International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

A Formal Approach for Proof Constructions in Cryptography

Markus Kaiser, Johannes Buchmann

Abstract**—** In this article we explore the application of a formal proof system to verification problems in cryptography. Cryptographic properties concerning correctness or security of some cryptographic algorithms are of great interest. Beside some basic lemmata, we explore an implementation of a complex function that is used in cryptography. More precisely, we describe formal properties of this implementation that we computer prove. We describe formalized probability distributions (σ-algebras, probability spaces and conditional probabilities). These are given in the formal language of the formal proof system Isabelle/HOL. Moreover, we computer prove Bayes' Formula. Besides, we describe an application of the presented formalized probability distributions to cryptography. Furthermore, this article shows that computer proofs of complex cryptographic functions are possible by presenting an implementation of the Miller-Rabin primality test that admits formal verification. Our achievements are a step towards computer verification of cryptographic primitives. They describe a basis for computer verification in cryptography. Computer verification can be applied to further problems in cryptographic research, if the corresponding basic mathematical knowledge is available in a database.

Keywords**—** prime numbers, primality tests, (conditional) probability distributions, formal proof system, higher-order logic, formal verification, Bayes' Formula, Miller-Rabin primality test.

I. Introduction

MATHEMATICAL proofs are often complex and hard to verify by their readers. Consequently, the application of formal proof systems are a useful approach in the area of verification. Formal and computer verification augment the traditional concept of software engineering by providing techniques that guarantee trustiness as well as correctness of software systems in a mathematical way. There are many possible applications of formal and computer verification like automotive, medical technology, information technology security and cryptography.

In cryptographic research, the concept of provable security is of great interest to ensure, whether a given cryptographic primitive can be regarded as secure, or its behaviour is not as intended. During the last ten years, the relevance of security proofs increased rapidly. Great improvements in that direction were achieved, e.g. by proving the Optimal Asymmetric Encryption Padding (OAEP) for the case of RSA ([4]), which is widely used in practice. Finding a proof of an encryption scheme is in general done by considering the computational complexity of the regarded cryptographic primitive. Proof constructions in the complexity theoretic approach consider reductions from the problem of breaking a cryptographic primitive to the underlying (probably) hard problem. These proof constructions are typically made by human work. Consequently, a proof about security of a cryptographic primitive may contain some incorrectness. An example of this consequence (with respect to OAEP) was described in [8].

Another direction in cryptographic verification is the formal approach that can be applied to protocol verification. With respect to this view on proofs, cryptographic operations are handeled as perfect cryptographic operations. Consequently, this approach can not be used to prove a cryptographic primitive secure. But proof constructions with respect to the formal view can gain from computer support that is possible, for example by using a formal proof system.

A. Idea

Our idea related to the above described problem is to construct proofs of cryptographic primitives like encryption schemes in the complexity theoretic approach with computer support. In this case, formal computational complexity and formal probability theory has to be studied. For this purpose, we use the proof assistant Isabelle/HOL, which is successfully applied in the *Verisoft project* [1].

B. Contents

In this article, we describe formalized probability distributions (σ-algebras, probability spaces and conditional probabilities). These are given in the formal language of the formal proof system Isabelle/HOL. Moreover, we computer prove (a formalized version of) Bayes' Formula. Besides we describe an application of the presented formalized probability distributions to cryptography.

The correctness of the implementation of cryptographic functions such as encryption and digital signature is crucial for the security of computer systems. But in general, such functions are very complex which makes formal proofs very hard. This article shows that computer proofs of complex cryptographic functions are possible by presenting an implementation of the Miller-Rabin primality test that admits formal verification. This test is a key ingredient of RSA and DSA key generation. We describe its computer verification with Isabelle/HOL. More precisely, it is possible to formally prove

$$k \text{ prime}, x < k, \gcd(x,k) = 1 \implies prim(x,k) = 1,$$

and

$$\text{All primality conditions hold for } k, x \Leftrightarrow prim(x,k) = 1.$$

The authors are with the Technische Universität Darmstadt, 64289 Darmstadt, Germany. This work was partially funded by the German Federal Ministry of Education and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the authors.

[1]Verisoft is a German industrial research project, where formal mathematics, as well as computer science are applied to engineering.

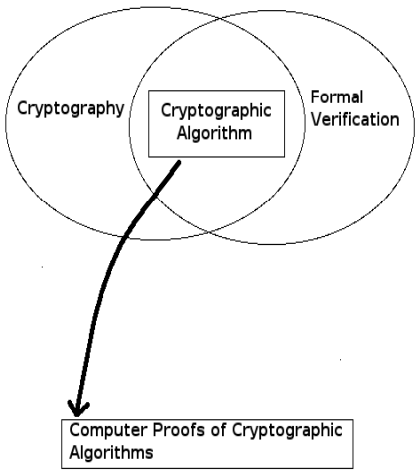

Fig. 1. Cryptography and Formal Verification

for an implementation $prim$. Besides, this implementation will be part of a fully verified cryptography client.

