a set of rules denoted $\mathcal{P}$. Usually, protocols are specified at the logical layer and generate an *END* event when the protocol finishes. We describe the three rules of $\mathcal{P}_{PingPong}$ corresponding to the Ping-Pong protocol.

$$(PingPong\_1)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}{(tr \cup \{send_\alpha(A, Ping, t)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(PingPong\_2)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad recv_\alpha(A, Ping, t_1) \in tr \qquad t_1 \leq t_2}{(tr \cup \{send_\alpha(A, \langle A, Pong\rangle, t_2)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(PingPong\_3)\frac{\begin{array}{c} tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \\ send_\alpha(A, Ping, t_1) \in tr \qquad send_\phi(A, Ping, t_2) \in tr \\ recv_\alpha(A, \langle B, Pong\rangle, t_3) \in tr \qquad t_1 \leq t_2 \leq t_3 \leq t_4 \end{array}}{(tr \cup \{END(A, B, t_2, t_4)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

The two first rules correspond to the counterpart in the protocol, properly specified with timestamps on the abstract layer. The last rule raises the *END* flag, which models the fact that the protocol reached a conclusion about A and B.

§ 8.2.3.2. Communication Rules. We present the set of rules inherent to all networks. The axiom ($Begin$) initializes a trace with the empty set. For generating events from one layer to the other, we have two rules ($Con0$) and ($Con1$), where δ denotes the delay a node needs to relay a message. Finally, we have the rule (Phy) which allows messages to go from a node's physical layer to another node in a time greater than or equal to $\mathbf{T}_{A,B}(t_1)$.

$$(Begin)\frac{}{\emptyset \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(Con0)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad send_\alpha(A, m, t_1) \in tr \qquad t_1 + \delta/2 \leq t_2}{(tr \cup \{send_\phi(A, m, t_2)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(Con1)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad recv_\phi(A, m, t_1) \in tr \qquad t_1 + \delta/2 \leq t_2}{(tr \cup \{recv_\alpha(A, m, t_2)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(Phy)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad send_{\phi}(A,m,t_1) \in tr \qquad \mathbf{T}_{A,B}(t_1) \leq (t_2 - t_1)}{(tr \cup \{recv_{\phi}(B,m,t_2)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

This rule may be applied multiple times for a single send event, to model signal reflections which may cause a message to be received twice or more, as shown in Figure 10. It is why there is no upper bounds on t_2. We also assume that all the nodes have access to a $New\_Nonce()$ function, which returns a fresh random value at each new call.

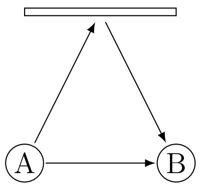


Figure 10 – Network where the same message hits the same node twice.

8.2.4 — Intruder Rules

Intruder capabilities are described by the set of rules denoted $\mathcal{I}$, common to all the intruder nodes. $\mathcal{I}$ describes both what an intruder node can know, and what it can do. We adapt the classical Dolev-Yao intruder models [DY83a] given by the following rules. Symmetric keys are labeled k, and asymmetric secret and public keys are denoted by (sk, pk). Encryptions are denoted by $\{m\}_k$. The set $IK_I(tr,t)$ represents the knowledge of the intruder node I at time t and $\widehat{IK}_I(tr,t)$ represents what I can deduce from $IK_I(tr,t)$. We denote $IK_I(\emptyset,0)$ the initial knowledge of the intruder I. It contains all agent names, public keys and intruder private keys.

$$(IK\_init)\frac{m \in IK_I(\emptyset,0)}{m \in IK_I(tr,t)} \qquad (IK\_hat)\frac{m \in IK_I(tr,t)}{m \in \widehat{IK_I(tr,t)}}$$

$$(IK\_left)\frac{\langle m,n\rangle \in \widehat{IK_I(tr,t)}}{m \in \widehat{IK_I(tr,t)}} \qquad (IK\_right)\frac{\langle m,n\rangle \in \widehat{IK_I(tr,t)}}{n \in \widehat{IK_I(tr,t)}}$$

$$(IK\_pair)\frac{m \in \widehat{IK_I(tr,t)} \qquad n \in \widehat{IK_I(tr,t)}}{\langle m,n\rangle \in \widehat{IK_I(tr,t)}}$$

$$(IK\_decrypt\_sym)\frac{\{m\}_k \in \widehat{IK_I(tr,t)} \qquad k \in \widehat{IK_I(tr,t)}}{m \in \widehat{IK_I(tr,t)}}$$

$$(IK\_encrypt\_sym)\frac{m \in \widehat{IK_I(tr,t)} \qquad k \in \widehat{IK_I(tr,t)}}{\{m\}_k \in \widehat{IK_I(tr,t)}}$$

$$(IK\_decrypt\_asym\_sk)\frac{\{m\}_{sk} \in \widehat{IK_I(tr,t)} \qquad pk \in \widehat{IK_I(tr,t)}}{m \in \widehat{IK_I(tr,t)}}$$

$$(IK\_encrypt\_asym\_sk)\frac{m \in \widehat{IK_I(tr,t)} \qquad sk \in \widehat{IK_I(tr,t)}}{\{m\}_{sk} \in \widehat{IK_I(tr,t)}}$$

$$(IK\_encrypt\_asym\_pk)\frac{m \in \widehat{IK_I(tr,t)} \qquad pk \in \widehat{IK_I(tr,t)}}{\{m\}_{pk} \in \widehat{IK_I(tr,t)}}$$

$$(IK\_nonce)\frac{N_I = New\_Nonce()}{N_I \in \widehat{IK_I(tr,t)}}$$

We have two other rules: (IK_recv) which increases the intruder's knowledge with received messages and $(Intrude\_forge)$ which represents the emission of a message forged by the intruder.

$$(IK\_recv)\frac{recv_\alpha(I,m,t_1) \in tr \qquad t_1 \leq t_2}{m \in IK_I(tr,t_2)}$$

$$(Intrude\_forge)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad m \in \widehat{IK}_I(tr,t_1) \qquad t_1 \leq t_2}{(tr \cup \{send_\alpha(I,m,t_2)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

8.2.5 — Mobility

Mobility of nodes is taken into account in the model, since the values of $\mathbf{T}(t)$ varies with time. We add two realistic assumptions according to modern communication channels.

Assumption 1 *A node moves much slower than a message.*

Assumption 2 *The time needed to forward a message δ is in the same order of magnitude of value as the message transfer time.*

It means that sending and relaying messages does not take a significant time with regard to node movement.

8.3 Neighborhood and (k)-Neighborhoods

8.3.1 — Neighborhood

A node A is *neighbor or* (1)*-neighbor to the node* B *at time* t if $\mathbf{T}_{A,B}(t)$ is finite which means that A is in communication range of B.

Definition 36 (Neighborhood) *Let* $\mathbf{N} = (V, \mathbf{T}(t))$ *be a network, the* neighborhood of a node at time t*, denoted by* $\mathbf{Ng}^1_A(t)$*, is the set of nodes that* A *can reach with a message sent at time* t*. It is formally defined by:*

$$\mathbf{Ng}^1_A(t) = \{X \mid X \in V \wedge \mathbf{T}_{A,X}(t) < +\infty\}.$$

If $tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}$ is a trace, we denote by $\widehat{tr}$ the set of all possible traces that can be inferred from tr using rules defined previously. It allows us to define $t_{transfer}(A, B,)$, the smallest time possible needed to send a fresh message m from A to B at time t using only communication rules.

We also define a protocol $\mathcal{P}_{Forward}$ and an intruder $\mathcal{I}_{Forward}$ which consist in the following unique rule which makes nodes forward message on abstract layer:

$$(Forward)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad recv_\alpha(A, m, t_1) \in tr \qquad t_1 \leq t_2}{(tr \cup \{send_\alpha(A, m, t_2)\}) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

We use $\mathcal{I}_{Forward}$ and $\mathcal{P}_{Forward}$ to establish the minimum transfer time between two nodes. For this we assume the simplest situation, where nodes can only forward messages.

Definition 37 ($t_{transfer}(A, B, t)$) *Let* $\mathbf{N} = (V, \mathbf{T}(t))$ *be a network, and* $tr = \{send_\phi(A, m, t)\}$ *be a trace in* $S_{\mathbf{N},\mathcal{I}_{Forward},\mathcal{P}_{Forward}}$*. We define* $t_{transfer}(A, B, t)$ *by:*

— $t_{transfer}(A, B, t) = \min\{x - t \mid recv_\phi(B, m, x) \in \widehat{tr} \subseteq S_{\mathbf{N},\mathcal{I}_{Forward},\mathcal{P}_{Forward}}\}$.

— *If the event* $recv_\phi(B, m, t)$ *does not belong to* $S_{\mathbf{N},\mathcal{I}_{Forward},\mathcal{P}_{Forward}}$ *then we state that* $t_{transfer}(A, B, t) = +\infty$.

The time $t_{transfer}(A, B, t)$ is different than $\mathbf{T}_{A,B}(t)$, because it can take into account multiple hops, and the relay times needed between nodes. Also, there may be routes which are faster than the direct ones (with some conditions on δ and $\mathbf{N}$, as in the first example in Figure 8).

We denote by $t_{max\_emitter}$ the maximum positive finite time in $\mathbf{T}(t)$, *i.e.* the maximal communication time of two connected nodes. If we assume that $\delta > t_{max\_emitter}$ then we are able to characterize the definition of $\mathbf{Ng}^1_A(t)$ by relations between $t_{max\_emitter}$ and $t_{transfer}(A, B, t)$. In [20] we show that the authenticated ranging protocol, proposed in [PPH08a], is secure under this assumption and exhibit an attack otherwise.

Our idea for (1)-neighborhood is to prove that if a protocol claims to satisfy (1)-neighborhood then there exists a direct way of communication between the nodes. It is captured by the following definition.

Definition 38 *Let $\mathcal{N}$ be a family of networks and $\mathcal{I}$ an intruder. A protocol $\mathcal{P}$ verifies the* (1)*-neighborhood relation in presence of an intruder $\mathcal{I}$ over $\mathcal{N}$ if and only if* $\forall \mathbf{N} = (V, \mathbf{T}(t)) \in \mathcal{N}, \forall A, B \in V$, *$\nexists tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}$ such that $END(A, B, t, t_x) \in tr \wedge B \notin \mathbf{Ng}^1_A(t)$.*

A protocol does not verify a neighborhood property in presence of an intruder $\mathcal{I}$ over a set of networks $\mathcal{N}$ if and only if there is at least a trace *tr* in the networks in $\mathcal{N}$, built by the given protocol and intruder rules, such that $END(A, B, t_{prop}, t) \in tr$ and there is no direct communication possibility between those nodes at time t_{prop}.

8.3.2 — (k)-neighborhood

We extend our definition in order to determine if a node is neighbor to another one after k hops. For simplicity's sake, we consider that all nodes have the same relaying time δ. We can easily generalize our results with a different time for each node in the network.

Definition 39 ((k)-or-less-neighborhood) *Let $\mathbf{N} = (V, \mathbf{T}(t))$ be a network, $\mathbf{Ng}^{\leq k}_A$ (the (k)-or-less-neighborhood of A) is the set of all the nodes B for which there is a path starting at A at time t and ending at B with k hops or less, taking into account the minimal flight time of messages and the relaying time δ. We define it recursively by:*

$$\mathbf{Ng}^{\leq k}_A(t) = \left\{ B \mid \left(X \in \mathbf{Ng}^1_A(t)\right) \wedge \left(B \in \mathbf{Ng}^{\leq (k-1)}_X\left(t + \mathbf{T}_{A,X}(t) + \delta\right)\right) \right\}.$$

This recursive definition means that the (k)-or-less neighborhood of A at time t is the union, for all the neighbors X reachable by A at time t, of the $(k-1)$-or-less-neighborhoods of the different X at time $t + \mathbf{T}_{A,X}(t) + \delta$. Colloquially, a (k)-neighbor of A is a node which can be reached in k or less hops by a message sent from A at time t, and relayed through any other nodes. As expected, the (1)-or-less-neighborhood is the same thing as the neighborhood given in Definition 36. According to our definition, (k)-or-less-neighborhood has the following straightforward property.

Property 1 *Let $\mathbf{N} = (V, \mathbf{T}(t))$ be a network, then we have:*

$$\mathbf{Ng}^{\leq k}_A(t) \subseteq \mathbf{Ng}^{\leq (k+1)}_A(t).$$

We now define the (k)-neighborhood, which is the set of nodes for which there is a path (in the sense of the previous definition) of length exactly k. It is the set of all the nodes B for which there is a path starting at A at time t and ending at B with k hops, but there is no such path with $k - 1$ hops or less.

Definition 40 ((*k*)-neighborhood) *Let* $\mathbf{N} = (V, \mathbf{T}(t))$ *be a network,* $\mathbf{Ng}_A^k$ *((k)-neighborhood of A) is defined by:*

$$\mathbf{Ng}_A^k(t) = \left(\mathbf{Ng}_A^{\leq k}(t)\right) \setminus \left(\mathbf{Ng}_A^{\leq k-1}(t)\right).$$

Now, as in the previous subsection, we can formally define what is a protocol which verifies the (k)-or-less-neighborhood relation in our framework.

Definition 41 *Let* $\mathcal{N}$ *be a network family. A protocol* $\mathcal{P}$ *verifies the* (k)*-or-less-neighborhood relation in presence of an intruder* $\mathcal{I}$ *if and only if* $\forall \mathbf{N} = (V, \mathbf{T}(t)) \in \mathcal{N}, \forall A, B \in V$, $\nexists tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}$ *such that* $\mathit{END}(A, B, t, t_x) \in tr \wedge B \notin \mathbf{Ng}_A^{\leq k}(t)$.

In [20], we present proofs using our model on several examples.

8.4 (k)-or-less-Neighbors Discovery Protocol

Assuming that we have a secure (1)-neighbor discovery protocol, we propose a (k)-or-less-neighbor discovery protocol based on it. This protocols aims to construct the (k)-or-less-neighborhood, by using the knowledge each node has about its neighborhood. For this protocol, we need to consider several *END* events, each being a stepping stone towards a final conclusion. For instance $\mathit{END}^k(A, B, t_{prop}, t)$ expresses the (k)-or-less neighborhood property between A and B at time t_{prop}. We notice that definitions of how a protocol verifies a property follows the same idea as before: there should not exist a trace where an *END* contradicts the physical property it claims. We call this protocol the *Sharek* protocol.

In the first phase of the protocol, each node floods the network with its signed (1)-neighborhood. We model this with the $View$ function, which corresponds to a part of the local neighborhood of a node. After this, no more communications happen.

Then, all the nodes try to map their (2)-or-less-neighborhoods based on **two or more of their** (1)**-neighbors claiming neighborhood to another node**. After that, each node continues by computing its upper neighborhood based on its previous deductions. We do not require that each node knows its whole (1)-neighborhood, but only a subset of it. If two nodes claim neighborhood with a third one, but are not at the same level of neighborhood, then the third one will be accepted at the highest level, since the $(k-1)$-or-less neighborhood is included in the (k)-or-less neighborhood.

This protocol is sensitive to the number of intruders, we present it for only one intruder in the network. This way, the single intruder can not create false conclusions by lying, since there will not be the same claim from another node. If we increase the number of approving nodes in the deduction, then we can resist to more intruders.

For proving our protocol we also assume one of the two following assumptions:

— Nodes have synchronized clocks. This is required by the protocol which makes nodes use timestamp sent by their neighbors. If honest nodes send erroneous times, then the intruder can easily leverage this and trick the protocol into a false conclusion. In the rest, we consider this assumption.

— Nodes are static. *i.e.* $\mathcal{N}_{still}$ as defined above. Then, the neighborhoods will not change over time, and the timestamps become trivially useless. It is easy to infer a proof with this assumption from the previous case.

We give the few rules modeling our protocol:

$$(Sharek\_begin)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}{tr.send_\alpha(A, \{\langle t_{prop}, A, View(A, \mathbf{Ng}^1_A(t_{prop}))\rangle\}_{sk_A}, t) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(Sharek\_basis)\frac{tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \qquad B \in View(A, \mathbf{Ng}^1_A(t_{prop})) \qquad t_{prop} \leq t}{tr.END^1(A, B, t_{prop}, t) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(Sharek\_forward)\frac{\begin{array}{c} tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \\ recv_\alpha(A, \{\langle t_{prop}, B, \{N_1, ..., N_k\}\rangle\}_{sk_B}, t_1) \in tr \\ t_1 \leq t_2 \end{array}}{tr.send_\alpha(A, \{\langle t_{prop}, B, \{N_1, ..., N_k\}\rangle\}_{sk_B}, t_2) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

$$(Sharek\_makestep)\frac{\begin{array}{c} tr \in S_{\mathbf{N},\mathcal{I},\mathcal{P}} \\ recv_\alpha(A, \{\langle t_{prop}, B, \{D, ...\}\rangle\}_{sk_B}, t_1) \in tr \\ recv_\alpha(A, \{\langle t_{prop}, C, \{D, ...\}\rangle\}_{sk_C}, t_2) \in tr \\ END^b(A, B, t_{prop}, t_b) \in tr \\ END^c(A, C, t_{prop}, t_c) \in tr \\ \forall t_x \in \{t_1, t_2, t_b, t_c\}, t_x \leq t_d \end{array}}{tr.END^{max(b,c)+1}(A, D, t_{prop}, t_d) \in S_{\mathbf{N},\mathcal{I},\mathcal{P}}}$$

In [20], we prove the security of this protocol. We use $\mathcal{N}_{1,sym}$ the family of networks where there is only a single intruder, and where all the connections are symmetric. First we prove that our protocol verifies (1)-neighborhood property then we assume Sharek protocol verifies all (i)-neighborhood properties for $i \leq k$ and we show that it verifies ($k+1$)-neighborhood property.

8.5 Conclusion

We modeled (k)-neighborhood property in a wireless network in order to verify protocols in the presence of intruders. We take into account time and movement of nodes. We also proposed the Sharek protocol, which securely discover (k)-neighbors based on the knowledge of the (1)-neighborhood, in presence of one intruder. We can generalize this protocol in order to be resistant to several intruders. We also provide a formal proof of the security of the Sharek protocol in our formal model.

Chapter 9

Independent Intruders

Une des principales caractéristiques du modèle d'intrus introduit par Dolev et Yao est que l'intrus contrôle le réseau. Dans ce chapitre, nous avons développé un modèle formel à base de contraintes pour vérifier des protocoles de routage dans les réseaux ad-hoc, en présence de nœuds malicieux intépendants : ces intrus ne partagent pas les informations collectées individuellement et ne collaborent pas entre eux. Etant donné une topologie, un nombre borné de session, nous avons proposé une procédure pour trouver une attaque en présence de tels intrus pour des protocoles de routage dans les réseaux sans fil ad-hoc.

