Journal of Universal Computer Science, vol. 20, no. 2 (2014), 193-212
submitted: 19/4/13, accepted: 31/10/13, appeared: 1/2/14 © *J.UCS*

Towards Formal Linear Cryptanalysis using HOL4

Osman Hasan and Syed Ali Khayam
(School of Electrical Engineering and Computer Science
National University of Sciences and Technology (NUST)
Sector H-12, Islamabad, Pakistan
{osman.hasan, ali.khayam}@seecs.nust.edu.pk)

Abstract: Linear cryptanalysis (LC), first introduced by Matsui, is one of the most widely used techniques for judging the security of symmetric-key cryptosystems. Traditionally, LC is performed using computer programs that are based on some fundamental probabilistic algorithms and lemmas, which have been validated using paper-and-pencil proof methods. In order to raise the confidence level of LC results, we propose to formally verify its foundational probabilistic algorithms and lemmas in the higher-order-logic theorem prover HOL4. This kind of infrastructure would also facilitate reasoning about LC properties within the HOL4 theorem prover. As a first step towards the proposed direction, this paper presents the formalization of two foundations of LC, which were initially proposed in Matsui's seminal paper. Firstly, we formally verify the Piling-up Lemma, which allows us to compute the probability of a block cipher's linear approximation, given the probabilities of linear approximations of its individual modules. Secondly, we provide a formal probabilistic analysis of Matsui's algorithm for deciphering information about the unknown bits of a cipher key. In order to illustrate the practical effectiveness and utilization of our formalization, we formally reason about a couple of LC properties.

Key Words: Formal Verification, Higher-order logic, Probability Theory, Cryptography, Theorem Proving.
Category: F.4, I.2.3, D.3.1

1 Introduction

Encryption is now used ubiquitously in computer systems to prevent information leakage and cyber crime. It was the requirement to keep unclassified information secure on computing systems that led to the development and standardization of the Data Encryption Standard (DES) – the first published digital encryption algorithm–in 1977. Despite its widespread use in personal, commercial, government, and banking sectors, the DES algorithm remained impervious to attack for over a decade.

Security evaluation of DES and other encryption algorithms falls in a subbranch of cryptology called cryptanalysis [Stamp and Low, 2007], which focuses on designing attacks that can reveal the strengths/weaknesses of a cryptographic system. Linear cryptanalysis (LC) [Matsui, 1993] is one of the most powerful cryptanalysis tools presently used to gauge the security of symmetric-key cryptosystems, see e.g., DES [Matsui, 1993], GOST [Shorin et al, 2001], Serpent [Biham et al., 2001], AES [Daemen and Rijmen, 2002], Blowfish [Nakahara, 2007] and SMS4 [Zhang et al., 2008]. The most popular class of attacks on encryption systems is called the *known plaintext attack*, in which the attacker has access to a set of *plaintexts* (inputs) and their corresponding *ciphertexts* (encrypted outputs). LC allows us to identify weaknesses of a cryptosystem by deciphering some bits of the key from the available $\langle$plaintext, ciphertext$\rangle$ pairs in a reasonable computational time. The process of LC can be broadly divided into three main steps.

First, for each one of the cipher modules E of a given cipher, we obtain linear relationships between the bits of the plaintext $\mathcal{P}$, the ciphertext $\mathcal{C}$, and the key $\mathcal{K}$ as:

$$(\bigoplus_{i \in a} \mathcal{P}_i) \oplus (\bigoplus_{j \in b} \mathcal{C}_j) = \bigoplus_{k \in c} \mathcal{K}_k, \tag{1}$$

where $\oplus$ denotes the Boolean Exclusive-OR (XOR) operator and the indices i, j and k respectively denote fixed bit locations in the bit vectors a, b and c of $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$, respectively. Ideally, Equation (1) must hold exactly with a probability $p = \frac{1}{2}$ in the case of a randomly given plaintext $\mathcal{P}$ and the corresponding ciphertext $\mathcal{C}$. Thus, the fact that $p \neq \frac{1}{2}$ implies that we have acquired an effective linear approximation of E.

The second step is to obtain the probability associated with the linear approximation for the entire cipher, based on the probabilities of the linear approximations of all of its modules calculated by Equation (1), using Matsui's fundamental probabilistic property called the *Piling-up Lemma* [Matsui, 1993].

As the last step, the probabilities associated with the linear approximations of the cipher are used, along with the known plaintext and ciphertext bits, to decipher information about the unknown key bits using an algorithm proposed by Matsui [Matsui, 1993], which will henceforth be referred to as the *Key-Deciphering algorithm* in this paper.

LC is usually performed by implementing the above method using computer programs. Despite the widespread usage of such software, they cannot ascertain 100% accurate LC results. Sources of error include the informal specifications of the cipher to be analyzed and the fundamental LC algorithms. Similarly, roundoff errors accumulated in the extensive LC computations using computer arithmetics also contribute to discrepancies in the results. Due to the safety- and security-critical natures of the underlying cryptosystems that are analyzed using LC, such inaccuracies could ultimately lead to enormous computation costs in ensuring the safety of national security and finances.

In order to overcome these accuracy limitations, we propose to use higher-order-logic theorem proving [Gordon, 1989] for LC. Higher-order logic is a system of deduction with a precise semantics and, due to its high expressiveness, can be used to describe any mathematical relationship that can be modeled in a closed form. Interactive theorem proving is the field of computer science and mathematical logic concerned with computer-based formal proof tools that require human assistance. We argue that the high expressibility of higher-order logic can be leveraged to formalize the fundamental probabilistic principles of LC. Such an infrastructure can hence be used to formally reason about LC properties of cryptosystems within the sound core of a theorem prover. Due to the high expressibility of higher-order logic, the proposed approach is very flexible in terms of analyzing a variety of cryptosystems and reasoning about their generic security properties, while providing exact results due to the inherent soundness of the underlying approach.

As a first step towards the proposed formal LC approach, we present the formal verification of the Piling-up Lemma and the formal probabilistic analysis of Matsui's Key-Deciphering algorithm in this paper. In view of the fundamentally important nature of the Piling-up Lemma and the Key-Deciphering algorithm in diverse LC applications, our results serve as the stepping stone for higher-order-logic theorem proving based LC research. The developed HOL4 proof script, corresponding to the above mentioned formalization, is publicly available at [Hasan, 2014].

