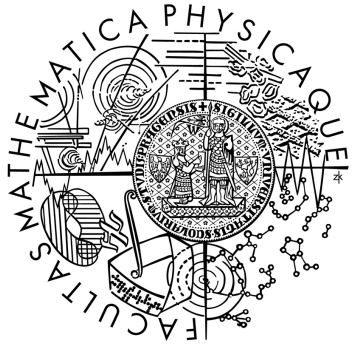
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Bc. Vojtěch Wagner

Logic and cryptography

Department of Algebra

Supervisor of the master thesis: prof. RNDr. Jan Krajíček, DrSc.

Study programme: Mathematics

Specialization: Mathematical Methods of Information Security

Prague 2015

I would like to thank the supervisor of my master thesis, prof. RNDr. Jan Krajíček, DrSc. for interesting topic, for lot of patience, for willingness to help and advise and for his valuable advices provided during this work.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date 20.5.2015 Bc. Vojtěch Wagner

Název práce: Logika a kryptografie

Autor: Bc. Vojtěch Wagner

Katedra: Katedra algebry

Vedoucí diplomové práce: prof. RNDr. Jan Krajíček, DrSc.

Abstrakt: Práce se zabývá studiem metod pro formalizaci kryptografických konstrukcí. Konkrétně metodou, která je založena na definování logické teorie T, která obsahuje řetězce, čísla a objekty třídy k - k-ární funkce. Povolíme jim určité operace a formulujeme axiomy, termy a formule. Budeme používat speciální typ termů - počítající term, který označuje počet prvků x v daném intervalu splňujících formuli $\varphi(x)$. Díky nim můžeme mluvit o pravděpodobnostech a používat další pojmy z teorie pravděpodobnosti. Práce nejprve popisuje detailně tuto teorii. Poté přináší formalizaci Goldreich-Levinovy věty. Cílem práce je předložit potřebné kryptografické pojmy a konstrukce v jazyce teorie T a následně dokázat větu pomocí objektů, pravidel a axiomů teorie T. Uvedené definice a principy jsou ilustrovány na příkladech. Cílem práce je ukázat, že takováto teorie je dostatečně silná, aby dokázala správnost a bezpečnost podobné kryptografické konstrukce.

Klíčová slova: kryptografie, ověřování protokolů, věta o správnosti, formální logická teorie, Goldreich-Levinova věta

Title: Logic and cryptography

Author: Bc.Vojtěch Wagner

Department: Department of Algebra

Supervisor: prof. RNDr. Jan Krajíček, DrSc.

Abstract: This work is devoted to a study of a formal method of formalization of cryptographic constructions. It is based on defining a multi-sorted formal logic theory T composed of strings, integers and objects of sort k - k-ary functions. We allow some operations on them, formulate axioms, terms and formulas. We also have a special type of integers called the counting integers. It denotes the number of x from a given interval satisfying formula $\varphi(x)$. It allows us to talk about probabilities and use terms of probability theory. The work first describes this theory and then it brings a formalization of the Goldreich-Levin theorem. The goal of this work is to adapt all needed cryptographic terms into the language of T and then prove the theorem using objects, rules and axioms of T. Presented definitions and principles are ilustrated on examples. The purpose of this work is to show that such theory is sufficiently strong to prove such cryptographic constructions and verify its correctness and security.

Keywords: cryptography, protocol verifying, Soundness theorem, formal logic theory, the Goldreich-Levin theorem

Contents

Introduction **2**

1 Formal system for cryptographic protocols verification **3**
1.1 General introduction . . . 3
1.2 Elements of the system . . . 3
1.3 The language . . . 5
1.4 Terms and formulas . . . 6
1.5 Axioms of the theory . . . 7
1.6 Properties of counting terms . . . 10
1.7 Soundness of the theory . . . 13

2 The Goldreich-Levin theorem **15**
2.1 Hard-Core predicate . . . 15
2.2 The Goldreich-Levin theorem . . . 16
2.3 Adaptation in our theory . . . 17
2.3.1 Cryptographic terms . . . 17
2.3.2 Inner product . . . 17
2.3.3 The theorem . . . 21

3 Proof of the Goldreich-Levin theorem **23**
3.1 Idea of the original proof . . . 23
3.2 Idea of the formalized proof . . . 24
3.3 Steps of the proof . . . 24
3.4 Formalization of φ_1 . . . 25
3.5 Formalization of φ_2 . . . 26
3.6 Formalization of ψ . . . 26
3.7 Construction of the Goldreich-Levin algorithm . . . 27
3.7.1 Preparation . . . 27
3.7.2 The algorithm . . . 28
3.7.3 The correctness and the success probability of the algorithm 30
3.8 Final step . . . 34

Conclusion **36**

Bibliography **37**

Appendix **38**

Introduction

With the masive expansion of interchange network protocols, big data storages and ICT technologies in general, there comes also a question regarding their security and correctness. Because of huge and continuously growing technological possibilities and opportunities, it is not sufficient to just provide some service but more and more attention goes to the quality of the provided service. Nowadays many companies have their data stored in some kind of ICT services. That immediately implies the question regarding correctness, security, integrity and availability of those services to protect and correctly store the data.