Tout d'abord, nous avons défini un système de contraintes adapté à de tels intrus, appelé *système de contraintes partiellement bien-défini*. Chaque intrus augmente uniquement sa propre connaissance, alors que les systèmes de contraintes existants considéraient une connaissance global. Pour trouver une solution à un système de contrainte il est usuel de donner des règles de simplification afin de réduire le système à un système plus simple. Les règles existantes ne fonctionnent pas pour les systèmes de contraintes partiellement bien-définis. Nous avons donc construit de nouvelles règles de simplification pour de tels systèmes. Bien entendu nous avons montré la correction, la complétude et la terminaison de ces règles.

Ensuite nous avons utilisé un calcul permettant de modéliser les protocoles de routage dans les réseaux ad-hoc proposé par M. Arnaud [Arn12], et l'avons adapté afin de prendre en compte les caractéristiques d'intrus indépendents. Nous avons aussi proposé des règles de transfomation qui étant donné un protocol de routage modèlisé dans ce calcul contruisent un système de contraintes partiellement bien-défini.

Contents

Ad-hoc network is a collection of wireless devices forming a network without the aid of any stand-alone infrastructure or centralized administration. Each device itself acts as a router for forwarding and receiving packets to and from other devices. A routing protocol is always needed to transport data and finding a path using control packet is a critical task. Several protocols have been presented to accomplish routing in ad-hoc networks such as [JMB01, HPJ05, PR99, BV04, PH02, ZA02]. An example of routing protocol for ad-hoc network is Dynamic Source Routing protocol (DSR)[JMB01]. The route discovery in DSR is divided into two phases:

— In the *request phase* a source S broadcasts a route request message which is received by all nodes currently within wireless communication range of S. Request message contains source and destination identifiers (S and D) and a unique request Id, chosen by S. Each route request also contains the list of forwarding nodes identifiers. At the beginning of the protocol, this route list is empty. When a certain node receives a request two situations occurs:
  – If the current node is the target of the request, it returns a route reply which contains the list of all forwarding nodes including the source and the destination.
  – Otherwise, the current node appends its own identifier to the list of all forwarding nodes and broadcasts it. Note that, if the current node has already seen another request message from the same source bearing the same request Id, or if its own identifier is already listed in the route list, it discards the request.

— In the *reply phase*, the node D (the destination) replies back the discovered route to S through the intermediate nodes appears on the list. The intermediate node that receives the reply message just forwards the message to the previous node in the list.

Figure 11 illustrates normal DSR route discovery between source node S and node D.

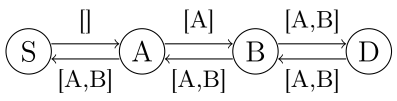


Figure 11 – DSR route discovery.

We describe the Secure Routing Protocol (SRP) which was claimed to be secure [PH02]. It is a generic way for securing on-demand source routing protocols. We use the following notations:

— [] is empty list and $a :: r$ the insertion of a to the list r.

— $< a, b, c >$ denotes a tuple.

— req, rep are identifiers indicating the phase of the execution.

— Id is request identifier.

— l, l' are lists of nodes

— K_{SD} is a shared key between S and D.

— *hmac* is a MAC function and m, m' represent MACs.

The protocol works in two phases:

Request phase:

— The source S broadcasts to its neighbors a request containing its identifier S, the destination identifier D, an Id that identifies the request, an empty list [], and a MAC computed over the content of the request with the key.

— Each intermediate node that receives a request message checks that the list representing the route ends with the identifier of one of its neighbors, appends its identifier to the list, and broadcasts the modified request.

In Figure 12, we give a more formal description of this phase.

Request phase:

The source S broadcasts $\langle req, S, D, Id, [], hmac(\langle req, S, D, Id\rangle, K_{SD})\rangle$

Intermediate node V receives $\langle req, S, D, Id, l, m\rangle$
V checks that the last element of l is one of its neighbors
V broadcasts $\langle req, S, D, Id, l :: V, m'\rangle$

The destination D receives $\langle req, S, D, Id, l', hmac(\langle req, S, D, Id\rangle, K_{SD})\rangle$

Figure 12 – Specification of SRP Request Phase Applied to DRS.

Reply phase:

— Once the destination node received a message, it checks that last node in the route is one of its neighbors and verifies the MAC. Then it initiates the reply phase by sending a message containing the discovered route with a MAC computed over it with the key K_{SD}.

— Each node checks if its identifier appears in the route list between two of its neighbors, if so they forward the replay to the previous node in the route.

— Then once the source S receives an answer containing a route to D with a MAC matching this route. It checks that the route does not contains a loop and that its neighbor in the route is indeed one of its neighbor in the network.

In Figure 13, we give a more formal description of this second phase.

Against a routing protocol, the attacker goal is to force S and D to believe in a false route. We now describe two attacks on SRP, one with colluding malicious nodes and one with independent adversaries.

Consider the network represented by graph G_0 in Figure 14, where the edges between nodes represent symmetric communication links. The node S initiates a route discovery procedure to reach D. Let M_1 and M_2 be two malicious nodes. We show that if M_1 and M_2 are collaborating and sharing their acquired knowledge, then there is an attack on SRP protocol applied to DSR, for any "sub-network" between them. We just assume that each node

Reply phase:

The destination D sends $\langle rep, S, D, Id, l', hmac(\langle req, D, S, Id, l'\rangle, K_{SD})\rangle$

Intermediate node V receives $\langle rep, D, S, Id, l', m'\rangle$
V checks if its identifier appears in the l' between two of its neighbors
V sends to next node in the list $\langle rep, D, S, Id, l', m'\rangle$

The source S receives $\langle rep, D, S, Id, l', hmac(\langle rep, D, S, Id, l'\rangle, K_{SD})\rangle$

Figure 13 – Specification of SRP Reply Phase Applied to DRS.

knows its neighbors.

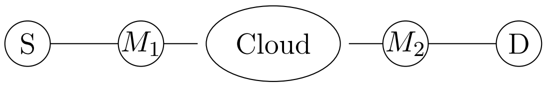


Figure 14 – Graph G_0

Request phase: To initiate the request phase S chooses a unique Id, in order to avoid replay attacks, and broadcasts $m_1 = \langle req, S, D, Id, [], hmac(\langle req, S, D, Id\rangle, K_{SD})\rangle$. The node M_1 receives the route request message m_1. As M_1 and M_2 share their knowledge, then M_2 also get the message m_1 (from M_1). So, M_2 can modify m_1 and obtain a fake message $m'_1 = \langle req, S, D, Id, [M_1, M_2], hmac(\langle req, S, D, Id\rangle, K_{SD})\rangle$ and send it to D. Since, M_2 is neighbor of D and the MAC of the message m'_1 is correct, then D accepts m'_1.

Reply phase: D builds the replay message $m_2 = \langle req, S, D, Id, [M_1, M_2], hmac(\langle req, S, D, Id, [M_1, M_2]\rangle, K_{SD})\rangle$ and sends it to M_2 in order to propagate back to S. When the node M_2 receives the message m_2, M_2 shares it with M_1. In order to complete the attack M_1 just forwards m_2 to S. As the MAC of m_2 built by D with the correct key K_{DS}, M_1 is neighbor of S, and the route $[M_1, M_2]$ is free from loops, then S believes that $[S, M_1, M_2, D]$ is a valid route.

This attack is know as *wormhole attack.*

We now assume that M_1 and M_2 are two malicious nodes, each knowing only its neighbors (local knowledge) and there is no collaboration between them. If we consider the network represented by graph G_1 in Figure 15, there is still an attack on SRP. We remark that M_1 nor M_2 can make alone an attack on SRP in network G_1.

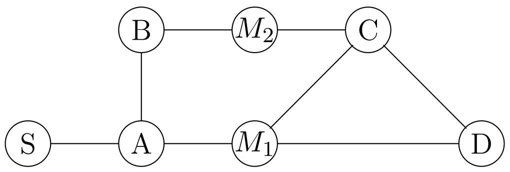


Figure 15 – Graph G_1

First request phase: The source node S initiates the SRP protocol by broadcasting a request message $\langle req, S, D, Id, [], h_s\rangle$, where $h_s = hmac(\langle req, S, D, Id\rangle, K_{SD})$. Then A receives the request message, checks that S is one of its neighbors, adds its identifier to the list and broadcasts the resulting message $\langle req, S, D, Id, [A], h_s\rangle$. The node B receives the message and does the same as A, then M_2 receives $m_1 = \langle req, S, D, Id, [A, B], h_s\rangle$. The malicious node M_2 forges the message $m_1' = \langle req, S, D, Id, [A, M_2], h_s\rangle$ and sends it to C which accepts it and broadcasts $m_2 = \langle req, S, D, Id, [A, M_2, C], h_s\rangle$. Then D and M_1 receive m_2. Since C is neighbor of D, D accepts m_2

First reply phase: The node D replies $m_3 = \langle req, S, D, Id, [A, M_2, C], h_D\rangle$, where $h_D = hmac(\langle req, D, S, Id, [A, M_2, C]\rangle, K_{SD})$. The malicious node M_1 receives the message $m_2 = \langle req, S, D, Id, [A, M_2, C], h_s\rangle$ from C. Now M_1 believes that M_2 is neighbor of A and C (since from the point of view of M_1, the node M_2 is honest).

Second request phase: Thus, M_1 forges $m_2' = \langle req, S, D, Id, [A, M_1, M_2, C], h_s\rangle$, and sends it to D in the name of C. Then D accepts the message.

Second reply phase: the node D generates $h_D' = \langle req, D, S, Id, [A, M_1, M_2, C]\rangle, K_{SD})$ over a false route $[A, M_1, M_2, C]$, and replies $\langle req,\ D, S, Id,\ [A, M_1, M_2, C],\ h_D'\rangle$ to C and M_1. M_1 forwards it to A who accepts the message and forwards it to S. Finally S verifies the MAC, checks that A is its neighbor and that the route is free from loops, and accepts the false route $[S, A, M_1, M_2, C, D]$, which concludes the attack.

We notice that M_2 cannot mount alone this attack since:

— m_3 is received by M_2 cannot directly be forwarded to B because he will drop it since B is not in the route list.

— even if M_2 adds B to the list so that B will accept the message, but the message will be rejected by S when MAC will be checked.

In wireless ad-hoc network where communication can only occur node to node, it is not realistic to assume that an intruder can listen to all messages exchanged in the network. The intruder has to be located somewhere, and can thus only receive the messages sent by his immediate neighbors. Similarly, he can only send messages directly to his neighbors. As a consequence we have several intruders (malicious nodes) each located somewhere on the network and listen only to his neighbors and thus has only local knowledge. Our aim is to propose a formal verification model which takes into account such *independent intruders*. This intruder model is weaker than the classical one proposed by Dolev-Yao, and also than the one proposed by M. Arnaud [Arn12]. In other words, it is possible that there is no attack for a given protocol in presence of independent intruder, but there is an attack in presence of collaborative intruders that share information. Both models are useful because the first one allows to verify protocols against possible wormhole attack, where intruder are very powerful, while the second one models the situation of several nodes independent malicious nodes.

Moreover if we consider that an adversary wants to attack nodes of an existing network then it is possible to take the control of some nodes without changing any communication settings. While it is very difficult to add an extra communication channel to such small devices in order to establish a private channel with a central intruder station that share knowledge collected by each corrupted nodes. Hence modeling intruders with independent nodes seems again more realistic.

Related Work: In [MS01a, MS03, Shm04, CLCZ10], the authors propose simplification rules to transform a constraints system into a simpler constraints system called *solved form*. They also provide a decision procedure for solving such systems. Generally they assume one intruder with a global knowledge, which is always increasing.

In [Arn12], a constraint based formal model was proposed to verify the security of ad-hoc network protocols in the presence of an intruder that controls several malicious nodes. These nodes collaborate between each other by sharing their acquired knowledge. This constraints system is an extension of the one proposed by Comon-Lundh et al. in [CLCZ10].

To the best of our knowledge, there is no work considering constraints systems that model several independent intruders.

Contributions: For a bounded number of session, we develop a constraint based model for the formal verification of secured ad-hoc routing protocols in a context where intruders are independent malicious nodes. We propose what we call *partially well-formed constraints system*, which is suitable to represent several independent intruders where each has his own knowledge. Known constraints systems permit to represent only one intruder. To decide if a certain constraints system has a solution or not, usually a set of simplification rules is used in order to transform the system into simpler one. But, the existing rules do not work for our system, we propose rules to simplify partially well-formed constraints systems. Naturally, we prove termination, soundness and completeness of our rules.

Moreover in [Kas12], we use pi-calculus to model ad-hoc network routing protocols. We extend the concrete transition system proposed in [Arn12] in order to represent all executions of certain protocol in presence of independent malicious nodes that do not share their knowledge. We also give transformation rules to construct a partially well-formed constraints system. Hence we obtain a decision procedure to decide if a given protocol is insecure in the presence of independent malicious nodes for a given network topology. The details of this modeling and transformations for obtaining a partially well-formed constraints system are given in [Kas12].

Outline: In the next section we introduce notations and background. In Section 9.2 we describe our intruder model. In Section 9.3 we define constraints system in order to model independent intruder. In Section 9.4, we propose a set of rules for transforming our constraints system into a quasi-solved form. Finally in Section 9.5, we give a resolution procedure for such constraints system.

9.1 Preliminaries

We assume the *perfect encryption assumption*, it means that the only way to obtain information of a cryptographic primitive is to known his inverse function. Hence all messages are modeled as elements of some term algebra where the constructors of that algebra are seen as abstractions of cryptographic algorithms.

Definition 42 (Signature). *A* signature *is a finite set of function symbols $\mathcal{F}$, where every $f \in \mathcal{F}$ associated with a positive natural number called* arity*, i.e. the number of arguments it has, denoted by ar. The set of function symbols of arity p is denoted by $\mathcal{F}_p$. We distinguish a set of secret function $\mathcal{F}_{priv}$ that contains private functions, i.e. functions that the intruder can not use.*

Definition 43 (Terms). *Let $\mathcal{F}$ be a signature, $\mathcal{X}$ be an infinite set of variables and $\mathcal{N}$ be an infinite set of names. The set of* terms *$\mathcal{T}(\mathcal{F},\mathcal{N},\mathcal{X})$ is defined to be the smallest set such that:*
— $\mathcal{N} \cup \mathcal{X} \subseteq \mathcal{T}(\mathcal{F},\mathcal{N},\mathcal{X})$.

— *Any $f \in \mathcal{F}_p$, if $f : s_1 \times \cdots \times s_p \rightarrow s$ and $t_1, \ldots, t_p \in \mathcal{T}(\mathcal{F},\mathcal{N},\mathcal{X})$ such that for any i, t_i is of the sort s_i, then $f(t_1, \ldots, t_p) \in \mathcal{T}(\mathcal{F},\mathcal{N},\mathcal{X})$.*

Definition 44 (Term Size). *Let t be a ground term. The* size *of t, denoted by $|t|$, is the number of function symbols used to build t, defined recursively by:*
— $|t| = 0$ *if* $t \in \mathcal{N}$

— $|t| = |u_1| + \cdots + |u_p| + 1$ *if* $t = f(u_1, \cdots, u_p)$ *for certain* $f \in \mathcal{F}_p$

Example Let $t = \langle senc(m,k), a\rangle$, then $|t| = 2$.

Definition 45 (Subterms). *Let t be a term, we define a set of* subterms *$st(t)$ as a smallest set such that:*
— $t \in st(t)$.

— $f(t_1, \ldots, t_p) \in st(t) \Rightarrow t_1, \ldots, t_p \in st(t)$, *for* $f \in \mathcal{F}_p$
For a set of terms T we define $st(T)$ as $\bigcup_{t \in T} st(t)$.

We consider a finite set of variables $\mathcal{X}$ and a finite set of names $\mathcal{N}$, that represents nonces, keys, agent names and some special names like $req, rep, Id, etc\ldots$. We assume that names and variables are given with sorts. We consider sort $Agent$ that contains agent names (nodes), sort Key that contains keys and sort $List$ that contains lists. Also we consider a special sort Msg that subsumes all other sorts. Furthermore, we consider a special set of names $\mathcal{N}_{agent}$, which represents only agent names, i.e $\mathcal{N}_{agent}$ contains the elements of $\mathcal{N}$ that are of sort $Agent$.

In the rest, we use the signature $\mathcal{F} = \{senc, aenc, \langle -,- \rangle, - :: -, hmac, sig, priv\}$ of function symbols, that are modeled as follows:
— $f : Msg \times Key \rightarrow Msg$ *for* $f \in \{senc, aenc, sig, hmac\}$

— $\langle -,- \rangle : Msg \times Msg \to Msg$

— $h : Msg \to Msg$

— $- :: - : List \times Agent \to List$

— $g : Agent \to Msg \quad for\ g \in \{pub, priv\}$

The symbol $\langle -,- \rangle$ represents pairing function, and $- :: -$ is list constructor. We write $\langle u_1, u_2, u_3 \rangle$ for the term $\langle u_1, \langle u_2, u_3 \rangle\rangle$, and $[u_1, u_2, u_3]$ for $(([] :: u_1) :: u_2) :: u_3$, where $[]$ represents the empty list. The term *priv*(A) represents the private key associated to the agent A. The intruder has no access to the *priv* function as he cannot generate private keys. For simplicity, we assume that agent identities are also their public keys. The function symbol *senc* (resp. *aenc*) is used to model symmetric (resp. asymmetric) encryption, *hmac* is used to compute message authentication code, and *sig* is used to sign messages.

Example Consider a set $T = \{hmac(\langle m, m' \rangle, k), [A, B]\}$, then $st(T) = \{hmac(\langle m,\ m' \rangle, k),$ $\langle m, m' \rangle, k,\ m, m', [A, B],\ [B], B,\ A, []\}$.

Definition 46 *Let t be a term, we define a set of its variables by $vars(t) = \mathcal{X} \cap st(t)$. For a set of terms T we define $vars(T)$ as $\bigcup_{t \in T} vars(t)$.*
A term t that does not contain variables, i.e. $vars(t) = \phi$ is called ground term.

Example Consider a set $T = \{[A, B, x], senc(y, K)\}$, where A and B are agent names, x and y are variables, and K is a symmetric key, then $vars(T) = \{x, y\}$.

Definition 47 (Substitution). *A* substitution *σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{N}, \mathcal{X})$ with the domain $dom(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$. We often use postfix notation $t\sigma$ instead of $\sigma(t)$. We consider only well-sorted substitutions, that is substitutions for which x and $x\sigma$ have the same sort. We call a substitution σ ground if for every $x \in dom(\sigma)$, the term $x\sigma$ is ground.*

Example The substitution $\sigma = \{x \to \langle A, N_A \rangle\}$, where A is an agent name, and N_A is a nonce cannot be applied to the term $aenc(m, pub(x))$.