In order to illustrate the practical effectiveness and utilization of our formalization, we use it to formally verify a couple of LC properties. In particular, we prove that a cipher cannot be broken if the probability associated with the linear approximation of any one of its submodules is exactly equal to one half (Theorem 7 in this paper) and that the Matsui's Key-Deciphering algorithm almost always succeeds if the probability that the linear expression, given in Equation (1), becomes very small (Theorem 8 in this paper) or the total number of $\langle \text{plaintext}, \text{ciphertext} \rangle$ becomes very large (Theorem 9 in this paper).

The foremost requirement for the proposed formalization and verification, and thus conducting formal LC, is to have access to the mathematical theories of natural and real numbers, sets and probability in a higher-order-logic theorem prover. The HOL4 system [Harrison et al., 2006], a higher-order-logic theorem prover, comes with a very rich built-in library of mathematical theories including the ones mentioned above and thus has been used for the present work.

The rest of the paper is organized as follows: Section 2 presents some related work. Section 3 provides an overview of HOL probabilistic analysis related foundations that we build upon for the proposed formalization of this paper. Then, we present the formalization and verification of the Piling-Up Lemma and the Key-Deciphering algorithm in Sections 4 and 5, respectively. This will be followed by a few illustrative applications of our formally verified results in the area of LC in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

Formal methods have been extensively used in cryptography, due to the dire need of precise analysis in this safety-critical domain. Formal tools, such as Avispa [Abadi and Cortier, 2006] and CryptoVerif [Blanchet and Pointcheval, 2006], can automatically verify cryptographic protocols for specific parameters. A SAT-based equivalence checking approach [Smith and Dill, 2008] for block ciphers has been successfully used to verify many commonly-used block ciphers, such as AES, DES, Blowfish and RC2. Similarly, the higher-order-logic theorem prover HOL4 has been used to formally specify and verify a number of widely used symmetric-key block ciphers, such as AES, MARS, Twofish, RC6, Serpent, IDEA and TEA [Duan et al., 2005]. Most of the research in the domain of formal analysis of cryptosystems concentrates on the functional correctness properties, such as checking that decryption and encryption are inverse functions. Our work is distinct from prior research because we focus on formal analysis of security aspects, such as checking that the cipher is not be easy to break. At the same time, our work can also be used to complement existing formalizations of various ciphers. For example, we can directly use the HOL4 formalization of AES, given in [Duan et al., 2005], to perform its formal LC using the infrastructure presented in this paper.

The ability to reason about probabilistic properties in a higher-order logic theorem prover holds the key to the proposed formal LC approach. Hurd's PhD thesis [Hurd, 2002] can be considered a pioneering work in this regard as it presents a methodology for the formalization and verification of probabilistic algorithms in the HOL4 theorem prover. Building upon Hurd's formalization, most of the commonly-used discrete [Hurd, 2002] and continuous [Hasan, 2008] random variables have been formalized in higher-order logic and their corresponding probabilistic [Hurd, 2002] and statistical [Hasan, 2008, Hasan et al., 2009a] properties have been verified using theorem proving techniques. In this paper, we use the above formalization infrastructure to develop a generic theorem proving based LC framework, which, to the best of our knowledge, has thus far not been presented in literature.

Affeldt *et al.*'s [Affeldt et al., 2007] work on the formalization of a game-playing framework and formally verifying security properties is worth mentioning here. The foundations of this work lie upon the formalization of a probabilistic programming language in the Coq theorem prover. This infrastructure is then used to formally verify some of the commonly used properties that are used to verify game-based security proofs. Affeldt *et al.* also illustrated the effectiveness of their formalization by verifying the PRP/PRF Switching Lemma. This approach, just like the proposed idea, is primarily based on formal probabilistic reasoning. But our final objectives are quite distinct. The focus of Affeldt *et al.*'s work is on the formalization of game-based security proofs. In contrast, our work is on the development of a formal reasoning framework that facilitates the formal verification of LC related properties.

Probabilistic model checking [Baier et al., 2003, Rutten et al., 2004] is another formal method that can be used for conducting precise probabilistic analysis. The main premise of this approach is to exhaustively verify the intended probabilistic properties against a state-based mathematical model of a given system. Besides the accuracy of the results, the most promising feature of probabilistic model checking is the ability to perform the analysis automatically. Probabilistic model checking has been used to assess security aspects of a few cryptosystems [Steel, 2006, Misra and Saha, 2009]. However, this method is not suitable for conducting cryptanalysis due to two inherent limitations: 1) Only systems that can be expressed as probabilistic finite state machines can be analyzed; 2) Probabilistic model checking often suffers from state-space explosion and does not support the verification of generic mathematical expressions. Our proposed higher-order-logic theorem proving based approach tends to overcome these limitations and thus can handle formal LC, which is fundamentally based on the mathematical expression of the Piling-up Lemma and the statistical properties of the Key-Deciphering algorithm.

3 Random Variables and their Properties in HOL4

A *measure space* is defined as a triple (Ω, Σ, μ) [Galambos, 1995], where Ω is a set, called the *sample space*, Σ represents a σ-algebra of subsets of Ω and the subsets are usually referred to as *measurable sets*, and μ is a *measure* with domain Σ. A *probability space* is a measure space $(\Omega, \Sigma, \mathbb{P})$ such that the measure, referred to as the *probability* and denoted by $\mathbb{P}$, of the sample space is 1.

Hurd [Hurd, 2002] formalized some measure theory in higher-order logic to define a measure space as a pair (Σ, μ), whereas the sample space on which this pair is defined is implicitly implied from the higher-order-logic definitions to be equal to the universal set of the appropriate data-type. Building upon this formalization, the probability space was then defined as a pair $(\mathcal{E}, \mathbb{P})$, where the domain of $\mathbb{P}$ is the set $\mathcal{E}$, which is a set of subsets of infinite Boolean sequences $\mathbb{B}^\infty$. Both $\mathbb{P}$ and $\mathcal{E}$ are defined using the Carathéodory's extension theorem, which ensures that $\mathcal{E}$ is a σ-algebra: closed under complements and countable unions.

Now, a random variable, which is one of the core concepts in probabilistic analysis, is fundamentally a probabilistic function and thus can be modeled in higher-order logic as a deterministic function, which takes an argument of type $\mathbb{B}^\infty$. These deterministic functions make random choices based on the result of popping the topmost bit in the infinite Boolean sequence and may pop as many random bits as they need for their computation. When the functions terminate, they return the result along with the remaining portion of the infinite Boolean sequence to be used by other programs. Thus, a random variable which takes a parameter of type α and ranges over values of type β can be represented in HOL4 by a function of type:

$$\mathcal{F} : \alpha \to B^{\infty} \to \beta \times B^{\infty}.$$

As an example, consider a Bernoulli random variable that returns 1 or 0 with probability $\frac{1}{2}$ in either case. It can be formalized in higher-order logic as follows:

$\vdash \forall$ `s. bit s = if shd s then 1 else 0, stl s`,

where the functions `shd` and `stl` are the sequence equivalents of the list operations *'head'* and *'tail'*, respectively. The function `bit` accepts the infinite Boolean sequence s and returns a pair. The first element of the returned pair is a random number that is either 0 or 1, depending on the Boolean value of the top most element of s, while the second element of the pair is the unused portion of the infinite Boolean sequence, which in this case is the tail of the sequence.

