J Autom Reasoning manuscript No.
(will be inserted by the editor)

Automated Proofs of Block Cipher Modes of Operation

Martin Gagné · Pascal Lafourcade · Yassine Lakhnech · Reihaneh Safavi-Naini

Received: date / Accepted: date

Abstract We present a Hoare logic for proving semantic security and determining exact security bounds of a block cipher mode of operation. We propose a simple yet expressive programming language to specify encryption modes, semantic functions for each command (statement) in the language, an assertion language that allows to state predicates and axioms, and rules to propagate the predicates through the commands of a program. We also provide heuristics for finding loop invariants that are necessary for the application of our rule on for-loops. This enables us to prove the security of protocols that take arbitrary length messages as input. We implemented a prototype that uses this logic to automatically prove the security of block cipher modes of operation. This prototype can prove the security of many standard modes of operation, such as Cipher Block Chaining (CBC), Cipher FeedBack mode (CFB), Output FeedBack (OFB), and CounTeR mode (CTR).

Keywords Automated Verification · Hoare Logic · Provable Cryptography · Symmetric Encryption · Block Cipher

1 Introduction

Block ciphers are symmetric key cryptographic primitives that take a fixed-length plaintext message, together with a key, and output a fixed-length ciphertext. A

Martin Gagné
Wheaton College, Norton, United States of America
E-mail: gagne_martin@wheatoncollege.edu

Pascal Lafourcade
LIMOS, Université Clermont 1
E-mail: lafourcade@sancy.univ-bpclermont.fr

Yassine Lakhnech
Université Joseph Fourier (Grenoble 1), CNRS, Grenoble, France
E-mail: yassine.lakhnech@imag.fr

Reihaneh Safavi-Naini
University of Calgary, Calgary, Canada
E-mail: rei@ucalgary.ca

mode of operation is a symmetric encryption scheme that uses a block cipher to encrypt arbitrary length messages. Modes of operation are commonly used to ensure the confidentiality of messages exchanged over a public channel. Their semantic security is proven by reducing the security of the scheme to the security of the block cipher.

While the security of early modes of operation, such as Cipher Block Chaining mode (CBC), Cipher FeedBack mode (CFB), Output Feedback mode (OFB), and CounTeR mode (CTR), can be proven relatively easily, the same cannot be said about more recent modes of operation. Many new modes of operation were designed in the past few years ([Jut01, LRW02, HR03, MV04, BRW04, HR04, Hal04, WFW05, CS06, MF07, Hal07, CN08, CS08] to name only a few), which often offer security properties that early modes did not possess. However, additional properties often come with increased complexity of the mode of operation, which, in turn, increase the complexity of the proof of security.

Automated verification tools can be used to provide an additional security argument for modes of operation using the security definition of block ciphers and the structure of the mode. In this paper, we propose an approach towards automating the process of providing a security proof for block cipher modes of operation that takes advantage of the structure of modes, expressed as a set of basic operations.

1.1 Contributions

We proposed a method based on a Hoare logic for proving the semantic security of modes of operation for symmetric key block ciphers. A Hoare logic is a set of logical rules used to propagate predicates through a program and has been originally designed to reason about the correctness of programs. In this paper, we adapt the technique for proving semantic security using the traditional indistinguishability-based security definition of symmetric encryption schemes. Constructing our method requires the following elements:

◇ **Simple Programming Language.** We notice that many modes use a small set of operations such as XOR, concatenation, computation of the block cipher, and sampling of random values. We introduce a programming language describing these operations which, while quite simple, is expressive enough to describe all traditional modes of operation.

◇ **Semantics.** We associate an initial state to the initial variables describing all the internal information needed to run the program of the mode of operation, and define a corresponding initial distribution of configurations. We define the effect of each command as a function on distributions of configurations. We found that using a straightforward definition for the semantic function of the block cipher greatly complicates the analysis, so we present an alternative, 'simulated' semantic function for the block cipher and show the precise event that causes the alternative semantic function to become distinguishable from the natural one.

◇ **Assertion Language.** The assertion language for our logic consists of six predicates, each with a precise definition based on properties of distributions of configurations. Intuitively, the predicates can be described as follows: one that indicates that the value of a variable is uniformly distributed, and is independent from the

values of a set of variables, three predicates that allow us to determine which variables are used as counters, one keeps track of the probability that the simulated semantics can be distinguished from the natural semantics, and one that ensures that the values on which the block cipher is computed are properly distributed. We prove relationships between these predicates and show which predicates hold in every initial distribution of configurations.

◇ **Encoding Security.** We prove that a symmetric mode of operation is semantically secure (IND-CPA) if a certain conjunction of predicates holds at the end of the execution of the program. Our theorem also shows how to derive precise security bounds on the security of the scheme from these predicates.

◇ **Hoare Logic.** Finally, we present a set of rules for a Hoare logic that can be used to propagate the predicates through the code (program) of the mode of operation.

With all these elements in place, proving the security of a mode of operation becomes a relatively simple matter: starting with the predicates holding in all the initial distributions, we use the Hoare logic to propagate the predicates through the entire program describing the mode of operation, and we verify whether the predicates that imply semantic security hold at the end.

We also show how to reason about for-loops, which enables us to argue about the security of modes of operation when applied to arbitrary length messages: we present two heuristics that can be used to discover stable loop invariants, and show how they can be applied, using examples. The first heuristic is designed specifically for modes that consist of a few initial steps followed by a single loop, each iteration of which produces a cipher block corresponding to a message block (this is how the overwhelming majority of modes of operation are designed). The heuristic examines the predicates that hold at the end of an iteration of the loop to determine if these predicates will be sufficient to prove the security of the mode, and if they are sufficient to obtain the same predicates at the end of a subsequent iteration of the loop. While relatively simple, this heuristic can successfully discover loop invariants for all single-loop modes of operations we examined.

In the interest of generality, we also present a second, far more robust heuristic based on widening in abstract interpretation. This heuristic attempts to find patterns that can be extrapolated after executing the loop a fixed number of times. While relatively easy to describe, this heuristic can quickly become very fastidious to implement since it requires the programmer to figure out every possible ways in which fields within predicates could be extrapolated, or predicates could be accumulated, for each predicate used in the analysis. In this, we greatly benefit from the relative simplicity of our predicates. This heuristic is capable of discovering loop invariants even when the loop causes an accumulation of new predicates at each iteration of the loop, and is particularly useful when the mode of operation is implemented using multiple loops.

Our method was implemented into an OCaml prototype [GLLSN]. The prototype requires about 2000 lines of code and can automatically produce proofs of security for several encryption modes including CBC, CFB, CTR and OFB. Of course our system does not prove ECB mode, because ECB is not semantically secure.

1.2 Related Work

The security of symmetric encryption has been extensively studied by cryptographers in the past twenty years. An extensive discussion on different security notions for symmetric encryption and a proof of the CBC mode of encryption is presented in [BDJR97]. A security analysis of the encryption mode CBC-MAC is given in [BKR00] and [ÉJJV01].

Another approach is to describe the security of symmetric encryption modes as a non-interference property. For example, Courant et al. [CEL07] present a computationally sound type system with exact security bounds for such programs describing deterministic encryption schemes. This type system has been applied to verify some symmetric encryption modes.

In [CDE+08], the authors proposed a Hoare logic for proving semantic security of asymmetric encryption schemes in the random oracle model. A similar method is used in [GLLSN09], and extended in [GLLSN11], to verify the security of block-cipher-based symmetric encryption modes, and in [GLL13] to verify the security of almost-universal hash functions and message authentication codes. This paper provides improvements over the techniques presented in [GLLSN09] and [GLLSN11] in the following ways:

- we present new heuristics that allow us to model and analyse for-loops and so handle encryption of arbitrarily long messages. Designing such heuristics is one of the hardest problems in program verification, and required us to create our own techniques, and adapt an existing method to the specifics of our analysis.
- our predicates now have clear semantic meanings, whereas in our previous work, predicates were sometimes *ad hoc* or purely syntactic.
- the predicates dealing with counters have been improved to be less dependent on the order of instructions than previous works [GLLSN09, GLLSN11]. In our previous work, inverting the order of two commands could sometimes prevent our logic from proving the security of a scheme. With the improved predicates, our analysis now succeeds regardless of the order of the commands.
- we provide exact security bounds on the reduction of the security of the mode to the security of the underlying block cipher. This required major modification to our semantics and to the definition of our predicates, which in turn required us to adapt the proof of all our rules.

The work [BDK+10] was a first step in order to define equality reasoning on probabilitic terms. This idea has been used in the construction of EasyCrypt. In our work we use an approach based on propagation of predicates and we do not consider direct equality of term, as is done in EasyCrypt [BGLB11, BGHB11] or in CIL [BDKL10].

While previous tools, such as Cryptoverif [BP06] and EasyCrypt [BGLB11, BGHB11], can be used to verify the security of cryptographic schemes, they are not well-suited to the task of proving the security of encryption schemes. Cryptoverif does not support loop constructs, which are necessary for arguing about the security of schemes for arbitrarily long messages. Easycrypt uses a game-based approach and requires a human to enter the sequence of games. Our method is complementary to these two papers, and could be integrated with the above tools to enable a more comprehensive analysis of cryptographic protocols.

Our prototype can be extended to construct a tool that automatically constructs and proves the security of new block cipher modes of operation, similar to what Zoocrypt does for padding-based public key encryption schemes [BCG+13].

Such an automated synthesizer for block cipher modes of operations has been published [MKG14], but their method for proving the security of the scheme is fundamentally different from ours. They model the mode of operation as a directed acyclic graph, then show that if there exists a labeling for the associated graph then the mode is secure against chosen-plaintext attacks. In order to construct the labeling they transform the problem into a constraint-satisfaction problem and they rely on an SMT solver to decide if the block cipher is secure or not. Their approach can be seen as a global approach, whereas our work consist in a local analysis of each command, hence our two works follow two different point of view. The fact that our method processes each command essentially out of its context necessitates the use of more complex predicates, but these complex predicates, in turn, enable to gather more information for our analysis, particularly concerning the analysis of counters. Moreover, they make a few simplifying assumptions (the mode is described in a single loop, little information passed from one iteration to the next) to reduce the search space for synthesis and to simplify the analysis. Using our method would enable the discovery of a greater variety of modes of operation thanks to our more flexible treatment of the increment operation, and loop invariant discovery. Finally, we are now able to automatically generate a security bound, which provides a more complete analysis of the mode.

1.3 Organization

Section 2 recalls the cryptographic background that is needed in the remainder of the paper. In Section 3, we present our grammar, the semantic functions of the commands and the assertion language that is used by our logic. In Section 4, we prove that if the ciphertext obtained by running the program is indistinguishable from a random value, as described in our predicates, then the mode of operation is semantically secure. In Section 5, we present the set of rules that is used to propagate the predicates. Section 6 combines the results of the two previous sections to prove the soundness of our method for proving semantic security. In Section 7, we present a set of heuristics for finding stable loop invariants and show how to use them to prove the security of some example cases. Section 8 concludes the paper.

2 Background

We describe the notation and cryptographic definitions that are used in this paper.

2.1 Notation and Conventions

For a probability distribution $\mathcal{D}$, we denote by $x \xleftarrow{\$} \mathcal{D}$ the operation of sampling a value x according to distribution $\mathcal{D}$. If S is a finite set, we denote by $x \xleftarrow{\$} S$ the operation of sampling x uniformly at random from the values in S. For a probabilistic algorithm $\mathcal{A}$, we denote by $x \xleftarrow{\$} \mathcal{A}$ the process of running algorithm

$\mathcal{A}$ on uniform random coins and assigning the result to x. If η is a positive integer, we denote by 1^η the string consisting of η consecutive 1's. The function $\mathsf{len} : \{0,1\}^* \to \mathbb{N}$ is the function that takes a string and returns the length of the string. 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 $\eta \geq \eta_0$, $f(\eta) \leq \frac{1}{p(\eta)}$.

2.2 Security Model

We recall the formal definition of symmetric encryption schemes, and show how their security is defined.

Definition 1 (Symmetric Encryption) A *symmetric encryption scheme* is a triple of probabilistic polynomial-time algorithms $(\mathbf{K}, \mathbf{E}, \mathbf{D})$ where,

- the key generation algorithm $\mathbf{K}(1^\eta)$ takes a security parameter η[1] and outputs a key k;
- the encryption algorithm $\mathbf{E}(k, m)$ takes a key k and a message m, and outputs a ciphertext; and
- the decryption algorithm $\mathbf{D}(k, c)$ takes a key k and a ciphertext c, and outputs a message.

These algorithms must satisfy the standard correctness requirement that for any key k generated by $\mathbf{K}$ and any message m, we have $\mathbf{D}(k, \mathbf{E}(k, m)) = m$.

Informally, we say that an encryption scheme is secure if no polynomial-time algorithm can distinguish between the encryption of two equal-length messages of its own choosing. To define security formally, we first describe a "left-right" selection function that, given two messages and a bit, chooses one of the messages depending on the value of the bit. We define the function $\mathsf{LR} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\} \to \{0,1\}^*$, as follows:

$$\mathsf{LR}(M_0, M_1, b) = \begin{cases} \bot \text{ if } |M_0| \neq |M_1| \\ M_0 \text{ if } |M_0| = |M_1| \text{ and } b = 0, \\ M_1 \text{ if } |M_0| = |M_1| \text{ and } b = 1. \end{cases}$$

We recall the formal definition of security for symmetric encryption, given by Bellare *et al.* in [BDJR97]. In the definition, the adversary has access to an oracle which, given two equal length messages, either always encrypts the first message, or always encrypts the second one (for the sake of definiteness, the oracle returns $\bot$ if the messages are not of equal length). The objective of the adversary is then to determine which of the two messages 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.

[1] The parameter η is given in unary notation because the algorithm $\mathbf{K}$ is required to run in time polynomial in the length of its input.

Definition 2 (LoR-CPA security [BDJR97]) Let $\mathcal{S} = (\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a symmetric encryption scheme. An *LoR-CPA adversary* is a probabilistic algorithm $\mathcal{A}$ that has access to an oracle $\mathbf{E}(k, \mathsf{LR}(\cdot, \cdot, b))$,[2] takes a security parameter η and outputs a bit b'. Consider the following experiment:

$$\begin{array}{l}\text{Experiment } \mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-b}(\mathcal{A}, \eta)\text{:} \\ \quad k \xleftarrow{\$} \mathbf{K}(1^\eta) \\ \quad b' \xleftarrow{\$} \mathcal{A}^{\mathbf{E}(k, \mathsf{LR}(\cdot,\cdot,b))}(1^\eta) \\ \quad \text{return } b'\end{array}$$

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

$$\mathsf{Adv}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}}(\mathcal{A}, \eta) = \left|\mathsf{Pr}[\mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-1}(\mathcal{A}, \eta) = 1] - \mathsf{Pr}[\mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-0}(\mathcal{A}, \eta) = 1]\right|.$$

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

This definition corresponds to the intuition that no information other than the length of the message should be leaked by the ciphertext.

We are interested in symmetric encryption schemes built using block ciphers, so we recall the definition of block ciphers and their security definition.

A *block cipher* is a family of efficiently computable permutations $\mathcal{E} : \mathcal{K} \times \{0,1\}^\eta \to \{0,1\}^\eta$ such that for each key $k \in \mathcal{K}$, the function $\mathcal{E}_k : \{0,1\}^\eta \to \{0,1\}^\eta$ defined by $\mathcal{E}_k(s) = \mathcal{E}(k, s)$ is a permutation. In most cases, $\mathcal{K}$ is equal to $\{0,1\}^{p(\eta)}$ for some polynomial p.

A block cipher is secure if, for a randomly sampled key, the block cipher is indistinguishable from a function sampled at random from the set of all functions $\{0,1\}^\eta \to \{0,1\}^\eta$. It is standard to model block ciphers as pseudo-random functions (PRF) instead of pseudo-random permutations (PRP) because PRFs and PRPs are statistically close [BDJR97] (this is often called the *PRP-PRF switching lemma*), and doing so makes the security arguments easier.

Definition 3 (Pseudo-Random Function) Let $F : \mathcal{K} \times \{0,1\}^\eta \to \{0,1\}^\eta$ be a family of functions and let $\mathcal{A}$ be an algorithm that takes an oracle and returns a bit. The *PRF-advantage* of an adversary $\mathcal{A}$ is defined as follows.

$$\mathsf{Adv}_{\mathcal{A},F}^{PRF} = \left|\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F(K,\cdot)}(1^\eta) = 1] - \Pr[R \xleftarrow{\$} \Phi_\eta : \mathcal{A}^{R(\cdot)}(1^\eta) = 1]\right|$$

where Φ_η is the set of all functions from $\{0,1\}^\eta$ to $\{0,1\}^\eta$. We say that F is a family of pseudo-random functions if, for any polynomial-time adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A},F}^{PRF}$ is a negligible function of η.

3 Model

In this section, we introduce a grammar for generating encryption modes. We present the semantics of each command and introduce the assertion language that will be used by our Hoare logic.

[2] Again, for the sake of definiteness, we define $\mathbf{E}(k, \perp) = \perp$ for any key k.

3.1 Grammar

We consider the language defined by the BNF grammar of Figure 1, where x, y and z are variables, and p and q are positive integers. For our purpose here, variables are simply strings of characters. The method for assigning them a value will be discussed in Section 3.2, which presents the semantics of the language. We refer to individual instructions as *commands*, and to lists of commands as *programs*. Informally, each command has the following effect:

- $x \xleftarrow{\$} \mathcal{U}(l)$ denotes 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 value obtained by incrementing the value of y by 1. The string value of y is interpreted as a binary number, this number is incremented by one modulo $2^{\mathsf{len}(y)}$ (the length of y as a string) and the result is converted back into a string of length $\mathsf{len}(y)$ by adding leading zeros if necessary.
- $x := \mathcal{E}(y)$ denotes the assignment to x of the value obtained by computing the function $\mathcal{E}$ on the value of y.
- $\mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}\colon [c_l]$ denotes the successive execution of $c_p, \ldots, c_q$ where $p \leq q$. For definiteness, if $p > q$, the command has no effect.
- $c_1; c_2$ denotes that c_2 is executed after c_1 is completed.

We assume that the encryption scheme uses only one block cipher. This allows us to simplify the notation by writing $\mathcal{E}(y)$ instead of $\mathcal{E}(k, y)$. It is understood that a key is selected at the initialization of the scheme, and remains the same throughout. This assumption allows us to greatly simplify the notation in the program and the semantics, and we show in Section 3.2 how one could remove this assumption if necessary.

$$\begin{aligned} \mathsf{cmd} \quad ::= \quad & x \xleftarrow{\$} \mathcal{U}(l) \mid x := y \mid x := \mathcal{E}(y) \mid x := y \oplus z \mid x := y \| z \\ & \mid x := y + 1 \mid \mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}\colon [\mathsf{cmd}_l] \mid \mathsf{cmd}_1; \mathsf{cmd}_2 \end{aligned}$$

Fig. 1 Grammar describing our programming language

Definition 4 (Generic Encryption Mode) A generic encryption mode on n message blocks is represented by $\mathcal{M}(m_1 \| \ldots \| m_n, c_n) : \mathsf{cmd}$, where $m_1, \ldots, m_n$ are the input variables containing equal length message blocks, c_n is the output variable, and the program cmd is built using the grammar of Figure 1.

We assume that, prior to executing the encryption mode, the message has been padded using some unambiguous padding scheme, so that all the message blocks $m_1, \ldots, m_n$ are of equal and appropriate length for the scheme, usually the input length of the block cipher. We also assume that each variable in cmd is assigned a value at most once by the program (this is analogous to *static single*

assignment). Any program can be transformed into an equivalent program with this property using standard methods [CFR+91]. Finally, we note that, apart for the input variables, it would not make sense for any variable to appear on the right side of a command before appearing on the left side of a previous command, so we assume that this does not happen.

In Figure 2, we present the standard encryption modes cipher block chaining (CBC) [EMST76] and counter mode (CTR). These two modes of operation will be used as examples in this paper.

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

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

Fig. 2 Description of CBC and CTR

3.2 Semantics

The precise effect of a command is given by its *semantic function*, which describes how the command modifies distributions of configurations. A *configuration* describes all the internal information needed to run a program, and includes a state, which assigns values to the variables, and a few more sets, functions and strings, which keep track of information needed for our analysis.