Definition 48 (Unification and mgu). *Two terms t_1 and t_2 are unifiable if there exists a substitution σ such that $t_1\sigma = t_2\sigma$. The* most general unifier *between two terms t_1 and t_2, denoted by $mgu(t_1, t_2)$ is a substitution, such that for any substitution θ if $t_1\theta = t_2\theta$, then there exists θ' such that $\theta' = mgu(t_1, t_2)\theta$.*

9.2 Intruder Model

As it has been done in [Arn12], we extend the classic Dolev-Yao intruder deduction system. The relation $T \vdash u$ represents the fact that the term t is deducible from the set of terms T. Our deduction system is described in Figure 16. The first inference rule (A) is an axiom, and the second rule (C) describes the composition rules. The remaining inference rules describes the decomposition rules. These deduction rules say that an intruder can compose messages by pairing, building lists, encrypting and signing messages providing he has the corresponding keys. Conversely, he can retrieve the components of a pair or a list, and he can also decrypt (resp. unsign) encrypted messages (resp. signed messages) provided he has the decryption key. However, he can not create private keys, for instance, as the deduction system does not allow him to use function *priv*.

$$(A)\ \frac{u \in T}{T \vdash u} \qquad (C)\ \frac{T \vdash u_1 \quad T \vdash u_2}{T \vdash f(u_1, u_2)}\ f \in \{senc, aenc, \langle\rangle, ::, hmac, sig\}$$

$$(UP)\ \frac{T \vdash \langle u_1, u_2\rangle}{T \vdash u_i}\ i \in \{1,2\} \qquad (UL)\ \frac{T \vdash u_1 :: u_2}{T \vdash u_i}\ i \in \{1,2\}$$

$$(SD)\ \frac{T \vdash senc(u_1, u_2) \quad T \vdash u_2}{T \vdash u_1} \qquad (AD)\ \frac{T \vdash aenc(u_1, pub(u_2)) \quad T \vdash priv(u_2)}{T \vdash u_1}$$

$$(US)\ \frac{T \vdash sig(u_1, priv(u_2)) \quad T \vdash pub(u_2)}{T \vdash u_1}$$

Figure 16 – Intruder deduction system.

Definition 49 — *The* size of a proof π*, denoted by* $|\pi|$*, is the number of nodes of* π*.*

— *Let T be any set of terms, and u be any term. A proof π of $T \vdash u$ is* minimal *if there is no proof π' of $T \vdash u$ with fewer nodes than π, i.e. $|\pi'| < |\pi|$.*

— *Let T be any set of terms, and u be any term. We say that a proof of $T \vdash u$ is* simple proof *if there is no repeated nodes on any branch. Remark that a sub-proof of a simple proof is also simple.*

9.3 Partially Well-Formed Constraints System

Several works [MS01b, CLCZ10] consider so-called well-formed constraints system of the form $T_1 \Vdash u_1 \wedge \cdots \wedge T_n \Vdash u_n$, where T_i is a set of terms and u_i is a term, that satisfies the two following assumptions:

- Intruder knowledge monotonicity: $T_1 \subseteq \cdots \subseteq T_n$.
- Variable origination: if x occurs in $vars(T_i)$ for certain T_i then there exists $k < i$ such that $x \in vars(u_k)$.

The first assumption models the fact that the intruder controls the network, as it was proposed by Dolev-Yao [DY83b]. So the intruder listens to all communications and thus his knowledge is always increasing. The second assumption models the behavior of the agents participating to the protocol saying that all used variables have values previously instantiated. In order to reason about protocol insecurity taking into account several independent intruders, we define a new kind of constraints systems called *partially well-formed constraints system.*

Definition 50 (PWFCS). *Let m be the number of intruders, n be the total number of constraints, and n_j be the number of constraints associated to the j^{th} intruder, so we have $\sum_{j=1}^{j=m} n_j = n$. A* partially well-formed constraints system *$\mathcal{C}$ is either $\perp$ or a finite conjunction of constraints satisfying origination and partial monotonicity, modeled as follows:*

$$\mathcal{C} \quad = \quad T_1^l \Vdash u_1 \wedge \cdots \wedge T_n^q \Vdash u_n$$

where l and $q \in \{1, \ldots, m\}$.

- *Origination, i.e. for any $j \in \{1, \ldots, m\}$, if $x \in vars(T_i^j)$ then there exists $k < i$ such that $x \in vars(u_k)$.*
- *Partial monotonicity, i.e. $T_k^j \subseteq T_i^j$ for every $j \in \{1, 2, \ldots, m\}$ such that $k < i$.*

The constraint $T_i^j \Vdash u_i$ represents the i^{th} constraint in the system, and it is associated to the j^{th} intruder. T_i^j is a finite set of terms called the left-hand side of the constraint, and u_i is a term called the right-hand side of the constraint.

The partial monotonicity condition states that the knowledge of each intruder is always increasing, and that when a certain intruder acquires a new message only his knowledge is increased.

The origination condition stays the same modulo the presence of several intruders. It says that each time a new variable is introduced, it first occurs in some right hand side of the constraint system.

Definition 51 (Right and Left Hand Sides). *Let $\mathcal{C}$ be a constraints system. We denote by $rhs(\mathcal{C})$ (resp. $lhs(\mathcal{C})$) the set of* right-hand side *terms (resp.* left-hand side *sets of terms) of $\mathcal{C}$. Formally, $rhs(\mathcal{C})$ and $lhs(\mathcal{C})$ are defined recursively in the following way:*

$$\begin{array}{ll} rhs(\perp) = \phi & lhs(\perp) = \phi \\ rhs(\mathcal{C} \wedge T_i^j \Vdash u_i) = rhs(\mathcal{C}) \cup \{u_i\} & lhs(\mathcal{C} \wedge T_i^j \Vdash u_i) = lhs(\mathcal{C}) \cup \{T_i^j\} \end{array}$$

The subterms of $\mathcal{C}$ is defined as $st(\mathcal{C}) = st(lhs(\mathcal{C})) \cup st(rhs(\mathcal{C}))$. Also we define right-hand variables of $\mathcal{C}$ by $rvars(\mathcal{C}) = vars(rhs(\mathcal{C}))$. Due to the origination property $rvars(\mathcal{C})$ contains all the variables that appear in $\mathcal{C}$. One other consequence of the origination is the following property, which says that for any variable there exists a constraint, denoted $T_x \Vdash u_x$, where the variable appears for the first time on the right hand side.

Property 2 *For any variable $x \in vars(\mathcal{C})$, there exist l and k such that: $T_k^l \Vdash u_k \in \mathcal{C}, x \notin vars(T_k^l), x \in vars(u_k)$ and for every $T_i^j \Vdash u_i \in \mathcal{C}$ such that $x \in T_i^j$, $i > k$. We denote T_k^l by T_x and u_k by u_x.*

Definition 52 (Solution) *A* solution θ *of a partially well-formed constraints system* $\mathcal{C}$ *is a well-sorted ground substitution whose domain is* $rvars(\mathcal{C})$ *and such that, for every* $T_i^j \Vdash u_i \in \mathcal{C}$, $T_i^j\theta \vdash u_i\theta$.

Definition 53 (Minimal Solution) *Let* $\mathcal{C}$ *be a partially well-formed constraints system and* $rvars(\mathcal{C}) = \{x_1, \cdots, x_p\}$. *A solution* θ *of* $\mathcal{C}$ *is* minimal *if for every solution* θ' *of* $\mathcal{C}$, *we have* $\sum_{j=1}^{j=p} |x_j\theta| \leq \sum_{j=1}^{j=p} |x_j\theta'|$.

Example Consider the partially well-formed constraints system $\mathcal{C}_2$ of Figure 17. $\mathcal{C}_2$ has a minimal solution $\theta = \{x \mapsto senc(m,k)\}$. Another possible solution of $\mathcal{C}_2$ is $\theta' = \{x \mapsto senc(senc(m,k), k_1)$ but it is not minimal.

$$\begin{aligned} T_1^1 &= \{m, k, k_1\} \Vdash x \\ T_2^2 &= \{x, k_1\} \Vdash senc(m,k) \end{aligned}$$

Figure 17 – Partially well-formed constraints system $\mathcal{C}_2$

9.4 Simplifying Constraints System

Deciding if a certain constraints system has a solution is not an easy task. There exist procedures to solve a well-formed constraints system according to the variant of intruder model considered. However the main idea of these procedure is to use a set of simplification rules in order to reduce the initial system of constraints into a simpler one that has a solution. A good simple form to decide if a certain constraints system has a solution or not is the one used in [CLCZ10], which is called *solved form.* A constraints system is in solved form if all its constraints are of the form $T \Vdash x$ where x is a variable. Any constraints system in this form has a solution. It is enough to assign a ground term from left-hand side of each constraint to the right-hand variable. Comon-Lundh et al. in [CLCZ10] prove that any well-formed constraints system that has a solution can be transformed to the solved form. But, this is not the case of partially well-formed constraints system. Consider the partially well-formed constraints system $\mathcal{C}_1$ of Figure 18, $\mathcal{C}_1$ has a solution $\theta = \{x \mapsto \langle a, b\rangle\}$. But, using simplification rules proposed in [CLCZ10] $\mathcal{C}_1$ can not be transformed into solved-form. For this reason we consider another form we called *quasi-solved form* defined below, where right hand side of a constraint can either be a variable or a constant. As a consequence $\mathcal{C}_1$ of Figure 18 is in quasi-solved form.

Definition 54 (Quasi-solved form). *A partially well-formed constraints system is in* quasi-solved form *if it is* $\perp$ *or each of its constraints* $T_i^j \Vdash u_i$ *satisfies:*

$$\begin{array}{rcl} T_1^1 & = & \{a, b\} \Vdash x \\ T_2^2 & = & \{x\} \Vdash a \\ T_3^3 & = & \{x\} \Vdash b \end{array}$$

Figure 18 – Partially well-formed constraints system $\mathcal{C}_1$

1. u_i is either variable or ground term.

2. $T_i^j \cup \{x \mid T_k^j \Vdash x \in \mathcal{C}, k < i\} \nvdash u_i$[1]

In Figure 19, we propose a simplification rules that transform any any partially well-formed constraints system that has a solution into a quasi-solved form that has a solution. The rule R_{ax} removes the redundant constraints, i.e when the constraint is a logical consequence of previous constraints. The rule R_{unif} guesses some equality between two parts of the sent messages. R'_{unif} allows assigning to a variable a concatenation (or list) of two other terms. The rule R_f decomposes a term $f(u, v)$ for certain $f \in \{senc, aenc, \langle -, - \rangle, - :: -, hmac, sig\}$, and R_{fail} states the constraints system has no solution if it contains a constraints $T_i^J \Vdash u_i$ free from variables with u_i is not deducible from T_i^j.

In fact all the rules are indexed by a substitution, when there is no index then the identity substitution is implicitly assumed. We write $\mathcal{C} \leadsto_\sigma^n \hat{\mathcal{C}}$ if there are $\mathcal{C}_1, \cdots, \mathcal{C}_n$ with $n \geqslant 1$, $\hat{\mathcal{C}} = \mathcal{C}_n$, $\mathcal{C} \leadsto_{\sigma_1} \mathcal{C}_1 \leadsto_{\sigma_2} \cdots \leadsto_{\sigma_n} \mathcal{C}_n$ and $\sigma = \sigma_n \cdots \sigma_2 \sigma_1$. We write $\mathcal{C} \leadsto_\sigma^* \hat{\mathcal{C}}$ if there exists $n \geqslant 1$ such that $\mathcal{C} \leadsto_\sigma^n \hat{\mathcal{C}}$.

$$\begin{array}{rlcl}
R_{ax} & : & \mathcal{C} \wedge T_i^j \Vdash u_i \leadsto \mathcal{C} & \text{if } T_i^j \cup \{x \mid T_k^j \Vdash x \in \mathcal{C}, k < i\} \vdash u_i \\
R_{unif} & : & \mathcal{C} \leadsto_\sigma \mathcal{C}\sigma & \sigma = mgu(t_1, t_2),\ t_1, t_2 \in st(\mathcal{C}) \\
R'_{unif} & : & \mathcal{C} \wedge T_i^j \Vdash u_i \leadsto_\sigma \mathcal{C}\sigma \wedge T_i^j\sigma \Vdash u_i\sigma & \sigma = mgu(t, f(t_1, t_2)), f \in \{\langle -, - \rangle, - :: -\}, \\
 & & & t \in vars(u_i), t_1, t_2 \in st(T_k^l),\ \text{where } k \leq i \\
R_f & : & \mathcal{C} \wedge T^j \Vdash f(u, v) \leadsto \mathcal{C} \wedge T^j \Vdash u \wedge T^j \Vdash v & \text{if } f \in \{senc, aenc, \langle -, - \rangle, - :: -, hmac, sig\} \\
R_{fail} & : & \mathcal{C} \wedge T_i^j \Vdash u_i \leadsto \bot & \text{if } T_i^j = \emptyset, \text{or } vars(T_i^j \cup \{u_i\}) = \emptyset \\
 & & & \text{and } T_i^j \nvdash u_i
\end{array}$$

Figure 19 – Simplification rules.

We first prove that using these rules still produce a partially well-formed constraints system, then we prove the completeness, soundness and termination.

[1]It means that we cannot apply the rule R_{ax} to $T_i^j \Vdash u_i$.

Lemma 17 *The simplification rules of Figure 19 transform a partially well-formed constraints system into a partially well-formed constraints system.*

Theorem 18 *Let $\mathcal{C}$ be a partially well-formed constraints system, we have that:*

— *Soundness (Correctness): If $\mathcal{C} \rightsquigarrow_{\sigma}^{*} \hat{\mathcal{C}}$ for some substitution σ, and if θ' is a solution of $\hat{\mathcal{C}}$ then $\theta = \sigma\theta'$ is a solution of $\mathcal{C}$.*

— *Completeness: If θ is a minimal solution of $\mathcal{C}$, then there exist a partially well-formed constraint system $\hat{\mathcal{C}}$ in quasi-solved form and substitutions σ and θ' such that $\theta = \sigma\theta'$, $\mathcal{C} \rightsquigarrow_{\sigma}^{*} \hat{\mathcal{C}}$ and θ' is a solution of $\hat{\mathcal{C}}$.*

— *Termination: There is no infinite derivation sequence $\mathcal{C} \rightsquigarrow_{\sigma_1} \mathcal{C}_1 \rightsquigarrow_{\sigma_2} \mathcal{C}_2 \cdots \rightsquigarrow_{\sigma_n} \mathcal{C}_n \cdots$.*

The difficult part is the completeness. Our idea for constructing θ' is to construct a minimal solution. Moreover we first to apply the rule R_{unif} then the rule R_f. After if the system is still not in quasi-solved form we construct a minimal solution using the rule R'_{unif}. Finally we apply rule R_{ax} in order to eliminate redundant constraints. Detailed proofs of these results can be found in [Kas12].

9.5 Solving quasi-solved form constraints system

Let $\mathcal{C}$ be a partially well-formed constraints system, in quasi-solved form different from $\perp$. We can see $\mathcal{C}$ as two systems:

- $\mathcal{C}_1$ contains constraints $T_i^j \Vdash x_i$ of $\mathcal{C}$ with x_i is variable.
- $\mathcal{C}_2$ contains constraints $T_i^j \Vdash u_i$ of $\mathcal{C}$ with u_i is a ground term.

Finding a solution on $\mathcal{C}_1$ corresponds to the usual solved form proposed in the literature. Usually it is enough to assign to x_i a ground term from T_i^j for each $T_i^j \Vdash x_i$ of $\mathcal{C}_1$ to get a solution of $\mathcal{C}_1$. After we just have to check that constraints system $\mathcal{C}_2$ is satisfied or not.

This concludes the resolution of partially well-formed constraints system. For verifying routing protocol, we adapt the modeling and transformations of M. Arnaud in applied π-calculus for the independent intruders in order to obtain a partially well-formed constraints system.

9.6 Conclusion

We considered an intruder model that is close to the reality by limiting the sharing ability of intruders. In this model, we assume the existence of several independent intruders that do not share any knowledge. In this setting, we show that deciding whether there is an attack against a given ad-hoc routing protocol for a bounded number of sessions can be done in NP time.

We define a new type of constraints system called partially well-formed constraints system and propose simplification rules that can be used to transform it into quasi-solved form. This new constraints system is convenient to model security protocols in the presence of several independents intruders.

We extend modeling in applied π-calculus and transformations for obtain constraints system proposed by M. Arnaud in [Arn12] to protocols considering the presence of malicious nodes that do not share their knowledge. It gives us a decision procedure for verifying ad-hoc routing protocols in presence to independent intruder.

Chapter 10

Routing with Shared Tabu List

Ce chapitre présente l'étude de marches aléatoires employant des listes taboues afin d'éviter que les messages passent deux fois par le même nœud. L'itinéraire du message est stocké dans son contenu. Il est donc posible d'éviter de repasser par un nœud déjà visité. Pour des raisons d'implémentations évidentes nos listes taboues sont finies. Il faut donc déterminer la taille de la liste mais aussi la politique appropriée pour gérer le cas des listes pleines. Nous definissons une large classe de politiques de gestion des marches aléatoires avec liste taboue. Nous donnons une condition nécessaire et suffisante sur ces politiques afin d'assurer la convergence de ces algorithmes en temps fini. De plus nous caractérisons en fonction de la taille des listes taboues la politique la plus efficace. Enfin nous comparons en fonction de leurs tailles trois politiques usuelles de gestion de listes taboues: FIFO, LRU, RAND.

Contents

10.1 Introduction

A *random walk* is a mathematical formalization of a route taken by a walker through a topology of locations: at each step, the next destination is randomly chosen. Random walks are probabilistic distributed algorithms and this is one of their main advantages. The main drawback of random walks is the large number of steps generally needed to reach one vertex starting from another one, namely the *hitting time*. This is mainly due to the fact that the walker may come back to previously visited vertexes, as a consequence forming useless loops. We study *partially self-avoiding* random walks on *finite* graphs. They are variants of the *simple random walk*, for which at each step the walker chooses its next destination uniformly at random among all its neighbors. We add a bounded memory to the walker in order to reduce the number of loops. This memory is called a *tabu list* and contains a view of previously visited vertexes. We say that a tabu list is of length m, if the list can contain at most m elements. The walker avoids every vertex contained in its tabu list unless it has no choice. One can expect that the tabu list helps to reduce the mean hitting time of the walk. For instance, sending 500 000 messages in a network of 200 nodes with one sink and an average degree of 8, the mean hitting time is 449 hops for a random walk and 310.4 hops for a random walk with a tabu list of size one. More experimental details are presented in the next chapter.