Higher-order-logic functions for probabilistic algorithms can also be expressed in the more general state-transforming monad, where the states are the infinite Boolean sequences. The concept of Monads originates from the category theory, but for expressing probabilistic programs no prior knowledge of category theory is required. The monadic operator `unit` is used to lift values to the monad, and the operator `bind` is the monadic analogue of function application. These operators are formally defined as follows:

$\vdash \forall$ `a s. unit a s = (a,s)`
$\vdash \forall$ `f g s. bind f g s = g (fst (f s)) (snd (f s))`.

The HOL4 functions `fst` and `snd` return the first and second components of their argument, which is a pair, respectively. All monad laws hold for this definition and the notation allows us to write functions without explicitly mentioning the sequence that is passed around, e.g.,

$\vdash$ `bit_monad = bind sdest (`λ`b. if b then unit 1 else unit 0)`

represents function `bit` in monadic notation. Function `sdest` gives the head and tail of a sequence as a pair `(`*shd* `s,`*stl* `s)` and (λ`x. fx`) denotes the lambda-abstraction function that maps its argument x to $f(x)$.

Once random variables are formalized, we can use the formalized probability theory infrastructure to reason about their probabilistic properties. For example, the following Probability Mass Function (PMF) property can be verified for the function `bit` using the HOL4 theorem prover.

$\vdash \mathbb{P}$ {`s | fst (bit s) = 1`} = $\frac{1}{2}$, where $\{x|C(x)\}$ represents a set of all x that satisfy the condition C in HOL4.

The measurability and independence of a probabilistic function are central concepts in probability theory. A property `indep`, called *strong function independence*, is introduced in [Hurd, 2002] such that if a function f preserves strong function independence, then f will be both measurable and independent. It has also been shown that a function is guaranteed to preserve *strong function independence* if it accesses the infinite Boolean sequence using only the `unit`, `bind` and `sdest` primitives [Hurd, 2002].

The above approach has been successfully used to formalize most of the commonly-used random variables and verify them based on their corresponding probability distribution properties. In this paper, we use the Bernoulli random variable, which is verified using the following PMF relation [Hurd, 2002]:

Theorem 1: *PMF of Bernoulli(p) Random Variable*

$\vdash \forall$ `p. 0` $\leq$ `p` $\wedge$ `p` $\leq$ `1` $\Rightarrow \mathbb{P}$ `{s | fst (bernoulli_rv p s)} = p.`

The function `bernoulli_rv` [Hurd, 2002] for the Bernoulli(p) random variable models an experiment with two outcomes, *True* and *False*, whereas p represents the probability of obtaining *True*. It has been formalized in [Hurd, 2002] as follows:

The above-mentioned formalization has also been used to formally model statistical properties, like expectation and variance [Hasan, 2008]. These properties can also be used to formally verify the corresponding relationships for random variables. For example, in this paper, we use the expectation and variance properties of the Binomial random variable, verified by mean of the following theorems [Hasan, 2008]:

Theorem 2: *Expectation of Binomial(m,p) Random Variable*

$\vdash \forall$ `m p. 0` $\leq$ `p` $\wedge$ `p` $\leq$ `1`
  $\Rightarrow$ `expec (`λ`s. binomial_rv m p s) = m p.`

Theorem 3: *Variance of Binomial(m,p) Random Variable*

$\vdash \forall$ `m p. 0` $\leq$ `p` $\wedge$ `p` $\leq$ `1`
  $\Rightarrow$ `variance (`λ`s. binomial_rv m p s) = m p (1 - p).`

The HOL4 functions `binomial_rv`, `expec` and `variance` above represent the binomial random variable, expectation and variance, respectively. The Binomial (m, p) random variable is formally modeled the sum of m Bernoulli(p) random variables. This way, the proofs of Theorems 2 and 3 are primarily based on the linearity of expectation and variance properties and the expectation and variance properties of the Bernoulli(p) random variable, respectively. Further details about their verification can be found in [Hasan, 2008].

Similarly, the main classical properties of expectation and variance have been verified to hold for these formal definitions. Chebyshev's inequality [Hasan, 2008] is one such property that can be used to reason about the tail distribution bounds associated with the Key-Deciphering algorithm in this paper.

Theorem 4: *Chebyshev's Inepuality*

$\vdash \forall$ `R a. (0 <` `a)` $\wedge$ `(0 <` `variance R)` $\wedge$
`(summable(`λ`n. n` $\mathbb{P}$`{s | fst (R s) = n}))` $\wedge$
`(summable(`λ`n. n`^{2} $\mathbb{P}$`{s | fst (R s) = n}))`
$\Rightarrow$ $\mathbb{P}$ `{s | abs (fst (R s) - expec R)` $\geq$ `a}` $\leq$ $\frac{\texttt{variance R}}{\texttt{a}^2}$.

In the above theorem, the predicate `summable` is $True$ if the infinite summation of its *real* sequence argument exists [Harrison, 1998], i.e., $\exists x. \lim_{k\to\infty} \sum_{n=0}^{k} f(n) = x$. Thus, the *summable* assumptions in the above theorem state that the theorem is only valid for a random variable R with a well-defined expectation and variance. The proof of Theorem 4 is quite lengthy and can be found in [Hasan, 2008]. It is primarily based on the relationship between variance and standard deviation and many probability axioms.

4 Formal Verification of the Piling-Up Lemma

Linear cryptanalysis (LC) essentially consists of deciphering a cryptosystem using some of its $\langle$plaintext, ciphertext$\rangle$ pairs and the probabilities associated with the linear approximations of each of its modules. The probability p_i that the linear approximation of a single module i of a block cipher holds can be exhaustively evaluated using Equation (1). But the process of evaluating the probability associated with the linear approximation of the entire cipher is much more complicated. The main challenge here is the computation complexity of the probability term, which requires one to exhaust the possible combinations of plaintexts and keys. Obviously, such a requirement is infeasible for modern ciphers which employ large bit widths. Matsui [Matsui, 1993] proposed the Piling-up Lemma as a solution to this problem.