More formally, a configuration is a tuple of the form $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_{\mathcal{E}}, \sigma)$ where:

- The *state* S is a function $S : \mathsf{Var} \to \{0,1\}^* \cup \bot$, where Var is the full set of variables in the program, that assigns bitstrings to variables. If x is a variable and $S(x) = s$, we call s the *value* of variable x. The symbol $\bot$ is used to denote that no value has been assigned to the variable yet.
- The set $\mathcal{T}$ contains arrays denoted by $\mathcal{T}_x$. A new array $\mathcal{T}_x$ is added to $\mathcal{T}$ whenever the result of a random sampling or a fresh query to the block cipher is assigned to the variable x, and for each integer $i \geq 0$, $\mathcal{T}_x[i]$ contains the set of variables whose values are known to be $S(x) + i$. The set $\mathcal{T}$ is used to keep track of the variables that contain values used as counters, that is, variables whose value is obtained by repeatedly adding 1 to a randomly sampled value, or to a value obtained from a new query to the block cipher. Each array $\mathcal{T}_x$ keeps track of the variables that contain values from a counter started at x.
- The set $\mathcal{Q}$ contains sets denoted by $\mathcal{Q}_x$, each associated with a corresponding array $\mathcal{T}_x \in \mathcal{T}$. Therefore, as in $\mathcal{T}$, a new set $\mathcal{Q}_x$ is added to $\mathcal{Q}$ whenever the result of a random sampling or a fresh query to the block cipher is assigned to the variable x. A set $\mathcal{Q}_x$ contains variables whose values are the result of a computation involving a variable in $\{v \in \mathsf{Var} \mid \exists i \geq 0, v \in \mathcal{T}_x[i]\}$, or a variable already in $\mathcal{Q}_x$. That is, $\mathcal{Q}_x$ contains variables that are not themselves counters,

but whose value might[3] depend on the value from a counter started at x. The set $\mathcal{Q}_x$ therefore contains all the values that are deemed "unsafe" when using x as a counter, and the use of x as a counter will be considered insecure if the block cipher is computed on any of the variables in $\mathcal{Q}_x$.
- $\mathcal{E}$ is a block cipher, and the multiset $\mathcal{L}_\mathcal{E}$ contains pairs (s, x) where s is a bitstring and x is a variable or the symbol $\perp$. A pair (s, x) for $x \in \mathsf{Var}$ is added to $\mathcal{L}_\mathcal{E}$ every time a command of the form $y := \mathcal{E}(x)$ is executed in the program, and $S(x) = s$. A pair $(s, \perp)$ is added to $\mathcal{L}_\mathcal{E}$ when the block cipher is computed in a previous encryption query on a value s. We write $\mathcal{L}_\mathcal{E}.\mathsf{dom}$ and $\mathcal{L}_\mathcal{E}.\mathsf{res}$ to denote the multisets obtained by projecting each pair in $\mathcal{L}_\mathcal{E}$ to its first or second element, respectively (we note that we could easily generalize our method for modes of operation that require more than one block cipher by adding more block ciphers $\mathcal{E}'$ and corresponding lists $\mathcal{L}_{\mathcal{E}'}$ to configurations).
- The tuple $\sigma = (\sigma_0, \ldots, \sigma_n)$ is an information-passing parameter. The bitstring σ_0 is used to store all the state information of the LoR-CPA adversary from Definition 2 when it makes an encryption query, so that the adversary may continue its execution with its previous state after the query is answered, whereas the bitstrings σ_1 to σ_n contain the values of the message blocks to be encrypted – that is, $\sigma_1 = S(m_1)$, …, $\sigma_n = S(m_n)$.

For any $x \in \mathsf{Var}$ for which there is an array $\mathcal{T}_x \in \mathcal{T}$, we denote by $Set(\mathcal{T}_x)$ the set $\{v \in \mathsf{Var} : v \in \mathcal{T}_x[i] \text{ for some integer } i\}$.

3.2.1 Initial and Constructible Distributions

While the semantic function of commands will be defined for any distribution of configurations, it is useful to first examine which distributions of configurations can occur during the course of our analysis. To define the set of initial distributions, the distributions that can occur when a LoR-CPA adversary has just issued an encryption query, we note the following:

- The LoR-CPA adversary may have already issued some encryption queries before issuing the query that is currently under study, so the multiset $\mathcal{L}_\mathcal{E}$ must already contain the value on which the block cipher has been computed to answer these queries. Those are the queries that will have the form $(s, \perp)$ in $\mathcal{L}_\mathcal{E}$.
- Since our analysis models the block cipher as a random function, the function $\mathcal{E}$ is sampled from the set of all functions from $\{0,1\}^\eta$ to $\{0,1\}^\eta$.
- When issuing the encryption query, all the state information of the LoR-CPA adversary is stored somewhere so that the adversary can be 'restarted' and given the ciphertext answering its encryption query. It is this state information that is stored in σ_0.
- For the purpose of defining the initial distributions, we assume that the bit b used in the LR function has already been sampled, and that the selection by the LR function has already been made, so that we have a single message to encrypt, whose blocks are $\sigma_1, \ldots, \sigma_n$.

[3] Note that our analysis will be very conservative, so a variable may be put in $\mathcal{Q}_x$ even if, on a closer analysis, one may discover that the value of the variable is not actually dependent on x.

- Since we assume that all the variables that are not input variables are assigned a value before appearing on the right side of a command, we can, without loss of generality, assign them all an initial value of $\perp$.
- We discard the values used as counters to answer the previous encryption queries, so the sets $\mathcal{T}$ and $\mathcal{Q}$ are initially empty.

Let $\mathcal{M}(m_1\| \ldots \|m_n, c_n) : \mathsf{cmd}$ be a generic encryption mode. Taking all the observations above into account, we define the set of *initial distributions*, denoted $\text{CDIST}_0^{\mathcal{M}}(\Gamma)$, as the set of all distributions of the form:

$$[\mathcal{E} \xleftarrow{\$} \Phi_\eta; \sigma \xleftarrow{\$} \mathcal{A}^{\mathcal{M}(\cdot)}(1^\eta) : (S_\sigma, \emptyset, \emptyset, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)]$$

where $\mathcal{A}$ is a polynomial-time algorithm with oracle access to $\mathcal{M}(\cdot)$, the encryption algorithm corresponding to the generic encryption mode $\mathcal{M}(m_1\| \ldots \|m_n, c_n) : \mathsf{cmd}$. The tuple σ must have the structure $\sigma = (\sigma_0, \ldots, \sigma_n)$ described above, and the state S_σ is the function that assigns the value σ_i to variable m_i for $1 \leq i \leq n$, and assigns $\perp$ to all other variables. All the block cipher computations that are required to answer $\mathcal{A}$'s calls to the oracle $\mathcal{M}(\cdot)$ are recorded in $\mathcal{L}_\mathcal{E}$ as pairs of the form $(s, \perp)$.

The set $\text{CDIST}^{\mathcal{M}}(\Gamma)$ of *constructible distributions* is the set of distributions $\{[\![p]\!]X : X \in \text{CDIST}_0^{\mathcal{M}}(\Gamma)$ and p is a program produced by the grammar of Figure 1$\}$. We describe how to compute the effect of each command on a distribution of configurations in the next section.

3.2.2 Natural Semantics

We first introduce a few notational shortcuts to simplify the writing of the semantic function of each command. We write the semantic functions of commands as functions $\mathsf{cmd} : \Gamma \to \text{DIST}(\Gamma)$, where Γ is the set of all configurations $\text{DIST}(\Gamma)$ is the set of all possible distributions on configurations, and it should be clear that each of these functions uniquely determines a function $\mathsf{cmd} : \text{DIST}(\Gamma) \to \text{DIST}(\Gamma)$ obtained by point-wise application (by abuse of notation, we denote both functions the same way). We write $\delta(\gamma)$ to denote the Dirac distribution, in which configuration γ has probability 1 and all other configurations have probability 0. The notation $S\{x \mapsto v\}$ 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 $S\{x \mapsto v\}(x) = v$. We denote by $\mathsf{new}(\mathcal{T}_x)$ the array that has $\mathcal{T}_x[0] = \{x\}$ and $\mathcal{T}_x[i] = \emptyset$ for any $i > 0$, and by $\mathsf{new}(\mathcal{Q}_x)$ an empty set associated with the variable x.

The effect of commands on the state is essentially what one would expect from the description of the commands. The process for updating $\mathcal{T}$ and $\mathcal{Q}$ is as explained below.

- When we execute a command $x \xleftarrow{\$} \mathcal{U}(l)$ or a command $x := \mathcal{E}(y)$ with $S(y) \notin \mathcal{L}_\mathcal{E}.dom$, a new array $\mathcal{T}_x$ and a new set $\mathcal{Q}_x$ is added to $\mathcal{T}$ and $\mathcal{Q}$ respectively. The array $\mathcal{T}_x$ has $\mathcal{T}_x[0] = \{x\}$ and, initially, $\mathcal{T}_x[i] = \emptyset$ for $i > 0$, and set $\mathcal{Q}_x$ is initially empty. This identifies the variable x as a possible starting point for a counter.
- When we execute the command $x := y + 1$ on a variable $y \in \mathcal{T}_w[i]$ for some variable w and integer i, then we add x to $\mathcal{T}_w[i + 1]$ because clearly the value of x is one more than the value of y.

$$
\begin{array}{l}
[\![x \xleftarrow{\$} \mathcal{U}(l)]\!](S,\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) = \\
\quad [u \xleftarrow{\$} \mathcal{U}(l) : (S\{x \mapsto u\},\mathcal{T} \cup \{\mathsf{new}(\mathcal{T}_x)\},\mathcal{Q} \cup \{\mathsf{new}(\mathcal{Q}_x)\},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma)] \\
\qquad \text{where } \mathcal{T}_x[0] = \{x\} \text{ and } \mathcal{T}_x[i] = \emptyset \text{ for } i > 0, \text{ and } \mathcal{Q}_x = \emptyset. \\
[\![x := y]\!](S,\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) = \\
\quad \begin{cases} \delta(S\{x \mapsto S(y)\},\mathcal{T}\{\mathcal{T}_z[i] \mapsto \mathcal{T}_z[i] \cup \{x\}\},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) \text{ if } \exists z \text{ for which } y \in \mathcal{T}_z[i], \\ \delta(S\{x \mapsto S(y)\},\mathcal{T},\mathcal{Q}_+(x,\{y\}),\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) \text{ otherwise.} \end{cases} \\
[\![x := y \oplus z]\!](S,\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) = \delta(S\{x \mapsto S(y) \oplus S(z)\},\mathcal{T},\mathcal{Q}_+(x,\{y,z\}),\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) \\
[\![x := y||z]\!](S,\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) = \delta(S\{x \mapsto S(y)||S(z)\},\mathcal{T},\mathcal{Q}_+(x,\{y,z\}),\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) \\
[\![x := y+1]\!](S,\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) = \\
\quad \begin{cases} \delta(S\{x \mapsto S(y)+1,\mathcal{T}\{\mathcal{T}_z[i+1] \mapsto \mathcal{T}_z[i+1] \cup \{x\}\},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) \text{ if } \exists z \text{ for which } y \in \mathcal{T}_z[i], \\ \delta(S\{x \mapsto S(y)+1,\mathcal{T},\mathcal{Q}_+(x,\{y\}),\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) \text{ otherwise.} \end{cases} \\
[\![\mathsf{for}\ l = p\ \mathsf{to}\ q\ \mathsf{do}\colon [c_l]]\!]\gamma = \begin{cases} [\![c_q]\!] \circ [\![c_{q-1}]\!] \circ \ldots \circ [\![c_p]\!]\gamma \text{ if } p \leq q \\ \gamma \text{ otherwise} \end{cases} \\
[\![x := \mathcal{E}(y)]\!](S,\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}},\sigma) = \\
\quad \begin{cases} \delta(S\{x \mapsto \mathcal{E}(S(y))\},\mathcal{T} \cup \{\mathsf{new}(\mathcal{T}_x)\},\mathcal{Q} \cup \{\mathsf{new}(\mathcal{Q}_x)\},\mathcal{E},\mathcal{L}_{\mathcal{E}} \cup \{(S(y),y)\},\sigma) \\ \quad \text{if } S(y) \notin \mathcal{L}_{\mathcal{E}}.\mathsf{dom}\ . \\ \delta(S\{x \mapsto \mathcal{E}(S(y))\},\mathcal{T},\mathcal{Q},\mathcal{E},\mathcal{L}_{\mathcal{E}} \cup \{(S(y),y)\},\sigma) \text{ otherwise} \end{cases} \\
[\![c_1;c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]
\end{array}
$$

Table 1 Natural semantics of the programming language

- When we execute the command $x := y$ on a variable $y \in \mathcal{T}_w[i]$ for some variable w and integer i, then we add x to $\mathcal{T}_w[i]$ because clearly x and y contain the same value.
- When we execute either the command $x := y + 1$ or $x := y$ for a variable $y \in \mathcal{Q}_w$ for some variable w, or a XOR or concatenation command in which the variable y appears on the right side of the command and either $y \in \mathcal{T}_w[i]$ for some variable w and integer i or $y \in \mathcal{Q}_w$, then the value assigned to x could be equal to that of one of the counters in $Set(\mathcal{T}_w)$, so we add x to $\mathcal{Q}_w$.
- If none of the variables on the right side of the command appear in either $Set(\mathcal{T}_w)$ or $\mathcal{Q}_w$, then the set $\mathcal{Q}_w$ is unchanged by the command.

We denote by $\mathcal{Q}_+(x,V)$ the set obtained from $\mathcal{Q}$ by adding the variable x to all the sets $\mathcal{Q}_w$ for variables w such that one of the variables in V is either in $Set(\mathcal{T}_w)$ or $\mathcal{Q}_w$, that is:

$$
\begin{aligned}
\mathcal{Q}_+(x,V) = &\{\mathcal{Q}_y \cup \{x\} : V \cap (Set(\mathcal{T}_y) \cup \mathcal{Q}_y) \neq \emptyset\} \cup \\
&\{\mathcal{Q}_y : V \cap (Set(\mathcal{T}_y) \cup \mathcal{Q}_y) = \emptyset\}.
\end{aligned}
$$

This variable is used in the semantics to update the set $\mathcal{Q}$. Typically, the variable x is the variable appearing on the left side of the command, while the set V contains the variables from the right side of the command. For example, if the command $x := y \oplus z$ is executed, the set $\mathcal{Q}$ gets updated to $\mathcal{Q}_+(x,\{y,z\})$.

The 'natural' semantic function of each command is given in Table 1.

3.2.3 Simulated Semantics

Studying the distribution of the values of the output of the block cipher becomes hard when the block cipher is computed more than once on any given input. We would like to be able to say that, since the block cipher is modeled as a function

$$[\![x := \mathcal{E}(y)]\!](S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_{\mathcal{E}}, \sigma) =$$
$$[u \xleftarrow{\$} \mathcal{U}(\eta) : (S\{x \mapsto u\}, \mathcal{T} \cup \{\mathsf{new}(\mathcal{T}_x)\}, \mathcal{Q} \cup \{\mathsf{new}(\mathcal{Q}_x)\}, \mathcal{E}, \mathcal{L}_{\mathcal{E}} \cup \{(S(y), y)\}, \sigma)]$$
where $\mathcal{T}_x[0] = \{x\}$, $\mathcal{T}_x[i] = \emptyset$ for $i > 0$, and $\mathcal{Q}_x = \emptyset$.

Table 2 Idealized semantics of the block cipher used in our analysis

sampled at random from Φ_η, whenever the block cipher is computed, the output of the function is indistinguishable from new randomly sampled values. However, this statement is clearly false when we compute the block cipher on a value on which it has been computed before, as the output of the block cipher on these values is clearly predictable. Still, as long as the block cipher is only queried on non-repeating input values, the output of the block cipher is indistinguishable from a random sampling. For this reason, we introduce a new semantics in Table 2 in which the semantic of the block cipher command is replaced precisely by the sampling of an independent random value (the semantic functions of all other commands are the same as in Table 1). As long as inputs to the block cipher do not repeat, this is indistinguishable from the first semantics because sampling the block cipher from Φ_η is the same as sampling in advance all the outputs of the block cipher. Therefore, we can use this second semantics for our analysis, provided that we also keep track of the probability that the block cipher is computed more than once on a value.

Noting that every value on which the block cipher is computed is added to $\mathcal{L}_{\mathcal{E}}$, the following allows us to determine when the block cipher has been computed on the same input more than once.

Definition 5 A configuration $\gamma = (S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_{\mathcal{E}}, \sigma)$ is *bad* if there exists two distinct elements (s, v) and (s', v') in $\mathcal{L}_{\mathcal{E}}$ with $s = s'$. For a configuration γ, the predicate $Bad(\gamma)$ is true if γ is bad, and false otherwise.

For any configuration γ, we define the function $NoBad$ as follows:

$$NoBad(\gamma) = \begin{cases} \gamma & \text{if } \gamma \text{ is not bad,} \\ \bot & \text{otherwise.} \end{cases}$$

Theorem 1 *Let $X \in \mathrm{CDIST}_0^{\mathcal{M}}(\Gamma)$ be an initial distribution and cmd be a program. Let $X_{nat} = [\![\mathsf{cmd}]\!]X$ using the "natural" semantic function of Table 1 for the block cipher command and $X_{sim} = [\![\mathsf{cmd}]\!]X$ using the "idealized" semantic function of Table 2. Then, the following holds:*

$$[\gamma \xleftarrow{\$} X_{nat} : NoBad(\gamma)] = [\gamma \xleftarrow{\$} X_{sim} : NoBad(\gamma)].$$

Proof We easily see that, as long as the block cipher is never computed on any value more than once, then the simulated semantic function simply delays the sampling of the value assigned until the command is executed, whereas the natural semantics samples the value corresponding to each domain point of the function $\mathcal{E}$ up front when sampling $\mathcal{E}$ from Φ_η. This change in the timing of the sampling clearly cannot change the distribution output of the function $NoBad$.

3.3 Assertion Language

Before we introduce the predicates that are the basis of our Hoare logic, we introduce a refinement on our definition of bad configuration that allows us to distinguish between the case when a configuration was bad before the program was executed – that is, the configuration is bad because there exists (s, v) and (s', v') in $\mathcal{L}_\mathcal{E}$ with $s = s'$ and $v = v' = \perp$ – and the case when a configuration becomes bad as a result of the execution of a program.

Definition 6 A configuration $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)$ is *locally bad* if there exists two elements (s, v) and (s', v') in $\mathcal{L}_\mathcal{E}$ with $s = s'$ and at least one of v or v' is not $\perp$. For a configuration γ, the predicate $Lbad(\gamma)$ is true if γ is locally bad, and false otherwise.

Intuitively, a configuration becomes locally bad after the execution of a command of the form $x := \mathcal{E}(y)$ such that the value of the variable y was already in $\mathcal{L}_\mathcal{E}.\mathsf{dom}$. Using this, it is easy to find that if $X \in \text{CDIST}(\Gamma, \mathcal{F})$ and p is a program generated by our grammar, then

$$\mathsf{Pr}[\gamma \xleftarrow{\$} [\![p]\!]X : Bad(\gamma)] \leq \mathsf{Pr}[\gamma \xleftarrow{\$} X : Bad(\gamma)] + \mathsf{Pr}[\gamma \xleftarrow{\$} [\![p]\!]X : Lbad(\gamma)]$$

In the following, if γ is a configuration, we denote by S_γ, $\mathcal{T}_\gamma$, $\mathcal{Q}_\gamma$, $\mathcal{E}_\gamma$, $\mathcal{L}_{\mathcal{E}\gamma}$ and σ_γ the state, sets, block cipher and information-passing parameter, associated with γ. For any set $V \subseteq \mathsf{Var}$ and configuration $\gamma = (S_\gamma, \mathcal{T}_\gamma, \mathcal{Q}_\gamma, \mathcal{E}_\gamma, \mathcal{L}_{\mathcal{E}\gamma}, \sigma_\gamma)$, we denote by $S_\gamma(V)$ the multiset resulting from the application of S on each variable in V. We also extend the domain of S_γ to include the symbol $\ell_\mathcal{E}$, a formal symbol which stands for all the values on which the block cipher has been computed; that is, $S_\gamma(V, \ell_\mathcal{E}) = S_\gamma(V) \cup \mathcal{L}_{\mathcal{E}\gamma}.\mathsf{dom}$. Finally, we use Var^* as a shorthand for $\mathsf{Var} \cup \{\ell_\mathcal{E}\}$.