Adding a tabu list in the message in order to share information considerably improve the performance of such routing algorithms. But, as the size of the tabu list is bounded, when the list is full, one element has to be removed before any new insertion. We call *update rule* the algorithm which drives the policy of insertion and removal in the list. For example, in the $FIFO_m$ update rule, the tabu list is of length m, the current position of the walker is always inserted, and when the list is full, the oldest element is firstly removed to make room for the current position. In particular, $FIFO_1$ is a *non-backtracking* random walk, where the walker never backtracks to the last visited vertex except if it is his only neighbor. Our goal is to give some answers to the following question: How do the update policy and the size of the tabu list influence the performance of such routing algorithms?

Contribution. A tabu random walk is characterized by its update rule. We define a large class of update rules and analytically study their associated tabu random walks. We compare all these tabu random walks *w.r.t.* the mean hitting time between every two given vertexes. Mainly, we try to figure out how to handle the memory of the tabu list and what is the best way to do so. The panel of answers proposed here helps to choice an update rule. More precisely, our contribution is threefold:

1. We provide a necessary and sufficient condition on our update rules that ensures the finiteness of the mean hitting time on every graph.
2. We partially answer to the question "What is the best update rule?" by exhibiting a large collection of graphs indexed by a positive integer m, called *m-free* graphs, in which $FIFO_m$ is the optimal (*w.r.t.* the mean hitting time) update rule, provided that the

length of the tabu list is at most m. In particular, the 1-*free* class contains all graphs. Therefore, $FIFO_1$ is the optimal policy on every graph if the tabu list contains at most one element.

3. We compare the performances of three collections of classical update rules, *i.e.*, $FIFO_m$, $RAND_m$, and LRU_m, according to the length m of the tabu list. Our results show that no general answer can be given. For some classes of topologies, the mean hitting time decreases when the size of the memory increases. But, counter-intuitively, there exist cases where having more memory is a penalty: We exhibit topologies where the mean hitting time increases when the length of the tabu list increases.
   A important (perhaps surprising) consequence of our results is that for every update rule $FIFO_m$ with $m \geqslant 2$, there exists a graph and two vertexes x, y such that the mean hitting time from x to y using $FIFO_m$ is strictly greater than that of the simple random walk. By contrast, $FIFO_1$ always yields smaller mean hitting times than the simple random walk.

Related Work. For a general account on *simple random walks*, we refer to the survey [Lov93] and the forthcoming book [AF01b].

Random walks with memories have received much less attention. Most analytic results deal with infinite graphs. For example, there are results on *self-avoiding random walks* for infinite graphs, see the survey [Sla10].

On finite graphs, [Li10] and [4] study random walks that attach memories on the vertexes of the graph. Nevertheless, no theoretical analysis of these solutions are yet available.

In [OW07], the authors study *non-backtracking random walk* on finite and infinite connected graphs with minimum degree two. In particular, they show that for each finite graph, except cycles, $FIFO_1$ is irreducible[1]. Consequently, the mean hitting time of $FIFO_1$ on these graphs is also finite. Our first result is more general than this latter assertion because it deals with all finite connected graphs and a class of update rules that includes $FIFO_1$.

Outline. The description of a random walk with a tabu list is given in Section 10.2. Section 10.3 we give necessary and sufficient condition on our update rules that ensures the finiteness of the mean hitting time on every graph. In Section 10.4 we propose a class of graph in which $FIFO_m$ is the optimal. Then in Section 10.5 we compare different policies according to the length of the tabu list. In Section 10.6 we conclude. All formal definitions and proofs are given in [2, 1].

10.2 Tabu Random Walks and Update Rules

In our framework, the walker evolves on a simple, undirected and connected graph. Besides, we assume that the vertex set is *finite* and contains at least two vertexes.

[1]It is possible to get to any state from any state.

10.2.1 — Tabu Random Walks

A *tabu random walk* on a simple graph is a partially self-avoiding random walk, where the walker is endowed with a finite memory and can jump from a node to another, provided that they are neighbors. At step n, the position of the walker is represented by the random variable X_n and the current tabu list by the random variable T_n. We will denote by T_n^i the i-th element of T_n. The successive ordered pairs $(X_n, T_n)_{n \geqslant 0}$ is a Markov chain, called *tabu chain*. The tabu random walk is the sequence $(X_n)_{n \geqslant 0}$ of the successive positions of the walker.

At each step, the walker avoids to revisit vertexes which are present in the current tabu list, unless he is forced to. More precisely, for every non-negative integer n, the next visited vertex X_{n+1} is uniform on the set formed by the neighbors of the current vertex X_n which are not in the tabu list T_n. If this is not possible because all neighbors of X_n are already in the tabu list T_n, then the next visited vertex X_{n+1} is uniform on the neighborhood of the current vertex X_n. Afterward, the next tabu list T_{n+1} is obtained using an *update rule.*

10.2.2 — Update Rules

The policy to insert or remove an entry in the tabu list is called the *update rule* and denoted by R_m, where m is the maximum number of elements of the tabu list. By extension, m also called the *length of the update rule.* Every update rule works as follows:

(1) First, concatenate the current vertex X_n and the current tabu list T_n, the concatenation being noted $X_n \cdot T_n$.

(2) Then, possibly remove an element of $X_n \cdot T_n$.

Note that according to (2), for some policies, the first element of the concatenation $X_n \cdot T_n$ might be directly discarded, which means that the current vertex X_n is actually not inserted in the tabu list, *i.e.*, $T_n = T_{n+1}$.

An update rule is *trivial* if the current vertex is never inserted in the tabu list when the tabu list contains no element. For example, the unique update rule with zero length is trivial. For every trivial update rule, if the tabu list is initially empty, then it remains empty forever and the walker performs a *simple random walk*: at each step, the next visited vertex is chosen uniformly at random among the neighbors of the current vertex.

10.2.3 — Examples of Update Rules

For every non-negative integer m, we describe three update rules $FIFO_m$, LRU_m and $RAND_m$ of length m by giving for every non-negative integer n, the law of the next tabu list T_{n+1} conditionally on (X_n, T_n):[2]

$FIFO_m$: The current vertex X_n is inserted at the beginning (left) of the current tabu list T_n. If T_n was already full, then its rightmost element is firstly removed.

[2]These rules match the requirements given in Subsection 10.2.2, see [1] for their formal definition.

LRU_m: All occurrences of the current vertex X_n in T_n are removed. If T_n is still full afterward, then its rightmost element is removed. Then, X_n is inserted at the beginning (left) of T_n.

$RAND_m$: If the current vertex X_n has an occurrence in T_n, then $T_{n+1} = T_n$. Otherwise, if T_n is full, then T_{n+1} is formed by removing one of the $m+1$ elements of $X_n \cdot T_n$ uniformly at random. If T_n is not full, then $T_{n+1} = X_n \cdot T_n$.

Remark that $FIFO_0$, LRU_0 and $RAND_0$ denote the unique trivial update rule with length 0. Note also that when m equals 1 or 2, the update rules $FIFO_m$ and LRU_m coincide and are both distinct from $RAND_m$. However, for every integer $m \geqslant 3$, $FIFO_m$, LRU_m and $RAND_m$ are distinct.

In general, a random walk equipped with a tabu list is not a Markovian process. However, when using $FIFO_m$ (for some positive integer m), the next visited vertex only depends on the current one and on the m previous steps: this is called a *Markov chain with internal states*; refer to [Hug95, p. 177].

10.3 Finite Mean Hitting Times

For every update rule R_m of length m, the *hitting time* $H_y(R_m)$ of every vertex y is the random number of steps needed by a walker to reach y. It is defined as the first instant when the tabu random walk $(X_n(R_m))_{n \geqslant 0}$ reaches y: $H_y(R_m) = \inf\{n \geqslant 0 : X_n(R_m) = y\}$. The *mean hitting time* $E_{(x,\varepsilon)} H_y(R_m)$ is the mean hitting time of y when the walker starts at x with an empty tabu list ε. Our goal is to characterize the class of update rules that have finite mean hitting times, for all graphs and all vertexes x and y.

Definition 55 *We define two conditions. For every tabu list t and every position of the walker x, if t is not full and does not contain x, then applying the update rule on x and t results in :*
($\mathbf{C_1}$) *in inserting x in t with a positive probability.*
($\mathbf{C_2}$) *removing rightmost element from t with a positive probability.*

The conjunction of ($\mathbf{C_1}$) and ($\mathbf{C_2}$) is a necessary and sufficient condition that ensures the finiteness of all mean hitting times for every associated tabu random walk on every graph. The update rules $FIFO_m$, LRU_m and $RAND_m$ satisfy both ($\mathbf{C_1}$) and ($\mathbf{C_2}$). On the contrary, an update rule of length 1 that keeps the unique element of the tabu list until the corresponding vertex is visited again does not satisfy ($\mathbf{C_2}$). Using such an update rule may lead to an infinite mean hitting time. Indeed, on the flower graph F_1 given in Figure 20,[3] if the walker starts at vertex 1 with the empty tabu list and does not hit vertex 0 at his first step, then its tabu list is 1 forever. Thus, the walker never comes back to vertex 1 and, consequently, the walker will never reach vertex 0. Hence, the mean hitting time to reach 0 from 1 is infinite.

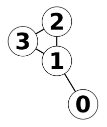


Figure 20 – The flower F_1.

[3] F_ℓ is defined in Section 10.5, for all values of ℓ.

Theorem 19 *Let R_m be an update rule of length m. The mean hitting time $E_{(x,\varepsilon)}H_y(R_m)$ is finite on all graphs, for all vertexes x and y, if and only if R_m is either trivial or satisfies $(\mathbf{C_1})$ and $(\mathbf{C_2})$.*

10.4 Optimal Update Rule for m-Free Graphs

For each positive integer m, we identify a non trivial class of graphs on which $FIFO_m$ gives the smallest mean hitting time, among all update rules of length at most m. Then, we describe a class of update rules that yield tabu random walks with the same law than $FIFO_m$ on this class of graphs.

Definition 56 (m-Free Graphs) *Let m be a positive integer. A graph is m-free if there does not exist any path $x_0, \ldots, x_k$ with length $k \geqslant 1$ that satisfies the four following conditions:*

1. *The vertex x_k has degree at least two.*
2. *For all integers j in $\{0, \ldots, k-1\}$, $x_j \neq x_k$.*
3. *All neighbors of x_k of degree at least two belong to $\{x_0, \ldots, x_{k-1}\}$.*
4. *$k + 2d \leqslant m$, where d is the number of neighbors of x_k of degree one.*

The idea behind Definition 56 is that for each m-free graph with $m \geq 2$, a walker who uses $FIFO_m$ always selects a destination that is not in his current tabu list. Hence, he avoids cycles of size less than or equal to m.

As direct consequences of the definition, *all* graphs are 1-free and that all $(m+1)$-free graphs are also m-free, for every positive integer m. Furthermore:

— A graph is 2-free if and only if it does not contain any triangle with one vertex of degree exactly two, namely, if and only if it does not possess any vertex x such that $V_x = \{y, z\}$ and $\{x, z\} \subseteq V_y$, where V_x and V_y are, respectively, the set of neighbors of x and y.

— Every $(m+1)$-*regular* graph, namely with all vertexes of degree $m+1$, is m-free. Indeed, assume that $x_0, \ldots, x_k$ is a path that satisfies the four conditions stated above. Since $d = 0$, we infer that $k \leqslant m$. Yet, x_k has $m+1 > k$ neighbors of degree at least two while the set $\{x_0, \ldots, x_{k-1}\}$ has k elements. Thus, the condition 3 is not satisfied and we reach a contradiction.

— Every graph with *girth*[4] strictly greater than $m+1$, that is, where every cycle has at least $m+2$ edges, is m-free.

The following theorem states that for m-free graphs, and with no more than m memories, $FIFO_m$ is the optimal update rule.

Theorem 20 *Let m be a positive integer and R_k be an update rule of length $k \leqslant m$. On a m-free graph, for every two vertexes x and y, $E_{(x,\varepsilon)}H_y(FIFO_m) \leqslant E_{(x,\varepsilon)}H_y(R_k)$.*

[4]The length of a shortest cycle contained in a graph

Note that being m-free is not a necessary condition to have, for every two vertexes x and y, $E_{(x,\varepsilon)}H_y(FIFO_m) \leqslant E_{(x,\varepsilon)}H_y(R_k)$ (Theorem 20 only gives a sufficient condition). Indeed, consider the clique with vertex set $\{0, 1, 2\}$. The path $(x_0, x_1, x_2) = (0, 1, 2)$ ensures that the graph is not 2-free. Yet, for every two distinct vertexes x and y and for every update rule R, $E_{(x,\varepsilon)}H_y(R) \geqslant 3/2$, while $E_{(x,\varepsilon)}H_y(FIFO_2) = 3/2$.

From the above theorem, we can deduce that the non-backtracking random walk ($FIFO_1$) is the fastest among all update rules of length less or equal to 1, since every graph is 1-free.

Corollary 21 *For every update rule R of length 0 or 1 and for every two vertexes x and y of every graph, $E_{(x,\varepsilon)}H_y(FIFO_1) \leqslant E_{(x,\varepsilon)}H_y(R)$.*

Proposition 3 completes Corollary 21 and states that $FIFO_1$ is the only update rule of length 1 such that, on all graphs, all hitting times are smaller than those associated to the simple random walk, here represented by $FIFO_0$.

Property 3 *If R is an update rule of length 1 distinct from $FIFO_1$, then there exists a positive integer ℓ such that on the graph F_ℓ,*[5] *$E_{(1,\varepsilon)}H_0(FIFO_0) < E_{(1,\varepsilon)}H_0(R)$.*

Theorem 20 shows that $FIFO_m$ is the best update rule of length at most m for m-free graphs. In Theorem 22 below, we characterize a larger class of update rules which are optimal policies in that specific case; for example, LRU is in this class as stated by Corollary 23 (as a direct application of Theorem 22).

Theorem 22 *Consider a positive integer m and an m-free graph. Every update rule of length k in $\{0, \ldots, m\}$ yields tabu random walks with the same law as those associated to $FIFO_k$ if and only if it satisfies the two following conditions:*

— *If the tabu list is not full and does not contain the current vertex, then it is inserted.*

— *If the tabu list is full and does not contain the current vertex, then the last element is removed and the current vertex is inserted.*

Corollary 23 *Let m be a positive integer. On every m-free graph, for every integer k in $\{0, \ldots, m\}$, the update rules LRU_k and $FIFO_k$ have the same probability law.*

As a conclusion, within the class of m-free graphs and no more than m memories, we identified a class of optimal update rules which contains $FIFO_m$. For other cases, namely, for graphs that are not m-free or using more that m memories, the question is still open.

10.5 Impact of the Length of the Update Rules

We study the effect of the size of the memory of the walker, on the mean hitting times using 3 collections of update rules: $(FIFO_m)_{m\geqslant 0}$, $(LRU_m)_{m\geqslant 0}$, and $(RAND_m)_{m\geqslant 0}$. Our results

[5]Flower graphs F_ℓ are defined in Section 10.5.

shows that there is no general trend, *i.e.*, having more memory does not always increase the performances. To see this, we present comparisons based on four particular collections of graphs:

• Cliques $(K_r)_{r\geqslant 2}$: For every integer $r \geq 2$, the clique K_r is the complete graph with vertex set $\{0, \dots, r-1\}$: each vertex is neighbor of all other vertexes. As an example, the clique K_6 is drawn in Figure 21. Note also that K_r, for $r > 2$ is $(r-2)$-free.

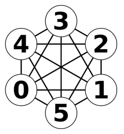


Figure 21 – The clique K_6.

• Lines $(P_r)_{\geqslant 2}$: For every integer $r \geqslant 2$, P_r denote the graph line with vertex set $\{0, \dots, r-1\}$ and edge set $\{\{i, i+1\}, i \in \{0, \dots r-2\}\}$.

• Lollipops $(L_r)_{r\geqslant 3}$: For every integer $r \geqslant 3$, L_r denotes the lollipop graph with vertex set $\{0, \dots, r-1\}$ such that the vertexes $2, \dots, r-1$ form a clique with $r-1$ elements and the vertex 1 has 0 and 2 as neighbors. As an example, the lollipop L_6 is drawn in Figure 22.

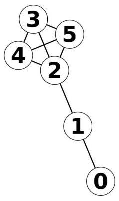


Figure 22 – The lollipop L_6.

• Flowers $(F_\ell)_{\ell\geqslant 1}$: For all positive integers ℓ, we define the graph F_ℓ with vertex set $\{0, \dots, 2\ell+1\}$ as follows. Initially, the vertexes 0 and 1 are isolated and for each integer x in $\{1, \dots, \ell\}$, the vertexes $2x$ and $2x+1$ are neighbor. Then, each vertex is linked to the vertex 1, except the vertex 1 itself. As an example, the flower F_3 is drawn in Figure 23.

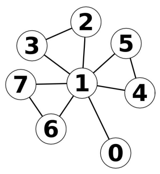


Figure 23 – The flower F_3.

For a given update rule, R_m of length m, we study the mean hitting time $h(R_m) = E_{(1,\varepsilon)} H_0(R_m)$ of the vertex 0 for a walker that starts from vertex 1 with an empty tabu list. The next proposition presents a complete view of the impact of the length on the update rules for $(FIFO_m)_{m\geqslant 0}$, $(LRU_m)_{m\geqslant 0}$ and $(RAND_m)_{m\geqslant 0}$.

Property 4 *On the graph written at k-th row and m-th column,*

1. $h(FIFO_k) > h(FIFO_m)$:

$k \backslash m$	0	1	2	$m \geqslant 3$
0	$\emptyset$	K_3	K_3	K_3
1	$\emptyset$	$\emptyset$	K_3	K_3
2	F_7	F_4	$\emptyset$	K_4
$k \geqslant 3$	P_4	P_4	P_4	$\begin{cases} P_{m+2} & \textit{if } m < k\,, \\ \emptyset & \textit{if } m = k\,, \\ K_{k+2} & \textit{if } m > k\,. \end{cases}$

2. $h(LRU_k) > h(LRU_m)$:

$k \backslash m$	0	1	2	$\geqslant 3$
0	$\emptyset$	K_3	K_3	K_3
1	$\emptyset$	$\emptyset$	K_3	K_3
2	F_7	F_4	$\emptyset$	K_4
$\geqslant 3$	P_4	L_4	L_4	$\begin{cases} K_{k+2} & \text{if } m > k\,, \\ \emptyset & \text{if } m = k\,, \\ L_{m+2} & \text{if } m < k\,. \end{cases}$

3. $h(RAND_k) > h(RAND_m)$:

$k \backslash m$	0	1	2	3	$m \geqslant 4$
0	$\emptyset$	K_3	K_3	K_3	K_3
1	F_5	$\emptyset$	K_3	K_3	K_3
2	F_5	F_6	$\emptyset$	K_4	K_4
3	P_4	P_4	L_5	$\emptyset$	K_5
$k \geqslant 4$	P_4	L_5	L_5	L_5	$\begin{cases} K_{k+2} & \text{if } m > k\,, \\ \emptyset & \text{if } m = k\,, \\ L_{m+2} & \text{if } m < k\,. \end{cases}$

In each table, the symbol $\emptyset$ means that no such graph exists. Actually, symbol $\emptyset$ appears in two disjoint cases : when $k = m$ or when we compare $FIFO_0$ to $FIFO_1$. (In this latter case, for all graphs and all vertexes x and y, $E_{(x,\varepsilon)}H_y(FIFO_0) \geqslant E_{(x,\varepsilon)}H_y(FIFO_1)$, by Corollary 21.)