In the remaining part of this paper, we give a computer proven version of Bayes' Formula and we show how a formal proof system can be used to prove the correctness of an implementation of a primality test. As a result, a verified applicable function and formally proven properties, that extend the used database, are received. For this purpose, we give a new functional/logical description (and implementation) of the well known Miller-Rabin primality test. This description is written down and (interactively) verified in the formal proof language of Isabelle/HOL. Moreover, this implementation can be integrated into a cryptographic client that is implemented and verified in the used formal proof language.

Altogether we prove that formal verification with computer support of cryptographic relevant algorithms improves cryptographic practice by providing correct implementations, as well as cryptographic research by extending a formal database for the purpose of cryptography. Both aspects are illustrated below (Figure 1).

C. Overview

This paper is organized as follows: We start in II with a description of the used formal proof system. In III we explore Bayes' Formula with a formal proof system, and in IV we apply formalized probabilities in cryptography. Moreover, in V we give some mathematical definitions, that describe a basis of a formal description and verification of the Miller-Rabin algorithm. A formal verification of the Miller-Rabin algorithm is described in VI. Besides, this paper contains formal definitions of functions describing the Miller-Rabin algorithm, computer verified properties of these functions, and further explanations. In VII some conclusions, as well as some comments on future work are given.

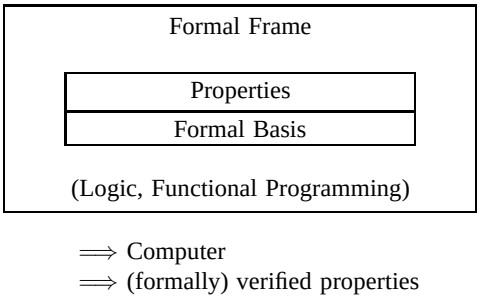

Fig. 2. Construction pattern in Isabelle/HOL described in II-B

II. Formal Frame

A formal frame usually provides a formal language where formal objects, as well as relations between these objects formally are described. That means, a formal property can be proven by using formally defined proof constructions. Consequently, a formal proof provides a clear structure that can be analyzed to minimize the number of errors. UML, Petri-nets, automata theory or logic are examples for formal frameworks used in computer science.

A. Formal Proof Systems

Formal proof systems provide computer support for formal verification. But the correctnes of a formal proof using a formal proof system relies on the correctness of the applied computer system. A formal proof system works on automated or interactive proof constructions. In the following, the interactive formal proof system Isabelle/HOL is described.

B. The Isabelle System

In the following, we give a short description of the verification tool Isabelle/HOL. For the reason of a better understanding of our formal version of (conditional) probability distributions, Bayes' Formula, cryptographic application of probability, and implemented version of the Miller-Rabin algorithm, which are formulated and verified in the formal language of Isabelle/HOL, we want to explain the main structures of the Isabelle proof system. It is a proof assistant for higher-order logic, which can be used for interactive proof constructions, formal specifications, as well as verification in higher-order logic and functional programming.

The formal language that consists of higher-order logic and functional programming, is used to give definitions and lemmata, which are based on a large database. These definitions and (proven) lemmata can be used to prove further lemmata and theorems, which results in an augmented data base for the purpose of building up new theories (compare to Figure II-B).

More information about Isabelle/HOL (that is successfully applied in the *Verisoft project* (http://www.verisoft.de)) are

given in [7], which describes constructions with this tool. A further useful reference is [9]. There, parts of the large database are mapped. Besides [9] contains other references about Isabelle/HOL.

III. Formal Probability Theory and Bayes' Formula

In this part some fundamental definitions and properties about probability distributions are described. These definitions and properties are often necessary for proofs in the area of provable security. Below, we present a formal description of the uniform distribution. Furthermore we describe formalized probability distibutions in general, and properties of probability distributions, which are needed to formally prove a useful lemma for the purpose of cryptography (Section IV).

A. Introduction

In the following, we give a computer proven version of Bayes' Formula that is formalized in the formal proof language of the used formal proof system. Bayes' Formula is a useful lemma to compute probabilities (for example computation of probabilities in cryptography). That means, a formalized version of this lemma is useful for formal proofs in cryptography.

In order to computer prove Bayes' Formula we explore formal definitions of basic mathematical concepts (σ-algebra, probability space, conditional probability).

Review: A probability space $(\Omega, \mathcal{A}, P)$ is built up of a set Ω (set of results), a σ-algebra $\mathcal{A}$ of Ω (system of possible events) and a probability distribution $P : \Omega \to [0, 1]$. A σ-algebra $\mathcal{A}$ of Ω is a system of carriers from Ω ($\mathcal{A} \subseteq \mathcal{P}(\Omega)$). In the following these components of a probability space (i.e. their formal description) are presented (compare [1], [3], [5], [6] for more mathematical background).

B. Classical Probability Theory

The following lines contain a description of the uniform distribution and its formal specification in the proof language of Isabelle/HOL.