Our Hoare Logic is based on statements in the following language:

$$\begin{aligned}\varphi &::= \mathsf{true} \mid \varphi \wedge \varphi \mid \psi \\ \psi &::= \mathsf{LBad}\,(\epsilon) \mid \mathsf{Indep} \mid \mathsf{Indis}(x; W) \mid \\ &\quad\ \mathsf{Lctr}(x; y; i; V; V') \mid \mathsf{Ectr}(x; y; i; V; V') \mid \mathsf{ctr}(x; y; i)\end{aligned}$$

where ϵ is a real number between 0 and 1, $x, y \in \mathsf{Var}$, $V, V' \subseteq \mathsf{Var}$, $W \subseteq \mathsf{Var}^*$ and i is a non-negative integer. We refer to the statements produced by this grammar as *formulas*.

The components of the formula, which we call *predicates*, are properties of the distribution of configurations. The first two predicates describe properties of the values on which the block cipher has been computed, and the remaining predicates are properties of the distribution of the values of variables within the distribution of configurations. Informally, our predicates can be described as follows:

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

Indep: means that each value on which the program has computed the block cipher is randomly distributed and independent from the value of the message blocks and from the values on which the block cipher was computed before the execution of the program. In more concrete terms, this means that for each element of the form $(s, v) \in \mathcal{L}_\mathcal{E}$ in which $v \neq \perp$, s is randomly distributed and independent from all the values s' in elements of the form $(s', \perp) \in \mathcal{L}_\mathcal{E}$.

$\mathsf{Indis}(x; W)$: means that the value of x is distributed uniformly, independently from the values $S(W)$ and the value of the message blocks.

$\mathsf{Lctr}(x; y; i; V; V')$: means that x contains the most recently computed value of a counter that started at a random value stored in the variable y, and that the value of x is $S(y) + i$. In addition, V contains all the variables with previous values of the counter (so $V = Set(\mathcal{T}_y)$), V' contains all the variables that are not counters, but whose value are dependent on values of the counter (so $V' = \mathcal{Q}_y$).

$\mathsf{Ectr}(x; y; i; V; V')$: means that x contains the value of a counter that started at a random value stored in the variable y, and that the value of x is $S(y) + i$, and that the block cipher has not been computed on the value of x. In addition, V contains all the variables other values of the counter (so $V = Set(\mathcal{T}_y)$) and V' contains all the variables that are not counters, but whose value are dependent on values of the counter (so $V' = \mathcal{Q}_y$).

$\mathsf{ctr}(x; y; i)$: means that x contains a counter that started with the value of y, and whose value is $S(y) + i$

The most important distinction between the predicates $\mathsf{Ectr}(x; y; i; V; V')$ and $\mathsf{Lctr}(x; y; i; V; V')$ is that $\mathsf{Lctr}(x; y; i; V; V')$ is true only for the most recently computed value of the counter (whereas $\mathsf{Ectr}(z; y; i; V; V')$ can be true of all the previous values z simultaneously), and $\mathsf{Ectr}(x; y; i; V; V')$ indicates that the block cipher has not been computed on the value of x (whereas this is not the case for $\mathsf{Lctr}(x; y; i; V; V')$). For example, if X satisfies both $\mathsf{Lctr}(x; y; i; V; V')$ and $\mathsf{Ectr}(x; y; i; V; V')$, then $[\![z := x+1]\!]X$ still satisfies $\mathsf{Ectr}(x; y; i; V; V')$ but no longer satisfies $\mathsf{Lctr}(x; y; i; V; V')$; whereas $[\![z := \mathcal{E}(x)]\!]X$ still satisfies $\mathsf{Lctr}(x; y; i; V; V')$, but no longer satisfies $\mathsf{Ectr}(x; y; i; V; V')$.

For each predicate ψ, we define that a distribution $X \in \textsc{Dist}(\Gamma)$ satisfies ψ, denoted $X \models \psi$ as follows, where $x, y \in \mathsf{Var}$, $V, V' \subseteq \mathsf{Var}$ and $W \subseteq \mathsf{Var}^*$:

- $X \models \mathsf{true}$.
- $X \models \varphi \wedge \varphi'$ iff $X \models \varphi$ and $X \models \varphi'$.
- $X \models \mathsf{LBad}\,(\epsilon)$ iff $\mathsf{Pr}[\gamma \xleftarrow{\$} X : LBad(\gamma)] \leq \epsilon$.
- $X \models \mathsf{Indep}$ iff for every variable $v \in \mathsf{Var}$ for which there exists a configuration γ with non-zero probability in X such that $(s, v) \in \mathcal{L}_{\mathcal{E}_\gamma}$ for some string s, the following holds:

$$[\gamma \xleftarrow{\$} X : (S_\gamma(v), \{s' : (s', \bot) \in \mathcal{L}_{\mathcal{E}_\gamma}\}, \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(v)) : (u, \{s' : (s', \bot) \in \mathcal{L}_{\mathcal{E}_\gamma}\}, \sigma_\gamma)]$$

- $X \models \mathsf{Indis}(x; W)$ iff the following holds:

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), S_\gamma(W - x), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, S_\gamma(W - x), \sigma_\gamma)]$$

We note that the variable x is removed from the set W in the equation above to remove the trivial case that, if $x \in W$, we can always distinguish the distribution of $(S_\gamma(x), S_\gamma(W), \sigma_\gamma)$ from $(u, S_\gamma(W), \sigma_\gamma)$ by simply looking for the value of $S_\gamma(x)$ in $S_\gamma(W)$. Removing the variable x from W simplifies the notation of the predicate, and as a result, we have that $X \models \mathsf{Indis}(x; V)$ if and only if

$X \models \mathsf{Indis}(x; V, x)$. We note also that $X \models \mathsf{Indis}(x; V)$ for any set V implies the independence of x from the message blocks since the message blocks are contained in the string σ_γ.

- $X \models \mathsf{Lctr}(x; y; i; V; V')$ iff for every configuration $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)$ that is not bad and has non-zero probability in X, $x \in \mathcal{T}_y[i]$, $\mathcal{T}_y[i+1] = \perp$, $V = Set(\mathcal{T}_y)$, $V' = \mathcal{Q}_y$, and the following holds:

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), W, \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, W, \sigma_\gamma)]$$

  where the set W is equal to $S_\gamma(\mathsf{Var} \setminus (V \cup V')) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}_\gamma} \wedge v \notin V\}$.
- $X \models \mathsf{Ectr}(x; y; i; V; V')$ iff for every configuration $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)$ that is not bad and has non-zero probability in X, $x \in \mathcal{T}_y[i]$, $V = Set(\mathcal{T}_x)$, $V' = \mathcal{Q}_y$, and the following holds:

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), W, \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, W, \sigma_\gamma)]$$

  where the set W is equal to $S_\gamma(\mathsf{Var} \setminus (V \cup V')) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}_\gamma} \wedge v \notin V \setminus \mathcal{T}_y[i]\}$.
- $X \models \mathsf{ctr}(x; y; i)$ iff for every configuration $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)$ that is not bad and has non-zero probability in X, $x \in \mathcal{T}_y[i]$.

Note that in all the predicates that involve the indistinguishability of a value of a variable from a random string (that is, Indep, $\mathsf{Indis}(x, V)$, $\mathsf{Lctr}(x; y; i; V; V')$ and $\mathsf{Ectr}(x; y; i; V; V')$), we require that the distribution of the value of the variable is *strictly* equal to the distribution obtained by replacing the value of the variable by a random value. This is different from previous works, which only required the distributions to be computationally indistinguishable. We can use this stronger definition because we are doing our analysis using the idealized semantics of Table 2, and so the probability of an adversary distinguishing the value of the variable from a random value is essentially contained in the probability that the configuration is bad. This strict equality of the distributions enables us to obtain better bounds in the security analysis.

The definition of the predicates $\mathsf{Lctr}(x; y; i; V; V')$ and $\mathsf{Ectr}(x; y; i; V; V')$ are somewhat complex, so we explain them in a few more details. The goal of the sets V and V' in the predicates is to partition the set of all variables in three sets:

1. The set V consists of the variables containing a counter with the same starting point as x (this starting point is the value contained in the variable y). We can easily determine whether or not the value contained in these variable is equal to the value of x because those variables will always be assigned a predicate $\mathsf{ctr}(z; y; i')$. If $i = i'$, then the value of z is equal to the value of x, otherwise, their values are different. This information can be used to "transfer" predicates from one variable to another if their value is the same. For example, if, say, $\mathsf{Lctr}(x; y; i; V; V')$ and $\mathsf{ctr}(z; y; i)$ are both true, then x and z contain the same value, therefore $\mathsf{Lctr}(z; y; i; V; V')$ will be true as well.
2. The set V' consists of the variables whose content might depend on the value of one of the variables in V, but whose exact values are unknown – in particular, their value could be equal to the value of one of the variables in V. These values are deemed "unsafe" when it comes to using x as a counter, and if the block cipher is queried on any of the values in V', the use of x as a counter will no longer be allowed.

3. Finally, by design, the value of x should be randomly distributed and independent from the value of the variables that are not in V or V'. Therefore, any operations on variables that are in neither V or V' will have no effect on the use of x as a counter.

We also highlight the distinctions between predicates $\mathsf{Ectr}(x; y; i; V; V')$ and $\mathsf{Lctr}(x; y; i; V; V')$. In $\mathsf{Lctr}(x; y; i; V; V')$, we require that $x \in \mathcal{T}_y[i]$ and $\mathcal{T}_y[i+1] = \perp$, whereas for $\mathsf{Ectr}(x; V; V')$, we only require that $x \in \mathcal{T}_y[i]$. In addition, $\mathsf{Lctr}(x; y; i; V; V')$ requires that x is indistinguishable from a random value when given the values of all the variables except for those in V $(= \mathcal{T}_y)$ and V' $(= \mathcal{Q}_y)$, and all the values in $\mathcal{L}_{\mathcal{E}}.\mathsf{dom}$, except for those associated with variables in V, whereas $\mathsf{Ectr}(x; V; V')$ requires that x is indistinguishable from a random values when given the values of all the variables except for those in V and V', and all the values in $\mathcal{L}_{\mathcal{E}}.\mathsf{dom}$ that are not associated with variables in $V \setminus \mathcal{T}_y[i]$. This enables us to prove that $\mathsf{Ectr}(x; V; V')$ implies that the value of x is unlikely to be in $\mathcal{L}_{\mathcal{E}}.\mathsf{dom}$.

Our method attempts to prove that when the configurations are not bad, the security of the mode of operation holds unconditionally. Therefore, the value contained in the predicate $\mathsf{LBad}(\epsilon)$ is the only value necessary for calculating the exact security bound in the analysis. This is also why the other predicates require certain conditions to hold in every configurations that are not bad.

3.4 Properties of Predicates and Useful Lemmas

We present a series of results that show the relations between our predicates. We also prove a few useful lemmas that we use repeatedly in the proofs of the rules of our Hoare logic.

We first present three results that should be self-evident, since all they state is that strong predicates imply weaker ones. We omit the proof since it is a direct consequence of the definition.

Lemma 1 *For any distribution $X \in \textsc{Dist}(\Gamma)$, any variable $x, y \in \mathsf{Var}$, any sets $V' \subseteq V \subseteq \mathsf{Var}$ and $V_0, V_1 \subseteq \mathsf{Var}$ and any non-negative integer i,*

$$X \models \mathit{Indis}(x; V) \Rightarrow X \models \mathit{Indis}(x; V')$$
$$X \models \mathit{Lctr}(x; y; i; V_0; V_1) \Rightarrow X \models \mathit{Indis}(x; \mathsf{Var} \setminus (V_0 \cup V_1))$$
$$X \models \mathit{Ectr}(x; y; i; V_0; V_1) \Rightarrow X \models \mathit{Indis}(x; \mathsf{Var} \setminus (V_0 \cup V_1))$$

Proof The first statement is obvious, the other two follow from the observation that the set $S(\mathsf{Var} \setminus \{V_0 \cup V_1\})$ is a subset of the set W in the definition of the predicates $\mathsf{Lctr}(x; y; i; V_0; V_1)$ and $\mathsf{Ectr}(x; y; i; V_0; V_1)$. □

The following lemma enables us to infer predicates on counters when multiple variables contain the same counter value.

Lemma 2 *For any distribution $X \in \mathit{Dist}(\Gamma)$, any variables $x, y, z \in \mathsf{Var}$, any non-negative integer i and any sets $V, V' \subseteq \mathsf{Var}$,*

$$X \models \mathit{Lctr}(x; y; i; V; V') \Rightarrow X \models \mathit{ctr}(x; y; i)$$
$$X \models \mathit{Ectr}(x; y; i; V; V') \Rightarrow X \models \mathit{ctr}(x; y; i)$$
$$X \models \mathit{Lctr}(x; z; i; V; V') \wedge \mathit{ctr}(y; z; i) \Rightarrow X \models \mathit{Lctr}(y; z; i; V; V')$$
$$X \models \mathit{Ectr}(x; z; i; V; V') \wedge \mathit{ctr}(y; z; i) \Rightarrow X \models \mathit{Ectr}(y; z; i; V; V')$$

Proof The first two statements are straightforward consequences of the definition.

Suppose $X \models \mathsf{Lctr}(x; z; i; V; V') \wedge \mathsf{ctr}(y; z; i)$. Therefore, $y \in \mathcal{T}_z[i]$ and $\mathcal{T}_z[i+1] = \bot$, $V = Set(\mathcal{T}_y)$ and $V' = \mathcal{Q}_y$. Then

$$[\gamma \xleftarrow{\$} X : (S_\gamma(y), W, \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)) : (u, W, \sigma_\gamma)]$$

where the set W is equal to $S_\gamma(\mathsf{Var} \setminus (V \cup V')) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$, trivially follows from $X \models \mathsf{Lctr}(x; z; i; V; V')$ and the fact that $\{x, y\} \subseteq \mathcal{T}_z[i]$, which trivially implies that the values of x and y are the same, and equal to the value of z plus i.

The proof of the last statement is done similarly. □

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 $X \in \textsc{Dist}(\Gamma)$, any variable $x \in \mathit{Var}$ and any set $V \subseteq \mathit{Var}$, if $X \models \mathit{Indis}(x; V)$, then*

$$\mathit{Pr}[\gamma \xleftarrow{\$} X : S_\gamma(x) \in S_\gamma(V - x)] \leq \frac{|S_\gamma(V - x)|}{2^{\mathit{len}(x)}}.$$

Proof Let $X \models \mathsf{Indis}(x; V)$. So we have that

$$\begin{aligned}\mathsf{Pr}[\gamma \xleftarrow{\$} X : S_\gamma(x) \in S_\gamma(V - x)] \leq & \mathsf{Pr}[\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : u \in S_\gamma(V - x)] \\ \leq & \frac{|S_\gamma(V - x)|}{2^{\mathsf{len}(x)}}\end{aligned}$$

as required. □

As a consequence of Lemma 3, if $X \models \mathsf{Indis}(x; V, \ell_\mathcal{E})$ – note the very important presence of the symbol $\ell_\mathcal{E}$, which means that the value of x has a uniform random distribution independent from all the values in $\mathcal{L}_\mathcal{E}.\mathsf{dom}$ – then the probability that the value of x is either in $V - x$ or in $\mathcal{L}_\mathcal{E}.\mathsf{dom}$ is bounded by $\frac{|S(V-x)|+|\mathcal{L}_\mathcal{E}.\mathsf{dom}|}{2^{\mathsf{len}(x)}}$. So by combining Lemma 3 with Lemma 1, we obtain the following:

Corollary 1 *For any distribution $X \in \textsc{Dist}(\Gamma)$, any variables $x \in \mathit{Var}$ and any set $V \subseteq \mathit{Var}$,*

$$X \models \mathit{Indis}(x; V, \ell_\mathcal{E}) \Rightarrow \mathit{Pr}[\gamma \xleftarrow{\$} X : S_\gamma(x) \in \mathcal{L}_{\mathcal{E}\gamma}.\mathit{dom}] \leq \frac{|\mathcal{L}_{\mathcal{E}\gamma}|}{2^{\mathit{len}(x)}}$$

Proof We have from Lemma 1 that $X \models \mathsf{Indis}(x; V, \ell_\mathcal{E})$ implies $X \models \mathsf{Indis}(x; \{\ell_\mathcal{E}\})$. The result follows by applying Lemma 3 on that last predicate. □

Using a similar reasoning, we also obtain the following:

Corollary 2 *For any distribution $X \in \textsc{Dist}(\Gamma)$, any variables $x, y \in \mathit{Var}$ any sets $V, V' \subseteq \mathit{Var}$, and any non-negative integer i,*

$$X \models \mathit{Ectr}(x; y; i; V; V') \Rightarrow \mathit{Pr}[\gamma \xleftarrow{\$} X : S_\gamma(x) \in \mathcal{L}_{\mathcal{E}\gamma}.\mathit{dom}] \leq \frac{|\mathcal{L}_{\mathcal{E}\gamma}|}{2^{\mathit{len}(x)}}$$

□

In the following lemma, we show that for all the distributions in $\text{CDIST}^{\mathcal{M}}(\Gamma)$, the predicate $\mathsf{Indis}(x; V)$ always implies independence from the values contained in the (input) message variables.

Lemma 4 *For any distribution $X \xleftarrow{\$} \text{CDIST}^{\mathcal{M}}(\Gamma)$, any variable x and set $V \subseteq \mathit{Var}^*$, if $X \models \mathit{Indis}(x; V)$, then $X \models \mathit{Indis}(x; V \cup \{m_1, \ldots, m_n\})$.*

Proof We know, by definition of $\text{CDIST}_0^{\mathcal{M}}(\Gamma)$ that for any $X_0 \xleftarrow{\$} \text{CDIST}_0^{\mathcal{M}}(\Gamma)$, for any configuration $\gamma \xleftarrow{\$} X_0$, $\sigma_\gamma = (\sigma_0, S_\gamma(m_1), \ldots, S_\gamma(m_n))$ for some bitstring σ_0. Using the fact that commands never modify σ, we easily obtain, using a simple induction, that for any $X \xleftarrow{\$} \text{CDIST}^{\mathcal{M}}(\Gamma)$, for any configuration $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma) \xleftarrow{\$} X$, $\sigma = (\sigma_0, S(m_1), \ldots, S(m_n))$ for some bitstring σ_0. Therefore, the value of σ uniquely determines the value of the message blocks.

Let $X \xleftarrow{\$} \text{CDIST}^{\mathcal{M}}(\Gamma)$ be such that $X \models \mathsf{Indis}(x; V)$, and let V_M be the set $V \cup \{m_1, \ldots, m_n\}$ Therefore, since $X \models \mathsf{Indis}(x; V)$, by definition

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), S_\gamma(V - x), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, S_\gamma(V - x), \sigma_\gamma)]$$

The following is then immediate since the value of σ_γ uniquely determines the value of the message blocks

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), S_\gamma(V_M - x), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, S_\gamma(V_M - x), \sigma_\gamma)]$$

Hence $X \models \mathsf{Indis}(x; V_M)$, as required. □

Finally, we show which predicates hold at the beginning of the program's execution.

Lemma 5 *If $X \in \text{CDIST}_0(\Gamma)$, then $X \models \mathit{LBad}(0) \wedge \mathit{Indep}$.*

Proof Since $X \in \text{CDIST}_0(\Gamma)$, then, for any configuration γ that has non-zero probability in X, all the elements of $\mathcal{L}_{\mathcal{E}\gamma}$ are of the form $(s, \perp)$ for some string s, because only the execution of a program can add an element of the form (s, v) with $v \in \mathsf{Var}$ to $\mathcal{L}_\mathcal{E}$. Therefore, all configurations γ that have non-zero probability in X are not locally bad, and the criterion for Indep is vacuously satisfied. Therefore $X \models \mathsf{LBad}(0) \wedge \mathsf{Indep}$. □

4 Semantic Security in Predicates

In this section, we prove the following theorem, which states that if the formula $\mathsf{Indis}(c_n; \emptyset) \wedge \mathsf{LBad}(\epsilon(q_\mathbf{E}, q_\mathcal{E})) \wedge \mathsf{Indep}$ holds at the end of the execution of the program of the mode of operation for some negligible function $\epsilon(q_\mathbf{E}, q_\mathcal{E})$, then the mode of operation is semantically secure.