The above proposition shows that, for the three studied collections of update rules, there is no general trend: increasing the length of the memory does not always lead to a gain of performance and may even be a penalty in some cases.

10.6 Conclusion

We analyzed classes of tabu random walks characterized by their update rules. Our goal was to study the impact of the choice of an update rule on the performance of tabu random walks. We focus on classes of update rules for which we give a necessary and sufficient condition that ensures the finiteness of the mean hitting time of the associated tabu random walk on every graph. Then, we exhibit non-trivial classes of graphs, namely the m-free graphs on which we exhibit the optimal update rules among those of length at most m. Finally, we demonstrate the impact of the tabu list length on the efficiency of the walk. This latter study shows that, except in one case (namely $FIFO_1$), modifying the length of the tabu list does not guarantee a better hitting time in all cases.

Chapter 11

Routing with Local Tabu List

Il est possible de mémoriser de l'information des listes taboues dans les nœuds afin de réduire la charge du réseau. Nous proposons deux algorithmes stockant des listes taboues dans les nœuds (TLN):

— La première solution enregistre les messages déjà envoyés dans une direction. Lorsque le message revient, s'il a déjà été envoyé alors il a fait une boucle, nous augmentons donc un compteur associé à ce lien. Ensuite nous utilisons les compteurs pour favoriser l'envoi des messages par des liens ayant une petite valeur, afin d'éviter les boucles. Cet algorithme est efficace sans la présence de nœuds malicieux. Nous montrons qu'un intrus peut tirer profit de notre algorithme pour attirer le traffic vers lui.

— Par conséquence nous proposons un second algorithme résilient en présence d'intrus et utilisant des primitives cryptographiques. Pour cela nous mettons en place un système de confirmation des messages reçus par le puits. Ceci nous permet de favoriser les routes "correctes". Pour cela nous utilisons plusieurs listes de stockage d'information par nœuds [3].

Nous comparons ces deux approches à des protocoles de la littérature en utilisant le simulateur SINALGO [Gro08].

Contents

A wireless sensor network (WSN) is a network made of numerous interconnected nodes using wireless communications and having batteries. Usually, a WSN is organized around a special node called the sink (or base station), which acts as a gateway between the wireless sensor network and more traditional data processing facilities. The role of a routing protocol is to allow communication between nodes that are not immediate neighbors, using other nodes in the network. In the case of WSNs, a routing protocol should scale up to thousands of nodes and does not consume too much energy. The energy consumed will depend on the internal node operations and the length of the route. We are interested in all-to-one routing algorithms, where all packets are destined to the sink, and all other nodes regularly process new data to route. We also consider only bidirectional communications.

Moreover routing protocols are often designed without security in mind. Adding security is not an easy task in WSN, because asymmetric cryptography require expensive computations, and so energy. Hence, they cannot be suitable for WSNs. Therefore we use lightweight cryptography, such as symmetric encryption, hash functions, in order to protect communications.

Contributions: We propose algorithms that try to maintain a good trade-off between the quality of the service (delivery rate) and energy consumption in presence of attackers or faults. These algorithms are called *r*esilient. We mean by resilient that even if there is a change in the environment, *e.g.*, in the topology, or a problem due to an intruder or a failure, then a resilient algorithm still ensures a certain quality of service [OMVK10, Pon11]. Our approach is based on local tabu list stored in node (TLN).

— Tabu lists store messages that have been already sent through a link. When a message is received, if it has already be send by a link, then we detect a loop. Then we increase a counter associated to this link. Finally we decide to route a message using a probability law which prefers nodes with smaller counter. We try to not send messages in potential loops. Without any malicious node, this algorithm is efficient comparing to a random walk or even to the algorithm presented in the previous chapter, cf Figure 26. However we propose a scenario where a strong intruder can clearly disturb our algorithm.

— As a consequence the second algorithm that we proposed aims to resist to such attacks. Hence it uses cryptographic primitives and an acknowledgement mechanism. The main idea of our second algorithm is to detect valid route to the sink and try to route messages only using such routes. In order to do so, we have several lists per node in order to store: i) a kind of routing table which stores acknowledgments send by the sink, ii) generated data messages sent through a link and iii) good links, i.e. links that reach the sink. Our algorithm uses a probability law which prefers nodes present in the list of good links [3, 3].

In order to evaluate our solutions we perform simulations using the simulator SINALGO [Gro08], considering different scenarios.

Outline: We first describe our routing algorithm that avoid loops then we present an attack in presence of malicious node. In the next section we give another resilient algorithm using cryptographic primitives. Then in the last section we conclude this chapter.

11.1 Tabu List with Counter in the Node (`TLCN`)

The efficiency of the random walks is mainly affected by the existence of cycles in the graph: a message can follow cycles many times before going out and reaching the sink. As we have seen in the previous chapter avoiding cycle of size one already improved such kind of algorithms. To overcome this problem, we propose to use local tabu lists to detect cycles. The protocols are basically random walks for which, at each hop, tabu lists are maintained and the message is preferably sent to some neighbor nodes, according to constraints on the tabu lists. We thus adapt the random walk optimized by Yamashita `RWLD` in [SIY09] and modify its probability to chose the next node to now depend on tabu list information. We call such protocols `TLCN` (Tabu List with Counter in the Node).

11.1.1 — Protocol Description

Tabu lists contain message identifiers. Actually, at each node v, there is a tabu list $TL[u]$ attached to each possible destination $u \in \mathcal{N}_v$. When a message leaves node v through some link (v, u), then the message identifier is added to the corresponding tabu list, $TL[u]$. The following invariant is maintained: at each node v, tabu lists are exclusive (any message identifier appears in at most one list at node v), and if a message identifier appears in a list $TL[u]$, then this means that the last time it visited v, the message left v through the corresponding link (v, u). This enables to detect that a message followed a cycle: when it comes back to some node, its identifier is present in the list of the link through which it previously left the node. Then, we are able to detect two links of the cycle: the link through which the message previously left the node and the link through which it comes back to the node.

At each node v, an *accusation counter*[1] $Counter[u]$ is also associated to each possible destination $u \in \mathcal{N}_v$ (henceforth, to each tabu list). Such a counter is used to remember how many times the node v detects that a message followed a cycle containing this link. Tabu lists and counters are therefore used to *detect cycles*: the higher the counter of a link, the more the link leads to some cycles. Thus, a node preferably sends the messages through the links with small counter values. The selection of the next link then uses the probability law $P^v_{\mathtt{TLCN}}$ which is derived from the one of `RWLD`. $P^v_{\mathtt{TLCN}}(Counter) : \mathcal{N}_v \to [0;1]$ selects a node u among the neighbors of a node v, is weighted by the array of counters $Counter$, and is defined as follows:

$$P^v_{\mathtt{TLCN}}(Counter)(u) = \frac{Counter[u]^{-1} \times \delta_u^{-\frac{1}{2}}}{\displaystyle\sum_{w \in \mathcal{N}_v} Counter[w]^{-1} \times \delta_w^{-\frac{1}{2}}}$$

Where δ_v is the degree of the node v, *i.e.* the number of neighbors of v. Notice that when all

[1]This method was originally used in fault-tolerant algorithms, see [ADGFT08].

counter values are equal, then the probability law is the one of RWLD.

Algorithm 2 TLCN (k,update)

```
Code for every source node $v$
1: Variables:
2:     $id$: integer
3:     $TL[i]$: array of lists defined for all $i \in \mathcal{N}_v$
4:     $Counter[i]$: array of integer defined for all $i \in \mathcal{N}_v$
5: End Variables
6: Initialization
7:     $id \leftarrow 1$
8:     for all $i \in \mathcal{N}_v$ do
9:         $TL[i] \leftarrow \perp$
10:        $Counter[i] \leftarrow 1$
11:    end for
12: End Initialization
13: On request to route $\langle m \rangle$
14:    randomly select $next \in \mathcal{N}_v$ using probability law $P^v_{\texttt{TLCN}}(Counter)$
15:    send $\langle m, (v, id) \rangle$ to $next$
16:    update($(v, id), TL[next], k$)
17:    $id \leftarrow id + 1$
18: End Request
19: Upon receiving $\langle m, tag \rangle$ from $u$
20:    if $\exists z \in \mathcal{N}_v : tag \in TL[z]$ then
21:        $Counter[u] \leftarrow Counter[u] + 1$
22:        $Counter[z] \leftarrow Counter[z] + 1$
23:        remove($tag$, $TL[z]$)
24:    end if
25:    randomly select $next \in \mathcal{N}_v$ using probability law $P^v_{\texttt{TLCN}}(Counter)$
26:    send ⟨m, tag⟩ to $next$
27:    update($tag, TL[next], k$)
28: End Receive

Code for the sink $s$
1: Upon receiving $\langle m, tag \rangle$ from $u$
2:     deliver($m$)
3: End Receive
```

Overview of Algorithm TLCN (Algorithm 2) Each source node is equipped with a tabu list and a counter for each outgoing link. The tabu list of a link contains identifiers of a part of the messages that were sent through the link. The counter of a link represents the number of time (plus one) the protocol detected that the link was involved in a cycle, namely some message moved along a cycle containing this link.

At the beginning (Lines 6-12), all the tabu lists are empty and the counters are set to one.[2]

When a source node v produces a new datum m (Lines 13-18), it first encapsulates m in a message labeled by $tag = (v, id)$ where id is a sequence number that uniquely identifies the message among all others whose source is v. Hence, the label (v, id) uniquely determines the message in the network. The message is then sent to the next node using the probability law $P^v_{\texttt{TLCN}}(Counter)$ and the tabu list of the link is updated with $tag = (v, id)$.

When a source node v receives a message $\langle m, tag \rangle$ from a node u (Lines 19-24), it first updates the counters. If the message was already in some of the tabu lists, *e.g.*, $TL[z]$, this means that it was formerly sent by v through the link (v, z). As it came back *via* the link

[2]We do not initialize the counter to zero because in $P^v_{\texttt{TLCN}}(Counter)$ we consider the inverse of counters.

(u,v), v detects $\langle m, tag\rangle$ moved along a cycle v, z,..., u, v. The counters of u and z are then incremented. The tabu lists are also updated: tag is removed from the tabu list of z and added to that of the node where the message will be sent next (to maintain the exclusion property of the lists). This next destination is randomly chosen using the probability law $P^{v}_{\texttt{TLCN}}$ which gives preferences to links with small counters.

Influence of the update policy The update policy of the tabu lists slightly influences the execution of the algorithms. For tabu lists of bounded size sz, FIFO update enables to store the sz last messages that were sent through the link, whereas with random update, some messages may be older. Both cases are heavily influenced by 1) the *load* of the node, *i.e.*, the number of messages treated by the node and 2) the size of the cycles to be detected.

11.1.2 — Attack

Experimentally without intruder this protocol improves considerably the `RWLD`. Unfortunately if some nodes are malicious and behave like a *black-hole*: is a node who is receiving all messages but does not emit any message. For our algorithm, such malicious nodes attract the traffic like a sink, because all received messages will not be considered in any loop so the counter to reach these nodes are anymore increased, as a consequence associated links will be preferred. Hence a black hole will attract part of the traffic and decrease the delivery rate of information to the sink as it is shown in Figure 27 where we consider a graph with 10% of black holes and only 3.2 % of sent messages are delivered.

11.2 A Secure Algorithm

In order to improve `TLCN`, we add cryptographic mechanism and a reputation mechanism in order to prevent the previous attack. We call this algorithm TLCNS only because it was proposed after `TLCN` and it is secure.

11.2.1 — Idea of the Protocol

Security: First, each node gets an unique public identity, and a pre-shared key with the sink. This enables symmetric encryption between the sink and each node, a step towards confidentiality. Then, each message is identified by a nonce. It is concatenated with the data and this is encrypted to protect data from eavesdropping. A hash of the identifier added to the message, moreover the identity of the emitter is part of the message, so that the sink knows which key to use. This whole construction guarantees the unforgeability and confidentiality of the packet. This means that an attacker cannot create or alter a message, so that it will be considered valid by the sink.

Self-adaptivity: Our algorithm implements a kind of learning mechanism. To achieve this, we added memory of previous decisions and their outcomes, which will in turn influence the routing decisions. The reputation mechanism determines if a message was actually delivered or not. Only the sink can reliably determine if it received a message. So, each time it receives one, it produces a control packet, called an acknowledgment. It contains the message identifier. The acknowledgment will then be routed to the original sender, who will then learn that its message was correctly delivered. The original sender keeps track of the emitted message in a finite memory and selects with high probability "correct" nodes when he is routing a message.

11.2.2 — TLCNS Algorithm

We use the following notations:

— $\textsc{Encrypt}(plaintext, k_{ab})$ computes the symmetric encryption of the *plaintext* with the key k_{ab} and $\textsc{Decrypt}(ciphertext, k_{ab})$ decrypts the *ciphertext*.

— $\textsc{Hash}(key)$ denotes the hash function $Hash$.

— $\textsc{New\_Nonce}()$ generates a new nonce.

— $Count(elem, L_{Routing})$ is the function which returns the number of occurrences of *elem* in $L_{Routing}$.

— Let the random variable X represent the identity of the chosen next node for a message. X takes its value among $\mathcal{N}g_v$[3]. Its law is denoted $\mathcal{L}^v_{TLCNS}(L_{Routing})$ for a node v, and defined by:

$$Pr(X_{\mathbf{M}} = n) = \frac{Count(n, L_{Routing}) + \delta_v^{-1}}{\sum_{x \in Candidates} Count(x, L_{Routing}) + \delta_v^{-1}}$$

Where δ_v denotes the degree of node v. When a node needs to route a message, it will follow this law, which means that it will select more often a route with "correct" nodes present in $L_{Routing}$ than other nodes, but it is still possible to select any neighbor.

In our algorithm, packets sent by nodes to the sink are made of five parts $\langle \mathbf{M}, C, H, o, f \rangle$, where:

— $\mathbf{M}$ is a flag (a constant) which indicates that this packet is a message.

— C is the encryption of the tuple $\langle Data, N_o \rangle$, where N_o is a nonce generated by the originator of the message, and $Data$ is the information the node wants to route to the sink.

— H is $Hash(N_o)$.

— o is the identity of the node that generated the message (the origin), so that the sink knows which key will open C.

— f is the identity of the last node to have forwarded that message.

On the other side packets emitted by the sink are called *Acknowledgments* and contain three parts $\langle \mathbf{A}, N_o, o \rangle$:

— $\mathbf{A}$ is a flag which indicates that this packet is an acknowledgment.

[3]Neighbors of node v

— N_o is the nonce used in the acknowledged message, which has been generated by o.

— o is the identity of the node that generated the initial message, which is now the destination of the acknowledgment.

In order to store information each nodes maintains the 3 following lists:

— $L_{Routing}$ is a FIFO[4] list of node identities with a maximal size, where insertion of a new element in a full list replaces the earliest inserted element. We add elements using the function $\textsc{Enqueue}(L, elem)$.

— M_{Queue} and $M_{AckRouting}$ are lists of tuples made out of a message identifier (denoted m) and a node identifier (v), with a maximal number of entries, and an uniqueness constraint on the first element of the tuple. It means that:

  – When a $\langle m, v\rangle$ pair is inserted in the list M, with function $\textsc{AddToLRM}(M, \langle m, v\rangle)$, if there is already an entry containing m, delete it. Otherwise , if the maximum number of entries is already reached, remove the least recently modified (LRM) tuple. Finally, insert the new pair $\langle m, v\rangle$ in the list.

  – When a message identifier m should be deleted from M, we remove the tuple containing m if it exists, by using function $\textsc{RemoveFromLRM}(M, m)$.

  – $\textsc{IsInLRM}(M, m)$ is the test that returns true if there is an entry containing m in M.

  – $\textsc{GetFromLRM}(M, m)$ is the function which returns the node identifier for that entry.

  – We denote $\{\}$ the empty list

In Algorithm 3 we give a pseudo-algorithm for a non-sink node and in Algorithm 4 we describe the behavior of the sink.

First, when a node v has data to send, it generates a message (line 7-14). This message is randomly routed to a neighbor of v, which we name n, and we store in the finite list M_{Queue} the tuple made of the message identifier and the neighbor n which was the first hop. When the sink receives that message (lines 48-54), if it is valid, it replies with an acknowledgment (lines 50-54), containing the message identifier and the initial sender v. If a message gets routed back to its initial sender, the corresponding entry is removed and the routing process is restarted (lines 28-44).

Each node maintains a finite list $M_{AckRouting}$ of the messages that it received, and their origin. Thanks to these lists, acknowledgments can follow the route of their corresponding messages, and revert to RW if there is a problem (lines 28-44).

Upon reception of that acknowledgment, the sender v will recover the identity of n from M_{Queue}, and insert it into $L_{Routing}$ to register its trust in n (lines 15-27). Each time v has to route a message, it takes into account the entries in $L_{Routing}$ to make a decision (lines 11).

[4]First In First Out

11.3 Implementations

11.3.1 — Delivery Rate

In order to evaluate the performance of the routing algorithms we observe the delivery rate, we use the following metrics.

Mean Length Route: We look at the mean number of edges traversed, for each message that reached its destination. If a message does loop, we count the length of the loop in the route.

Average Delivery Rate: The delivery rate of a routing algorithm is the proportion of data-bearing packets received by the sink. The average delivery rate is a measure of how the algorithm manages to bring packets to the sink.

Delivery Rate Classes In order to compare different algorithms on a lot of graphs, we need a way to average the measures over different runs of the algorithm, so that they still make sense. For this, we expanded the idea of delivery rate classes from [EOKMV11].

From a bundle of simulation runs in the same setting (same degree, number of nodes, number of intruders, and simulation parameters), we compute the overall delivery rate of each node, and the global delivery rate over all simulations. At the end, we categorize them in 12 classes as follows: 0% delivery rate, each slice of 10%, and exactly 100%. We look at the relative sizes of the classes and indicate the average delivery rate of the simulations.

Fairness: We compute delivery rates of each individual node and the *fairness* is the standard deviation of this data for one simulation run. It characterizes the fact that the algorithm tries to have a high delivery rate for most of the nodes. For instance if we consider a deterministic algorithm and a black-hole on the path of a node then his delivery rate is 0%, as a consequence it is decreasing his fairness.

11.3.2 — Results

To evaluate the performance and resilience of our algorithm, we compare with the following routing algorithms.