1) The Uniform Distribution: The *uniform distribution* $P : \mathcal{P}(\Omega) \to [0, 1]$ is given as follows,

$$P(A) = \frac{|A|}{|\Omega|},$$

$A \subseteq \Omega$, and Ω is a finite non-empty set, i.e. $(\Omega, \mathcal{P}(\Omega), P)$ is the corresponding probability space.

For a given probability space $(\Omega, \mathcal{P}(\Omega), P)$ many properties are well known and are well described in the literature about probability theory. But verification of corresponding formal properties needs further investigation.

2) Formal Specification: The uniform distribution and its properties are well known. As mentioned before, the main reason for its presentation in this aricle is to give a formalized version of the uniform distribution and some of its properties.

This formal specification given in the proof language of Isabelle/HOL is based on a function *prb* representing the uniform distribution. A function in this proof language is declarated as constant with arguments of given datatypes. Moreover, a function can be defined in a mathematical way.

consts
prb :: " 'a set => 'a set => real"

defs
prb_def: "prb E A == real (card A) / real (card E)";

Besides the datatype *real* representing the real numbers, there is a function *real*, which converts an argument of a type describing natural numbers to the type representing real numbers. The function *card* returns the cardinality of its argument of type *set*.

In addition to this definition of the uniform distribution, we formalized and proved several properties of the uniform distribution within the proof language of Isabelle/HOL. The verification of these properties needed further proven properties, e.g. properties about real numbers or sets.

C. Axiomatic Probability Theory

Below our formalized version of some fundamental definitions, lemmata and theorems from probability theory is described. This formal description is given in a mathematical notation, furthermore examples given in the Isabelle proof language are added.

The following formal description of a σ-algebra, probability space, conditional probability and some related properties are based on the Isabelle theory *Main*, and Isabelle theories about real numbers and sets, respectively. These Isabelle theories contain many basic properties about number theory, real numbers and sets.

1) Probability Distributions: We formalized a σ-algebra $\mathcal{A}$ of a set E as a predicate sigma_algebra with a set of a fixed type and a set of sets of the same type as arguments. The declaration and definition of this predicate (constant) in the formal language of the Isabelle system is given as follows.

consts sigma_algebra :: "['a set, 'a set set] => bool";

defs
sigma_algebra_def:
" sigma_algebra E A == E : A ∧ (ALL K : A. E-K : A)
∧ (ALL B. ALL (i::nat). (B i) : A ⟶ (⋃ i. B i) : A)";

This constant describes a predicate, which holds, if and only if the second argument A is a σ-algebra of the set E (first argument).

Moreover, we described a probability space through the declaration and definition of another predicate.

consts
pr_space :: "['a set, 'a set set, ('a set => real)] => bool";

defs
pr_space_def:
"pr_space E F Pr == (sigma_algebra E F ∧ E ≠ {}
∧ (ALL A : F. 0 ≤ (Pr A))
∧ (Pr E) = 1 ∧ (ALL A : F. ALL B : F. ((A ∩ B = {})
⟶ (Pr (A ∪ B)) = (Pr A) + (Pr B))))";

This predicate holds, if and only if its arguments form a probability space.

Besides we formalized the term of conditional probability in the formal language of Isabelle. Therefore the following predicate describes conditional probabilities.

consts cond_pr :: "['a set, 'a set set, ('a set => real), (['a set, 'a set] => real)] => bool";

defs
cond_pr_def:
"cond_pr E Alg Pr CoPr == pr_space E Alg Pr
∧ (ALL A : Alg. ALL Z : Alg. 0 < Pr Z
⟶ CoPr A Z = Pr (A ∩ Z) / Pr Z)";

D. Fundamental Lemmata and Bayes' Formula

In the following fundamental lemmata about probability distributions and our version of Bayes' Formula are presented.

1) Fundamental Lemmata: Because our aim of formalizing σ-algebras, probability distributions and conditional probabilities was to build up a data base of useful (proven) properties for their application in proofs about cryptography, we formulated and proved fundamental lemmata about probability spaces. Our formal definition of a predicate describing a probability space is based on a formal definiton of a σ-algebra, i.e. formal proofs about probability spaces need lemmata about σ-algebras. Consequently, we formulated and proved the needed properties.

Example:

$\mathcal{A}$ σ-algebra of E, $A_0 \in \mathcal{A}$, $A_1 \in \mathcal{A} \Longrightarrow A_1 \setminus A_0 \in \mathcal{A}$.

In the proof language of Isabelle this property is formalized as follows.

lemma "[| sigma_algebra E A; A0 ∈ A; A1 ∈ A |] ==> (A1 - A0) ∈ A";

With this lemma and further lemmata we were able to prove fundamental properties of probability distributions. While some of these properties are formal consequences of our definition of a probability space, the others need more care to be proven formally. In the lines below a formal description of these properties within the proof language of Isabelle is given.

lemma "[| pr_space E A Pr; A0 ∈ A |] ==> Pr (E-A0)";

lemma "[| pr_space E A Pr; A0 ∈ A |] ==> Pr A0 ≤ 1";

lemma " pr_space E A Pr ==> Pr {} = 0 ";

lemma "[| pr_space E A Pr; A0 ∈ A; A1 ∈ A; A1 ⊆ A0 |] ==> Pr (A0-A1) = Pr A0 - Pr A1";

lemma "[| pr_space E A Pr; A0 ∈ A; A1 ∈ A; A1 ⊆ A0 |] ==> Pr A1 ≤ Pr A0";

2) Bayes' Formula: Besides a formal definition of conditional probabilities, we have formalized some lemmata and Bayes' Formula.