Theorem 2 *Let $\mathcal{M}(m_1|\dots|m_n, c_n)$: cmd be a generic encryption mode. If for every distribution $X \in \text{CDIST}_0^{\mathcal{M}}(\Gamma)$, $[\![\mathsf{cmd}]\!]X \models \mathsf{Indis}(c_n; \emptyset) \wedge \mathsf{LBad}(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}$, where $q_{\mathbf{E}}$ is the number of calls made to the* LR *encryption oracle by the algorithm that created the distribution X and $q_{\mathcal{E}}$ is the number of computations of the block cipher made to answer those oracle queries, then, under the assumption that the block cipher is a random function in Φ_η, for any* LoR-CPA *adversary $\mathcal{B}$, the following holds:*

$$\mathsf{Adv}_{\mathcal{M}}^{\mathsf{LoR\text{-}CPA}}(\mathcal{B}) \leq \sum_{i=1}^{Q} \epsilon(i, q_{\mathcal{E}}^{(i)})$$

where Q is an upper bound on the number of queries made by $\mathcal{B}$ to the LR *encryption oracle, 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 $\mathcal{B}$.*

Proof Our strategy for proving this theorem is to show that if answering the LR encryption queried never causes the block cipher to be computed on the same value twice, then the adversary is unable to gain any significant advantage in the LoR-CPA game. This is done using a hybrid argument that progressively replaces the answer to each query to the LR oracle by random bits. We can then bound the adversary's advantage by bounding the probability that the block cipher is queried on the same value twice using the bound inside the predicate LBad.

Let $\mathcal{B}$ be an LoR-CPA adversary. We have that

$$\mathsf{Adv}_{\mathcal{M}}^{\mathsf{LoR\text{-}CPA}}(\mathcal{B}, \eta) = \left|\mathsf{Pr}[\mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-1}(\mathcal{B}, \eta) = 1] - \mathsf{Pr}[\mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-0}(\mathcal{B}, \eta) = 1]\right|$$

where $\mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-b}(\mathcal{B}, \eta)$ is the experiment described in Definition 2. For the sake of conciseness, in the following, we abbreviate $\mathbf{Exp}_{\mathcal{S}}^{\mathsf{LoR\text{-}CPA}-b}(\mathcal{B}, \eta)$ with $\mathbf{Exp}^b(\mathcal{B}, \eta)$. The reason for the superscript will soon become clear. Let bad be the event that the block cipher has to be computed more than once on an input value to answer one of $\mathcal{B}$'s oracle encryption queries during the execution of $\mathbf{Exp}^b(\mathcal{B}, \eta)$. The event has the same probability regardless of the bit b since, from $[\![\mathsf{cmd}]\!]X \models \mathsf{Indep}$, the distribution of the calls to the block cipher is independent from the message encrypted. So, we have that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{M}}^{\mathsf{LoR\text{-}CPA}}(\mathcal{B}, \eta) &= \left|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1]\right| \\
&= \Big|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] + \mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \mathsf{bad}] \\
&\quad - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \mathsf{bad}]\Big| \\
&\leq \left|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right| \\
&\quad + \left|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \mathsf{bad}] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \mathsf{bad}]\right| \\
&\leq \left|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right| \\
&\quad + \mathsf{Pr}[\mathsf{bad}]
\end{aligned}$$

The first inequality is an application of the triangle inequality, and the second is a consequence of the fact that the value of both $\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \mathsf{bad}]$ and $\mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \mathsf{bad}]$ is between zero and $\mathsf{Pr}[\mathsf{bad}]$.

We prove the result by showing that the quantity in absolute value in the last inequality is equal to zero, and $\mathsf{Pr}[\mathsf{bad}]$ is equal to the sum in the statement of the Theorem.

To do this, we first need to define algorithms $\mathcal{C}_i^b$ and $\mathcal{D}_i^b$ for $0 \leq i \leq Q-1$.

$\mathcal{C}_i^b$: is an algorithm that has oracle access to the encryption algorithm $\mathcal{M}$ and outputs a tuple $\sigma = (\sigma_0, \ldots, \sigma_n)$. On input 1^η, it runs algorithm $\mathcal{B}$ on input 1^η until it makes its i^{th} oracle encryption query. $\mathcal{C}_i^b$ answers $\mathcal{B}$'s $i-1$ first encryption oracle queries $(M_{j,0}, M_{j,1})$ by sending $\mathsf{LR}(M_{j,0}, M_{j,1}, b)$ to its encryption oracle and relaying the answer to $\mathcal{B}$. When $\mathcal{B}$ makes its i^{th} oracle encryption query $(M_{j,0}, M_{j,1})$, $\mathcal{C}_i^b$ splits $M_b = \mathsf{LR}(M_{i,0}, M_{i,1}, b)$ into blocks of length η and outputs $(\sigma_0, \ldots, \sigma_n)$, where σ_1 to σ_n are the blocks of M_b, and σ_0 is all the state information of algorithm $\mathcal{B}$.

$\mathcal{D}_i$: is an algorithm that, on input (c, σ), restarts algorithm $\mathcal{B}$ just after its i^{th} oracle encryption query using state information σ_0 and c as the answer to the oracle query. Algorithm $\mathcal{D}_i$ then answers all of $\mathcal{B}$'s remaining oracle encryption queries with random strings of the appropriate length.

For $0 \leq i \leq Q-1$ and $b \in \{0,1\}$, we define $\textsc{Dist}_i^b$ as the following distribution:

$$[\mathcal{E} \xleftarrow{\$} \Phi_\eta; \sigma \xleftarrow{\$} \mathcal{C}_i^{b\,\mathcal{M}(\cdot)}(1^\eta) : (S_\sigma, \emptyset, \emptyset, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)].$$

Clearly, all the $\textsc{Dist}_i^b$'s are in $\mathrm{CDist}_0(\Gamma)$. We define the experiments $\mathbf{E}^{(i,b)}(\mathcal{B}, \eta)$ as follows:

$$\begin{array}{l}\text{Experiment } \mathbf{E}^{(i,b)}(\mathcal{B}, \eta):\\ \quad \gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}_i^b\\ \quad b' \xleftarrow{\$} \mathcal{D}_i(S_\gamma(c_n), \sigma_\gamma)\\ \quad \text{return } b'\end{array}$$

So by definition, $\mathbf{E}^{(Q-1,b)}(\mathcal{B}, \eta)$ is exactly the same as $\mathbf{Exp}_\mathcal{S}^{\mathsf{LoR\text{-}CPA}-b}(\mathcal{B}, \eta)$, the experiment of the $\mathsf{LoR\text{-}CPA}$ security game, and that as i gets smaller, $\mathbf{E}^{(i,b)}(\mathcal{B}, \eta)$ progressively answers more and more of $\mathcal{B}$'s LoR encryption queries with random bits, starting from the end.

The result follows from the two following claims.

Claim 1:

$$\mathsf{Pr}[\mathsf{bad}] \leq \sum_{i=0}^{Q-1} \epsilon(i, q_\mathcal{E}^{(i)})$$

Proof It is clear that bad occurs exactly when a configuration in $[\![\mathsf{cmd}]\!]\textsc{Dist}_{Q-1}^b$ is bad. We have seen in Section 3.3 that

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}_{Q-1}^b : Bad(\gamma)]\\ &\qquad \leq \mathsf{Pr}[\gamma \xleftarrow{\$} \textsc{Dist}_{Q-1}^b : Bad(\gamma)] + \mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}_{Q-1}^b : Lbad(\gamma)]\\ &\qquad \leq \mathsf{Pr}[\gamma \xleftarrow{\$} \textsc{Dist}_{Q-1}^b : Bad(\gamma)] + \epsilon(Q-1, q_\mathcal{E}^{(Q-1)})\end{aligned}$$

The last inequality above is a direct consequence of the fact that $[\![\mathsf{cmd}]\!]\textsc{Dist}_{Q-1}^b \models \mathsf{LBad}\left(\epsilon(Q-1, q_\mathcal{E}^{(Q-1)})\right)$, which follows directly from the hypothesis in the statement of the Theorem. Then, the probability that a configuration in $\textsc{Dist}_{Q-1}^b$ is bad

is precisely the probability that a configuration in $[\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-2}$ is bad, therefore:

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-1} : Bad(\gamma)]\\ &\qquad \leq \mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-2} : Bad(\gamma)] + \epsilon(Q-1, q_{\mathcal{E}}^{(Q-1)})\end{aligned}$$

We then repeat the reasoning above on $\mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-2} : Bad(\gamma)]$ and combine it with the above to get that:

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-1} : Bad(\gamma)]\\ &\quad \leq \mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-3} : Bad(\gamma)] + \epsilon(Q-2, q_{\mathcal{E}}^{(Q-2)}) + \epsilon(Q-1, q_{\mathcal{E}}^{(Q-1)})\end{aligned}$$

Repeating this all the way down to $\textsc{Dist}^b_0$, we obtain that:

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]\textsc{Dist}^b_{Q-1} : Bad(\gamma)]\\ &\qquad \leq \mathsf{Pr}[\gamma \xleftarrow{\$} \textsc{Dist}^b_0 : Bad(\gamma)] + \sum_{i=0}^{Q-1} \epsilon(i, q_{\mathcal{E}}^{(i)})\\ &\qquad \leq \sum_{i=0}^{Q-1} \epsilon(i, q_{\mathcal{E}}^{(i)})\end{aligned}$$

because in all configurations of $\textsc{Dist}^b_0$, $\mathcal{L}_{\mathcal{E}}$ is empty and therefore the probability that a configuration is bad is zero. □

Claim 2:

$$\left|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right| = 0$$

Proof We know that

$$\begin{aligned}&\left|\mathsf{Pr}[\mathbf{Exp}^1(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{Exp}^0(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right| =\\ &\qquad \left|\mathsf{Pr}[\mathbf{E}^{(Q-1,1)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{E}^{(Q-1,0)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right|\end{aligned}$$

since the experiments are the same. Our plan is to show that for each i, $1 \leq i \leq Q-1$, and $b \in \{0,1\}$,

$$\left|\mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{E}^{(i-1,b)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right| = 0$$

As a result, we obtain that

$$\begin{aligned}&\left|\mathsf{Pr}[\mathbf{E}^{(Q-1,1)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{E}^{(Q-1,0)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right|\\ &\qquad = \left|\mathsf{Pr}[\mathbf{E}^{(0,1)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{E}^{(0,0)}(\mathcal{B}, \eta) = 1 \wedge \neg\mathsf{bad}]\right| = 0\end{aligned}$$

Since experiment $\mathbf{E}^{0,1}(\mathcal{B}, \eta)$ is indistinguishable from experiment $\mathbf{E}^{0,0}(\mathcal{B}, \eta)$ as in both cases, the LoR encryption queries are answered with random bits independent from the query's input.

So we only have left to prove that for each i, $1 \leq i \leq Q-1$, and $b \in \{0,1\}$,

$$\left|\mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}] - \mathsf{Pr}[\mathbf{E}^{(i-1,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}]\right| = 0$$

We note that the only difference between experiment $\mathbf{E}^{(i,b)}(\mathcal{B},\eta)$ and experiment $\mathbf{E}^{(i-1,b)}(\mathcal{B},\eta)$ is that in the former, the i^{th} LoR encryption query is answered using the encryption mode, whereas in the latter, it is answered with random bits.

Let $\mathsf{bad}_{[1,i]}$ be the event that the block cipher is computed on a given value more than once while answering one of the first i LoR queries, and let $\mathsf{bad}_{[i+1,Q]}$ be the event that that the block cipher is queried on a previous value while answering the LoR queries $i+1$ to Q. It is clear that $\mathsf{bad} = \mathsf{bad}_{[1,i]} \vee \mathsf{bad}_{[i+1,Q]}$. Also, we get from $[\![\mathsf{cmd}]\!]X \models \mathsf{Indep}$ that $\mathsf{bad}_{[i+1,Q]}$ is independent from $\mathsf{bad}_{[1,i]}$, and it is also clearly independent from $\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1$ and $\mathbf{E}^{(i-1,b)}(\mathcal{B},\eta) = 1$ since in these experiments, the queries $i+1$ to Q are answered with random bits rather than using the encryption mode. Therefore,

$$\begin{aligned}\mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}] &= \mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}_{[1,i]} \wedge \neg\mathsf{bad}_{[i+1,Q]}] \\ &= \mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}_{[1,i]}] \cdot \mathsf{Pr}[\neg\mathsf{bad}_{[i+1,Q]}],\end{aligned}$$

and similarly for $\mathsf{Pr}[\mathbf{E}^{(i-1,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}]$. So it suffices to prove that

$$\left|\mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}_{[1,i]}] - \mathsf{Pr}[\mathbf{E}^{(i-1,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}_{[1,i]}]\right| = 0.$$

We note that the event $\neg\mathsf{bad}_{[1,i]}$ is exactly the same as the event that the configuration sampled in $[\![\mathsf{cmd}]\!]\textsc{Dist}_i^b$ is not bad. However, since $\textsc{Dist}_i^b \in \textsc{CDist}_0$, the statement of the Theorem tells us that $[\![\mathsf{cmd}]\!]\textsc{Dist}_i^b \models \mathsf{Indis}(c_n;\emptyset)$. Thus, the distribution of the value of $(S_\gamma(c_n),\sigma_\gamma)$ in $[\![\mathsf{cmd}]\!]\textsc{Dist}_i^b$ given that the configuration γ is not bad is the same as the distribution of (u,σ_γ) for a random string u of the same length as $S_\gamma(c_n)$. That is, the distribution of the answer to the i^{th} query in $\mathbf{E}^{(i,b)}$ is the same as that of a randomly selected value of the same length, exactly the same as in $\mathbf{E}^{(i-1,b)}$. Hence

$$\left|\mathsf{Pr}[\mathbf{E}^{(i,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}_{[1,i]}] - \mathsf{Pr}[\mathbf{E}^{(i-1,b)}(\mathcal{B},\eta) = 1 \wedge \neg\mathsf{bad}_{[1,i]}]\right| = 0.$$

which concludes the proof. □

This completes the proof of the theorem. □

5 Hoare Logic

Hoare logic rules have the form $\{\varphi\}\mathsf{cmd}\{\varphi'\}$, and mean 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 first present the rules grouped together according to their corresponding commands in Section 5.1, then we prove the soundness of all the rules in Section 5.2.

5.1 Logic Rules

In all the rules, unless stated otherwise, we assume that, in a predicate $\mathsf{Indis}(t; V)$, the set V is a subset of $\mathsf{Var} \cup \{\ell_{\mathcal{E}}\}$, and in a predicate $\mathsf{Lctr}(t; w; i; V; V')$ or $\mathsf{Ectr}(t; w; i; V; V')$, the sets V and V' are subsets of Var and the variables t, u and w are not (syntactically) equal to x, y or z.

5.1.1 General rules

First, we recall a few general rules for consequence, sequential composition and conjunction. Let $\phi_1, \phi_2, \phi_3, \phi_4$ be any four formulas 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 $\{\phi_1 \wedge \phi_3\}\mathsf{cmd}\{\phi_2 \wedge \phi_4\}$

We present these rules without proof because they are well known.

5.1.2 Generic preservation rules:

The generic preservation rules show how predicates are preserved by most commands when the variables contained in the predicates are distinct from the variables contained in the command. We have generic preservation rules for all commands except for $x := \mathcal{E}(y)$, which requires special treatment because the function $\mathcal{E}$ is unknown to the adversary, because that command is the only command that modifies $\mathcal{L}_{\mathcal{E}}$, and because it can have a more complicated effect on some of the predicates.

Before we introduce the preservation rules, we first need to define the concept of constructible expression. Generally, we say that an expression e is *constructible from* V if e has polynomial size and it can be constructed by combining variables in V using polynomial-time operations. Since our Hoare logic treats commands individually, we are only interested in expressions that consist of the right side of a command.

Definition 7 We say that a variable x is *constructible from* V if one of the following holds:

- the variable x is assigned a value by a command of the form $x \xleftarrow{\$} \mathcal{U}(l)$,
- the variable x is assigned a value by a command of the form $x := y$ or $x := y+1$, where $y \in V$,
- the variable x is assigned a value by a command of the form $x := y \oplus z$ or $x := y \| z$, where $y, z \in V$.

This concept is useful in our preservation rules because if, say, the value of a variable t is randomly distributed and independent from variables V, then it seems clear the value of t will also be randomly distributed and independent from any expression that is constructible from V.

Let cmd be either $x \xleftarrow{\$} \mathcal{U}$, $x := y$, $x := y \| z$, $x := y \oplus z$ or $x := y + 1$. Then the the preservation rules are as follows:

(G1) $\{\mathsf{LBad}(\epsilon)\}$ cmd $\{\mathsf{LBad}(\epsilon)\}$
(G2) $\{\mathsf{Indep}\}$ cmd $\{\mathsf{Indep}\}$
(G3) $\{\mathsf{Indis}(t;V)\}$ cmd $\{\mathsf{Indis}(t;V)\}$ if $x \notin V$ unless x is constructible from $V - t$, even if $t = y$ or $t = z$
(G4) $\{\mathsf{Lctr}(t;w;i;V;V')\}$ cmd $\{\mathsf{Lctr}(t;w;i;V;V')\}$ if $y, z \notin V \cup V'$
(G5) $\{\mathsf{Lctr}(t;w;i;V;V')\}$ cmd $\{\mathsf{Lctr}(t;w;i;V;V',x)\}$ if [cmd is not $x := y$ nor $x := y + 1$, and $y \in V \cup V'$ or $z \in V \cup V'$, even if $t = y$ or $t = z$] or [cmd is $x := y$ or $x := y + 1$, and $y \in V'$]
(G6) $\{\mathsf{Ectr}(t;w;i;V;V')\}$ cmd $\{\mathsf{Ectr}(t;w;i;V;V')\}$ if $y, z \notin V \cup V'$
(G7) $\{\mathsf{Ectr}(t;w;i;V;V')\}$ cmd $\{\mathsf{Ectr}(t;w;i;V;V',x)\}$ if [cmd is not $x := y+1$ or $x := y$, and $y \in V \cup V'$ or $z \in V \cup V'$, even if $t = y$ or $t = z$] or [cmd is $x := y + 1$ and $y \notin V$]
(G8) $\{\mathsf{ctr}(t;w;i)\}$ cmd $\{\mathsf{ctr}(t;w;i)\}$ even if $\{t, w\} \cap \{y, z\} \neq \emptyset$ or if cmd is $y := \mathcal{E}(y)$

5.1.3 Random sampling:

We have only one rule for the random sampling, which states that if a variable is assigned a value sampled at random, then that value can be used as a counter and it is, quite naturally, indistinguishable from a random value.

(R1) $\{true\}\ x \xleftarrow{\$} \mathcal{U}\ \{\mathsf{Indis}(x;\mathsf{Var}^*) \wedge \mathsf{Lctr}(x;x;0;\{x\};\emptyset) \wedge \mathsf{Ectr}(x;x;0;\{x\};\emptyset)\}$

5.1.4 Assignment:

The rules for assignment are mostly trivial consequences of the fact that after executing $x := y$, the variable x ends up containing the same value as the variable y. We note however that the predicates $\mathsf{Lctr}(t;w;i;V;V')$ and $\mathsf{Ectr}(t;w;i;V;V')$ must be updated when $y \in V$ to account for the fact that x is now also a counter variable containing the same value as y.

(A1) $\{\mathsf{Indis}(y;V)\}\ x := y\ \{\mathsf{Indis}(x;V)\}$ provided $y \notin V$
(A2) $\{\mathsf{ctr}(y;w;i)\}\ x := y\ \{\mathsf{ctr}(x;w;i)\}$ even if $y = w$
(A3) $\{\mathsf{Lctr}(t;w;i;V;V')\}\ x := y\ \{\mathsf{Lctr}(t;w;i;V,x;V')\}$ if $y \in V$, even if $t = y$ or $w = y$
(A4) $\{\mathsf{Ectr}(t;w;i;V;V')\}\ x := y\ \{\mathsf{Ectr}(t;w;i;V,x;V')\}$ if $y \in V$, even if $t = y$ or $w = y$

5.1.5 XOR operator:

The rule for the exclusive-or is reminiscent of the application of a one-time-pad encryption of z using y as a key: if z is XOR-ed with a random value independent from z, then the result is indistinguishable from a random value provided that the value of y is not given. For this rules, it should be clear that the roles of y and z can be reversed since the exclusive-or operation is commutative.

(X1) $\{\mathsf{Indis}(y;V,z)\}\ x := y \oplus z\ \{\mathsf{Indis}(x;V,z)\}$ if $y \neq z$ and $x, y \notin V$

5.1.6 Concatenation:

The rule for concatenation states that the concatenation of two strings that are indistinguishable from random and independent from each other is also indistinguishable from random.