Lemma 1: *Let X_i $(1 \leq i \leq n)$ be independent random variables whose values are 0 with probability p_i or 1 with probability $1 - p_i$. Then the probability that $X_1 \oplus X_2 \oplus \ldots \oplus X_n = 0$ is*

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} e_i \tag{2}$$

where e_i, usually termed as the probability bias, represents the amount by which the probability p_i deviates from $\frac{1}{2}$, i.e., $p_i = \frac{1}{2} + e_i$.

Assuming that the linear approximations of all the modules in a block cipher are independent from one other, we can model them as the random variables X_i above and then the probability of the linear approximation of the entire block cipher can be directly evaluated using Equation (2). Matsui analyzed the DES cipher [Matsui, 1993] this way and since then it has become the de-facto method for LC of symmetric key block ciphers.

In this paper, we formally verify the Piling-up Lemma in HOL4. As a first step, we develop a higher-order-logic model of the XOR operation of a set of independent random variables $X_1 \oplus X_2 \oplus \ldots \oplus X_n = 0$. We formalize the XOR operation for multiple Boolean variables as follows:

Definition 1: *XOR operation for multiple Boolean Variables*

```
⊢ xor_list [] = F ∧
  ∀ h t. xor_list (h::t) =
         (h ∧ ¬(xor_list t)) ∨ (¬h ∧ (xor_list t)).
```

In the above definition, `h::t` denotes the list with head `h` and tail `t`. The function `xor_list` accepts a list of Boolean variables and recursively computes their XOR. It is important to note here that the function `xor_list` returns *False* (F) for the base case when its argument is an empty list [ ]. The reason behind this choice is that the XOR operation of any value x with *False* always returns x and thus it does not affect the output of the function `xor_list`.

By looking at the behavior of the X_i's in Lemma 1, it can be observed that each one of them is essentially a Bernoulli random variable with success probability equal to $1 - p_i$. Therefore, we formalize these n random variables as a list of Bernoulli random variables with distinct probabilities as follows:

Definition 2: *Bernoulli Random Variable List*

```
⊢ bernoulli_rv_list [] = [] ∧
  ∀ h t. bernoulli_rv_list (h::t) =
         bernoulli_rv h :: (bernoulli_rv_list t).
```

The function `bernoulli_rv_list` accepts a list of *real* numbers and returns a corresponding list of Bernoulli random variables, such that the probability of success for each one of these random variables is equal to the corresponding real numbers from the input list. The function `bernoulli_rv` above represents the formal model of the Bernoulli random variable that returns a $True$ with a probability equal to its argument, as explained in Section 3.

Now we define a list of independent random variables, as all the Bernoulli random variables in the Piling-up Lemma are independent.

Definition 3: *Independent Random Variable List*

```
⊢ indep_rv_list [] = unit [] ∧
  ∀ h t. indep_rv_list (h::t) =
         bind h (λx. bind (indep_rv_list t)
           (λy. unit (x::y))).
```

The function `indep_rv_list` accepts a list of random variables and returns the corresponding list of the same random variables such that the outcome of each one of these random variables is independent of the outcomes of all the others. This is done by recursively using the remaining portion of the infinite Boolean sequence of each random variable to model its subsequent random variable in the list using the monadic functions `unit` and `bind`. In other words, the value for the first random variable in the input list would be evaluated using a certain number, say n, of the top most bits of the initial infinite Boolean sequence B_i. The next random variable in the input list would be then evaluated using again a certain number, say m, of the remaining infinite Boolean sequence, i.e., $B_i - B_i[1-n]$ and so on and so forth. Thus, the infinite Boolean sequence bits that are utilized for modeling a random variable are never utilized again, which ensures the independence between the random variables.

Based on the above definitions, we are now in the position of formally expressing the XOR operation of a set of independent Bernoulli random variables that is used in the Piling-up Lemma as follows

Definition 4: *Piling-Up*

```
⊢ ∀ P. piling_up P = bind
           (indep_rv_list (bernoulli_rv_list P))
             (λx. unit (xor_list x)).
```

The function `piling_up` accepts a list of *real* numbers P that represents the probabilities of successes for Bernoulli random variables. It returns a Boolean type random variable that corresponds to the XOR operation between all the Bernoulli random variables with the given probabilities in the input list. The function `bernoulli_rv_list`, given in Definition 2, is used to construct a list of Bernoulli random variables, whereas each one of the Bernoulli random variables models a Bernoulli experiment with success probability equal to a distinct element of the input list of *real* numbers P. The function `indep_rv_list`, given in Definition 3, then transforms this list of Bernoulli random variables to a list of independent Bernoulli random variables. Finally, the function `xor_list`, given in Definition 1, evaluates the XOR of all the elements of the independent Bernoulli random variables in the list returned from the function `indep_rv_list`. It is important to note that the final output of the function `piling_up` is a pair with the first element equal to the result of the above-mentioned XOR operation and the second element equal to the remaining portion of the infinite Boolean sequence, which is consistent with the formal modeling of randomized functions approach outlined in Section 3. The usage of infinite Boolean sequence is implicit since monadic operators are used.

Definition 4 can now be used to formally express the Piling-up Lemma statement, given in Lemma 1, as the following higher-order-logic theorem:

Theorem 5: *Piling-Up Lemma*

$\vdash$ ∀P. (2≤LENGTH P)∧(∀e.mem e P ⇒ (-1/2 ≤ e) ∧ (e≤1/2))
⇒ (ℙ {s | fst (piling_up (prob_list P) s) = F} =
1/2 + ($2^{\text{length(P)}-1}$) list_mul P).

The theorem is universally quantified over a list of real numbers P, which represents the list of probability biases for all of the Bernoulli random variables in the Piling-up Lemma. The function `prob_list`, used in Theorem 5, recursively converts the list of probability biases to their corresponding success probabilities for the Bernoulli random variables:

Definition 5: *Probability Bias to Success Probability Conversion*

```
⊢ prob_list [] = [] ∧
  ∀ h t. prob_list (h::t) = (1 - (1/2 + h))::(prob_list t).
```

The function `list_mul`, used in Theorem 5, recursively models the multiplication of a list of real numbers as follows:

Definition 6: *Product of a list of Real Numbers*

```
⊢ list_mul [] = 1 ∧
  ∀ h t. list_mul (h::t) = h * (list_mul t).
```

The first assumption in Theorem 5 ensures that there are at least two elements in the probability bias list as the HOL4 function `length`, defined below, returns the length of its argument list.