Other Algorithms: We select either well-known algorithms (RW, GFG, GBR), and algorithms designed to be resilient (PRGBR, PRDGBR). We now describe these algorithms:

Uniform Random Walk (RW): Each packet to route is sent to a random uniformly selected neighbor.

Gradient-based routing and variants Gradient-based routing (GBR) algorithm described in [SS01] works in two phases. First, the sink floods the network with an INTEREST packet containing a counter (indicating the distance to the sink, also called the height of the node), which is then broadcasted by all nodes after incrementing the counter. Nodes keep the minimal value of the counter they have seen, which denotes its height, and also keeps track of the height of each of its neighbors. If a new INTEREST packet with a smaller height than the node has seen until now, the node updates its height, and notifies its neighbors.

On each routing request, nodes choose a next hop according to the specific variant[5].

— **Deterministic GBR (GBR):** Nodes send their messages to a fixed lower-height neighbor, this is the original GBR protocol proposed in [SS01].

— **Probabilistic-random GBR (PRGBR):** With probability p, the message is routed to a random same-height node if one exists, and otherwise, to a random lower-height node. We used $p = 0.4$, as suggested in [EOKMV11].

— **Duplicating probabilistic-random GBR (PRDGBR):** It is PRGBR where at each hope messages are duplicated [EOKMV11] and messages already seen by a node are discarded.

By design, GBR variant always routes packets to lower-height nodes. This means that neighbors of the sink will always send to the sink, and (n)-height nodes will send to $(n-1)$-height nodes. Thus, by induction, it will always route packets in the minimal number of hops. The two other variants do not verify that property.

Greedy-Face-Greedy (GFG): It is a combination of Greedy routing algorithm and Face-based routing algorithm (FBR). They both require that each node knows its position (GPS coordinates), its neighbors positions, and the sink position. Also, FBR requires a planar graph (where no edges intersect) which can be done using the algorithm proposed in [BMSU01a].

— Greedy (or geographical) routing consists of routing each message to the neighbor with the closest position to the sink. However, this algorithm can get blocked in graph components where a node is closer to the sink than all its neighbors, creating a routing loop. When this happens, FBR is executed until it reaches a position closer to the sink than the point where the message got stuck. Then, Greedy runs, and so on.

— Face-based routing [BMSU01a] works on a planar graph as follow. A line between the origin of the routing and the sink is computed, and the face adjacent to the emitter is selected. This face is followed until to come back to the initial node. The line to the sink intersect some edges. Then we select the closet node to the sink of these edges as a new starting point.

Settings and Scenario: We use the following setting for testing these algorithms:

— For PRGBR p is set at 0.4.

— For TLCNS, we set at 10 the size of $L_{Routing}$, 5 the size of M_{Queue} and 3 the size of

[5]We used the variants and notations from [EOKMV11].

$M_{AckRouting}$, according to experimental results obtained over several graphs.

We consider networks containing 200 UDG[6] nodes, here we chose a radius of 20, in a square 177*177. The sink is at the center of the square, the network is connected and the mean degree per node is 8. We consider a simulation of 500 000 messages. Among the 200 nodes, 180 are honest, 10 are black-holes and 10 are wormholes with the sink. Here a wormhole with the sink is a node of the network which has an extra direct link with the sink, as it is illustrated in Figure 24.

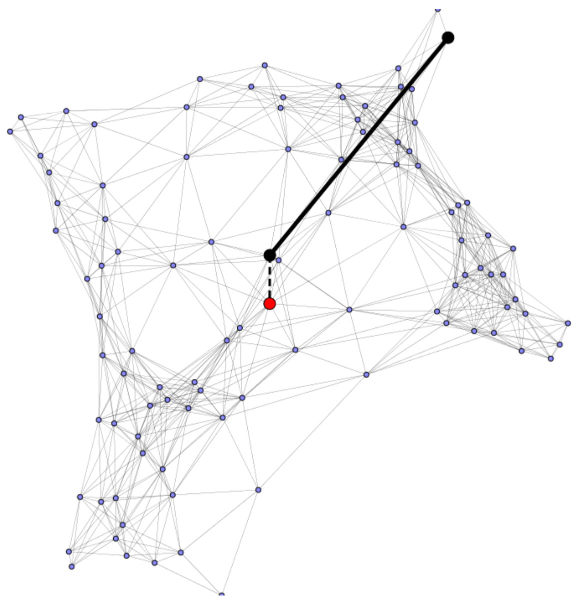


Figure 24 – An example wormhole (sink in red)

We consider the scenario where the 10 black-holes and 10 wormhole are active. Then once one third of messages are emitted wormholes become black-holes.

Performance: We randomly generate a first network described in Figure 25. We first consider that all nodes are honest. Of course the delivery rate is 100% for all our algorithms and the fairness is zero. In Figure 26, we class algorithms according to the mean length of all routes. We see that GBR is the best, then all variants follow. After GFG is close to TLCNS. Then come TLCN with a tabu list of size 8 and TLM with a tabu list of one element. Finally random walk has a high mean number of hops.

We now consider that10 % of the nodes are black holes (pink and black nodes in Figure 25). In Figure 27 we class algorithm according to the mean delivery rate, and we indicate the fairness and the mean number of hop. We see that TLCNS has the best delivery rate and fairness but the worst mean number of hop because he has to follow longer path in order to avoid the black holes. The worst algorithms are TLCN8, RW and TLM1 which fall very often in the black holes. It is why the mean number of hop is so small and the fairness so good. due

[6]UDG (Unit Disc Graph) is a usual communication model, where two nodes are connected if and only if their distance is at most at a fixed positive distance.

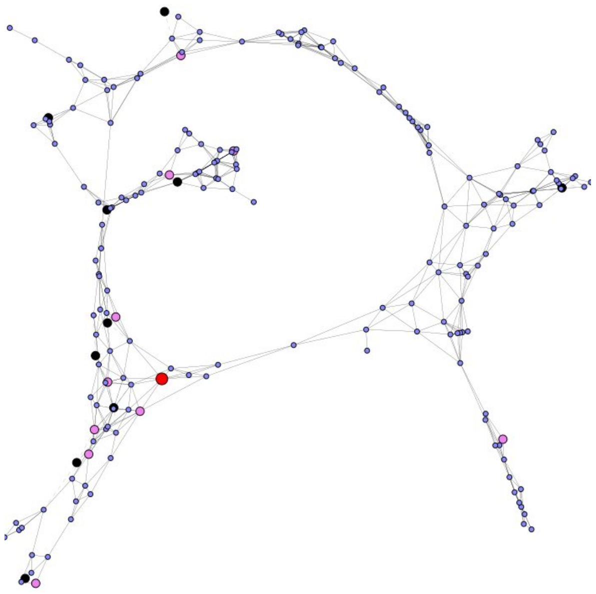


Figure 25 – First network where the sink is red, wormholes are pink and black holes are black.

Algorithms	Mean length route
GBR	5.4
PGBR	5.4
PRGBR	8.4
PRDGBR	8.9
GFG	14.2
TLCNS	14.3
TLCN8	93.7
TLM1	310.4
RW	449

Figure 26 – Mean number of hops for all the algorithms without intruder.

to the position of the black holes 72.7 % of traffic falls in the black holes. All variants of GBR are in the same range around 50 % of delivery rate, except PRDGBR due to the duplication which avoids in a better way the black holes.

In Figure 28, we present ours results for the same graph with our scenario with 5% of wormholes that become black-holes after a third of messages sent. We see that the second phase does not change the order, except for GFG that becomes the second due to the particular place of intruders on this graph. We also see that this change of behavior affects drastically all algorithms, except TLCNS.

In order to look more precisely this phenomenon we represent the delivery rate of each algorithm in presence of black-hole and wormhole, and after to see how the algorithms behave once the wormholes become black-hole. For this we sort by emission date all the messages, and group them in blocks of 100 messages. We average block's messages emission date to have

Algorithms	Total delivery rate	Fairness	Mean number of hop
TLCNS	90.1 %	0.043	12.3
PRDGBR	57.7 %	0.376	8.3
GBR	49.5 %	0.500	4.7
PGBR	48.9 %	0.469	4.8
PRGBR	38,4 %	0.354	6.9
GFG	28.3 %	0.451	6.7
TLM1	4.3 %	0.082	10.1
RW	4 %	0.084	11.4
TLCN8	3.2 %	0.104	2.8

Figure 27 – Total delivery rate, fairness and mean number of hop for all the algorithms with 10% of black holes.

Algorithms	Total Delivery Rate	Fairness	Mean Number of Hop
TLCNS	89.8 %	0.206	9.490
PRDGBR	44.1 %	0.272	5.097
GFG	38.2 %	0.410	6.361
RGBR	36 %	0.288	2.442
GBR	34.3 %	0.312	2.417
PRGBR	33.4 %	0.239	3.720
TLM1	16.8 %	0.140	10.911
RW	16.6 %	0.143	13.856
TLCN8	16.6 %	0.162	4.173

Figure 28 – Total delivery rate, fairness and mean number of hop for all the algorithms with 5% of black holes and 5% of worm holes.

the x coordinate. We calculate the proportion of the block's messages which were delivered to the sink to get the y coordinate. This leads us to several points for each algorithm then we take the "mean curve" using Bezier method, the result is given in Figure 29. This explain the small variations that we can observe at the end of the simulation which are not significant. In order to help the reader to distinguish these curve we add some special symbols.

During the first phase TLCNS is the best, then comes PRDGBR with more than 80%. TLCNS prefers to take routes that are "good", then is avoiding black-hole for reaching the sink. PRDGBR tries to choose shortest routes but in a non deterministic manner and duplicating the message at each step which allows it to avoid the black-holes but it clearly augment the load of the network as a consequence it is very energy consuming. Algorithms GFG and GBR are deterministic and according to the position of the black-hole they have more or less influence on the delivery rate. Finally other random algorithms, with optimization or not to avoid loops, have more chance to meet one of the black-hole than the sink, as a consequence have smaller delivery rates.

In the second phase, we double the number of black-holes by transforming all wormholes

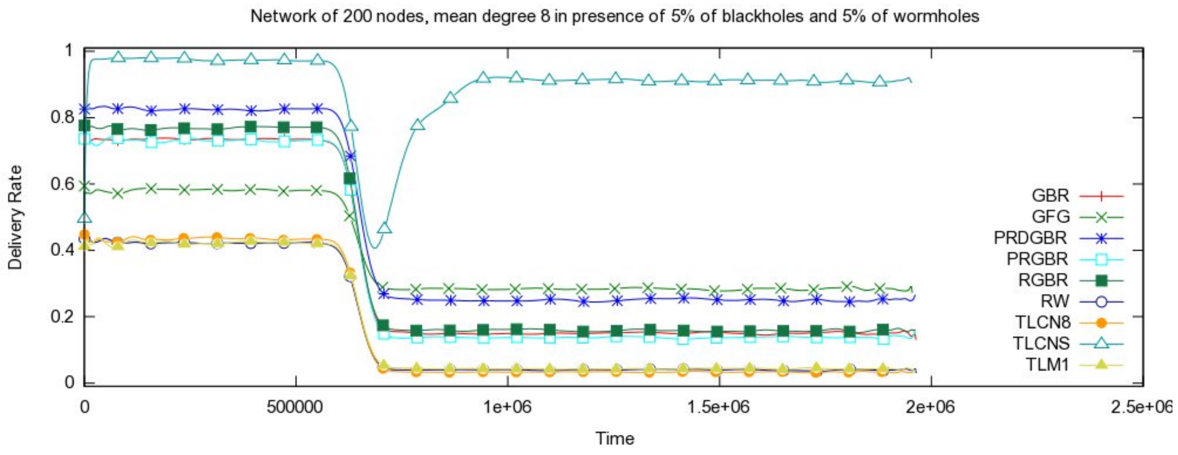


Figure 29 – Delivery rate, we add symbols on the curves in order to increase its readability.

in black-holes. It first affects drastically the delivery rates of all algorithms we notice that the order is respected. We also notice that TLCNS re-converges toward a high delivery rate. It is due to the fact that TLCNS adapts his behavior in function of the acknowledgment of correct routes that he received. The speed of the re-convergence depend of the size of the list $L_{Routing}$, delivery rate is also influenced by the size of the two other lists.

We can also look at the total delivery rate over all the simulation which is represented in Figure 27. We see that TLCNS has the best total delivery rate in presence of black-holes and wormholes, and all other are less than 50%.

We now run the same scenario over 20 similar networks: average degree 8, 200 nodes, 5 % black-holes, 5% wormholes. We compute the average of the delivery rate's average, which is represented by a pink dot in Figure 30. We also represent the percentage of node having a certain delivery rate by slices of 10 % (it i a kind of distribution of delivery rate over nodes). We can see that TLCNS has only nodes over 70 % of delivery rate, whereas deterministic algorithms like GFG or GBR have nodes with a null delivery rate, other with 100 % and some between. Other probabilistic algorithms like Random walk or TLM1 have very low delivery rate node because they have more probability to hit one black-hole than the sink, specially in the second phase.

11.4 Conclusion

We first give an algorithm that stores in local tabu lists information that allows to avoid cycles. But we clearly show that simple black-holes can attract the traffic like a sink. Then we propose another solution concerning security in presence of malicious nodes. We show using simulation that our second solution is more resilient than existing solution due to the learning mechanism we have developed. Finding a good trade-off between efficiency, energy consumption and security is still a real challenge. A precise model for energy consumption is

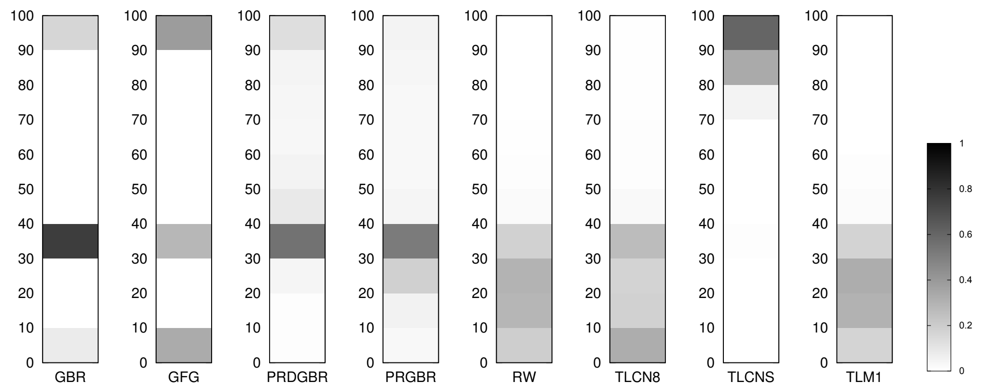


Figure 30 – Delivery rates classes for all the algorithms.

needed in order to evaluate the energy-consumption quality of our solution which might be worst than existing algorithms.

Algorithm 3 TLCNS routing for a non-sink node v

```
1: Variables:
2:     $M_{Queue}$ : fixed-size LRM list of tuples $\langle nonce, node\ identity \rangle$, initially empty
3:     $M_{AckRouting}$ : fixed-size LRM list of tuples $\langle digest, node\ identity \rangle$, initially empty
4:     $L_{Routing}$ : fixed-size FIFO list of nodes identities,, initially empty
5:     $k_{vs}$ : the symmetric key of this node $v$, shared with the sink $s$
6: End Variables

7: On data generation of (Data)
8:     $N_v \leftarrow$ NEW_NONCE()
9:     $H \leftarrow$ HASH($N_v$)
10:    $C \leftarrow$ ENCRYPT($\langle Data, N_v \rangle, k_{vs}$)
11:    $next \leftarrow$ DRAWACCORDINGTO($\mathcal{L}^v_{\mathbf{M}}(\mathcal{N}g_v, L_{Routing})$)
12:    ADDTOLRM($M_{Queue}, \langle N_v, next \rangle$)          ▷ Store this decision for feedback
13:    SEND($next, \langle \mathbf{M}, C, H, v, v \rangle$)
14: End

15: On reception of a message ($\langle \mathbf{M}, C, H, o, f \rangle$)
16:    $next \leftarrow$ DRAWACCORDINGTO($\mathcal{L}^v_{\mathbf{M}}(\mathcal{N}g_v, L_{Routing})$)
17:    if $v = o$ then                                          ▷ The message looped
18:        $\langle Data, N_o \rangle \leftarrow$ DECRYPT($C, k_{vs}$)
19:        if HASH($N_o$) $= H$ then                            ▷ If legitimate
20:            ADDTOLRM($M_{Queue}, \langle N_o, next \rangle$)
21:            SEND($next, \langle \mathbf{M}, C, H, o, v \rangle$)
22:        end if
23:    else                                                     ▷ The message is from elsewhere
24:        ADDTOLRM($M_{AckRouting}, \langle H, f \rangle$)      ▷ Reverse chaining
25:        SEND($next, \langle \mathbf{M}, C, H, o, v \rangle$)
26:    end if
27: End

28: On reception of an ack ($\langle \mathbf{A}, N_o, o \rangle$)
29:    if $v = o$ then                                          ▷ Ack was intended for $v$
30:        if $N_o \in M_{Queue}$ then                          ▷ $v$ remembers that Ack
31:            $first\_hop \leftarrow$ GETVALUEFROMLRM($M_{Queue}, N_o$)
32:            ENQUEUE($L_{Routing}, first\_hop$)                ▷ Use the feedback
33:            REMOVEFROMLRM($M_{Queue}, N_o$)
34:        end if
35:    else                                                     ▷ Ack was for another node, route it
36:        $H \leftarrow$ HASH($N_o$)
37:        if $H \in M_{AckRouting}$ then                       ▷ Reverse chaining
38:            $next \leftarrow$ GETVALUEFROMLRM($M_{AckRouting}, H$)
39:        else                                                 ▷ Random walk
40:            $next \leftarrow$ DRAWACCORDINGTO($\mathcal{L}^v_{\mathbf{A}}(\mathcal{N}g_v)$)
41:        end if
42:        SEND($next, \langle \mathbf{A}, N_o, o \rangle$)
43:    end if
44: End
```

Algorithm 4 TLCNS routing for a sink s

```
45: **Variables:**
46:     $keys[n]$ : the array of symmetric keys defined for all nodes in the network.
47: **End Variables**

48: **On reception of a message** $(\langle \mathbf{M}, C, H, o, f \rangle)$
49:     $\langle Data, N_o \rangle \leftarrow \textsc{Decrypt}(C, keys[o])$
50:     **if** $(\textsc{Hash}(N_o) = H)$ **then**                         ▷ Check validity
51:         Use the Data
52:         $\textsc{Send}(f, \langle \mathbf{A}, N_o, o \rangle)$                  ▷ And send the ack
53:     **end if**
54: **End**

55: **On reception of an ack** $(\langle \mathbf{A}, N_o, o \rangle)$
56:     $next \leftarrow \textsc{DrawAccordingTo}(\mathcal{L}^s_{\mathbf{A}}(\mathcal{N}g_s))$   ▷ Just forward it
57:     $\textsc{Send}(next, \langle \mathbf{A}, N_o, o \rangle)$
58: **End**
```

Chapter 12

Conclusion

Encryption or more generally cryptographic primitives have a cost in term of time and energy. Finding the good trade-off between security, efficiency and energy will be one of the main challenge in the next years. Before presenting some perspectives, we first summarize our works.