$\mathcal{A}$ σ-algebra of E, $A_0 \in \mathcal{A}$, $A_1 \in \mathcal{A}$, and $0 < Pr(A_0), 0 < Pr(A_1)$

$\Longrightarrow Pr(A_0|A_1) = \frac{Pr(A_1|A_0) \cdot Pr(A_0)}{Pr(A_1)}$.

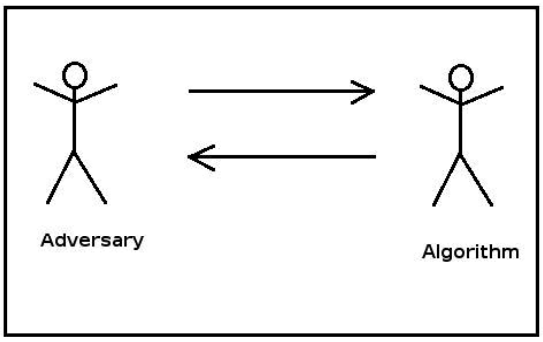

Fig. 3. Proof in Cryptography: $m_b \in \{m_0, m_1\}$, encryption of $m_b \longrightarrow$ attacker tries to find out, if $b = 0$ or $b = 1$

Our formalized version of Bayes' Formula within the proof language of Isabelle is given below.

lemma: "[| cond_pr E A Pr CoPr;
A0 ∈ A; A1 ∈ A; 0 < Pr A0; 0 < Pr A1 |] ==>
CoPr A0 A1 = (CoPr A1 A0) ∗ (Pr A0) / (Pr A1)";

IV. Cryptographic Application

In the lines below, we describe an application of the above presented formal framework to cryptography. Besides the explored application of formalized probability theory to cryptography, there are further possibilities of application (the term of *perfect secrecy* and related proofs provide an interesting area of application in provable security, but a deeper discussion won't be given at this place).

A. Bellare-Rogaway

In [2] a well known proof technique is described (Bellare-Rogaway model, 1993; compare Figure 3). The following lines give a definition of semantical security against chosen ciphertext attack (IND-CCA).

1) IND-CCA:

Definition 4.1: IND-CCA

A given encryption scheme with security parameter z is IND-CCA in the Bellare-Rogaway model, if the following holds for all polynomial time algorithms A and for all $f \in \mathbf{Z}$ with $f > 0$.

For f there is x, for all $z > x$:

IND-CCA: $\mid 2 \cdot Pr(A \text{ finds correct bit}) - 1 \mid = Adv(A) < \frac{1}{z^f}$,

where the adversary is represented by A.

In this definition, the term $Pr(A$ finds correct bit$)$ specifies the probability an algorithm A has to learn, whether a certain bit is equal to 0 or 1, if it attacks a cryptosystem with security parameter z.

Below we want to define the term of IND-CCA in the formal language of Isabelle/HOL.

2) Formal Definition of IND-CCA: The following formal definition of IND-CCA depends on our version of formalized probability theory (which is based on the theory *Main* and on theories about real numbers and sets).

consts IND_CCA ::
"['a set, 'a set set, ('a set => real), 'a set] => bool";

defs IND_CCA_def:
" IND_CCA E Alg Pr A == pr_space E Alg Pr
∧ ∀ (f::int). 0 < f ⟶ (∃ (x::int). 0 < x ⟶
(∀ (z::int). x < z ⟶ | 2 ∗ (Pr A) - 1 | < 1/z^f))";

In the lines above IND_CCA represents a predicate, that describes the dependencies between the probability of A (represented by the term *Pr A*) and the value of z^f.

B. Application of Formal Probability Theory

Below we describe a useful lemma for the purpose of cryptography, which we formalized and proved in the proof language of Isabelle/HOL. For this purpose, several properties of sets, σ-algebras, probability distributions, and conditional probabilities, respectively, are proven with the Isabelle system.

Lemma 4.1: $A_0, A_1 \in \mathcal{A}$ (where $\mathcal{A}$ is a σ-Algebra of E), and $Pr(A_0 \mid E - A_1) = \frac{1}{2}$ implies that the following holds: $Pr(A_0) \leq \frac{1}{2} + Pr(A_1)$.

A consequence from the definition of IND-CCA is, that $Pr(A_0) = \frac{1}{2} + \frac{Adv(A)}{2}$, if $Pr(A_0) \geq \frac{1}{2}$ (A_0: A finds correct bit).