Definition 7: *Length of a list*

```
⊢ length [] = F ∧
  ∀ h t. length (h::t) = length t + 1).
```

This minimum constraints has been placed because the XOR operation is only defined for two or more elements. The second assumption of Theorem 5 ensures that each member of the input list P, or the probability bias, lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ using another list function `mem`, which returns a $True$ if its first argument is an element of the list that is given as its second argument as follows:

Definition 8: *Member of a list*

```
⊢ ∀ x. mem x [] = F ∧
  ∀ x h t. mem x (h::t) = (x = h) ∨ mem x t).
```

This constraint is imposed in order to keep the probability terms within their interval $[0, 1]$. The conclusion of the theorem represents the Piling-up Lemma, given in Equation (2), using the probability theory fundamentals of Section 3.

We proceed with the proof of Theorem 5 by inducting on P, which generates the following subgoals after some simplifications.

Subgoal 5.1: `(∀e.mem e [a; b] ⇒ (-1/2 ≤ e) ∧ (e ≤ 1/2))`
`⇒ (ℙ {s | fst (piling_up (prob_list [a; b]) s) = F} =`
`1/2 + (2`$^{\mathtt{length([a;b])-1}}$`) list_mul [a; b]).`

Subgoal 5.2: `(∀e.mem e (a::b::P) ⇒(-1/2 ≤ e) ∧ (e ≤ 1/2)) ⇒`
`(ℙ {s | fst(piling_up (prob_list (a::b::P)) s) = F} =`
`1/2 + (2`$^{\mathtt{length(a::b::P)-1}}$`) list_mul (a::b::P))`
`⇒ ∀c.(∀e.mem e (a::b::c::P) ⇒ (-1/2≤e) ∧ (e≤1/2))⇒`
`(ℙ {s | fst(piling_up (prob_list (a::b::c::P)) s) = F} =`
`1/2 + (2`$^{\mathtt{length(a::b::c::P)-1}}$`) list_mul (a::b::c::P)).`

The first subgoal corresponds to the base case whereas the second subgoal is for the step case. We now tackle them one-by-one. Subgoal 1 represents the case when the list P consists of only two elements, i.e., $P = [a; b]$. After simplifying this subgoal with Definitions 1 to 5 and rewriting with Boolean Theory principles we are left with the following expression to prove in the HOL4 theorem prover.

Subgoal 5.1.1:
```
(∀e. mem e [a; b] ⇒ (-1/2 ≤ e) ∧ (e ≤ 1/2))
    ⇒ (ℙ {s | fst (bernoulli_rv (1 - (1 / 2 + a)) s) ∧
           fst (bernoulli_rv (1 - (1/2 + b))
               (snd (bernoulli_rv (1 - (1/2 + a)) s))) ∨
            ¬fst (bernoulli_rv (1 - (1/2 + a)) s) ∧
            ¬fst (bernoulli_rv (1 - (1/2 + b))
               (snd (bernoulli_rv (1 - (1/2 + a)) s)))} =
    1/2 + (2 a b)).
```

The set on the left-hand-side (LHS) of the above subgoal contains two independent Bernoulli random variables. We know that they are independent because the remaining portion of the infinite Boolean sequence, i.e, `(snd (bernoulli_rv (1 - (1 / 2 + a)) s))` is used to model the second Bernoulli random variable. In order to simplify this subgoal, we first verify the following equality:

Subgoal 5.1.2:
```
1/2 + (2 a b) =
  (1 - (1/2 + a)) (1 - (1/2 + b)) +
  (1 - (1 - (1/2 + a))) (1 - (1 - (1/2 + b))).
```

Using this result, we replace the right-hand-side (RHS) term of Subgoal 5.1.1. Next, we replace the terms `(1 - (1 / 2 + a))` and `(1 - (1 / 2 + b))` with variables `x` and `y`, such that both of these newly introduced variables lie in the interval $[0, 1]$ based on the assumption `(∀ e. mem e [a; b] ⇒ (-`$\frac{1}{2}$ `≤ e) ∧ (e ≤` $\frac{1}{2}$`))`. These simplifications, along with some set theoretic reasoning, allow us to rewrite Subgoal 1.1 as follows:

Subgoal 5.1.3: `(0 ≤ x) ∧ (x ≤ 1) ∧ (0 ≤ y) ∧ (y ≤ 1)`

$\Rightarrow$ ($\mathbb{P}$ {s | fst (bernoulli_rv x s)} $\cap$
{s | fst (bernoulli_rv y (snd (bernoulli_rv x s)))} $\bigcup$
$\overline{\text{{s | fst (bernoulli\_rv x s)}}}$ $\cap$
$\overline{\text{{s | fst (bernoulli\_rv y (snd (bernoulli\_rv x s)))}}}$ =
x y + (1 - x) (1 - y).

The LHS of the above subgoal can now be simplified based on the formally verified basic probability laws [Hurd, 2002], i.e., the additive law ($P(A \cup B) = P(A) + P(B)$), the law of independent events ($P(A \cap B) = P(A)P(B)$) and the complement law ($P(\overline{A}) = 1 - P(A)$). After this simplification, the subgoal can be discharged using the PMF relation of the Bernoulli random variable given in Theorem 1. This also concludes the proof of Subgoal 5.1.

Subgoal 5.2, obtained after performing induction on the variable P of Theorem 5, represents the step case where we want to verify the given expression holds for a list `(a::b::c::P)` for all values of variable c under the assumption that the given expression is $True$ for the list `(a::b::P)`. After simplifying with the definition of list function `mem` and some arithmetic rewriting, we obtain:

Subgoal 5.2.1: ($\forall$e.mem e (a::b::c::P) $\Rightarrow$ (-1/2$\leq$e) $\wedge$ (e$\leq$1/2)) $\wedge$
($\mathbb{P}$ {s | fst(piling_up (prob_list (a::b::P)) s) = F} =
1/2 + ($2^{\texttt{length(P)}+1}$) list_mul (a::b::P))
$\Rightarrow$ ($\mathbb{P}$ {s | fst(piling_up (prob_list (a::b::c::P)) s) = F} =
1/2 + ($2^{\texttt{length(P)}+2}$) list_mul (a::b::c::P)).

The second assumption given above is the key to verify the subgoal but it cannot be applied as is because the newly introduced element c is positioned in such a way that just rewriting the conclusion with Definitions 1 to 5 does not result in the list `(a::b::P)`, which is found in the assumption. Therefore, we used the commutativity and associativity of XOR and the independence of variables in the `piling_up` list to formally verify the following relationship:

Subgoal 5.2.2: $\mathbb{P}${s | fst(piling_up (prob_list(a::b::c::P)) s)=F} =
$\mathbb{P}${s | fst(piling_up (prob_list(c::a::b::P)) s)=F}.