(C1) $\{\mathsf{Indis}(y;V,z) \wedge \mathsf{Indis}(z;V,y)\}\ x := y \| z\ \{\mathsf{Indis}(x;V)\}$ if $y, z \notin V$ and $y \neq z$

5.1.7 Block cipher:

We have ten rules for the computation of the block cipher. The first two rules update the probability that a configuration is locally bad according to the probability that the value of y is in $\mathcal{L}_{\mathcal{E}}.\mathsf{dom}$. The next two preserve the predicate about the independence of the values on which the block cipher is computed from previous values. The fifth states that, after the execution of $x := \mathcal{E}(y)$, the distribution of the value of x is uniform random and independent from all other values, which is immediate from the ("simulated") semantics of the command. Rules (B6) to (B11) are preservations rules.

(B1) $\{\mathsf{Indis}(y;\{\ell_{\mathcal{E}}\}) \wedge \mathsf{LBad}\,(\epsilon)\}\ x := \mathcal{E}(y)\ \{\mathsf{LBad}\left(\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^n}\right)\}$

(B2) $\{\mathsf{Ectr}(y;w;i;V;V') \wedge \mathsf{LBad}\,(\epsilon)\}\ x := \mathcal{E}(y)\ \{\mathsf{LBad}\left(\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^n}\right)\}$

(B3) $\{\mathsf{Indis}(y;\{\ell_{\mathcal{E}}\}) \wedge \mathsf{Indep}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indep}\}$

(B4) $\{\mathsf{Ectr}(y;w;i;V;V') \wedge \mathsf{Indep}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indep}\}$

(B5) $\{\mathsf{true}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(x;\mathsf{Var}^*) \wedge \mathsf{Lctr}(x;x;0;\{x\};\emptyset) \wedge \mathsf{Ectr}(x;x;0;\{x\};\emptyset)\}$

(B6) $\{\mathsf{Indis}(t;V)\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t;V,x)\}$ even if $t = y$

(B7) $\{\mathsf{Indis}(t;V,y,\ell_{\mathcal{E}})\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t;V,x,y,\ell_{\mathcal{E}})\}$ (here, $t \neq y$)

(B8) $\{\mathsf{Lctr}(t;w;i;V;V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Lctr}(t;w;i;V;V')\}$ provided $y \notin V'$, even if $t = y$

(B9) $\{\mathsf{Lctr}(t;w;i;V;V') \wedge \mathsf{Indis}(t;\mathsf{Var}^* \setminus W)\}\ x := \mathcal{E}(y)\ \{\mathsf{Lctr}(t;w;i;V;V')\}$ where $W = (V \cup V') \setminus \{y\}$, provided $t \neq y$

(B10) $\{\mathsf{Ectr}(t;w;i;V'',V''') \wedge \mathsf{Ectr}(y;u;j;V;V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Ectr}(t;w;i;V'';V''')\}$ provided $[w = u$ and $i \neq j]$ or $[w \neq u]$

(B11) $\{\mathsf{Ectr}(t;w;i;V;V') \wedge \mathsf{Indis}(t;\mathsf{Var}^* \setminus W)\}\ x := \mathcal{E}(y)\ \{\mathsf{Ectr}(t;w;i;V;V')\}$ where $W = (V \cup V') \setminus \{y\}$, provided $t \neq y$

5.1.8 Increment:

We have five rules for the increment operation. The first rule states that if the value being incremented was uniformly distributed, then so is the incremented value. The following two rules update the counter predicates. The last two rules are preservation rules that cover cases that were not yet covered by rules (G4) to (G7).

(I1) $\{\mathsf{Indis}(y;V)\}\ x := y+1\ \{\mathsf{Indis}(x;V)\}$ if $y \notin V$
(I2) $\{\mathsf{Lctr}(y;z;i;V;V')\}\ x := y+1\ \{\mathsf{Lctr}(x;z;i+1;V,x;V') \wedge \mathsf{Ectr}(x;z;i+1;V,x;V')\}$
(I3) $\{\mathsf{ctr}(y;z;i)\}\ x := y+1\ \{\mathsf{ctr}(x;z;i+1)\}$
(I4) $\{\mathsf{Lctr}(t;w;i;V;V') \wedge \mathsf{ctr}(y,w,j)\}\ x := y+1\ \{\mathsf{Lctr}(t;w;i;V,x;V')\}$ if $j < i$
(I5) $\{\mathsf{Ectr}(t;w;i;V;V')\}\ x := y+1\ \{\mathsf{Ectr}(t;w;i;V,x;V')\}$ if $y \in V$, even if $t = y$

5.1.9 For loop:

We have only one rule for the For loop, which states that if there is a formula $\psi(k)$ such that whenever $\psi(k-1)$ holds before the execution of iteration k in the loop, $\psi(k)$ will hold at the end of the k^{th} iteration, if the loop runs from index i to index j, and that $\psi(i-1)$ holds before the execution of the loop, then $\psi(j)$ will hold at the end.

(F1) $\{\psi(p-1)\}$ for $k = p$ to q do: $[c_k]\ \{\psi(q)\}$ provided $\{\psi(k-1)\}\ c_k\ \{\psi(k)\}$ for $p \le k \le q$

Finding an invariant $\psi(i)$ such that the rule above can be applied can be difficult. We discuss heuristics to discover this invariant in Section 7.1.

5.2 Proof of our Rules

Our full set of rules is summarized in Table 3. We now prove the soundness of all our rules.

5.2.1 Generic preservation rules:

Before we state the preservation rules, we prove two lemmas that will be used in the proof of many of the rules. First, the following shows that the indistinguishability from random values is preserved by a command when the variable that is assigned a value by the command is not present in any of the sets under consideration.

Lemma 6 *Let $x, t \in \mathsf{Var}$, $V \subseteq (\mathsf{Var}^* - x)$ and $X \in \mathrm{CDIST}(\Gamma)$ be such that*

$$[\gamma \xleftarrow{\$} X : (S_\gamma(t), S_\gamma(V-t), \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u, S_\gamma(V-t), \sigma_\gamma)].$$

In addition, let cmd be either $x \xleftarrow{\$} \mathcal{U}$, $x := y$, $x := y \| z$, $x := y \oplus z$ or $x := y+1$, with $x \neq t$. Then,

$$[\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]X : (S_\gamma(t), S_\gamma(V-t), \sigma_\gamma)] = [\gamma \xleftarrow{\$} [\![cmd]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u, S_\gamma(V-t), \sigma_\gamma)].$$

NOTE: Unless stated otherwise, the variables t, u and v are not (syntactically) equal to x, y or z.

Generic Preservation

cmd is either $x \xleftarrow{\$} \mathcal{U}$, $x := y$, $x := y \| z$, $x := y \oplus z$ or $x := y + 1$

(G1) $\{\mathsf{LBad}\,(\epsilon)\}\ \mathsf{cmd}\ \{\mathsf{LBad}\,(\epsilon)\}$

(G2) $\{\mathsf{Indep}\}\ \mathsf{cmd}\ \{\mathsf{Indep}\}$

(G3) $\{\mathsf{Indis}(t; V)\}\ \mathsf{cmd}\ \{\mathsf{Indis}(t; V)\}$ if $x \notin V$ unless x is constructible from $V - t$, even if $t = y$ or $t = z$

(G4) $\{\mathsf{Lctr}(t; w; i; V; V')\}\ \mathsf{cmd}\ \{\mathsf{Lctr}(t; w; i; V; V')\}$ if $y, z \notin V \cup V'$

(G5) $\{\mathsf{Lctr}(t; w; i; V; V')\}\ \mathsf{cmd}\ \{\mathsf{Lctr}(t; w; i; V; V', x)\}$ if $\lceil \mathsf{cmd}$ is not $x := y$ nor $x := y+1$, and $y \in V \cup V'$ or $z \in V \cup V'$, even if $t = y$ or $t = z\rceil$ or $\lceil \mathsf{cmd}$ is $x := y$ or $x := y+1$, and $y \in V'\rceil$

(G6) $\{\mathsf{Ectr}(t; w; i; V; V')\}\ \mathsf{cmd}\ \{\mathsf{Ectr}(t; w; i; V; V')\}$ if $y, z \notin V \cup V'$

(G7) $\{\mathsf{Ectr}(t; w; i; V; V')\}\ \mathsf{cmd}\ \{\mathsf{Ectr}(t; w; i; V; V', x)\}$ if $\lceil \mathsf{cmd}$ is not $x := y$ or $x := y+1$, and $y \in V \cup V'$ or $z \in V \cup V'$, even if $t = y$ or $t = z\rceil$ or $\lceil \mathsf{cmd}$ is $x := y$ or $x := y+1$ and $y \in V'\rceil$

(G8) $\{\mathsf{ctr}(t; w; i)\}\ \mathsf{cmd}\ \{\mathsf{ctr}(t; w; i)\}$ even if $\{t, w\} \cap \{y, z\} \neq \emptyset$ or if cmd is $y := \mathcal{E}(y)$

Random Assignment

(R1) $\{true\}\ x \xleftarrow{\$} \mathcal{U}\ \{\mathsf{Indis}(x; \mathsf{Var}^*) \wedge \mathsf{Lctr}(x; x; 0; \{x\}; \emptyset) \wedge \mathsf{Ectr}(x; x; 0; \{x\}; \emptyset)\}$

Assignment

(A1) $\{\mathsf{Indis}(y; V)\}\ x := y\ \{\mathsf{Indis}(x; V)\}$ provided $y \notin V$

(A2) $\{\mathsf{ctr}(y; w; i)\}\ x := y\ \{\mathsf{ctr}(x; w; i)\}$ even if $y = w$

(A3) $\{\mathsf{Lctr}(t; w; i; V; V')\}\ x := y\ \{\mathsf{Lctr}(t; w; i; V, x; V')\}$ if $y \in V$ even if $t = y$ or $w = y$

(A4) $\{\mathsf{Ectr}(t; w; i; V; V')\}\ x := y\ \{\mathsf{Ectr}(t; w; i; V, x; V')\}$ if $y \in V$, even if $t = y$ or $w = y$

XOR Operation

(X1) $\{\mathsf{Indis}(y; V, z)\}\ x := y \oplus z\ \{\mathsf{Indis}(x; V, z)\}$ if $y \neq z$ and $x, y \notin V$
and similarly with the roles and y and z reversed.

Concatenation

(C1) $\{\mathsf{Indis}(y; V, z) \wedge \mathsf{Indis}(z; V, y)\}\ x := y \| z\ \{\mathsf{Indis}(x; V)\}$ if $y, z \notin V$ and $y \neq z$

Block Cipher

(B1) $\{\mathsf{Indis}(y; \{\ell_{\mathcal{E}}\}) \wedge \mathsf{LBad}\,(\epsilon)\}\ x := \mathcal{E}(y)\ \{\mathsf{LBad}\left(\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^{\eta}}\right)\}$

(B2) $\{\mathsf{Ectr}(y; w; i; V; V') \wedge \mathsf{LBad}\,(\epsilon)\}\ x := \mathcal{E}(y)\ \{\mathsf{LBad}\left(\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^{\eta}}\right)\}$

(B3) $\{\mathsf{Indis}(y; \{\ell_{\mathcal{E}}\}) \wedge \mathsf{Indep}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indep}\}$

(B4) $\{\mathsf{Ectr}(y; w; i; V; V') \wedge \mathsf{Indep}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indep}\}$

(B5) $\{\mathsf{true}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(x; \mathsf{Var}^*) \wedge \mathsf{Lctr}(x; x; 0; \{x\}; \emptyset) \wedge \mathsf{Ectr}(x; x; 0; \{x\}; \emptyset)\}$

(B6) $\{\mathsf{Indis}(t; V)\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t; V, x)\}$ even if $t = y$, provided $\ell_{\mathcal{E}} \notin V$

(B7) $\{\mathsf{Indis}(t; V, y, \ell_{\mathcal{E}})\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t; V, x, y, \ell_{\mathcal{E}})\}$ (here, $t \neq y$)

(B8) $\{\mathsf{Lctr}(t; w; i; V; V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Lctr}(t; w; i; V; V')\}$ provided $y \notin V'$, even if $t = y$

(B9) $\{\mathsf{Lctr}(t; w; i; V; V') \wedge \mathsf{Indis}(t; \mathsf{Var}^* \setminus W)\}\ x := \mathcal{E}(y)\ \{\mathsf{Lctr}(t; w; i; V; V')\}$ where $W = (V \cup V') \setminus \{y\}$, provided $t \neq y$

(B10) $\{\mathsf{Ectr}(t; w; i; V'', V''') \wedge \mathsf{Ectr}(y; u; j; V; V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Ectr}(t; w; i; V''; V''')\}$ provided $\lceil w = u$ and $i \neq j\rceil$ or $\lceil w \neq u\rceil$

(B11) $\{\mathsf{Ectr}(t; w; i; V; V') \wedge \mathsf{Indis}(t; \mathsf{Var}^* \setminus W)\}\ x := \mathcal{E}(y)\ \{\mathsf{Ectr}(t; w; i; V; V')\}$ where $W = (V \cup V') \setminus \{y\}$, provided $t \neq y$

Increment

(I1) $\{\mathsf{Indis}(y; V)\}\ x := y + 1\ \{\mathsf{Indis}(x; V)\}$ if $y \notin V$

(I2) $\{\mathsf{Lctr}(y; z; i; V; V')\}\ x := y + 1\ \{\mathsf{Lctr}(x; z; i + 1; V, x; V') \wedge$
$\mathsf{Ectr}(x; z; i + 1; V, x; V')\}$

(I3) $\{\mathsf{ctr}(y; z; i)\}\ x := y + 1\ \{\mathsf{ctr}(x; z; i + 1)\}$

(I4) $\{\mathsf{Lctr}(t; w; i; V; V') \wedge \mathsf{ctr}(y, w, j)\}\ x := y + 1\ \{\mathsf{Lctr}(t; w; i; V, x; V')\}$ if $j < i$

(I5) $\{\mathsf{Ectr}(t; w; i; V; V')\}\ x := y + 1\ \{\mathsf{Ectr}(t; w; i; V, x; V')\}$ if $y \in V$, even if $t = y$

For Loop

(F1) $\{\psi(p - 1)\}$ for $k = p$ to q do: $[c_k]\ \{\psi(q)\}$ provided $\{\psi(k - 1)\}\ c_k\ \{\psi(k)\}$ for $p \leq k \leq q$

Table 3 Hoare logic rules

Proof Since the value of t and the value of the variables in V are unchanged by the command, we easily find that

$$[\gamma \xleftarrow{\$} X : (S_\gamma(t), S_\gamma(V-t), \sigma_\gamma)] = [\gamma \xleftarrow{\$} [\![\mathsf{cmd}]\!]X : (S_\gamma(t), S_\gamma(V-t), \sigma_\gamma)]$$

and

$$[\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u, S_\gamma(V-t), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} [\![cmd]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u, S_\gamma(V-t), \sigma_\gamma)].$$

The result then follows trivially from these two equalities. □

Next, we show that if a value of a variable t is indistinguishable from a random value when given the values of variable in the set V, then adding to V variables whose values is constructible from $V - t$ does not help in distinguishing t from a random value. The following lemma shows the result for single commands, but it should be clear that this can be generalized to any poly-time computable value using a simple structural induction.

Lemma 7 *For any distribution $X \in \textsc{Dist}(\Gamma)$, if $X \models \mathit{Indis}(t; V)$, cmd is either $x \xleftarrow{\$} \mathcal{U}(l)$, $x := y$, $x := y + 1$, $x := y \oplus z$ or $x := y \| z$ and x is constructible from $V - t$, then $[\![\mathit{cmd}]\!](X) \models \mathit{Indis}(t; V, x)$.*

Proof Let $X \models \mathsf{Indis}(t; V)$. We first prove the result when the command is $x \xleftarrow{\$} \mathcal{U}(l)$, and then we can prove the result for all other commands together.

When the command is $x \xleftarrow{\$} \mathcal{U}(l)$ for some l, then, following the semantics, we have that

$$\begin{aligned}
&[\gamma \xleftarrow{\$} [\![x \xleftarrow{\$} \mathcal{U}(l)]\!]X : (S_\gamma(t), S_\gamma(V, x-t), \sigma_\gamma)] \\
&\quad = [\gamma \xleftarrow{\$} X; u_0 \xleftarrow{\$} \mathcal{U}(l) : (S_\gamma(t), S_\gamma(V-t) \cup \{u_0\}, \sigma_\gamma)] \\
&\quad = [\gamma \xleftarrow{\$} X; u_0 \xleftarrow{\$} \mathcal{U}(l); u_1 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u_1, S_\gamma(V-t) \cup \{u_0\}, \sigma_\gamma)] \\
&\quad = [\gamma \xleftarrow{\$} [\![x \xleftarrow{\$} \mathcal{U}(l)]\!]X; u_1 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u_1, S_\gamma(V, x-t), \sigma_\gamma)],
\end{aligned}$$

as required.

The proofs for the cases when the command is either $x := y$, $x := y + 1$, $x := y \oplus z$ or $x := y \| z$ are very similar so we prove the result for $x := y \oplus z$ and leave the others to the reader. Since x is constructible from $V - t$ (and therefore $y, z \in V - t$), we get that the values of the variables in $V - t$ uniquely determine the value of x. Therefore, following the semantic function for $x := y \oplus z$, for any possible value ν_x for x, ν_σ for σ and set of values ν_{V-t} for $V - t$, which contains a value ν_y for y and ν_z for z, we have

$$\begin{aligned}
&\mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(t) = \nu_t) \wedge (S_\gamma(V-t) = \nu_{V-t}) \wedge (\sigma_\gamma = \nu_\sigma)] \\
&\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : \\
&\qquad (S_\gamma(t) = \nu_t) \wedge (S_\gamma(V-t) = \nu_{V-t}) \wedge (S_\gamma(x) = \nu_y \oplus \nu_z) \wedge (\sigma_\gamma = \nu_\sigma)].
\end{aligned}$$

And we can get the following similarly:

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u = \nu_t) \wedge (S_\gamma(V - t) = \nu_{V-t}) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : \\ &\qquad (u = \nu_t) \wedge (S_\gamma(V - t) = \nu_{V-t}) \wedge (S_\gamma(x) = \nu_y \oplus \nu_z) \wedge (\sigma_\gamma = \nu_\sigma)].\end{aligned}$$

Combining these two with the fact that $X \models \mathsf{Indis}(t; V)$, that is,

$$\begin{aligned}&[\gamma \xleftarrow{\$} X : (S_\gamma(t), S_\gamma(V - t), \sigma_\gamma)] = \\ &\qquad [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u, S_\gamma(V - t), \sigma_\gamma)]\end{aligned}$$

we immediately obtain the following

$$\begin{aligned}&[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : (S_\gamma(t), S_\gamma(V - t) \cup \{S(x)\}, \sigma_\gamma)] = \\ &\qquad [\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(t)) : (u, S_\gamma(V - t) \cup \{S(x)\}, \sigma_\gamma)]\end{aligned}$$

which by definition, means $[\![x := y \oplus z]\!]X \models \mathsf{Indis}(t; V, x)$. □

Lemma 8 *Let* cmd *be either* $x \xleftarrow{\$} \mathcal{U}$, $x := y$, $x := y \| z$, $x := y \oplus z$ *or* $x := y + 1$. *Then the following rules are sound:*

- *(G1)* $\{\mathsf{LBad}\,(\epsilon)\}$ cmd $\{\mathsf{LBad}\,(\epsilon)\}$
- *(G2)* $\{\mathsf{Indep}\}$ cmd $\{\mathsf{Indep}\}$
- *(G3)* $\{\mathsf{Indis}(t; V)\}$ cmd $\{\mathsf{Indis}(t; V)\}$ *if* $x \notin V$ *unless* x *is constructible from* $V - t$, *even if* $t = y$ *or* $t = z$
- *(G4)* $\{\mathsf{Lctr}(t; w; i; V; V')\}$ cmd $\{\mathsf{Lctr}(t; w; i; V; V')\}$ *if* $y, z \notin V \cup V'$
- *(G5)* $\{\mathsf{Lctr}(t; w; i; V; V')\}$ cmd $\{\mathsf{Lctr}(t; w; i; V; V', x)\}$ *if* ⌈cmd *is not* $x := y$ *nor* $x := y + 1$, *and* $y \in V \cup V'$ *or* $z \in V \cup V'$, *even if* $t = y$ *or* $t = z$⌋ *or* ⌈cmd *is* $x := y$ *or* $x := y + 1$, *and* $y \in V'$⌋
- *(G6)* $\{\mathsf{Ectr}(t; w; i; V; V')\}$ cmd $\{\mathsf{Ectr}(t; w; i; V; V')\}$ *if* $y, z \notin V \cup V'$
- *(G7)* $\{\mathsf{Ectr}(t; w; i; V; V')\}$ cmd $\{\mathsf{Ectr}(t; w; i; V; V', x)\}$ *if* ⌈cmd *is not* $x := y+1$ *or* $x := y$, *and* $y \in V \cup V'$ *or* $z \in V \cup V'$, *even if* $t = y$ *or* $t = z$⌋ *or* ⌈cmd *is* $x := y + 1$ *and* $y \notin V$⌋
- *(G8)* $\{\mathsf{ctr}(t; w; i)\}$ cmd $\{\mathsf{ctr}(t; w; i)\}$ *even if* $\{t, w\} \cap \{y, z\} \neq \emptyset$ *or if* cmd *is* $y := \mathcal{E}(y)$

Proof

(G1) Since cmd does not alter $\mathcal{L}_\mathcal{E}$, and that $\mathsf{LBad}\,(\epsilon)$ is a property of $\mathcal{L}_\mathcal{E}$ alone, the execution of cmd does not change the probability that a configuration is locally bad.