Even, if this lemma covers only a small part of a proof about semantic security of an encryption scheme, it is a step towards computer verification of cryptographic primitives. Further studies in that direction, could complete the computer support. Hence, they improve proving in the area of cryptography.

V. Miller-Rabin Algorithm

In this section we present the Miller-Rabin primality test. That presentation will be the basis of the Isabelle/HOL implementation described in the next section.

A. Primality or Compositeness

In general, a primality test is an algorithm that reveals, whether a given integer number k is prime or composite. In this context, a concrete prime factor decomposition of a composite number is not discussed.

Public-key cryptography often uses large prime numbers, consequently efficient algorithms are needed. A well known algorithm is division by possible factors, but because of its complexity ($O(\sqrt{k})$) this algorithm is not used for large numbers. More efficient algorithms are the *Fermat algorithm* and the *Miller-Rabin algorithm* ([3] or [10]), but their results are incorrect with a certain (small) probability.

B. Miller-Rabin Algorithm – Overview

In order to test an odd positive integer k for primality, the Miller-Rabin test proceeds as follows. As a precomputation the Miller-Rabin test determines the decomposition

$$k - 1 = 2^z \cdot v,$$

where $z \in \mathbf{N}_{\geq 1}$ and v is an odd positive integer. Functions that compute these two numbers are given in VI.

The Miller-Rabin test looks for witnesses that attest the compositeness of k. Such a witness is a number $x \in \{0, \ldots, k-1\}$, $\gcd(x, k) = 1$, such that the following property holds.

Theorem (1).

For $k \in \mathbf{N}$ prime, $z = \max\{r \in \mathbf{N} : 2^r \mid k - 1\}$, $v = (k-1)/2^z$ ($k - 1 = 2^z \cdot v$), and $x \in \{0, \ldots, k-1\}$ randomly chosen with $\gcd(x, k) = 1$ the following holds:

(1.1)
$$x^v \equiv 1 \bmod k$$

or there is a number $r \in \{0, \ldots, z-1\}$ with

(1.2)
$$x^{2^r \cdot v} \equiv -1 \bmod k$$

If (1.1) and (1.2) do not hold for a given k, x is a witness for the compositeness of k. Theorem (1) can be used to implement an algorithm that looks for witnesses that attest the compositeness of k (compare [3]).

C. Miller-Rabin Algorithm – Implementation

In this section we describe an implementation of the Miller-Rabin test. We use Theorem (1) to implement Algorithm (1) that looks for witnesses that attest the compositeness of k. If no such witness is found, primality of k is assumed.

Algorithm (1) uses the precomputation introduced above, i.e. input of Algorithm (1) are an odd positive integer k (the primality/compositeness of k is tested), a randomly chosen number $x \in \{0, \ldots, k-1\}$ with $\gcd(x, k) = 1$, and the decomposition $k - 1 = 2^z \cdot v$. Output of Algorithm (1) is *composite* or *prime*. We use the two functions $prim_1 : \mathbf{N}^4 \to \mathbf{N}$ and $f_1 : \mathbf{N}^3 \to \mathbf{N}$ that are based on Theorem (1) for a search for witnesses. These functions are defined as follows.

$$\begin{aligned} \text{prim}_1(x, k, z, v) \;=\; & \text{if } x^v \equiv 1 \bmod k \text{ then } 1 \\ & \text{else } f_1(z-1, x^v \bmod k, k) \end{aligned}$$

uses f_1, where

$$\begin{aligned} f_1(z, x, k) \;=\; & \text{if } 0 < z \text{ then} \\ & \quad \text{if } x \equiv -1 \bmod k \text{ then } 1 \\ & \quad \text{else } f_1(z-1, x^2, k) \\ & \text{else } 0. \end{aligned}$$

$prim_1$ and f_1 are applied to x, k, z, and v as described in the following lines.

Algorithm (1).

Input: $k \in \mathbf{N}$, $x \in \{0, \ldots, k-1\}$ with $\gcd(x, k) = 1$, $k - 1 = 2^z \cdot v$ with properties given above

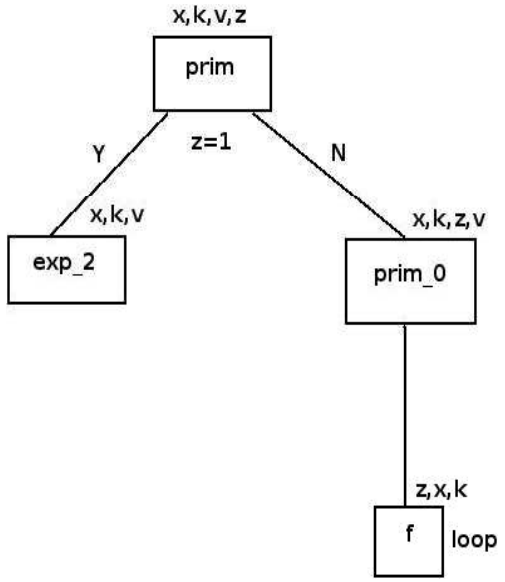

Fig. 4. Function *prim*

Output: *composite* or *prime*

$$\text{If } \text{prim}_1(x,k,z,v) = 1 \text{ then return } prime \text{ else return } composite$$