Now after rewriting Subgoal 5.2.1 with the result of Subgoal 5.2.2 and simplifying the resulting subgoal with Definitions 1 to 5 we get

Subgoal 5.2.3: ($\forall$e. mem e (a::b::c::P)
$\Rightarrow$ (-1/2 $\leq$ e) $\wedge$ (e $\leq$ 1/2)) $\wedge$
($\mathbb{P}$ {s | fst (piling_up (prob_list (a::b::P)) s) = F} =
1/2 + ($2^{\texttt{length(P)}+1}$) list_mul (a::b::P))
$\Rightarrow$ ($\mathbb{P}$ {s | fst (bernoulli_rv (1 - (1/2 + c)) s) $\wedge$
fst (piling_up (prob_list (a::b::P))
(snd (bernoulli_rv (1 - (1/2 + c)) s))) $\vee$

```
¬fst (bernoulli_rv (1 - (1/2 + c)) s) ∧
¬fst (piling_up (prob_list (a::b::P)))
       (snd (bernoulli_rv (1 - (1/2 + c)) s))} =
1/2 + (2^(length(P)+2)) list_mul (a::b::c::P)).
```

The above subgoal is very similar to Subgoal 5.1.1 except the fact that instead of one of the Bernoulli random variables, we have the variable (piling_up (prob_list (a::b::P)). Therefore, just like Subgoal 5.1.1., the proof of Subgoal 5.2.3 is primarily based on the basic probability laws, the PMF relation for the Bernoulli random variable given in Theorem 1, and the PMF relation for the random variable (piling_up (prob_list (a::b::P)) which can be extracted from its assumption list using the complement law of probability. This proof also concludes the formal verification of the Piling-up Lemma, given in Theorem 5.

The formal verification of the Piling-up Lemma ensures the correctness of our formalization of the Piling-up principle, given in Definition 4. Besides that, the formally verified statement of the Piling-up Lemma can also be built upon to formally reason about interesting LC properties within the sound core of HOL4 theorem prover, as will be illustrated in Section 6 of this paper.

5 Probabilistic Analysis of the Key-Deciphering algorithm

Matsui's algorithm, given in Algorithm 1, is based on the maximum-likelihood method and can be used to obtain one bit of information about the key, i.e., $(\bigoplus_{k\in c}\mathcal{K}_k)$, once a linear expression of the entire cipher is obtained [Matsui, 1993]. By observation, it can be deduced that the success rate of the above algorithm would increase with N or e, which denotes the probability bias $|p - \frac{1}{2}|$. This deduction forms the basis of LC and has been used to break many block ciphers. In this paper, we formally reason about this result using the probabilistic theorem proving based approach, outlined in Section 3. Algorithm 1 is formalized as follows:

Definition 9: *Key-Deciphering Algorithm*

```
⊢ ∀ N p. key_deciphering N p s =
    (xor (fst (binomial_rv N p s) > N/2) (p > 1/2)).
```

The function key_deciphering accepts three parameters: N p and s that represent the variables N and p used in Algorithm 1 and the infinite Boolean sequence, which is the source of randomness in our formalization. It returns a Boolean quantity $\bigoplus_{k\in c}\mathcal{K}_k$ by modeling variable T of Algorithm 1 as the Binomial random variable with parameters N and p.

In this paper, we analyze Algorithm 1 under the assumptions that $e > 0$ and $\bigoplus_{k\in c}\mathcal{K}_k = False$, which have been adapted from Junod's [Junod, 2000] informal paper-and-pencil based analysis of the same algorithm. We formally

Algorithm 1 Key-Deciphering Algorithm

$N :=$ Total number of $\langle \text{plaintext}, \text{ciphertext} \rangle$ pairs
$p :=$ Probability that the linear expression given in Equation (1) holds
$T :=$ Number of plaintexts for which the LHS of Equation (1) is $False$
if $(T > \frac{N}{2})$ then
    if $(p > \frac{1}{2})$ then
        $\bigoplus_{k \in c} \mathcal{K}_k = False$
    else
        $\bigoplus_{k \in c} \mathcal{K}_k = True$
else
    if $(p > \frac{1}{2})$ then
        $\bigoplus_{k \in c} \mathcal{K}_k = True$
    else
        $\bigoplus_{k \in c} \mathcal{K}_k = False$

reason about the following relationship between the probability of success of Algorithm 1 and the parameters N and e.

Theorem 6: *Lower Bound on the Probability of Success of Algorithm 1*

$\vdash \forall$ `k N p e.0 < N` $\wedge$ `(p=1/2 - e)` $\wedge$ `0 < e` $\wedge$ `e < 1/2` $\wedge$ `(k=F)`
$\Rightarrow$ `1 - (1/N)(1/(4 e`^{2}`) - 1)` $\leq$
$\mathbb{P}$ $\{$`s | (key_deciphering N p s) = k`$\}$.

The first assumption ensures that we have at least one $\langle \text{plaintext}, \text{ciphertext} \rangle$ pair, while the others define the relationship between variables `e` and `p` and the above-mentioned constraints under which we analyze the algorithm. The RHS of the above inequality represents the success probability of Algorithm 1 as it gives the probability when the algorithm returns `k`, which represents $\bigoplus_{k \in c} \mathcal{K}_k$. The LHS of the inequality is an expression in terms of variables `N` and `e`, the value of which increases when the value of any one of these two variables increases.

We proceed with the proof of Theorem 6 by first rewriting the success probability term with Definition 9, simplifying it based on the given assumptions and the complement law of probability and rearranging the terms as follows:

Subgoal 6.1: `0 < N` $\wedge$ `0 < e` $\wedge$ `e < 1/2`
$\Rightarrow$ $\mathbb{P}$ $\{$`s | N/2 < (fst (binomial_rv N (1 / 2 - e) s))`$\}$ $\leq$
`(1/N)(1/(4 e`^{2}`) - 1)`.

The LHS of the above subgoal can be rewritten based on the expectation and variance relations of the Binomial random variable, given in Theorems 2 and 3, respectively, and some arithmetic reasoning as follows:

Subgoal 6.2: $\texttt{0 < N} \wedge \texttt{0 < e} \wedge \texttt{e < 1/2}$
$\Rightarrow \mathbb{P}$ {s | fst ((λs. binomial_rv N (1/2 - e) s) s) -
expec (λs. binomial_rv N (1/2 - e) s) > N e} $\leq$
(variance (λs. binomial_rv N (1/2 - e) s))/(N e)^2.