(G2) Since cmd does not alter $\mathcal{L}_\mathcal{E}$, and that Indep is a property of $\mathcal{L}_\mathcal{E}$ alone, the execution of cmd does not change whether the predicate holds.

(G3) If $x \notin V$, the result follows directly from Lemma 6. If $x \in V$ and x is constructible from $V - t$, we note that x being constructible from $V - t$ precludes the cases, when applicable, that $t = y$ or $t = z$, and so we obtain the result by combining the above with Lemma 7.

(G4) Let $X \models \mathsf{Lctr}(t; w; i; V; V')$. First, we find from the semantics that since $t \notin \{x, y, z\}$ and $y, z \notin V \cup V' = \mathcal{TQ}(w)$, the command will not modify either the table $\mathcal{T}_w$ or the set $\mathcal{Q}_w$, so we only have left to verify that t is indistinguishable from a random value when given the values of all the variables in $\mathsf{Var} \setminus (V \cup V')$ and the first coordinates of all the values in $\mathcal{L}_\mathcal{E}$ whose second coordinate is not in V, which follows from Lemma 6.

(G5) We note that when cmd is not $x := y$ nor $x := y+1$, then the variable x gets added to $\mathcal{Q}_w$ if and only if one of the variables appearing on the right side of the command is in $\mathcal{TQ}(w) = V \cup V'$, whereas when cmd is $x := y$ or $x := y+1$, the variable x gets added to $\mathcal{Q}_w$ if and only if $y \in \mathcal{Q}_w$. These are precisely the conditions listed in the rule. Otherwise, the proof proceeds exactly as in (G4)

(G6),(G7) The proof are essentially the same as for (G4) and (G5) respectively, except that t has to be indistinguishable from a random value when given the values of all the variables in $\mathsf{Var} \setminus (V \cup V')$ and the first coordinates of all the values in $\mathcal{L}_\mathcal{E}$ whose second coordinate is not in $V - x$. This follows from Lemma 7.

(G8) This is trivial since once a variable gets added to $\mathcal{T}_w[i]$, it is never removed.

□

5.2.2 Random sampling:

Lemma 9 *The following rule is sound:*

(R1) $\{true\}\ x \xleftarrow{\$} \mathcal{U}\ \{\mathit{Indis}(x; \mathsf{Var}^*) \wedge \mathit{Lctr}(x; x; 0; \{x\}; \emptyset) \wedge \mathit{Ectr}(x; x; 0; \{x\}; \emptyset)\}$

Proof Let X be any distribution. The fact that $[\![x \xleftarrow{\$} \mathcal{U}]\!]X \models \mathsf{Indis}(x; \mathsf{Var}^*)$ is immediate, and we can see from the semantics that the execution of the command will create a new table $\mathcal{T}_x$ containing only x, and the indistinguishability from randomness required for $\mathsf{Lctr}(x; x; 0; \{x\}; \emptyset)$ and $\mathsf{Ectr}(x; x; 0; \{x\}; \emptyset)$ are implied by $\mathsf{Indis}(x; \mathsf{Var}^*)$, therefore, $[\![x \xleftarrow{\$} \mathcal{U}]\!]X \models \mathsf{Lctr}(x; x; 0; \{x\}; \emptyset)$ and $[\![x \xleftarrow{\$} \mathcal{U}]\!]X \models \mathsf{Ectr}(x; x; 0; \{x\}; \emptyset)$. □

5.2.3 Assignment:

Lemma 10 *The following rule is sound:*

(A1) $\{\mathit{Indis}(y; V)\}\ x := y\ \{\mathit{Indis}(x; V)\}$ *provided* $y \notin V$

(A2) $\{\mathit{ctr}(y; w; i)\}\ x := y\ \{\mathit{ctr}(x; w; i)\}$ *even if* $y = w$

(A3) $\{\mathit{Lctr}(t; w; i; V; V')\}\ x := y\ \{\mathit{Lctr}(t; w; i; V, x; V')\}$ *if* $y \in V$, *even if* $t = y$ *or* $w = y$

(A4) $\{\mathit{Ectr}(t; w; i; V; V')\}\ x := y\ \{\mathit{Ectr}(t; w; i; V, x; V')\}$ *if* $y \in V$, *even if* $t = y$ *or* $w = y$

Proof The first two rules are trivial consequences of the semantics, particularly the fact that the value of x is the same as the value of y. The last two are simple consequences of the fact that if, for any configuration $(S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)$ we have $y \in \mathcal{T}_w[j]$ for some j, then we have that $x \in \mathcal{T}'_w[j]$ for any configuration $(S', \mathcal{T}', \mathcal{Q}', \mathcal{E}', \mathcal{L}_\mathcal{E}', \sigma') \xleftarrow{\$} [\![x := y]\!](S, \mathcal{T}, \mathcal{Q}, \mathcal{E}, \mathcal{L}_\mathcal{E}, \sigma)$. □

5.2.4 XOR operator:

Lemma 11 *The following rule is sound:*
(X1) $\{\mathit{Indis}(y;V,z)\}\ x := y \oplus z\ \{\mathit{Indis}(x;V,z)\}$ *if* $y \neq z$ *and* $x, y \notin V$

Proof Define $V' = V \cup \{z\}$, and let $X \models \mathsf{Indis}(y;V',y)$ with $y \neq z$, $y \notin V'$, so $V' - y = V'$ and therefore

$$[\gamma \xleftarrow{\$} X : (S_\gamma(y), S_\gamma(V'), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)) : (u, S_\gamma(V'), \sigma_\gamma)].$$

So for any fixed value ν for y, ν_σ for σ and set of values $\nu_{V'}$ for V' that occurs with non-zero probability in X, we have that

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) = \nu) \wedge \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \cdot \\ &\qquad \mathsf{Pr}[\gamma \xleftarrow{\$} X : \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)].\end{aligned}$$

Now, from $X \models \mathsf{Indis}(y;V',y)$, we easily get that

$$\mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] = \frac{1}{2^{\mathsf{len}(y)}},$$

and, since conditioning on $S(V') = \nu_{V'}$ fixes the value of the variable z (because $z \in V'$), we also get

$$\mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) \oplus S_\gamma(z) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] = \frac{1}{2^{\mathsf{len}(y)}}.$$

Since the semantic function of the command $x := y \oplus z$ does not change the value of σ or that of any variable in V', we find that

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} X : \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)],\end{aligned}$$

and by definition of the semantic function of the command $x := y \oplus z$, we get

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) \oplus S(z) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : (S_\gamma(x) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)]\end{aligned}$$

Combining all the above, we get

$$\begin{aligned}&\mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) = \nu) \wedge \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} X : (S_\gamma(y) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \cdot \\ &\qquad \mathsf{Pr}[\gamma \xleftarrow{\$} X : \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : (S_\gamma(x) = \nu) \mid \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \cdot \\ &\qquad \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)] \\ &\quad = \mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : (S_\gamma(x) = \nu) \wedge \big(S_\gamma(V') = \nu_{V'}\big) \wedge (\sigma_\gamma = \nu_\sigma)]\end{aligned}$$

It therefore follows that

$$[\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X : (S_\gamma(x), S_\gamma(V'), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} [\![x := y \oplus z]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, S_\gamma(V'), \sigma_\gamma)],$$

and so $[\![x := y \oplus z]\!]X \models \mathsf{Indis}(x; V, x, z)$ as required. □

5.2.5 Concatenation:

Lemma 12 *The following rule is sound:*
(C1) $\{\mathsf{Indis}(y; V, z) \wedge \mathsf{Indis}(z; V, y)\}\ x := y\|z\ \{\mathsf{Indis}(x; V)\}$ *if* $y, z \notin V$ *and* $y \neq z$

Proof Let X be a distribution such that $X \models \mathsf{Indis}(y; V, z) \wedge \mathsf{Indis}(z; V, y)$ with $y, z \notin V$. From $X \models \mathsf{Indis}(y; V, z)$ and $y \neq z$, we get that

$$[\gamma \xleftarrow{\$} X : (S_\gamma(y)\|S_\gamma(z), S_\gamma(V), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)) : (u\|S_\gamma(z), S_\gamma(V), \sigma_\gamma)].$$

This is obtained simply by rearranging the information in the definition of $X \models \mathsf{Indis}(y; V, z)$. Similarly, from $X \models \mathsf{Indis}(z; V, y)$, we get that $X \models \mathsf{Indis}(z; V)$ from Lemma 1, so that

$$[\gamma \xleftarrow{\$} X; u_1 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)) : (u_1\|S_\gamma(z), S_\gamma(V), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} X; u_1 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)); u_2 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(z)) : (u_1\|u_2, S_\gamma(V), \sigma_\gamma)].$$

Since the concatenation of two independent, randomly sampled string has the same distribution as a single randomly sampled string of the same length, and since $x := y\|z$ does not change the value of the variables in V, we get

$$[\gamma \xleftarrow{\$} X; u_1 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)); u_2 \xleftarrow{\$} \mathcal{U}(\mathsf{len}(z)) : (u_1\|u_2, S_\gamma(V), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} [\![x := y\|z]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y) + \mathsf{len}(z)) : (u, S_\gamma(V), \sigma_\gamma)].$$

Combining all these, we obtain that

$$[\gamma \xleftarrow{\$} [\![x := y\|z]\!]X : (S_\gamma(x), S_\gamma(V), \sigma_\gamma)] = \\ [\gamma \xleftarrow{\$} [\![x := y\|z]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y) + \mathsf{len}(z)) : (u, S_\gamma(V), \sigma_\gamma)],$$

which means that $[\![x := y\|z]\!]X \models \mathsf{Indis}(x; V)$, as required. □

5.2.6 Block cipher:

Lemma 13 *The following rules are sound:*

(B1) $\{\mathsf{Indis}(y;\{\ell_{\mathcal{E}}\}) \wedge \mathsf{LBad}(\epsilon)\}\ x := \mathcal{E}(y)\ \{\mathsf{LBad}\left(\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^{\eta}}\right)\}$

(B2) $\{\mathsf{Ectr}(y;w;i;V;V') \wedge \mathsf{LBad}(\epsilon)\}\ x := \mathcal{E}(y)\ \{\mathsf{LBad}\left(\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^{\eta}}\right)\}$

(B3) $\{\mathsf{Indis}(y;\{\ell_{\mathcal{E}}\}) \wedge \mathsf{Indep}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indep}\}$

(B4) $\{\mathsf{Ectr}(y;w;i;V;V') \wedge \mathsf{Indep}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indep}\}$

(B5) $\{\mathsf{true}\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(x;\mathsf{Var}^*) \wedge \mathsf{Lctr}(x;x;0;\{x\};\emptyset) \wedge \mathsf{Ectr}(x;x;0;\{x\};\emptyset)\}$

(B6) $\{\mathsf{Indis}(t;V)\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t;V,x)\}$ *even if* $t = y$

(B7) $\{\mathsf{Indis}(t;V,y,\ell_{\mathcal{E}})\}\ x := \mathcal{E}(y)\ \{\mathsf{Indis}(t;V,x,y,\ell_{\mathcal{E}})\}$ *(here,* $t \neq y$*)*

(B8) $\{\mathsf{Lctr}(t;w;i;V;V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Lctr}(t;w;i;V;V')\}$ *provided* $y \notin V'$*, even if* $t = y$

(B9) $\{\mathsf{Lctr}(t;w;i;V;V') \wedge \mathsf{Indis}(t;\mathsf{Var}^* \setminus W)\}\ x := \mathcal{E}(y)\ \{\mathsf{Lctr}(t;w;i;V;V')\}$ *where* $W = (V \cup V') \setminus \{y\}$*, provided* $t \neq y$

(B10) $\{\mathsf{Ectr}(t;w;i;V'',V''') \wedge \mathsf{Ectr}(y;u;j;V;V')\}\ x := \mathcal{E}(y)\ \{\mathsf{Ectr}(t;w;i;V'';V''')\}$ *provided [*$w = u$ *and* $i \neq j$*] or [*$w \neq u$*]*

(B11) $\{\mathsf{Ectr}(t;w;i;V;V') \wedge \mathsf{Indis}(t;\mathsf{Var}^* \setminus W)\}\ x := \mathcal{E}(y)\ \{\mathsf{Ectr}(t;w;i;V;V')\}$ *where* $W = (V \cup V') \setminus \{y\}$*, provided* $t \neq y$

Proof

(B1) Let $X \models \mathsf{Indis}(y;\{\ell_{\mathcal{E}}\}) \wedge \mathsf{LBad}(\epsilon)$. Since the command $x := \mathcal{E}(y)$ has the effect of adding the pair $(S(y), y)$ to $\mathcal{L}_{\mathcal{E}}$, if a configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}_{\mathcal{E}}) \leftarrow X$ was not locally bad, but becomes locally bad as a result of the execution of $x := \mathcal{E}(y)$, then it must be the case that $S(y) \in \mathcal{L}_{\mathcal{E}}.\mathsf{dom}$. Therefore, the probability $\mathsf{Pr}[\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X : LBad(\gamma)]$ is no more than $\mathsf{Pr}[\gamma \xleftarrow{\$} X : LBad(\gamma)] + \mathsf{Pr}[\gamma \xleftarrow{\$} X : S(y) \in \mathcal{L}_{\mathcal{E}}.\mathsf{dom}]$, which, as we saw in Corollary 1, is precisely $\epsilon + \frac{|\mathcal{L}_{\mathcal{E}}|}{2^{\eta}}$ since the block cipher is a function from $\{0,1\}^{\eta}$ to $\{0,1\}^{\eta}$

(B2) The proof is similar to the proof of (B1), the only difference is that we need Corollary 2 instead of Corollary 1.

(B3) Let $X \models \mathsf{Indis}(y;\{\ell_{\mathcal{E}}\}) \wedge \mathsf{Indep}$. Since $X \models \mathsf{Indep}$, and the command $x := \mathcal{E}(y)$ only adds $(S(y), y)$ to $\mathcal{L}_{\mathcal{E}}$, to obtain $[\![x := \mathcal{E}(y)]\!]X \models \mathsf{Indep}$, we only need to prove the following:

$$[\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X : (S_{\gamma}(y), \{s' : (s', \bot) \in \mathcal{L}_{\mathcal{E}\gamma}\}, \sigma_{\gamma}))] =$$
$$[\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)) :$$
$$(u, \{s' : (s', \bot) \in \mathcal{L}_{\mathcal{E}\gamma}\}, \sigma_{g}amma)]$$

which is clearly implied by $X \models \mathsf{Indis}(y;\{\ell_{\mathcal{E}}\})$, and the fact that the command $x := \mathcal{E}(y)$ does not modify the set $\{s' : (s', \bot) \in \mathcal{L}_{\mathcal{E}\gamma}\}$.

(B4) This is proven the same way as rule (B3).

(B5) Since, in our modified semantics, the value assigned to the variable x is sampled uniformly at random, $[\![x := \mathcal{E}(y)]\!] \models \mathsf{Indis}(x;\mathsf{Var}^*)$ is immediate and, as in the proof of rule (R1), we can see from the semantics that the execution of the command will create a new table $\mathcal{T}_x$ containing only x, and the indistinguishability from randomness required for $\mathsf{Lctr}(x;x;0;\{x\};\emptyset)$ and $\mathsf{Ectr}(x;x;0;\{x\};\emptyset)$

are implied by $\mathsf{Indis}(x; \mathsf{Var}^*)$, therefore, $[\![x \xleftarrow{\$} \mathcal{U}]\!]X \models \mathsf{Lctr}(x; x; 0; \{x\}; \emptyset)$ and $[\![x \xleftarrow{\$} \mathcal{U}]\!]X \models \mathsf{Ectr}(x; x; 0; \{x\}; \emptyset)$. □

.

(B6) Suppose $x \notin V$. Since $x, \ell_{\mathcal{E}} \notin V$, then the values in V are unchanged by the application of the command, so we get that $X \models \mathsf{Indis}(t; V)$ implies $[\![x := \mathcal{E}(y)]\!]X \models \mathsf{Indis}(t; V)$ from Lemma 6. We obtain $[\![x := \mathcal{E}(y)]\!]X \models \mathsf{Indis}(t; V, x)$ using an argument similar to Lemma 7 and the fact that the value of x is sampled uniformly at random.

(B7) The proof of this rule is similar to the proof of rule (B6), we only need the additional observation that, for $\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X$, $S_\gamma(y) \cup \mathcal{L}_{\mathcal{E}\gamma}.\mathsf{dom} = \mathcal{L}_{\mathcal{E}\gamma}.\mathsf{dom}$ since $(S_\gamma(y), y)$ was added to $\mathcal{L}_{\mathcal{E}\gamma}$ as a result of the execution of the command.

(B8) Let $X \models \mathsf{Lctr}(t; w; i; V; V')$, so for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}_{\mathcal{E}})$ that has non-zero probability in X, $\mathcal{T}_w[i] = t$ and $\mathcal{T}_w[i+1] = \perp$, $V = Set(\mathcal{T}_x)$, $V' = \mathcal{Q}_w$, and the following holds:

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), W, \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, W, \sigma_\gamma)]$$

where the set W is equal to $S_\gamma(\mathsf{Var} \setminus (V \cup V')) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$.
Since the the command $x := \mathcal{E}(y)$ has no effect on either $\mathcal{T}_w$ or $\mathcal{Q}_w$, we will clearly have that for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}_{\mathcal{E}})$ that is not bad and has non-zero probability in $[\![x := \mathcal{E}(y)]\!]X$, $t \in \mathcal{T}_w[i]$ and $\mathcal{T}_w[i+1] = \emptyset$, $V = Set(\mathcal{T}_w)$, $V' = \mathcal{Q}_w$ for the same i and w as in the corresponding configuration in X.
Also, since $y \notin V'$, then either $y \in V$, $t = y$ or $y \in \mathsf{Var} \setminus (V \cup V' \cup \{t\})$. We first note that $t = y$ implies that $y \in \mathcal{T}_w[i]$ and so $y \in V$. If $y \in V$, then the variable y is excluded in both $\mathsf{Var} \setminus (V \cup V')$ and the value of y is excluded from $\{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$, and therefore the composition of $W = S_\gamma(\mathsf{Var} \setminus (V \cup V' \cup \{t\})) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$ is not altered by the command. If $y \in \mathsf{Var} \setminus (V \cup V' \cup \{t\})$, then the value of the variable y is in $W = S_\gamma(\mathsf{Var} \setminus (V \cup V' \cup \{t\})) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$ both before and after the execution of the command. In any of these cases, for $\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X$, the composition of $W = S_\gamma(\mathsf{Var} \setminus (V \cup V' \cup \{t\})) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$ is unchanged compared to what it was for the corresponding configuration in X, therefore fact that the following holds:

$$[\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X : (S_\gamma(x), W, \sigma_\gamma)] = [\gamma \xleftarrow{\$} [\![x := \mathcal{E}(y)]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, W, \sigma_\gamma)]$$

is an immediate consequence from the fact that it did in X.
Hence, $[\![x := \mathcal{E}(y)]\!]X \models \mathsf{Lctr}(t; w; i; V; V')$.