For some technical reasons we use the functions $prim : \mathbf{N}^4 \to \mathbf{N}$, $prim_0 : \mathbf{N}^4 \to \mathbf{N}$, $f : \mathbf{N}^3 \to \mathbf{N}$, and $exp_2 : \mathbf{N}^3 \to \mathbf{N}$ to implement and verify the Miller-Rabin test with Isabelle/HOL. These functions are defined as follows (compare Figure 4)

$$\text{prim}(x,k,z,v) = \text{if } z = 1 \text{ then } \text{exp}_2(x,k,v) \text{ else } \text{prim}_0(x,k,z,v)$$

$prim$ provides a case split, i.e. for $z = 1$ the function exp_2 computes a corresponding output value, while the function $prim_0$ looks for witnesses otherwise. This case distinction results from an abbreviation used in f that corresponds to f_1.

$$\text{prim}_0(x,k,z,v) = \text{if } x^v \bmod k \equiv 1 \text{ or } x^v \bmod k \equiv -1 \text{ then } 1 \text{ else } f(z-1, x^v \bmod k, k)$$

$$f(z,x,k) = \text{if } 0 < z \text{ then if } x^v \bmod k \equiv -1 \text{ then } 1 \text{ else if } x^v \bmod k \equiv 1 \text{ then } 0 \text{ else } f(z-1, x^2 \bmod k, k) \text{ else } 0$$

$$\text{exp}_2(x,k,v) = \text{if } x^v \bmod k \equiv 1 \text{ or } x^v \bmod k \equiv -1 \text{ then } 1 \text{ else if } x^{2\cdot v} \bmod k \equiv -1 \text{ then } 1 \text{ else } 0$$

A version of the Miller-Rabin test that uses similar case distinctions is given in [10].

D. Miller-Rabin Algorithm – Correctness

Up to now we gave no justification why the described search for witnesses for compositeness of odd positive integer k should be successful. But there are sufficient many witnesses, if k is composite, what is explained in Theorem (2), that describes the probability of correctness.

Theorem (2**).**

For $k \in \mathbf{N}$, $k \geq 3$ odd, composite number, the set $\{1, \ldots, k-1\}$ contains at most $(k-1)/4$ numbers x, where $\gcd(x,k) = 1$ and (1.1), (1.2) hold.

That means, if an integer number is composite, there are sufficient many witnesses to attest the compositeness of this number (compare [3]).

VI. Verification of the Miller-Rabin Algorithm

Algorithm (1) (with $prim$ instead of $prim_1$) is the main foundation of a formal description of the Miller-Rabin algorithm with Isabelle/HOL. This computer representation, that is based on functional programming and logic (of higher order), is a direct implementation of the functions *prim*, *prim*$_0$, *exp*$_2$ and *f* given in (3).

A. Formalized Mathematical Properties

In the following, we give a formalized version of Theorem (1) that can be verified with Isabelle/HOL.

lemma "[| (k::int) ∈ zprime; zgcd((x::int),k) = 1; 2 < k; x < k |] ⟹ (∃ (z::num). z < (f_02 (number k)) ∧ (xˆ((2ˆ z) * (mult2_value (number k)))) mod k = k − (1::int) | (xˆ (mult2_value (number k))) mod k = 1";

In the lines below some Isabelle/HOL code examples are explained.

Remark: (Isabelle/HOL code)

- int: integer number datatype
- num: natural number datatype
- ˆz: exponent z
- zprime: set of (integer) prime numbers
- zgcd: (integer) greatest common divisor
- f_02 and mult2_value compute a decomposition of $x-1 = 2^{(\text{f\_02}x)} \cdot (\text{mult2\_value}x)$, if $x \in \mathbf{N}$ is odd $((\text{f\_02}x) = \max\{r \in \mathbf{N} : 2^r \mid x-1\}$, $(\text{mult2\_value}x) = (x-1)/2^{(\text{f\_02}x)})$.
- function *number* describes the type change of an integer number z to a natural number corresponding to z

The presentation of this lemma not only shows that the implementation of the Miller-Rabin test given below was computer proven. Furthermore, the underlying mathematical theory can be verified with Isabelle/HOL.

B. Implementation

In this formal description, the functions *prim*, *prim*$_0$, *exp*$_2$ and *f* are implemented by *recovered_value01* and *primality*,

recovered_value0, *recovered_value2*, and *rep_operator*, respectively. Moreover, some auxiliary functions are used. These auxiliary functions are given below.

Auxiliary Functions:

We implemented the two auxiliary functions f_02 and mult2_value as follows.

consts f_02 :: "num ⇒ num";

defs
f_02_def: "f_02 x ≡ (GREATEST (z::num). (2^ z dvd (x−(1::num)))";

consts mult2_value :: "num ⇒ num";

defs
mult2_value_def: "mult2_value x ≡ (x−(1::num)) div (2^ f_02 x))";

These two functions compute a decomposition of $x-1 = 2^{(\text{f\_02}x)} \cdot (\text{mult2\_value}x)$, if $x \in \mathbf{N}$ is odd. That means $(\text{f\_02}x) = \max\{r \in \mathbf{N} : 2^r \mid x-1\}$ and $(\text{mult2\_value}x) = (x-1)/2^{(\text{f\_02}x)}$.