The above subgoal can now be discharged by using the increasing law of probabilities ($A \subseteq B \Rightarrow Pr(A) \leq Pr(B)$) and the Chebyshev's inequality, given in Theorem 4. This also concludes the proof of Theorem 6.

Theorem 6 provides a formal proof of Matsui's deduction that the probability of success of Algorithm 1 increases with both N and e. The HOL4 proof script, corresponding to the formalization and verification presented in Sections 4 and 5, is available at [Hasan, 2014].

Just like the Piling-Up lemma, the key deciphering algorithm is also a fundamental concept in LC and thus its formal verification confirms the accuracy of LC research that is based on this principle. Similarly, this verification paves the path to reason about many interesting LC properties using theorem proving.

6 Application: Formal Reasoning about LC Properties

To illustrate the usability and practical effectiveness of the proposed formalization, we use it to reason about some of the commonly used LC characteristics in this Section.

The first property that we verify states that the probability of linear approximation of a cipher is equal to one half, i.e., it can never be broken with the given linear approximation, if the probability associated with the linear approximation of at least one of its submodule is exactly equal to one half. The Piling-up Lemma is the core foundation that deals with the above-mentioned relationship of probabilities of linear approximations between the cipher and its submodules and thus would be used to verify it. Therefore, the given characteristic can be formalized as follows:

Theorem 7: *Probability of Linear Approximation is equal to 1/2*
$\vdash \forall$ P. (2 $\leq$ LENGTH P) $\wedge$
($\forall$ e. mem e P $\Rightarrow$ (-1/2 $\leq$ e) $\wedge$ (e $\leq$ 1/2)) $\wedge$
($\exists$ e. mem e P $\wedge$ (e = 0))
$\Rightarrow$ ($\mathbb{P}$ {s | fst (piling_up (prob_list P) s) = F} = 1/2).

The third assumption ensures that at least one element of the probability bias list would be zero and thus, from Definition 5, the corresponding probability of linear approximation would be exactly equal to $\frac{1}{2}$. The proof of the above theorem is a direct instantiation of Theorem 5 along with some basic arithmetic reasoning and was done in one HOL4 rewriting step.

Next, we want to formally reason about the impact of the total number of $\langle$plaintext, ciphertext$\rangle$ pairs N and the probability that the linear expression, given in Equation (1), holds p on the success of Matsui's Key-Deciphering algorithm. We first verify that the success probability of Matsui's deciphering algorithm approaches 1 when p approaches to 0.

Theorem 8: *Success Probability of Matsui's Deciphering Algorithm when p tends to 0*

$\vdash \forall$ `n. (0 < n)`
$\Rightarrow$ `(right_lim (`λ`p.`$\mathbb{P}\{$`s | key_deciphering n p s = F`$\}$`) 1 0).`

The predicate `right_lim f l a`, formalized in [Hasan et al., 2009b], represents the *limit from the right* $lim_{x \to a^+} f(x) = l$, i.e., the limit value of the function f at point a is l when its argument x decreases in value approaching a. Theorem 8 basically states that the success probability of the Matsui's Key-Deciphering algorithm, formalized in Definition 9, approaches 1 as its argument `p` approaches 0 when we have at least one $\langle$plaintext, ciphertext$\rangle$ pair. We proceed with the proof of Theorem 8 by first rewriting the proof goal with the definition of the function `right_lim f l a`, which after simplification gives us the following subgoal

Subgoal 8.1: `0 < n` $\wedge$ `0 < e`
$\Rightarrow \exists$`d. 0 < d` $\wedge$ `(`$\forall$`x. 0 < x` $\wedge$ `x < d` $\Rightarrow$
`abs(`$\mathbb{P}$ $\{$`s |` $\neg$`(key_deciphering_algo n x s)`$\}$ `- 1) < e).`

where `abs` is the absolute function in HOL. Next, we remove the existential quantifier on the variable d using the term `(1/2) - (1/2) * (1/(sqrt(1 + (n) * e)))`, where the HOL4 function `sqrt` models the square root of a real number. This term is positive, which allows us to discharge the first conjunct in the above subgoal. Using arithmetic simplification and the fact that the probability can never be greater than 1, the second conjunct of Subgoal 8.1 can be rewritten as follows:

Subgoal 8.2: `0 < n` $\wedge$ `0 < e` $\wedge$ `0 < x`
$\wedge$ `x < (1/2) - (1/2) * (1/(sqrt(1 + (n) * e)))` $\Rightarrow$
`1 - e <` $\mathbb{P}$ $\{$`s |` $\neg$`(key_deciphering_algo n x s)`$\}$.

The above inequality can now be verified using Theorem 6 and some arithmetic reasoning, which concludes the proof of Theorem 8.

Now, to reason about the impact of the total number of $\langle$plaintext, ciphertext$\rangle$ pairs N on the success of Matsui's deciphering algorithm, we prove that as N becomes very large, the success probability of Matsui's deciphering algorithm approaches 1.

Theorem 9: *Success Probability of Matsui's Deciphering Algorithm when n tends to* ∞

$$\vdash \forall\ \mathtt{p\ e.}\ (\mathtt{p} = \tfrac{1}{2} - \mathtt{e}) \wedge (0 < \mathtt{e}) \wedge (\mathtt{e} < \tfrac{1}{2}) \Rightarrow (\mathtt{lim}\ (\lambda \mathtt{n}.\ \mathbb{P}\ \{\mathtt{s} \mid \mathtt{key\_deciphering\ n\ p\ s = F}\}) = 1).$$

The function `lim f`, formalized in [Harrison, 1998], represents the *limit of a real sequence* $lim_{n\rightarrow\infty} f(n)$, i.e., the limit value of a real sequence f as its natural number arguments becomes very large. Theorem 9 states that the success probability of the Matsui's Key-Deciphering algorithm, formalized in Definition 9, approaches 1 as its natural number argument `n` becomes very large under the assumptions when `p` lies in the interval $(0, \frac{1}{2})$. The proof of Theorem 9 is very similar to the one of Theorem 8.

The straightforward formal verification of the above LC characteristics clearly demonstrates the effectiveness of the formalization of the previous two sections. It is also important to note that all the above properties are generic and independent of the cipher behavior. In case, we want to reason about properties specific to a given cipher, the cipher's behavior can be expressed in higher-order logic and the corresponding bias list can then be reasoned about, which in turn can be used to formally verify interesting LC properties.