(B9) Let $X \models \{\mathsf{Lctr}(t; w; i; V; V') \wedge \mathsf{Indis}(t; \mathsf{Var}^* \setminus W)$ where $W = (V \cup V') \setminus \{y\}$, provided $t \neq y$. Similarly as in the proof of rule (B8), since the the command $x := \mathcal{E}(y)$ has no effect on either $\mathcal{T}_w$ or $\mathcal{Q}_w$, we will clearly have that for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}_{\mathcal{E}})$ that is not bad and has non-zero probability in $[\![x := \mathcal{E}(y)]\!]X$, $t \in \mathcal{T}_w[i]$ and $\mathcal{T}_w[i+1] = \emptyset$, $V = Set(\mathcal{T}_w)$, $V' = \mathcal{Q}_w$ for the same i and w as in the corresponding configuration in X.

Now, from the proof of rule (B7), since $X \models \mathsf{Indis}(t; \mathsf{Var}^* \setminus W)$ and $y, \ell_{\mathcal{E}} \in \mathsf{Var}^* \setminus W$, we have that $[\![x := \mathcal{E}(y)]\!]X \models \mathsf{Indis}(t; \mathsf{Var}^* \setminus W)$. It then follows from the first line of Lemma 1 that

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), W', \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, W', \sigma_\gamma)]$$

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

(B10) The proof of this rule is similar to the proof of rule (B8), except that the value of s such that $(s, v) \in \mathcal{L}_{\mathcal{E}} \wedge v \in \mathcal{T}_w[i]$ is no longer removed from the set W, so, in additions to the conditions for the rule (B8), we must ensure that y is not in $\mathcal{T}_w[i]$. If $u = w$ and $i \neq j$, then $y \in V = \mathcal{T}_w$ and since $i \neq j$, clearly, $y \notin \mathcal{T}_w[i]$ since $y \in \mathcal{T}_w[j]$ and so the proof proceeds just like the case $y \in V$ in the proof rule (B8). If $u \neq w$, then $y \in \mathcal{T}_u[j]$ and $y \notin \mathcal{T}_w[i]$ since $\mathcal{T}_u$ and $\mathcal{T}_w$ are mutually exclusive. In addition, $y \notin V'$ since $y \in \mathcal{T}_u$ means that y must have been obtained by repeatedly incrementing the value of u, and so the value of w was never involved in its computation. Therefore, the proof proceeds just like the case $y \in \mathsf{Var} \setminus (V \cup V' \cup \{t\})$ in the proof of rule (B8).

(B11) The proof is exactly the same as the proof of rule (B9). The condition $\mathsf{Indis}(t; \mathsf{Var}^* \setminus W)$ with $y \in \mathsf{Var}^* \setminus W$ ensures, using Lemma 3 that, as in the previous proof, $y \notin \mathcal{T}_w[i]$.

□

5.2.7 Increment:

Lemma 14 *The following rules are sound:*

(I1) $\{\mathit{Indis}(y; V)\}\ x := y + 1\ \{\mathit{Indis}(x; V)\}$ *if* $y \notin V$

(I2) $\{\mathit{Lctr}(y; z; i; V; V')\}\ x := y + 1\ \{\mathit{Lctr}(x; z; i + 1; V, x; V') \wedge \mathit{Ectr}(x; z; i + 1; V, x; V')\}$

(I3) $\{\mathit{ctr}(y; z; i)\}\ x := y + 1\ \{\mathit{ctr}(x; z; i + 1)\}$

(I4) $\{\mathit{Lctr}(t; w; i; V; V') \wedge \mathit{ctr}(y, w, j)\}\ x := y + 1\ \{\mathit{Lctr}(t; w; i; V, x; V')\}$ *if* $j < i$

(I5) $\{\mathit{Ectr}(t; w; i; V; V')\}\ x := y + 1\ \{\mathit{Ectr}(t; w; i; V, x; V')\}$ *if* $y \in V$*, even if* $t = y$

Proof

(I1) This rule is a simple consequence of the fact that if a value s is randomly distributed in $\{0, 1\}^l$, then so is $s + 1$. Let $X \models \mathsf{Indis}(y; V, y)$ and $x, y \notin V$. Then,

$$\begin{aligned}
[\gamma \xleftarrow{\$} [\![x := y + 1]\!]X : (S_\gamma(x), S_\gamma(V, x - x), \sigma_\gamma)] \\
&= [\gamma \xleftarrow{\$} X : (S_\gamma(y) + 1, S_\gamma(V), \sigma_\gamma)] \\
&= [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u + 1, S_\gamma(V), \sigma_\gamma)] \\
&= [\gamma \xleftarrow{\$} [\![x := y + 1]\!]X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, S_\gamma(V, x - x), \sigma_\gamma)].
\end{aligned}$$

Hence $[\![x := y + 1]\!]X \models \mathsf{Indis}(x; V, x)$.

(I2) Let $X \models \mathsf{Lctr}(y; z; i; V; V')$, so, by definition, for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}_{\mathcal{E}})$ that has non-zero probability in X, $\mathcal{T}_z[i] = y$ and $\mathcal{T}_z[i+1] = \perp$, $Set(\mathcal{T}_x) = V$, $\mathcal{Q}_y = V'$, and

$$[\gamma \xleftarrow{\$} X : (S_\gamma(y), W, \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(y)) : (u, W, \sigma_\gamma)]$$

where the set W is equal to $S_\gamma(\mathsf{Var} \setminus (V \cup V' \cup \{y\})) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$. Therefore, using the semantics, it should be clear that for every configuration $(S, \sigma, \mathcal{E}, \mathcal{T}, \mathcal{Q}, \mathcal{L}_{\mathcal{E}})$ that is not bad and has non-zero probability in $[\![x := y+1]\!]X$, $\mathcal{T}_z[i+1] = x$ and $\mathcal{T}_z[i+2] = \perp$, $Set(\mathcal{T}_x) = V, x$, $\mathcal{Q}_y = V'$. In addition, we obtain that

$$[\gamma \xleftarrow{\$} X : (S_\gamma(x), W', \sigma_\gamma)] = [\gamma \xleftarrow{\$} X; u \xleftarrow{\$} \mathcal{U}(\mathsf{len}(x)) : (u, W', \sigma_\gamma)]$$

where the set W' is equal to $S_\gamma(\mathsf{Var} \setminus (V, x \cup V' \cup \{x\})) \cup \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$ by applying the same technique as in the proof of rule (I1), because the value of neither x nor y is in W and $W' \subset W$. Hence $[\![x := y+1]\!]X \models \mathsf{Lctr}(x; z; i+1; V, x; V')$.
Noting that since the variable x cannot appear on the right-hand side of any previous command because the value of x had not been assigned, and in particular, it cannot appear in any command of the form $z := \mathcal{E}(x)$ before the execution of the command $x := y + 1$, for any configuration $\gamma \xleftarrow{\$} [\![x := y+1]\!]X$, there cannot be any pair of the form (s, x) in $\mathcal{L}_{\mathcal{E}\gamma}$ for any string s. Thus, $\{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V - x\} = \{s : (s, v) \in \mathcal{L}_{\mathcal{E}\gamma} \wedge v \notin V\}$ and the proof of $[\![x := y+1]\!]X \models \mathsf{Ectr}(x; z; i+1; V, x; V')$ is done exactly as the proof of $[\![x := y+1]\!]X \models \mathsf{Lctr}(x; z; i+1; V, x; V')$ above.

(I3) Follows trivially from the semantics.

(I4) This rule covers the case that was not included in rules (G4) and (G5). It follows from the semantics that if $X \models \mathsf{Lctr}(t; w; i; V; V') \wedge \mathsf{ctr}(y, w, j)\}$, and $j < i$, then x gets added to $\mathcal{T}_w[j+1]$ and so x is added to $Set(\mathcal{T}_w)$. We must have that $j < i$ because if we had $j = i$, then the predicate $\mathsf{Lctr}(t; w; i; V; V')$ would have to be 'transferred' to x, i.e. rule (I1) needs to be applied. The case $j > i$ is clearly impossible since $X \models \mathsf{Lctr}(t; w; i; V; V')$.

(I5) This is similar to the previous rule, except that the case $t = y$ is no longer an issue.

□

5.2.8 For loop:

Lemma 15 *The following rule is sound:*

- *(F1)* $\{\psi(p-1)\}$ *for* $k = p$ *to* q *do:* $[c_k]$ $\{\psi(q)\}$ *provided* $\{\psi(k-1)\}$ c_k $\{\psi(k)\}$ *for* $p \leq k \leq q$

Proof This is a trivial induction on the number of iterations of the loop. □

6 A Method for Proving Semantic Security

Combining Theorem 2 with our Hoare logic, we can now prove the following results, which demonstrate the soundness of our method.

Theorem 3 *Let $\mathcal{M}(m_1|\ldots|m_n, c_n)$: cmd be a generic encryption mode. If the Hoare triple $\{\mathsf{LBad}(0) \wedge \mathsf{Indep}\}$ cmd $\{\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}\}$ is valid, where $q_{\mathbf{E}}$ and $q_{\mathcal{E}}$ are the number of calls made to the* LR *encryption oracle by the algorithm that created the distribution X and the number of computations of the block cipher made to answer those oracle queries respectively, then, under the assumption that the block cipher is a random function in Φ_η, for any* LoR-CPA *adversary $\mathcal{B}$, the following holds:*

$$\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{M}}(\mathcal{B}) \leq \sum_{i=1}^{Q} \epsilon(i, q_{\mathcal{E}}^{(i)})$$

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 $\mathcal{B}$.

Proof Let X be any distribution in $\text{CDIST}_0(\Gamma)$. Then, by Lemma 5, we have that $X \models \mathsf{LBad}(0) \wedge \mathsf{Indep}$. Using all the lemmas above about the soundness of our rules, if the Hoare triple $\{\mathsf{LBad}(0) \wedge \mathsf{Indep}\}$ cmd $\{\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}\}$ is valid, then we can conclude that $[\![\mathsf{cmd}]\!]X \models \mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}$. So all the hypotheses of Theorem 2 are satisfied and the result is obtained by simply applying it. □

We note that, as a result of this, if we need only to obtain a proof that the advantage of any adversary is negligible instead of getting the exact security analysis, then we can replace the predicate $\mathsf{LBad}(\epsilon)$ by a simpler predicate LBad which holds in a distribution X iff $\mathsf{Pr}[\gamma \xleftarrow{\$} X : LBad(\gamma)]$ is negligible.[4] Using this modified predicate, we obtain the following corollary.

Corollary 3 *Let $\mathcal{M}(m_1|\ldots|m_n, c_n)$: cmd be a generic encryption mode. If the Hoare triple $\{\mathsf{LBad} \wedge \mathsf{Indep}\}$ cmd $\{\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad} \wedge \mathsf{Indep}\}$ is valid, then, under the assumption that the block cipher is a random function in Φ_η, the encryption scheme $\mathcal{M}$ is* LoR-CPA*-secure.*

All our analysis so far has assumed that the block cipher was a function selected uniformly at random in Φ_η. Our final step is to remove this assumption and relate the security of the scheme to the probability that an adversary can distinguish the block cipher from a pseudorandom function (PRF).

Theorem 4 *Let $\mathcal{M}(m_1|\ldots|m_n, c_n)$: cmd be a generic encryption mode. If the Hoare triple $\{\mathsf{LBad}(0) \wedge \mathsf{Indep}\}$ cmd $\{\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}\}$ is valid, then for any* LoR-CPA *adversary $\mathcal{B}$ against an instantiation of $\mathcal{M}$ with block*

[4] To use our logic with this predicate, we have to modify rules (G1), (B1) and (B2) in the obvious way.

cipher F, there exists an algorithm $\mathcal{C}$ that can distinguish F from a pseudorandom function such that the following holds:

$$\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{M}}(\mathcal{B}) \leq 2\mathsf{Adv}^{PRF}_{\mathcal{C},F} + \sum_{i=1}^{Q} \epsilon(i, q_{\mathcal{E}}^{(i)})$$

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 $\mathcal{B}$.

Proof This proof is done exactly like that of Theorem 11 in [BDJR97], we include it here for completeness.

Let $\mathcal{B}$ be a LoR-CPA adversary against an instantiation of $\mathcal{M}$ with block cipher F.

We construct adversary $\mathcal{C}$ against PRF F as follows[5]:

1. First, $\mathcal{C}$ randomly selects a bit $b \in \{0,1\}$.
2. Then, $\mathcal{C}$ runs algorithm $\mathcal{B}$. When $\mathcal{B}$ makes a query (M_0, M_1) to its LR oracle, $\mathcal{C}$ answers it by encrypting M_b using the algorithm described by $\mathcal{M}$, using its oracle for the computation of the block cipher.
3. When $\mathcal{B}$ terminates and outputs a bit b', $\mathcal{C}$ outputs 1 if $b = b'$, and 0 otherwise.

Define the event $\mathsf{correct}(F)$ to be the event that $\mathcal{B}$ correctly guesses the bit b when the block cipher F is used, and $\mathsf{correct}(\Phi)$ the event that $\mathcal{B}$ correctly guesses when a random function from Φ_η is used in place of the block cipher. Then, by definition, $\mathsf{Adv}^{PRF}_{\mathcal{C},F} = \mathsf{correct}(F) - \mathsf{correct}(\Phi)$. Therefore, using the notation of Definition 2, we can obtain that

$$\begin{aligned}
\mathsf{correct}(F) &= \mathsf{Pr}[\mathcal{B} \text{ outputs } 1 \wedge b = 1] + \mathsf{Pr}[\mathcal{B} \text{ outputs } 0 \wedge b = 0] \\
&= \mathsf{Pr}[\mathcal{B} \text{ outputs } 1 | b = 1]\mathsf{Pr}[b = 1] + \mathsf{Pr}[\mathcal{B} \text{ outputs } 0 | b = 0]\mathsf{Pr}[b = 0] \\
&= \frac{1}{2}\mathsf{Pr}[\mathbf{Exp}^{\mathsf{LoR\text{-}CPA}-1}_{\mathcal{M}}(\mathcal{B}, \eta) = 1] + \frac{1}{2}\mathsf{Pr}[\mathbf{Exp}^{\mathsf{LoR\text{-}CPA}-0}_{\mathcal{M}}(\mathcal{B}, \eta) = 0] \\
&= \frac{1}{2}\left(\mathsf{Pr}[\mathbf{Exp}^{\mathsf{LoR\text{-}CPA}-1}_{\mathcal{M}}(\mathcal{B}, \eta) = 1] + (1 - \mathsf{Pr}[\mathbf{Exp}^{\mathsf{LoR\text{-}CPA}-1}_{\mathcal{M}}(\mathcal{B}, \eta) = 0])\right) \\
&= \frac{1}{2}\left(1 + \mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{M}}(\mathcal{B})\right).
\end{aligned}$$

The same can be done for $\mathsf{correct}(\Phi)$, and combining this with the upper bound on $\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{M}}(\mathcal{B})$ when the block cipher is a random function given by Theorem 3 we get the following:

$$\mathsf{correct}(\Phi) \leq \frac{1}{2}\left(1 + \sum_{i=1}^{Q} \epsilon(i, q_{\mathcal{E}}^{(i)})\right).$$

Therefore,

$$\begin{aligned}
\mathsf{Adv}^{PRF}_{\mathcal{C},F} &= \mathsf{correct}(F) - \mathsf{correct}(\Phi) \\
&\geq \frac{1}{2}\left(1 + \mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{M}}(\mathcal{B})\right) - \frac{1}{2}\left(1 + \sum_{i=1}^{Q} \epsilon(i, q_{\mathcal{E}}^{(i)})\right),
\end{aligned}$$

and the result follows. □

[5] Recall that a PRF adversary is given access to an oracle and has to determine whether his oracle computes the block cipher or a random function.

7 Implementation

Theorem 3 states that to prove the security of a mode of operation, it suffices to prove that $\{\mathsf{LBad}\,(0) \wedge \mathsf{Indep}\}\ \mathsf{cmd}\ \{\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}\,(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}\}$ is valid, where cmd is the program of the mode of operation. Two methods can be used to show this. The first method consists in starting at the beginning of cmd and, at each simple command, applying every possible rule until we reach the end, and see if $\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}\,(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}$ holds at the end for some values $q_{\mathbf{E}}$ and $q_{\mathcal{E}}$. The second method consists in starting at the end of cmd with the formula $\mathsf{Indis}(c_n;\emptyset) \wedge \mathsf{LBad}\,(\epsilon(q_{\mathbf{E}}, q_{\mathcal{E}})) \wedge \mathsf{Indep}$, where the values of $q_{\mathbf{E}}$ and $q_{\mathcal{E}}$ is for now undetermined, and, at each command, determining which formula needs to hold before the command in order to have the predicate we need afterwards.

The second method lets us keep only those invariants that we need, but sometimes, several rules may lead to the desired postconditions, and each possibility may need to be explored separately to obtain a proof, which could require a very large (in the worst case, exponential) number of branches being explored. In addition, if using the backwards method, some of our predicates, particularly those relating to counters such as $\mathsf{Lctr}(x; y; i; V; V')$, would require us to make "guesses" about some of the elements of the predicate (such as the starting variable y, the index i, etc), and an erroneous choice in the guess could result in the need to backtrack. For these reasons, we decided to go through the program of the mode of operation from beginning to end. This tends to be more straightforward, but may require to keep track of many invariants that are not necessary to obtain the final result. This can however be mitigated by adapting the predicate filter of [GLL13].

7.1 Loop Predicate Discovery

The code of a general encryption mode will generally contain at least one 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 heuristics that can be used to discover such an invariant, and we show how to apply them on $CBC_{\mathcal{E}}$ and a modified version of $CTR_{\mathcal{E}}$. The first heuristic runs the code of the loop once and attempts to find an invariant by simply looking at the predicates that the formulas before and after the execution of the loop have in common. The second heuristic runs the code of the loop several times and attempts to determine the patterns that emerge after each execution and extrapolates them.

When we hit a command "$\mathsf{for}\ k = p\ \mathsf{to}\ q\ \mathsf{do}$: $[\mathsf{cmd}_k]$", we express the formula that holds before the command in the form $\varphi(p-1)$. The basic method for finding a stable loop invariant consists in processing the program of the loop cmd to find the formula $\psi(k)$ such that $\{\varphi(k-1)\}\ \mathsf{cmd}_k\ \{\psi(k)\}$ holds. We then find the maximal formula $\varphi'(k)$ that is implied by both $\varphi(k)$ and $\psi(k)$, immediately replacing the predicate $\mathsf{Indis}(c_k; V)$ for any set V, if present, by its weakening $\mathsf{Indis}(c_k, \{m_1, \ldots, m_n\})$. Replacing $\mathsf{Indis}(c_k; V)$ by $\mathsf{Indis}(c_k, \{m_1, \ldots, m_n\})$ is allowed by Lemmas 1 and 4. This simplifies the predicate and increases the probability that the heuristic finds an invariant. It is often sufficient for finding proofs since we only need the empty set in the predicate $\mathsf{Indis}(c_n;\emptyset)$ to apply Theorem 3. We test if $\{\varphi'(k-1)\}\ \mathsf{cmd}_k\ \{\varphi'(k)\}$ holds, and if so, we have found a stable loop invariant and we can apply rule $(F1)$.

This first heuristic is generally sufficient to find the loop invariant of all modes of operations whose description consists of a short sequence of initial steps followed by a single loop that does all the encrypting. In particular, this should cover all the modes produced by the automated synthesizer for block cipher modes of operations of [MKG14].

Example 1 We show how to apply this heuristic in the case of $CBC_{\mathcal{E}}$, whose description can be found in Figure 2. In this example, we assume that we do not want to find the exact security bound for $CBC_{\mathcal{E}}$ and we use the simplified predicate LBad described in Corollary 3. To simplify the writing, we also apply only the rules that are necessary to obtain the proof.