Functions describing the Miller-Rabin algorithm:

We directly implemented the functions *prim* (recovered_value01 and primality), $prim_0$ (recovered_value0), exp_2 (recovered_value2), f (rep_operator).

consts rep_operator :: "(int × int × int) ⇒ int";

recdef rep_operator "measure (λ (z,x,k). (number z))"
rep_operator_def: "rep_operator (z,x,k) = (if (0 < z) then
(if ((x^ 2 mod k) = (k − (1 :: int))) then (1 :: int)
else (if ((x^ 2 mod k) = 1) then (0 :: int)
else (rep_operator ((z − (1 :: int)), (x^ 2 mod k), k))))
else 0)"
(hints recdef_simp: number_lemma);

consts primality :: "[int, int, int, int] ⇒ int";

defs
primality_def: "primality x k z v ≡ (if ((x^ (number v) mod k) = 1)
| ((x^ (number v) mod k) = (k − (1 :: int)) then (1 :: int)
else (rep_operator ((z − (1 :: int)), (x^ number v)k, k))))";

consts recovered_value0 :: "int ⇒ int ⇒ int";

defs
recovered_value0_def: "recovered_value0 x k ≡ primality x k (int (f_02 (number k))) (int (mult2_value (number k)))";

consts recovered_value2 :: "int ⇒ int ⇒ int";

defs
recovered_value2_def: "recovered_value2 x k ≡ (if ((x^ (number v) mod k) = 1)
| ((x^ (number v) mod k) = (k − (1 :: int)) then (1 :: int)
else if (x^ (number v) mod k) = k−(1 :: int) then (1 :: int)
else (0 :: int)";

consts recovered_value01 :: "int ⇒ int ⇒ int";

defs
recovered_value01_def: "recovered_value01 x k ≡ (if (f_02 (number k) = 1)
then (recovered_value2 x k)
else (recovered_value0 x k)";

In this description we used the function *number*. This function describes the type change of an integer number z to a natural number corresponding to z.

C. Verification

The new functional and logically compiled description as well as its implementation (formal and machinery description) of the Miller-Rabin algorithm given above, illustrate that formalized mathematical knowledge can improve verification of correctness of primality tests or other (cryptographic) algorithms. This improvement is reached by using Isabelle/HOL as computer verification tool. We interactively proved the following main results that we give in a mathematical description).

Theorem (3). $x, k \in \mathbf{Z}$, $k \in$ Prime, $\gcd(x,k) = 1$, $2 < k$, $x < k$, $0 <$ f_02(k), $0 <$ mult2_value$(k) \Longrightarrow$ prim$(x,k) = 1$.

(If an integer number k is prime and some preconditions hold, the function $prim$ outputs 1 (what indicates that the Miller-Rabin test is passed).)

Theorem (4). $x, k \in \mathbf{Z}$, $x < k$, $2 < k$, $0 <$ f_02(k), $0 <$ mult2_value$(k) \Longrightarrow ((\exists z \in \mathbf{N}. z \leq \text{f_02}(k) \wedge x^{2^z \cdot \text{mult2_value}(k)} \bmod k = k-1) \vee (x^{\text{mult2_value}(k)} \bmod 1)) = (\text{prim}(x,k) = 1)$.

(Under some preconditions, $((\exists z \in \mathbf{N}. z \leq \text{f\_02}(k) \wedge x^{2^z \cdot \text{mult2\_value}(k)} \bmod k = k-1) \vee (x^{\text{mult2\_value}(k)} \bmod k = 1))$ is equivalent to the fact that $prim$ outputs 1.)

In the following, we illustrate a computer verification of these results by giving some examples of properties we proved with Isabelle/HOL.

After the implementation of the above introduced functions describing the Miller-Rabin algorithm, we started the proof construction with interactive proofs of some easy lemmata.

Example: (For a better reading we skipped the data types (e.g.int) of the variables)

lemma "[| 2 < k; k−1 = 2^(number z) ∗ v; x < k; x^(number v) mod k = k−1 |] ⟹ (primality x k z v) = 1";
apply (unfold primality_def);
apply (auto);
done;

(lemma with computer proof script)

Many lemmata of that kind and further lemmata of increasing difficulty lead us to a proof of the following property.

lemma "[| 0 < z; ∃ i. (∀ j. j < i ⟶ (x^(2^j) mod k)^2 mod k ≠ k−1) ∧ (∀ j. j < i ⟶ (x^(2^j) mod k)^2 mod k ≠ 1) ∧ ((x^(2^i) mod k)^2 mod k = k−1) ∧ 0 < i ∧ 0 < z−(int i) ∧ ∀ j. j < i ⟶ 0 < z−(int j) |] ⟹ (rep_operator (z,(x mod k),k)) = 1";