7 Conclusions

This paper presents the formal verification of the Piling-up Lemma and Matsui's Key-Deciphering algorithm using the HOL4 theorem prover. These results are of foundational nature in the LC domain and thus their formalization facilitates the proposed approach for analyzing security aspects of block ciphers within the sound core of a higher-order-logic theorem prover. Accuracy is the main strength of our approach, which is a very useful feature while analyzing block ciphers deployed in safety or financial-critical systems. For illustration purposes, we used our formalization infrastructure to formally reason about a few LC properties in HOL4.

To the best of our knowledge, this paper represents the first attempt to formally reason about fundamental LC properties in a mechanical theorem prover. Our formalization opens the doors to many new areas in the direction of theorem proving based LC. First of all, we plan to build upon our formalization infrastructure to formally analyze security aspects of the DES cipher, which also served as a case study in Matsui's seminal paper on LC [Matsui, 1993]. Similarly, LC of other modern block ciphers, such as AES, MARS, Twofish, RC6, Serpent, IDEA, and TEA, can also be performed in HOL4. Another useful direction to explore would be to use one model of a block cipher to formally analyze both the functional and the security properties. Similar to our formalization of LC, differential cryptanalysis algorithms and lemmas can also be formalized in higher-order logic to facilitate the development of a formal differential cryptanalysis infrastructure.

References

[Abadi and Cortier, 2006] Abadi, M., Cortier, V.: "Deciding Knowledge in Security Protocols under Equational Theories"; Theoretical Computer Science; 367 (2006), 1, 2–32.

[Affeldt et al., 2007] Affeldt, R., Tanaka, M., Marti, N.: "Formal Proof of Provable Security by Game-Playing in a Proof Assistant"; Provable Security; volume 4784 of LNCS; 151–168; Springer, 2007.

[Baier et al., 2003] Baier, C., Haverkort, B., Hermanns, H., Katoen, J.: "Model Checking Algorithms for Continuous time Markov Chains"; IEEE Transactions on Software Engineering; 29 (2003), 4, 524–541.

[Biham et al., 2001] Biham, E., Dunkelman, O., Keller, N.: "Linear Cryptanalysis of Reduced Round Serpent"; Fast Software Encryption Workshop; 16–27; Springer, 2001.

[Blanchet and Pointcheval, 2006] Blanchet, B., Pointcheval, D.: "Automated Security Proofs with Sequences of Games"; Advances in Cryptology - CRYPTO; volume 4117 of LNCS; 537–554; Springer, 2006.

[Daemen and Rijmen, 2002] Daemen, J., Rijmen, V.: The Design of Rijndael; Springer, 2002.

[Duan et al., 2005] Duan, J., Hurd, J., Li, G., Owens, S., Slind, K., Zhang, J.: "Functional Correctness Proofs of Encryption Algorithms"; Logic for Programming, Artificial Intelligence, and Reasoning; volume 3835 of LNCS; 519–533; 2005.

[Galambos, 1995] Galambos, J.: Advanced Probability Theory; Marcel Dekker Inc., 1995.

[Gordon, 1989] Gordon, M.: "Mechanizing Programming Logics in Higher-0rder Logic"; Current Trends in Hardware Verification and Automated Theorem Proving; 387–439; Springer, 1989.

[Harrison, 1998] Harrison, J.: Theorem Proving with the Real Numbers; Springer, 1998.

[Harrison et al., 2006] Harrison, J., Slind, K., Arthan, R.: "HOL"; The Seventeen Provers of the World; volume 3600 of LNCS; 11–19; Springer, 2006.

[Hasan, 2008] Hasan, O.: Formal Probabilistic Analysis using Theorem Proving; PhD Thesis; Concordia University; Montreal, QC, Canada (2008).

[Hasan, 2014] Hasan, O.: "Towards Formal Linear Cryptanalysis using HOL Theorem Prover - HOL code"; (2014); http://save.seecs.nust.edu.pk/projects/lc.html.

[Hasan et al., 2009a] Hasan, O., Abbasi, N., Akbarpour, B., Tahar, S., Akbarpour, R.: "Formal Reasoning about Expectation Properties for Continuous Random Variables"; Formal Methods; volume 5850 of LNCS; 435–450; Springer, 2009.

[Hasan et al., 2009b] Hasan, O., Afshar, S. K., Tahar, S.: "Formal Analysis of Optical Waveguides in HOL"; Theorem Proving in Higher-Order Logics; volume 5674 of LNCS; 228–243; Springer, 2009.

[Hurd, 2002] Hurd, J.: Formal Verification of Probabilistic Algorithms; PhD Thesis; University of Cambridge; UK (2002).

[Nakahara, 2007] Nakahara, J.: "A Linear Analysis of Blowfish and Khufu"; Information Security Practice and Experience; volume 4464 of LNCS; 20–32; Springer, 2007.

[Junod, 2000] Junod, P.: Linear Cryptanalysis of DES; Master's thesis; Swiss Institute of Technology (2000).

[Matsui, 1993] Matsui, M.: "Linear Cryptanalysis Method for DES Cipher"; EUROCRYPT: Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology; volume 765 of LNCS; 386–397; Springer, 1993.

[Misra and Saha, 2009] Misra, J., Saha, I.: "A Reinforcement Model for Collaborative Security and its Formal Analysis"; New Security Paradigms Workshop; 101–114; ACM, 2009.

[Rutten et al., 2004] Rutten, J., Kwaiatkowska, M., Normal, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilisitc Systems; volume 23 of CRM Monograph Series; American Mathematical Society, 2004.

[Shorin et al, 2001] Shorin, V., Jelezniakov, V., Gabidulin, E.: "Linear and Differential Cryptanalysis of Russian GOST"; Electronic Notes in Discrete Mathematics; 6 (2001), 538–547.

[Smith and Dill, 2008] Smith, E., Dill, D.: "Automatic Formal Verification of Block Cipher Implementations"; Formal Methods in Computer Aided Design; 1–7; IEEE, 2008.

[Stamp and Low, 2007] Stamp, M., Low, R. M.: Applied Cryptanalysis: Breaking Ciphers in the Real World; Wiley-IEEE Press, 2007.

[Steel, 2006] Steel, G.: "Formal Analysis of PIN Block Attacks"; Theoretical Computer Science; 367 (2006), 1-2, 257–270.

[Zhang et al., 2008] Zhang, L., Zhang, W., Wu, W.: "Cryptanalysis of Reduced-Round SMS4 Block Cipher"; Information Security and Privacy; volume 5107 of LNCS; 216–229; Springer, 2008.