We easily find that, after processing the first two commands, $z_0 \xleftarrow{\$} \mathcal{U}(\eta)$; $c_0 := z_0$;, and applying rules (G1), (G2) and (R1), then (G1), (G2), (G3) and (A1), we obtain the formula $\mathsf{LBad} \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_0; \mathsf{Var} - c_0) \wedge \mathsf{Indis}(c_0; \mathsf{Var} - z_0)$. Parameterizing this in terms of k and weakening the indistinguishability predicate, we obtain the following:

$$\varphi(k) = \mathsf{LBad} \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \mathsf{Indis}(c_k; \{m_1, \ldots, m_n\})$$

We recall that the three instructions of the loop in $CBC_{\mathcal{E}}$ are the following:

$$y_k := z_{k-1} \oplus m_k; \qquad z_k := \mathcal{E}(y_k); \qquad c_k := c_{k-1} \| z_k;$$

After processing the program of the loop on $\varphi(k-1)$, we obtain the following:

$$\begin{aligned}\psi(k) =& \mathsf{LBad} \wedge \mathsf{Indep} \wedge \\ & \mathsf{Indis}(z_{k-1}; \mathsf{Var} \setminus \{y_k, c_{k-1}, c_k\}) \wedge \mathsf{Indis}(c_{k-1}; \{m_1, \ldots, m_n\}\}) \wedge \\ & \mathsf{Indis}(y_k; \mathsf{Var} \setminus \{z_k, c_{k-1}, c_k\}\}) \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \\ & \mathsf{Indis}(c_k; \{m_1, \ldots, m_n\})\end{aligned}$$

This is obtained after applying rules (G1), (G2), (G3) and (X1) for the first command, (B1), (B3), (B5) and (B6) for the second and (G1), (G2), (G3) and (C1) for the third. We find that the maximal formula implied by both $\varphi(k)$ and $\psi(k)$, after weakening the indistinguishability predicate pertaining to c_k, is the following:

$$\varphi'(k) = \mathsf{LBad} \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \mathsf{Indis}(c_k; \{m_1, \ldots, m_n\})$$

Testing this formula through the program of the loop again, we find that it is a loop invariant, therefore we can apply rule (F1) and obtain that, at the end of the program of $CBC_{\mathcal{E}}$, the following formula holds:

$$\mathsf{LBad} \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_n; \mathsf{Var}^* - c_n) \wedge \mathsf{Indis}(c_n; \{m_1, \ldots, m_n\}).$$

Therefore, we can apply Corollary 3 and find that $CBC_{\mathcal{E}}$ is $\mathsf{LoR\text{-}CPA}$-secure. □

Unfortunately, this heuristic usually fails when the proof requires to accumulate predicates at each iteration of the loop. For example, Figure 3 shows an alternative description of $CTR_{\mathcal{E}}$ for which our first heuristic would not return a loop invariant strong enough to find a proof. In addition, the heuristic does not enable us to collect enough information to extrapolate the security parameter contained in the predicate $\mathsf{LBad}\,(\epsilon)$ that is necessary for an exact security analysis. For these

reasons, we present a second heuristic based on widening in abstract interpretation. We start with formula $\varphi(k-1)$, and process the program of the loop once to find formula $\psi_1(k)$ such that $\{\varphi(k-1)\}\ \mathsf{cmd}_k\ \{\psi_1(k)\}$ (we do this by applying at each step every rule whose precondition is met). Then, we repeat this starting with formula $\psi_1(k-1)$ to find formula $\psi_2(k)$ such that $\{\psi_1(k-1)\}\ \mathsf{cmd}_l\ \{\psi_2(k)\}$. The idea is then to inspect formulas $\varphi(k)$, $\psi_1(k)$ and $\psi_2(k)$ for patterns that can be extrapolated. For example, we can try to identify a predicate $\gamma(k)$ such that:

1. $\gamma(k)$ appears in $\varphi(k)$,
2. $\gamma(k-1) \wedge \gamma(k)$ appears in $\psi_1(k)$,
3. $\gamma(k-2) \wedge \gamma(k-1) \wedge \gamma(k)$ appears in $\psi_2(k)$.

We then use a new starting formula $\varphi'(k)$ which is just like $\varphi(k)$, except that the occurrence of $\gamma(k)$ in $\varphi(k)$ is replaced by $\bigwedge_{j=p-1}^{j=k} \gamma(j)$ in $\varphi'(k)$. Note that, by construction, $\varphi(p-1)$ is equal to $\varphi'(p-1)$, so we know that $\varphi'(p-1)$ is satisfied at the beginning of the loop. We can similarly try to find patterns that appear only after the first iteration of the loop, that is, $\gamma(k)$ appears in $\psi_1(k)$ and $\gamma(k-1) \wedge \gamma(k)$ appears in $\psi_2(k)$, in which case occurrences of $\gamma(k)$ in $\varphi(k)$ are replaced by $\bigwedge_{j=p}^{j=l} \gamma(j)$ in $\varphi'(l)$.

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

Fig. 3 Alternative description of $CTR_{\mathcal{E}}$

The predicate $\mathsf{LBad}\,(\epsilon)$ can be treated similarly, except that it is the security parameter inside the predicate that may change from one iteration to the next instead of new invariants appearing at each iteration. We examine how the parameter in the predicate evolves in $\varphi(k)$, $\psi_1(k)$ and $\psi_2(k)$ – it will either remain the same, or new terms are added at each iteration – and, if necessary, we extrapolate this evolution in the form of a sum. The example below shows how this is done. One may decide to first find a stable loop invariant $\varphi'(k)$ using either of our heuristics while disregarding the security parameter in LBad (that is, using the predicate LBad without its parameter, as suggested before Corollary 3), and then to repeat the process a second time, that is, process the program of the loop twice to obtain $\varphi'(k)$, $\psi_1'(k)$ and $\psi_2'(k)$, this time looking *only* at how the parameter in LBad evolves to obtain a stable invariant $\varphi''(k)$ *with* the security parameter.

Our choice to process the program of the loop *two* times (to obtain $\psi_1(k)$ and $\psi_2(k)$) for the second heuristic is rather arbitrary and was made because it seems to work well in practice. It is however possible to construct artificial examples for which the pattern would not emerge after only two iterations[6], and for those cases, it would be necessary to repeat the process more times.

[6] For example, a variant of CBC in which the line $y_k := z_{k-1} \oplus m_k$ is replaced by $y_k := z_{k-3} \oplus z_{k-2} \oplus z_{k-1} \oplus m_k$ would require four iterations before the definitive pattern emerges.

Example 2 We found in the previous example that the following formula was a stable invariant for the loop in $CBC_{\mathcal{E}}$:

$$\varphi'(k) = \mathsf{LBad} \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \mathsf{Indis}(c_k; \emptyset)$$

Adding the parameter to LBad for the first two commands before executing the loop, we find that the predicate $\mathsf{LBad}\,(0)$ holds before executing the loop. Let q be the number of elements in $\mathcal{L}_{\mathcal{E}}$. Then we find that, after processing the program of the loop once to obtain $\psi_1'(k)$ we get the following:

$$\psi_1'(k) = \mathsf{LBad}\left(\frac{q}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \mathsf{Indis}(c_k; \emptyset)$$

We note that there are now $q+1$ elements in $\mathcal{L}_{\mathcal{E}}$. We repeat this a second time and obtain the following:

$$\psi_2'(k) = \mathsf{LBad}\left(\frac{q+(q+1)}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \mathsf{Indis}(c_k; \emptyset)$$

Extrapolating this, we get the following stable invariant with security parameter:

$$\varphi''(k) = \mathsf{LBad}\left(\frac{\sum_{i=0}^{k-1} q+i}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_k; \mathsf{Var}^* - c_k) \wedge \mathsf{Indis}(c_k; \emptyset)$$

So that, at the end of the program, the following formula holds:

$$\mathsf{LBad}\left(\frac{\sum_{i=0}^{n-1} q+i}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(z_n; \mathsf{Var}^* - c_n) \wedge \mathsf{Indis}(c_n; \emptyset)$$

Using this in Theorem 4, we find that for any $\mathsf{LoR\text{-}CPA}$ adversary $\mathcal{B}$ against an instantiation of $CBC_{\mathcal{E}}$ with block cipher F, there exists an algorithm $\mathcal{C}$ that can distinguish F from a pseudorandom function such that the following holds:

$$\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{CBC_{\mathcal{E}}}(\mathcal{B}) \leq 2\mathsf{Adv}^{prf}_{\mathcal{C},F} + \sum_{i=1}^{Q} \frac{\sum_{j=0}^{q_{\mathcal{E}}^{(i)}-1}(\sum_{k=1}^{i-1} q_{\mathcal{E}}^{(k)}) + j}{2^\eta}$$

where $\sum_{k=1}^{i-1} q_{\mathcal{E}}^{(k)}$ is the number of computation of the block cipher necessary to answer the first $i-1$ encryption queries, that is, the size of $\mathcal{L}_{\mathcal{E}}$ after answering the first $i-1$ encryption queries. Let $\Omega_i = \sum_{k=1}^{i-1} q_{\mathcal{E}}^{(k)}$ denote this sum. Note that the numerator of the term in this sum with $i = i_0$ is simply a sum of all the integers from Ω_{i_0} to $\Omega_{i_0} + q_{\mathcal{E}}^{(i_0)} - 1$. Similarly, the numerator of the term in this sum with $i = i_0 + 1$ is the sum of all the integers from Ω_{i_0+1} to $\Omega_{i_0+1} + q_{\mathcal{E}}^{(i_0+1)} - 1$. Since $\Omega_{i_0+1} = \Omega_{i_0} + q_{\mathcal{E}}^{(i_0)}$, we obtain that the sum of those two terms is the sum of all integers from Ω_{i_0} to $\Omega_{i_0} + q_{\mathcal{E}}^{(i_0)} + q_{\mathcal{E}}^{(i_0+1)} - 1$. Generalizing this, if we let $\mathcal{Q}_{\mathcal{E}}$ be the total number of computations of the block cipher necessary to answer all of the adversary's encryption queries, the expression in the previous equation can be simplified to

$$\begin{aligned}\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{CBC_{\mathcal{E}}}(\mathcal{B}) &\leq 2\mathsf{Adv}^{prf}_{\mathcal{C},F} + \sum_{i=0}^{\mathcal{Q}_{\mathcal{E}}-1} \frac{i}{2^\eta} \\ &\leq 2\mathsf{Adv}^{prf}_{\mathcal{C},F} + \frac{\mathcal{Q}_{\mathcal{E}}(\mathcal{Q}_{\mathcal{E}}-1)}{2^\eta}\end{aligned}$$

which is exactly the same security bound that one would obtain by analysing $CBC_{\mathcal{E}}$ manually ([BDJR97]). □

Example 3 We show how the second heuristic is used to find a stable predicate for the second loop in $CTR'_{\mathcal{E}}$ (see Figure 3). Again, to simplify the writing, we also apply only the rules that are necessary to obtain the proof and remove unnecessary predicates.

After processing the first two commands of $CTR'_{\mathcal{E}}$, and using rules (G1), (G2), (R1), (A1), (A3), we obtain the following formula:

$$\begin{aligned}&\mathsf{LBad}\,(0) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(c_0; \mathsf{Var}^* - ctr_0) \wedge \\ &\mathsf{Lctr}(ctr_0; ctr_0; 0; \{ctr_0, c_0\}; \emptyset)\end{aligned}$$

Parameterizing this in terms of k, we obtain the following:

$$\begin{aligned}\varphi(k) =&\mathsf{LBad}\,(0) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(c_k; \mathsf{Var}^* - ctr_k) \wedge \\ &\mathsf{Lctr}(ctr_k; ctr_k; k; \{ctr_k, c_k\}; \emptyset)\end{aligned}$$

The first heuristic would fail at finding a loop invariant, so we use the second heuristic. Let q be the number of elements in $\mathcal{L}_{\mathcal{E}}$. We recall that the program contained in the first loop is:

$$ctr_i := ctr_{i-1} + 1; \qquad y_i := \mathcal{E}(ctr_i)$$

We process the program of the loop on $\varphi(k-1)$ and obtain the following formula:

$$\begin{aligned}\psi_1(k) =&\mathsf{LBad}\left(\frac{q}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(c_{k-1}; \mathsf{Var} \setminus \{ctr_{k-1}, ctr_k\}) \wedge \\ &\mathsf{Indis}(y_k; \mathsf{Var}^*) \wedge \\ &\mathsf{Lctr}(ctr_k; ctr_{k-1}; k; \{ctr_k, ctr_{k-1}, c_{k-1}\}; \emptyset)\end{aligned}$$

To obtain this, we apply rules (G1), (G2), (G3) and (I2) after the first command, and (B2), (B4), (B5), (B6), (B8) after the second. We note that there are now $q+1$ elements in $\mathcal{L}_{\mathcal{E}}$. We repeat this a second time and obtain the following:

$$\begin{aligned}\psi_2(k) =&\mathsf{LBad}\left(\frac{q + (q+1)}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(c_{k-2}; \mathsf{Var} \setminus \{ctr_k, ctr_{k-1}, ctr_{k-2}\}) \wedge \\ &\mathsf{Indis}(y_{k-1}; \mathsf{Var}^*) \wedge \mathsf{Indis}(y_k; \mathsf{Var}^*) \wedge \\ &\mathsf{Lctr}(ctr_k; ctr_{k-2}; k; \{ctr_k, ctr_{k-1}, ctr_{k-2}, c_{k-2}\}; \emptyset)\end{aligned}$$

The same rules are applied, except that we additionally need to use (B7) for the second command. Extrapolating this pattern, we obtain the following:

$$\begin{aligned}\varphi'(k) =&\mathsf{LBad}\left(\frac{\sum_{i=0}^{k-1} q + i}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(c_0; \mathsf{Var} \setminus \{ctr_k, \ldots, ctr_0\}) \wedge \\ &\left(\bigwedge_{i=1}^{k} \mathsf{Indis}(y_i; \mathsf{Var}^*)\right) \wedge \\ &\mathsf{Lctr}(ctr_k; ctr_0; k; \{ctr_k, \ldots, ctr_0, c_0\}; \emptyset)\end{aligned}$$

Testing this formula through the program of the loop again, we find that it is a loop invariant, therefore we can apply rule (F1) and obtain that, after processing the loop, the following formula holds:

$$\mathsf{LBad}\left(\frac{\sum_{i=0}^{n-1} q+i}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \mathsf{Indis}(c_0; \mathsf{Var}^* \setminus \{ctr_n, \ldots, ctr_0\}) \wedge$$
$$\left(\bigwedge_{i=1}^{n} \mathsf{Indis}(y_i; \mathsf{Var}^*)\right) \wedge$$
$$\mathsf{Lctr}(ctr_n; ctr_0; n; \{ctr_n, \ldots, ctr_0, c_0\}; \emptyset)$$

Repeating this process with the second loop (after dropping the counter predicate since it is no longer needed), we find that the stable formula for the second loop is the following:

$$\begin{aligned}\varphi''(k) = &\mathsf{LBad}\left(\frac{\sum_{i=0}^{n-1} q+i}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \\ &\left(\bigwedge_{i=1}^{k} \mathsf{Indis}(y_i; \mathsf{Var}^* \setminus \{z_i, c_k, \ldots, c_1\})\right) \wedge \\ &\left(\bigwedge_{i=k+1}^{n} \mathsf{Indis}(y_i; \mathsf{Var}^*)\right) \wedge \\ &\left(\bigwedge_{i=1}^{k} \mathsf{Indis}(z_i; \mathsf{Var}^* \setminus \{y_i, c_k, \ldots, c_1\})\right) \wedge \\ &\left(\bigwedge_{i_1}^{k} \mathsf{Indis}(c_i; \mathsf{Var}^* \setminus \{y_k, z_k, \ldots, y_1, z_1, c_{k-1}, \ldots, c_0\})\right)\end{aligned}$$

So that, at the end of the program, the following formula holds:

$$\begin{aligned}&\mathsf{LBad}\left(\frac{\sum_{i=0}^{n-1} q+i}{2^\eta}\right) \wedge \mathsf{Indep} \wedge \\ &\left(\bigwedge_{i=1}^{n} \mathsf{Indis}(y_i; \mathsf{Var}^* \setminus \{z_i, c_n, \ldots, c_1\})\right) \wedge \\ &\left(\bigwedge_{i=1}^{n} \mathsf{Indis}(z_i; \mathsf{Var}^* \setminus \{y_i, c_n, \ldots, c_1\})\right) \wedge \\ &\left(\bigwedge_{i_1}^{n} \mathsf{Indis}(c_i; \mathsf{Var}^* \setminus \{y_n, z_n, \ldots, y_1, z_1, c_{n-1}, \ldots, c_0\})\right)\end{aligned}$$

Using this in Theorem 4, and simplifying as in the previous example (because the factor inside LBad is the same as in the previous example), we find that for any LoR-CPA adversary $\mathcal{B}$ against an instantiation of $CTR'_{\mathcal{E}}$ with block cipher F, there exists an algorithm $\mathcal{C}$ that can distinguish F from a pseudorandom function such that the following holds:

$$\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{CTR'_{\mathcal{E}}}(\mathcal{B}) \leq 2\mathsf{Adv}^{prf}_{\mathcal{C},F} + \frac{\mathcal{Q}(\mathcal{Q}-1)}{2^\eta}$$

which differs from the security bound one could find manually only by a factor of two (the manual proof is tighter). We note that we would have obtained the exact same bound for the mode of operation $CTR_{\mathcal{E}}$ of Figure 2, but the proof would have been much easier because our first heuristic would have worked on that program. □

7.2 Prototype

We programmed an OCaml prototype of our method for proving the security of modes of operation [GLLSN]. The program requires about 2000 lines of code, and can successfully produce proofs of security for all the examples discussed in this paper, in addition to other standard modes of operations such as CFB and OFB, and more exotic ones like PCBC, in less than one second on a personal workstation.

8 Conclusion

We proposed an automatic method for proving the semantic security of symmetric encryption modes. We introduced a small programming language in order to describe these modes. We construct a Hoare logic to make assertions about variables and propagate the assertions with the execution of the commands in the language. If the program which represents an encryption mode satisfies some invariants at the end of our automatic analysis then we conclude that the encryption mode is IND-CPA secure and we can provide an exact analysis for this security.

As future work, we are also considering an extension of our work to prove CCA security of encryption modes using approaches such as the one proposed in [Des00] or the method proposed in [CDE+08].

Another more complex and challenging direction is to propose an extended version of our Hoare Logic in order to be able to analyze "modern" encryption modes which use more complex mathematical operation or primitives, such as tweakable block ciphers [LRW02]. The main challenge here is that many modern modes make ad-hoc uses of mathematical operations (such as arithmetic operations in extension fields), and each mode tends to integrate the "tweak" in a slightly different way. As a result, it is very difficult to find the appropriate abstraction level and the appropriate equational theories to model such primitives.

References

[BCG+13] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella-Béguelin. Fully automated analysis of padding-based encryption in the computational model. In *Proceedings of the 20th ACM Conference on Computer and Communications Security, (CCS'13)*, November 2013.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:394, 1997.

[BDK+10] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 46–63. Springer, 2010.

[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.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[BGLB11] 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.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[BP06] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[CDE+08] Judicael Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lahknech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *Proceedings of the 15th ACM Conference on Computer and Communications Security, (CCS'08)*, Alexandria, USA, October 2008.

[CEL07] Judicaël Courant, Cristian Ene, and Yassine Lakhnech. Computationally sound typing for non-interference: The case of deterministic encryption. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2007.

[CFR+91] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13(4):451–490, October 1991.

[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, Berlin, Heidelberg, 2008. Springer-Verlag.

[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.

[Des00] Anand Desai. New paradigms for constructing symmetric encryption schemes secure against chosen-ciphertext attack. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 394–412, London, UK, 2000. Springer-Verlag.

[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.

[GLL13] Martin Gagné, Pascal Lafourcade, and Yassine Lakhnech. Automated security proofs for almost-universal hash for mac verification. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2013.

[GLLSN] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Prototype implementation of hoare logic. Available at http://sancy.univ-bpclermont.fr/∼lafourcade/Tools/.

[GLLSN09] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated proofs for encryption modes. In *13th Annual Asian Computing Science Conference Focusing on Information Security and Privacy: Theory and Practice (ASIAN'09)*, volume 5913 of *LNCS*, pages 39–53, 2009.

[GLLSN11] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated verification of block cipher modes of operation, an improved method. In Joaquín García-Alfaro and Pascal Lafourcade, editors, *FPS*, volume 6888 of *Lecture Notes in Computer Science*, pages 23–31. Springer, 2011.

[Hal04] Shai 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, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2004.

[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.

[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.

[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.

[LRW02] 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.

[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*, volume 4876 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.

[MKG14] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 140–152. IEEE, 2014.

[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, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[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.

[ÉJJV01] Éliane Jaulmes, Antoine Joux, and Frédéric Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit - a new construction. In *Fast Software Encryption 02, Lecture Notes in Computer Science*, pages 237–251. Springer-Verlag, 2001.