In Part I, we have proposed three Hoare logics for proving the security of generic public encryption schemes, generic symmetric encryption modes, and generic MACs. In the last chapter we decided to present a version without the concrete security parameters in order to keep rules and invariants simple but also to explain clearly our model with two parallel executions. Of course it is possible to add security parameters to all the rules as we did in Chapter 2 and 3.

We did not provide one logic with all rules presented in part I, because each logic has his own goal and therefore specific invariants to achieve. It is possible to define a general configuration with all tables, variables needed, adapted preservation and propagation rules,but we do not know any primitive that perform at the same time the goals of three studied primitives.

In Part II, we first described a flaw in Benaloh's homomorphic encryption scheme and proposed a corrected version in Chapter 5. The corrected version of Benaloh is still a manual proof. A tool which takes as input a description of these encryption and decryption primitives and provide for instance a proof in Coq [BC04, Tea04] would be very useful.

Then in Chapter 6, we model privacy properties of electronic voting protocols in applied π-calculus. First our taxonomy allows us to assess the level of privacy provided by a voting protocol. We illustrated on several case studies, including a case with multiple votes per voter (a variant of FOO [FOO92]) and forced abstention attacks. In particular we were able to show that the different dimensions of our definitions correspond to different properties of real-world protocols, and that in many cases the verification can be done automatically using existing tools. a natural continuation of this work would be to to translate our definitions to a computational setting. As we are in a symbolic model, we do not consider exact analysis.

Hence the adversary may in reality still have a certain probability of detecting that the coerced voter tried to resist coercion. A computational translation of our definitions should be able take this into account.

In Chapter 7, we presented an intuitive definition of Privacy, Receipt-Freeness and Coercion-Resistance for voting protocols using weighted votes. We considered situations where only one voter is under attack, and others where multiple voters are attacked. We were able to link these notions to existing ones and to prove that the multi-voter case is equivalent to the single-voter case if the protocol is correct and respects a modularity condition. Finally, we illustrated our work by analyzing two existing protocols. As future work, we would like to investigate the conditions necessary for the equivalence of the single and multi-voter cases more precisely. We hope to remove some of the hypotheses or to give further examples underlining the necessity of these assumptions. Another direction is also to translate these symbolic definitions to the computational setting.

In Part III, we studied WSNs. In Chapter 8, we modeled (k)-neighborhood property and proposed a trace-based verification technique that takes into account time and movement of nodes. We also described the Sharek protocol, which securely discover (k)-neighbors based on the knowledge of the (1)-neighborhood, in presence of one intruder. This formal modeling of the (k)-neighborhood is the first step toward an automatic tool, but due to the large number of possible traces we believe that dedicated techniques should be found in order to obtain a practical tool.

In Chapter 9, we verified routing protocols in presence of independent intruders. All the intruders have their own knowledge, as a consequence corresponding constraint systems does not satisfy the classical monotonicity assumption. We proposed a decision procedure for solving such constraint systems. For this purpose we introduced partially well-formed constraint systems and proposed simplification rules that can be used to transform it into quasi-solved form.

In Chapter 10, we improved random walks by adding to each message a tabu list containing already visited nodes. We studied the impact of the choice of an update rule on the performance of such algorithms. We provided a class of update rules that ensures the finiteness of the mean hitting time on every graph. Then, we exhibited non-trivial classes of graphs, namely the m-free graphs on which we exhibited the optimal update rules among those of length at most m. Our results could be extended to a larger class of update rules for which more removals are allowed; for example, the list could be reset regularly. A preliminary study shows that this extension should be carefully done: we believe that the necessary and sufficient condition of Theorem 19 could be adapted and that the result on m-free graphs still holds. We would also like to compare the relative performance of different update rules of same length. As a first step, we know that given a positive integer m, $FIFO_m$ is faster than LRU_m on line graphs, and LRU_m is faster than $RAND_m$ on lollipop graphs (see [1]).

Finally in Chapter 11, we give another class of probabilistic routing algorithms which store local information in each node. This allows us in a first version to send with a high probability messages in route that are not a loop. Then in a secure version using a probabilistic adaptative reputation mechanism and some acknowledgments, we forward messages preferably in "correct" routes. We also show that such algorithms are resilient to several types of attacks and compare to other existing algorithms.

Perspectives:

I would like to continue to work in automatic formal verification of security protocols according to the following three axis:

- *Formal analysis of non functional properties of WSNs*
- *Verification of e-auction protocols*
- *Security proofs of cryptographic primitives based on pairings*

Formal analysis of non functional properties of WSNs

My first aim is to define non functional properties in WSNs like energy consumption, connectivity. Then develop verification technique for energy consumption which are able also to consider security issue, because adding security has a non negligible energy, or computation resource cost. Finally I would like to study a intruder model than can take into account such constraints. An intruder model which assumes that intruders have also a limited battery as normal nodes seems to be more realistic.

Verification of e-auction protocols

Many e-actions protocols have been proposed and are now used over the Internet. They have to guarantee for instance anonymity of users, confidentiality of transactions, or fairness. Moreover often e-auction protocols use cryptographic primitives that use specific algebraic properties. These protocols can be analyzed according to this point of view but also according to the new properties that they should achieved. These properties are closed but also different of the of e-voting protocols.

Security proofs of primitives cryptographic primitives based on pairing

This axis is the most challenging.

In groups equipped with a bilinear mapping such as the Weil pairing or Tate pairing, generalizations of the computational Diffie-Hellman problem are believed to be a difficult problem while the simpler decisional Diffie-Hellman problem can be easily solved using the pairing function. Several cryptographic primitives have been designed based on pairing like identity-based encryptions, attribute-based encryptions, or signcryption. These primitives manipulate complex algebraic structures. Some errors have been found in some proofs of such schemes. S. Kremer et al [KM10] proposed a formal approach for verifying pairing. This result prove that if using their symbolic model the indistinguishability property holds on pairing the using a soundness theorem it also holds in computational model. As we have done in Part I, we would like to first identify the main algebraic properties used in pairing-based primitives, then propose a formal method directly in the computational model for verifying these primitives.

Bibliography

[ABB$^+$05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, Michael R., J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. of CAV'2005*, LNCS 3576, pages 281–285. Springer, 2005.

[ACD10] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. Modeling and verifying ad hoc routing protocols. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 59–74, Edinburgh, Scotland, UK, July 2010. IEEE Computer Society Press.

[ADGFT08] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing omega in systems with weak reliability and synchrony assumptions. *Distributed Computing*, 21(4):285–314, 2008.

[AF01a] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '01, pages 104–115, New York, 2001. ACM.

[AF01b] D.J. Aldous and J.A. Fill. Reversible Markov chains and random walks on graphs. Book in preparation, 2001.

[And01] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.

[Arn12] Mathilde Arnaud. Formal verification of secured routing protocols. *PHD, Ecole normale supérieure de Cachan-ENS Cachan*, 2012.

[AVI] AVISPA Project. AVISPA protocol library. Available at `http://www.avispa-project.org/`.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[BBU08] Michael Backes, Matthias Berg, and Dominique Unruh. A formal language for cryptographic pseudocode. In *LPAR 2008*, pages 353–376. Springer, November 2008.

[BC94] S. Brands and D. Chaum. Distance-bounding protocols. In *Advances in Cryptology (EUROCRYPT'93)*, pages 344–359. Springer, 1994.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. CoqâĂŹArt: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS series. Springer Verlag, 2004.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO '96, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1996.

[BCM11] David Basin, Cas Cremers, and Catherine Meadows. *Model Checking Security Protocols*, chapter 24. Springer, 2011. To appear.

[BCT04] Gilles Barthe, Jan Cederquist, and Sabrina Tarento. A Machine-Checked Formalization of the Generic Model and the Random Oracle Model. In D. Basin and M. Rusinowitch, editors, *Proceedings of IJCAR'04*, volume 3097 of *LNCS*, pages 385–399, 2004.

[BDD$^+$06] Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, and Mathieu Turuani. Compositional analysis of contract-signing protocols. *Theor. Comput. Sci.*, 367(1-2):33–56, 2006.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. *Annual IEEE Symposium on Foundations of Computer Science*, 0:394, 1997.

[BDKL10] Gilles Barthe, Marion Daubignard, Bruce Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 375–386. ACM, 2010.

[Ben87] Josh Daniel Cohen Benaloh. *Verifiable Secret-Ballot Elections.* PhD thesis, Yale University, New Haven, CT, USA, 1987.

[Ben94] Josh Benaloh. Dense Probabilistic Encryption. In *In Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, 1994.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, pages 71–90, 2011.

[BGLB11a] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella Béguelin. Beyond provable security verifiable ind-cca security of oaep. In *CT-RSA*, Lecture Notes in Computer Science, pages 180–196. Springer, 2011.

[BGLB11b] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella Béguelin. Beyond provable security verifiable ind-cca security of oaep. In *CT-RSA*, pages 180–196, 2011.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 90–101, New York, NY, USA, 2009. ACM.

[BHK$^+$99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. Umac: Fast and secure message authentication. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 216–233, 1999.

[BHM08] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. *Computer Security Foundations Symposium, IEEE*, 0:195–209, 2008.

[BJST08] Bruno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *Proceedings of ASIACCS'08*, pages 87–99. ACM, 2008.

[BKR94] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, pages 341–358, 1994.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, Cape Breton (Canada), 2001. IEEE Comp. Soc. Press.

[Bla08] Bruno Blanchet. *Vérification automatique de protocoles cryptographiques : modèle formel et modèle calculatoire. Automatic verification of security protocols: formal model and computational model.* Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, November 2008. En français avec publications en anglais en annexe. In French with publications in English in appendix.

[Bla11] Bruno Blanchet. A second look at shoup's lemma. In *Workshop on Formal and Computational Cryptography (FCC 2011)*, Paris, France, June 2011.

[BLP03] L. Bozga, Y. Lakhnech, and M. Perin. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In *Computer Aided Verification*, 2003.

[BMQR07] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In *E-Voting and Identity*, volume 4896, pages 111–124. Springer Berlin / Heidelberg, 2007.

[BMSU01a] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[BMSU01b] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7:609–616, 2001. 10.1023/A:1012319418150.

[BMU10] Michael Backes, Matteo Maffei, and Dominique Unruh. Computationally sound verification of source code. In *ACM CCS 2010*, pages 387–398. ACM Press, October 2010. Preprint on IACR ePrint 2010/416.

[Bor01] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP)*, LNCS. Springer, 2001.

[BP06] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology – CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[BP09] Jens-Matthias Bohli and Andreas Pashalidis. *Relations Among Privacy Notions*, pages 362–380. Springer-Verlag, 2009.

[BP10] Bruno Blanchet and David Pointcheval. The computational and decisional Diffie-Hellman assumptions in CryptoVerif. In *Workshop on Formal and Computational Cryptography (FCC 2010)*, Edimburgh, United Kingdom, July 2010.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.

[BR00] John Black and Phillip Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. In *Advances in Cryptology - CRYPTO'00, Lecture Notes in Computer Science*, pages 197–215. Springer-Verlag, 2000.

[BR02] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Advances in Cryptology - EUROCRYPT 2002. Lecture Notes in Computer Science*, pages 384–397. Springer-Verlag, 2002.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free Secret-Ballot Elections (extended abstract). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 544–553, New York, NY, USA, 1994.

[BT04] Gilles Barthe and Sabrina Tarento. A machine-checked formalization of the random oracle model. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *Proceedings of TYPES'04*, volume 3839 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2004.

[BV04] L. Buttyan and I. Vajda. Towards provable security for ad hoc routing protocols. *In Proc. of the ACM Workshop on Security in Ad Hoc and Sensor Networks*, pages 94–105, 2004.

[CB87] Josh Cohen Benaloh. Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret. In *Proceedings on Advances in Cryptology—CRYPTO '86*, pages 251–260, London, UK, 1987. Springer-Verlag.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. *IEEE Symposium on Security and Privacy*, 0:354–368, 2008.

[CDD12] Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. Analysing routing protocols: four nodes topologies are sufficient. In Pierpaolo Degano and Joshua D. Guttman, editors, *Proceedings of the 1st International Conference on Principles of Security and Trust (POST'12)*, volume 7215 of *Lecture Notes in Computer Science*, pages 30–50, Tallinn, Estonia, March 2012. Springer.

[CdH06] Ricardo Corin and Jerry den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.

[CE02] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *Proc. 9th International Static Analysis Symposium (SAS)*, volume 2477 of *LNCS*, pages 326–341. Springer, Sep 2002.

[CG96] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation (extended abstract). In *FOCS*, pages 504–513, 1996.

[CH05] Srdjan Capkun and Jean-Pierre Hubaux. Secure positioning of wireless devices with application to sensor networks. In *IEEE INFOCOM*. IEEE Journal on Selected Areas in Communications: Special Issue on Security in Wireless Ad Hoc Networks, 2005.

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[CLCZ10] H. Comon-Lundh, V. Coltier, and E. Zalinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Transactions on Computational Logic*, pages 496–520, 2010.

[CMFP$^+$06] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On some incompatible properties of voting schemes. In *Proceedings of the IAVoSS Workshop on Trustworthy Elections*, 2006.

[CN08] Debrup Chakraborty and Mridul Nandi. An improved security bound for HCTR. In *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, pages 289–302. Springer-Verlag, Berlin, Heidelberg, 2008.

[CR88] Benny Chor and Ronald L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, pages 901–909, 1988.

[CRC11] Cas Cremers, Kasper Bonne Rasmussen, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. Cryptology ePrint Archive, Report 2011/129, 2011. `http://eprint.iacr.org/`.

[Cre08] C.J.F. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[CRS05] David Chaum, Peter Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In Sabrina di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer Berlin, Heidelberg, 2005.

[CS06] Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2006.

[CS08] Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-counter-hash approach. *IEEE Transactions on Information Theory*, 54(4):1683–1699, 2008.

[Dau12] Marion Daubignard. *Formalisation de preuves de sÃľcuritÃľ concrÃĺte.* PhD thesis, Université de Grenoble, Janvier 2012.

[DB97] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, volume 1, pages 376 –380 vol.1, jun 1997.

[DDMR07] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007.

[DDMW06] Anupam Datta, Ante Derek, John C. Mitchell, and Bogdan Warinschi. Computationally sound compositional logic for key exchange protocols. In *CSFW '06: Proceedings of the 19th IEEE workshop on Computer Security Foundations*, pages 321–334, Washington, DC, USA, 2006. IEEE Computer Society.

[DDS10] Morten Dahl, Stéphanie Delaune, and Graham Steel. Formal analysis of privacy for vehicular mix-zones. In Dimitris Gritzalis and Bart Preneel, editors, *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10)*, volume 6345 of *Lecture Notes in Computer Science*, pages 55–70, Athens, Greece, September 2010. Springer.

[DKPR11] J. Du, E. Kranakis, O. Morales Ponce, and S. Rajsbaum. Neighbor discovery in a sensor network with directional antennae. In *Algosensors*, Saarbruecken, Germany, September 2011.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17:435–487, December 2009.

[DKR10] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Mirosław Kutyłowski, and Ben Adida, editors, *Towards Trustworthy Elections – New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 289–309. Springer, May 2010.

[DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Proc. of the 22nd Symp. on Foundations of ComputerScience*, pages 350–357. IEEE Computer Society Press, 1981.

[DY83a] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[DY83b] D. Dolev and A. Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, pages 198–207, 1983.

[EMST76] William F. Ehrsam, Carl H. W. Meyer, John L. Smith, and Walter L. Tuchman. Message verification and transmission error detection by block chaining. US Patent 4074066, 1976.

[EOKMV11] O. Erdene-Ochir, A. Kountouris, M. Minier, and F. Valois. Enhancing resiliency against routing layer attacks in wireless sensor networks: Gradient-based routing in focus. *International Journal On Advances in Networks and Services*, 4(1 and 2):38–54, 2011.

[EZ06] Charlott Eliasson and André Zúquete. An electronic voting system supporting vote weights. *Internet Research*, 16(5):507–518, 2006.

[FKS11] C. Fournet, M. Kohlweiss, and P. Strub. Modular code-based cryptographic verification. In Y.Chen, G. Danezis, and V. Shmatikov, editors, *ACM-CCS'11*, pages 341–350. ACM, 2011.

[Flo67] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology – AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251. Springer Berlin / Heidelberg, 1992.

[FOPS04a] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. *J. Cryptology*, 17(2):81–104, 2004.

[FOPS04b] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. *J. Cryptology*, 17(2):81–104, 2004.

[Gjo05] Kristian Gjosteen. Homomorphic cryptosystems based on subgroup membership problems. In Ed Dawson and Serge Vaudenay, editors, *Progress in Cryptology - Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 314–327. Springer Berlin / Heidelberg, 2005.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In *STOC*, pages 365–377, 1982.

[Gro08] Distributed Computing Group. Sinalgo - simulator for network algorithms, 2008. http://disco.ethz.ch/projects/sinalgo/.

[Hal04] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, Proceedings*, volume 3348 of *LNCS*, pages 315–327. Springer, 2004.

[Hal05] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. `http://theory.lcs.mit.edu/~shaih/pubs.html`, 2005.

[Hal07] Shai Halevi. Invertible universal hashing and the tet encryption mode. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer, 2007.

[HK05] G.P. Hancke and M.G. Kuhn. An rfid distance bounding protocol. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 67–73. IEEE, 2005.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, October 1969.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[HPJ03] Y. C. Hu, A. Perrig, and D. B. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, volume 3, pages 1976–1986 vol.3, 2003.

[HPJ05] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. *Wirel. Netw.*, 11(1-2):21–38, January 2005.

[HR03] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.

[HR04] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.

[Hug95] B.D. Hughes. *Random walks and random environments*, volume 1. Oxford University Press, 1995.

[IK03a] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[IK03b] Tetsu Iwata and Kaoru Kurosawa. On the security of a new variant of OMAC. In Jong In Lim and Dong Hoon Lee, editors, *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2003.

[IK03c] Tetsu Iwata and Kaoru Kurosawa. Stronger security bounds for OMAC, TMAC, and XCBC. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT*, volume 2904 of *Lecture Notes in Computer Science*, pages 402–415. Springer, 2003.

[IK06] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. *Journal of Computer and Systems Sciences*, 72(2):286–320, 2006.

[JCJ02] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections, 2002. Cryptology ePrint Archive, Report 2002/165, `http://eprint.iacr.org/`.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05, pages 61–70. ACM, 2005.