Consequently, we were able to prove the following lemma.

lemma "[| 2 < k; k−1 = 2^(number z) ∗ v; x < k; 0 < z−1; (x^(number v) mod k) = k−1) ∨ (x^(number v) mod k) = 1) ∨ (∃ i. (∀ j. j < i ⟶ (x^(2^j) mod k)^2 mod k ≠ k−1) ∧ (∀ j. j < i ⟶ (x^(2^j) mod k)^2 mod k ≠ 1) ∧ ((x^(2^ i) mod k)^2 mod k = k−1) ∧ 0 < i ∧ (int i) < z−1) |] ⟹ (primality x k z v) = 1";

For a complete formal verification, we introduced the following facts to the formal proof system:

- "((k::int) − (1::int)) = ((2^(number (int (f_02 (number k))))) ∗ (int (mult2_value (number k))))"
- "[| 2 < k; x < k; 0 < f_02 (number k); 0 < mult2_value (number k); recovered_value01 x k = 1 |]"

Fact 1 was used to proof the following properties.

lemma "[| k ∈ zprime; zgcd(x,k) = 1; 2 < k; x < k; 0 < f_02 (number k)−1; 0 < mult2_value (number k) |] ⟹ recovered_value0 x k = 1";
(correctness of auxiliary function recovered_value0)

lemma "[| k ∈ zprime; zgcd(x,k) = 1; 2 < k; x < k; f_02 (number k) = 1 |] ⟹ recovered_value2 x k = 1";
(correctness of auxiliary function recovered_value2)

lemma "[| k ∈ zprime; zgcd(x,k) = 1; 2 < k; x < k; 0 < f_02 (number k); 0 < mult2_value (number k) |] ⟹ recovered_value01 x k = 1";
(correctness of recovered_value01 (implementation of $prim$) (Theorem (3))

More (proven) lemmata and Fact 2 were used to verify the property below.

lemma "[| x < k; 2 < k; 0 < f_02 (number k); 0 < mult2_value (number k) |] ⟹ ((∃ z. z < (f_02 (number k)) ∧ (x^((2^z) ∗ (mult2_value (number k)))) mod k = k−1) ∨ (x^(mult2_value (number k))) mod k = 1) = (recovered_value01 x k = 1)";
(Theorem (4))

Altogether we implemented a functional/logic version of the Miller-Rabin primality test, that means we are able to computer verify the correctness of our algorithm. Besides, the used functions can be applied to integer numbers, what results in a applicable computer program for primality tests. In the following lines this result is illustrated.

Example: Application of ML code of

recovered_value0 x k

yields 0 or 1 corresponding to the values of $x, k \in \mathbf{Z}$, $x < k$.

VII. Conclusion

We explored formalized probability distributions (σ-algebras, probability spaces as well as conditional probabilities). These are given in the formal language of the formal proof system Isabelle/HOL. Moreover, we computer proved a formalized version of Bayes' Formula. Besides, we described an application of the presented formalized probability distributions to cryptography. Furthermore, this paper showed that computer proofs of complex cryptographic functions are possible by presenting an implementation of the Miller-Rabin primality test that admits formal verification. In this paper we gave a new description of a well known algorithm, that means we implemented the Miller-Rabin algorithm with computer support. Our new functional/logic implementation is computer verified and applicable as ML code. Moreover, we are able to computer verify the correctness of our algorithm. This paper is a further step towards a formalized basis of cryptographic algorithms. In cryptographic applications large prime numbers are often needed and therefore efficient correct algorithms for primality tests must be implemented and verified. Formal verification is a appropriate account to achieve correct algorithms. Moreover, cryptographic research can gain profit from formalized knowledge and formal verification. Formal proofs of security may support verification of cryptographic protocols or algorithms like encryption schemes. For that purpose, further mathematical knowledge must be implemented in a formal proof system.

References

[1] Heinz Bauer. *Wahrscheinlichkeitstheorie*. Walter de Gruyter Verlag, 5 edition, 2001.
[2] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *proceedings of the First ACM Conference on Computer and Communication Security*, 1993.
[3] Johannes Buchmann. *Einführung in die Kryptographie*. Springer-Verlag, 2 edition, 2001.
[4] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *proceedings of CRYPTO 2001*, 2001.
[5] Hans-Otto Georgii. *Stochastik*. Walter de Gruyter Verlag, 1 edition, 2002.
[6] B.W. Gnedenko. *Lehrbuch der Wahrscheinlichkeitstheorie*. Verlag Harri Deutsch, 10 edition, 1997.
[7] Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
[8] V. Shoup. OAEP Reconsidered. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes of Computer Science*, pages 239–259. Springer-Verlag, 2001.
[9] http://isabelle.in.tum.de.
[10] Dietmar Wätjen. *Kryptographie – Grundlagen, Algorithmen, Protokolle*. Spektrum Akademischer Verlag, 2004.