[JMB01] David B. Johnson, David A. Maltz, and Josh Broch. Dsr: the dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad hoc networking*, pages 139–172. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Jr.91] Hendrik W. Lenstra Jr. On the chor-rivest knapsack cryptosystem. *J. Cryptology*, 3:149–155, 1991.

[Jut01] Charanjit S. Jutla. Encryption modes with almost free message integrity. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 529–544, London, UK, 2001. Springer-Verlag.

[JZF03] Rui Joaquim, André Zúquete, and Paulo Ferreira. Revs - a robust electronic voting system. In *IADIS International Conference e-Society 2003, Lisboa (Portugal), June 3-6*, 2003.

[KAF$^+$10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel ThomÃľ, Pierrick Gaudry, Peter L. Montgomery, Dag Arne Osvik, Herman Te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus, 2010.

[Kas12] Ali Kassem. Formal verification of routing protocols in ad-hoc networks in presence of independent intruders. Master's thesis, Université Joseph Fourrier, 2012.

[KI03] Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-key CBC MAC. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2003.

[KM10] Steve Kremer and Laurent Mazaré. Computationally sound analysis of protocols using bilinear pairings. *Journal of Computer Security*, 18(6):999–1033, 2010.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *Proceedings of the 15th European Symposium on Research in Computer Security, ESORICS 2010*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010.

[KSR10] Petr Klus, Ben Smyth, and Mark D. Ryan. Proswapper: Improved equivalence verifier for proverif. `http://www.bensmyth.com/proswapper.php`, 2010.

[KT09] R. Küsters and T. Truderung. An Epistemic Approach to Coercion-Resistance for Electronic Voting Protocols. In *2009 IEEE Symposium on Security and Privacy (S&P 2009)*, pages 251–266. IEEE Computer Society, 2009.

[LBD$^+$04] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *Information Security and Cryptology - ICISC 2003*, volume 2971 of *LNCS*, pages 245–258. Springer Berlin / Heidelberg, 2004.

[Li10] K. Li. Performance analysis and evaluation of random walk algorithms on wireless networks. In *IPDPS Workshops*, pages 1–8, 2010.

[LJP10] Lucie Langer, Hugo Jonker, and Wolter Pieters. Anonymity and verifiability in voting: understanding (un)linkability. In *Proceedings of the 12th international conference on Information and communications security*, ICICS'10, pages 296–310. Springer-Verlag, 2010.

[Lov93] L. Lovász. Random walks on graphs: A survey. In T. Szőnyi D. Miklós, V.T. Sós, editor, *Combinatorics, Paul Erdős is Eighty*, volume 2 of *Bolyai Society Mathematical Studies*, pages 1–46. János Bolyai Mathematical Society, 1993.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166, Berlin (Germany), 1996. Springer-Verlag.

[Low98] G. Lowe. Casper: a compiler for the analysis of security protocols. *J. Comput. Secur.*, 6(1-2):53–84, 1998.

[LRW02a] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 31–46, London, UK, 2002. Springer-Verlag.

[LRW02b] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[Mea96] C. Meadows. Language generation and verification in the NRL protocol analyzer. In *Proc. 9th Computer Seurity Foundation Workshop (CSFW'96)*, pages 48–62, Kenmare (Ireland), 1996. IEEE Comp. Soc. Press.

[MF07] David A. McGrew and Scott R. Fluhrer. The security of the extended codebook (XCB) mode of operation. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.

[MH06] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inf. Theor.*, 24(5):525–530, September 2006.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *IEEE Symposium on Security and Privacy*, May 1997.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO 2006*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.

[MPP$^+$06] Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks, edited volume, Springer, Nov. 2006*, Nov 2006.

[MS01a] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.

[MS01b] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. *In Proc. of the ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[MS03] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. 16th Computer Security Foundation Workshop (CSFW'03)*, pages 47–62, Pacific Grove (California, USA), 2003. IEEE Comp. Soc. Press.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, Proceedings*, LNCS, pages 343–355. Springer, 2004.

[Now07] David Nowak. A framework for game-based security proofs. In *ICICS*, pages 319–333, 2007.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communication of the ACM*, 21(12):993–999, 1978.

[NS98] David Naccache and Jacques Stern. A New Public Key Cryptosystem Based on Higher Residues. In *ACM Conference on Computer and Communications Security*, pages 59–66, 1998.

[Oka96] Tatsuaki Okamoto. An electronic voting scheme. In *Proceedings of the IFIP World Conference on IT Tools*, pages 21–30, 1996.

[OMVK10] E.-O. Ochirkhand, M. Minier, F. Valois, and A. Kountouris. Resilient networking in wireless sensor networks. Research report, Inria, 2010.

[OW07] R. Ortner and W. Woess. Non-backtracking random walks and cogrowth of graphs. *Canad. J. Math.*, 59(4):828–844, 2007.

[Pau98] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, 6(1-2):85–128, 1998.

[PB12] Miriam Paiola and Bruno Blanchet. Verification of security protocols with lists: From length one to unbounded length. In Pierpaolo Degano and Joshua D.

Guttman, editors, *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2012.

[PCTS02] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5, 2002.

[PH02] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. *Proc.of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, January 2002.

[Poi00] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In *PKC'00*, pages 129–146, 2000.

[Pol74] J. M. Pollard. Theorems on Factorization and Primality Testing. *Mathematical Proceedings of the Cambridge Philosophical Society*, 76(03):521–528, 1974.

[Pon11] C. Ponsonnet. Secure probabilistic routing in wireless sensor networks. Master thesis, Laboratoire Verimag, 2011.

[PPH07] Marcin Poturalski, Panos Papadimitratos, and Jean-Pierre Hubaux. Secure Neighbor Discovery in Wireless Networks: Formal Investigation of Possiblity. Technical report, EPFL, 2007.

[PPH08a] Marcin Poturalski, Panos Papadimitratos, and Jean-Pierre Hubaux. Secure neighbor discovery in wireless networks: formal investigation of possibility. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 189–200. ACM, 2008.

[PPH08b] Marcin Poturalski, Panos Papadimitratos, and Jean-Pierre Hubaux. Towards Provable Secure Neighbor Discovery in Wireless Networks. In *The 6th ACM Workshop on Formal Methods in Security Engineering*, pages 31–42, Alexandria, VA, 2008. ACM.

[PPS$^+$08] Panagiotis Papadimitratos, Marcin Poturalski, Patrick Schaller, Pascal Lafourcade, Davin Basin, Srdjan Capkun, and Jean-Pierre Hubaux. Secure Neighborhood Discovery: A Fundamental Element for Mobile Ad Hoc Networking. *IEEE Communications Magazine*, 46(2), 2008.

[PR99] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. *Pro. 2nd Workshop on Mobile Computing Systems and Appliations*, pages 99–100, 1999.

[PR00] Erez Petrank and Charles Rackoff. CBC MAC for Real-Time Data Sources. *J. Cryptology*, 13(3):315–338, 2000.

[RDDM07] Arnab Roy, Anupam Datta, Ante Derek, and John C. Mitchell. Inductive proofs of computational secrecy. In *ESORICS*, pages 219–234, 2007.

[RDM07] Arnab Roy, Anupam Datta, and John C. Mitchell. Formal proofs of cryptographic security of diffie-hellman-based protocols. In *TGC*, pages 312–329, 2007.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *IEEE Symposium on Foundations of Secure Systems*, 1995.

[RP05] Peter Y.A. Ryan and Thea Peacock. Prêt à voter: a systems perspective. Technical Report 929, Newcastle University, School of Computing Science, 2005.

[RSG$^+$00] P.Y.A. Ryan, S.A. Schneider, M.H. Goldsmith, G. Lowe, and A.W. Roscoe. *The modelling and analysis of security protocols: the CSP approach.* Addison-Wesley, 2000.

[S$^+$10] W. A. Stein et al. *Sage Mathematics Software (Version 4.5.1).* The Sage Development Team, 2010. `http://www.sagemath.org`.

[SBP01] D. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[SC10] Ben Smyth and Veronique Cortier. Attacking and fixing helios: An analysis of ballot secrecy. Cryptology ePrint Archive, Report 2010/625, 2010. `http://eprint.iacr.org/`.

[SC11] Ben Smyth and Veronique Cortier. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 297–311. IEEE, 2011.

[Sch96] S.A. Schneider. Security properties and CSP. In *Proc. of the Symposium on Security and Privacy*, pages 174–187. IEEE Computer Society Press, 1996.

[Shm04] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proc. 13th European Symposium On Programming (ESOP'04)*, volume 2986 of *LNCS*, pages 355–369, Barcelona (Spain), 2004. Springer-Verlag.

[Sho02] Victor Shoup. Oaep reconsidered. *J. Cryptology*, 15(4):223–249, 2002.

[Sho04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs, 2004. URL: `http://eprint.iacr.org/2004/332`.

[SIY09] Izumi Kubo Satoshi Ikeda and Masafumi Yamashita. The hitting and cover times of random walks on finite graphs using local degree information. *Theoretical Computer Science*, 410:94–100, 2009. Contents lists available at ScienceDirect.

[Sla10] G. Slade. The self-avoiding walk: A brief survey. To appear in Surveys in Stochastic Processes, Proceedings of the Thirty-third SPA Conference in Berlin, 2009, to be published in the EMS Series of Congress Reports, eds. J. Blath, P. Imkeller, S. Roelly, 2010.

[SPRH09] R. Shokri, Marcin Poturalski, G. Ravot, and Panos Papadimitratosand Jean-Pierre Hubaux. A practical secure neighbor verification protocol for wireless sensor networks. In *Proceedings of the second ACM conference on Wireless network security*, pages 193–200. ACM, 2009.

[SRKK10] Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, volume 6186 of *LNCS*, pages 146–163. Springer, 2010.

[SS01] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, volume 1, pages 357–361. Ieee, 2001.

[SSBC09a] Patrick Schaller, Benedikt Schmidt, David Basin, and Srdjan Capkun. Modeling and verifying physical properties of security protocols for wireless networks. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*. IEEE, 2009.

[SSBC09b] Patrick Schaller, Benedikt Schmidt, David Basin, and Srdjan Capkun. Modeling and verifying physical properties of security protocols for wireless networks. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, pages 109–123. IEEE, 2009.

[SSQ02] David Soldera, Jennifer Seberry, and Chengxin Qu. The analysis of zheng-seberry scheme. In Lynn Margaret Batten and Jennifer Seberry, editors, *ACISP*, volume 2384 of *Lecture Notes in Computer Science*, pages 159–168. Springer, 2002.

[Tea04] The Coq Development Team. *The Coq proof assistant reference manual.* LogiCal Project, 2004. Version 8.0.

[TSG10] F. Javier Thayer, Vipin Swarup, and Joshua D. Guttman. Metric strand spaces for locale authentication protocols. In Masakatsu Nishigaki, Audun Jøsang, Yuko Murayama, and Stephen Marsh, editors, *Trust Management IV - 4th IFIP WG*

11.11 International Conference, IFIPTM 2010, Morioka, Japan, June 16-18, 2010. Proceedings, volume 321 of *IFIP Conference Proceedings*, pages 79–94. Springer, 2010.

[UMQ10] Dominique Unruh and Jörn Müller-Quade. Universally composable incoercibility. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*, pages 411–428. Springer, 2010.

[Vau98] Serge Vaudenay. Cryptanalysis of the chor-rivest cryptosystem. In *CRYPTO '98*, pages 243–256. Springer-Verlag, 1998.

[VKT05] S. Vasudevan, J. Kurose, and D. Towsley. On neighbor discovery in wireless networks with directional antennas. In *INFOCOM 2005*, volume 4, pages 2502–2512, 2005.

[WB09] Roland Wen and Richard Buckland. Masked ballot voting for receipt-free online elections. In *Proceedings of the 2nd International Conference on E-Voting and Identity*, VOTE-ID '09, pages 18–36, Berlin, Heidelberg, 2009. Springer-Verlag.

[WC19] M. Wegman and J. L. Carter. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1919.

[WC81] M. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

[WFW05] Peng Wang, Dengguo Feng, and Wenling Wu. On the security of tweakable modes of operation: TBC and TAE. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2005.

[WL99] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '99, pages 7–14, New York, NY, USA, 1999. ACM.

[ZA02] M. Guerrero Zapata and N. Asokan. Securing ad-hoc routing protocols. *Proc. 1st ACM workshop on Wireless Security (WiSE' 02)*, 2002.

[Zha08] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. IACR ePrint Archive 2008/434, 2008. Also in Proc. of Typed Lambda Calculi and Applications 2009.

[ZS93] Yuliang Zheng and Jennifer Seberry. Immunizing public key cryptosystems against chosen ciphertext attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):715–724, 1993.

My Publications List

[1] K. Altisen, S. Devismes, A. Gerbaud, and P. Lafourcade. Comparisons of mean hitting times for tabu random walks on finite graphs. Technical report, Laboratoire Verimag, 2011. `http://www-verimag.imag.fr/TR/TR-2012-6.pdf`.

[2] K. Altisen, S. Devismes, A. Gerbaud, and P. Lafourcade. Analysis of random walks using tabu lists. In G. Even and M. M. Halldórsson, editors, *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers*, volume 7355 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2012.

[3] K. Altisen, S. Devismes, P. Lafourcade, and C. Ponsonnet. Routage par marche aléatoire à listes tabous. In *Algotel*, 2011. to appear.

[4] K. Altisen, S. Devismes, P. Lafourcade, and C. Ponsonnet. Routage par marche aléatoire à listes tabous. In *Algotel'2011 (23–26 Mai 2011, Cap Estérel)*, 2011.

[5] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In P. Ning, P. F. Syverson, and S. Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 371–380, Alexandria, Virginia, USA, october 2008. ACM.

[6] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Automated proofs for asymmetric encryption. *Journal of Automated Reasoning*, 2010.

[7] C. J. F. Cremers, P. Lafourcade, and P. Nadeau. Comparing state spaces in automatic security protocol analysis. In V. Cortier, C. Kirchner, M. Okada, and H. Sakurada, editors, *Formal to Practical Security - Papers Issued from the 2005-2008 French-Japanese Collaboration*, volume 5458 of *Lecture Notes in Computer Science*, pages 70–94. Springer, 2009.

[8] J. Dreier, P. Lafourcade, and Y. Lakhnech. Vote-independence: A powerful privacy notion for voting protocols. In *Proceedings of the 4th Workshop on Foundations & Practice of Security (FPS)*, LNCS. Springer, 2011.

[9] J. Dreier, P. Lafourcade, and Y. Lakhnech. Vote-independence: A powerful privacy notion for voting protocols. Technical Report TR-2011-8, Verimag Research Report, April 2011. Available at `http://www-verimag.imag.fr/TR/TR-2011-8.pdf`.

[10] J. Dreier, P. Lafourcade, and Y. Lakhnech. Defining privacy for weighted votes, single and multi-voter coercion. In S. Foresti, M. Yung, and F. Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, volume 7459 of *Lecture Notes in Computer Science*, Pisa, Italy, sept 2012. Springer.

[11] J. Dreier, P. Lafourcade, and Y. Lakhnech. A formal taxonomy of privacy in voting protocols. In *First IEEE International Workshop on Security and Forensics in Communication Systems (ICC'12 WS - SFCS)*, 2012.

[12] J. Dreier, P. Lafourcade, and Y. Lakhnech. On defining privacy in the presence of weighted votes and the equivalence of single and multi-voter coercion. Technical Report TR-2012-2, Verimag Research Report, March 2012. Available at `http://www-verimag.imag.fr/TR/TR-2012-2.pdf`.

[13] J. Dreier, P. Lafourcade, and Y. Lakhnech. On parallel factorization of processes in the applied pi calculus. Technical Report TR-2012-3, Verimag, March 2012. Available at `http://www-verimag.imag.fr/TR/TR-2012-3.pdf`.

[14] J. Dreier, P. Lafourcade, and Y. Lakhnech. On unique decomposition of processes in the applied π-calculus. In *Technichal repoart*, 2012.

[15] L. Fousse, P. Lafourcade, and M. Alnuaimi. Benaloh's dense probabilistic encryption revisited. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 2011.

[16] M. Gagné, P. Lafourcade, and Y. Lakhnech. OCaml implementation of our method. Laboratoire VERIMAG, Université Joseph Fourier, France, April 2012. Available at `http://www-verimag.imag.fr/~gagne/macChecker.html`.

[17] M. Gagné, P. Lafourcade, and Y. Lakhnech. Automated security proofs for message authentication codes. Technical report, Verimag, 2012. Technical Report TR-2012-5.

[18] M. Gagne, P. Lafourcade, Y. Lakhnech, and S. Reihaneh. Automated proofs for encryption modes. In *13th Annual Asian Computing Science Conference Focusing on Information Security and Privacy: Theory and Practice (ASIAN0'9)*, Urumqi, China, Oct. 2009.

[19] N. Ghoualmi, N. Kahya, and P. Lafourcade. Key management protocol in wimax revisited. In *The Third International Conference on Communications Security and Information Assurance (CSIA 2012)*, Delhi, India, May 2012. Springer.

[20] R. Jamet and P. Lafourcade. *Formal Model for (k)-Neighborhood Discovery Protocols*, chapter in Advances in Network Analysis and its Applications. Springer, 2012. to appear.

[21] B. Ksieżopolski and P. Lafourcade. Attack and revison of an electronic auction protocol using OFMC. Technical Report 549, Department of Computer Science, ETH Zurich, Switzerland, Feb. 2007. 13 pages.

[22] P. Lafourcade. Application de la résolution de conflits « logiques », à l'aide à la décision pour la résolution de conflits des problèmes d'ordonnancement. Rapport de DEA, DEA Représentation de la Connaissance et Fomalisation du Raisonnement, Toulouse, France, June 2003. 66 pages.

[23] P. Lafourcade. *Vérification des protocoles cryptographiques en présence de théories équationnelles.* Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, ENS Cachan, Sept. 2006. 209 pages.

[24] P. Lafourcade, V. Terrade, and S. Vigier. Comparison of cryptographic verification tools dealing with algebraic properties. In J. Guttman and P. Degano, editors, *sixth International Workshop on Formal Aspects in Security and Trust, (FAST'09)*, Eindhoven, Netherlands, Nov. 2009.

[25] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Čapkun, and J.-P. Hubaux. Secure Neighborhood Discovery: A Fundamental Element for Mobile Ad Hoc Networking. *IEEE Communications Magazine*, 46(2):132–139, February 2008.

[26] J. Tharaud, S. Wohlgemuth, I. Echizen, N. Sonehara, G. Müller, and P. Lafourcade. Privacy by data provenance with digital watermarking - a proof-of-concept implementation for medical services with electronic health records. In I. Echizen, J.-S. Pan, D. W. Fellner, A. Nouak, A. Kuijper, and L. C. Jain, editors, *Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2010), Darmstadt, Germany, 15-17 October, 2010, Proceedings*, pages 510–513. IEEE Computer Society, 2010.