In general, we deal with two things. One is the construction of a correct and a secure protocol. Although this is the main task in cryptography, we will not deal with this task in this work. In fact, the construction of a secure protocol is just the first part. We also need the possibility to verify that our constructed protocol is correct and secure enough. To do that, we can define a formal logic system. We allow as objects just strings and integers that are the length of some string. We allow specific operations on them and present some axioms. To be able to verify protocols, we present functions, terms and formulas and special terms that give us a number of elements from a given interval satisfying some formula. We will call them *counting integers* and having them, we can talk about probablities. Such theory was introduced by Russel Impagliazzo and Bruce Kapron in [IK03]. The idea is to present formal logic system that can be easily implemented for automated protocol verifying.

The goal of this work is to show how this theory can be used for proving some cryptologic construction. Namely we will try to formalize and prove the well-known *Goldreich-Levin theorem* using this logic theory.

The first chapter is devoted to a description of this formal system of logic. We define objects we will work with - strings and integers and a special way how to view string as integers and vice versa. We present operations on them and define axioms, terms and formulas. We allow functions of arity $k > 0$. This will give us a strong enough tool to present *the Soundness theorem* - a general way how to prove cryptographic properties. We present this theorem at the end of the chapter. This chapter is closely based on the original work [IK03]. Besides that, we show the presented principles on some examples.

In the second chapter we present a formalization of the Goldreich-Levin theorem in our theory. We show how to transform the cryptographic primitives that are used in the theorem into our formal system of logic and finally we transform the theorem itself to our theory.

In the third chapter we try to prove the adapted Goldreich-Levin theorem using objects, axioms and rules from the defined theory. By giving the exact proof of the theorem, we will show that the theory is strong enough to prove such cryptographic constructions. This proof uses ideas from the original proof (as presented for example in [G01b]), but the whole composition and transformation into the language of the theory is the result of my work.

1. Formal system for cryptographic protocols verification

1.1 General introduction

To create a secure and a correct protocol which solves one or more tasks regarding privacy of data, integrity of data, availability of data or authentication is one of the most fundamental and important tasks in cryptography. But it is also one of the most difficult ones.

A simple straightforward solution comes to our mind immediately. We could take cryptographic primitives which are considered secure and correct and put them simply together - let every single task be solved by an appropriate primitive.

Although every particular primitive is secure and correct, the resulting protocol may not be secure and correct. In fact, with a high probability such protocol will not be secure and correct unless we took care of the right connection of primitives together and a correct usage of them. This makes from a cryptographic protocol construction very difficult task. Naturally there arises a question how to correctly verify that some protocol is sound and secure enough.

To have a possibility to verify protocols generally enough and independently on all factors, we need some formal system of axioms, rules, variables and functions in which we can interpret the whole protocol and prove the soundness easily by using axioms from this system. Such system was proposed by Russel Impagliazzo and Bruce Kapron and we will describe it in detail in the first chapter. The description is closely based on work of R. Impagliazzo and B. Kapron presented in [IK03]. We then present some examples for better understanding.

At first, we will give a description of this system and at the end of this chapter we will also show that this system allows us to verify cryptographical constructions within the Soundness theorem.

1.2 Elements of the system

We consider a formal logic system which uses syntax of first order logic. It means we have elements of propositional logic (i.e. syntax, derivation rules and axioms) and also quantifiers and predicates. We also have the use of function symbols. Having all this, we get strong enough tool to talk about probabilities, polynomial functions and asymptotical properties.

For complete introduction of the system, we have to define axioms, objects of the system and rules which we will use. Let us start with the primary objects of the system, i.e. objects that system takes as basic.

Security parameter. Let us suppose that we have some property which is asymptotically true. For example, we can look for the number of primes less than or equal to some value x. As we know, this number is asymptotically equal to $\Pi(x) \approx \frac{x}{\log x}$. But it is clear that taking some particular number x, the formula may not holds exactly. But it always holds that $\lim_{n\to\infty} \frac{\Pi(n)\cdot\log n}{n} = 1$.

Considering the situation that our goal is to create a tool which could be used for automatic protocols verifying, we surely need to have a value such that all asymptotically true statements are true at this value. Let us fix a variable name $\boldsymbol{n}$ for such value and call it *the security parameter*. Next, denote $\mathbf{s}$ a string such that $|\mathbf{s}| = \boldsymbol{n}$ (by $|x|$ we mean the length of x in a specific representation which will be described later).

Strings. Next objects, that we will work with, are strings. I.e. constant $\mathbf{s}$, inputs and outputs from (probabilistic) algorithms, outputs from functions applied to strings. We denote ϵ the empty string. Other string variables we denote by letters $a, b, c, \ldots$.

Integers. The next category of objects, which we will use in our system, are integers. We consider only those which are the length of some string. We denote $|t|$, meaning length of a string t. So we have numbers $\boldsymbol{n}$ for $\mathbf{s}$ (the security parameter), 0 for string ϵ, 1 for strings $\mathsf{0}$, $\mathsf{1}$ etc.

Using the dyadic representation we can also view strings as numbers and vice versa. The dyadic notation can be created as follows. We assign 0 to the empty string ϵ, 1 to a string $\mathsf{0}$, 2 to a string $\mathsf{1}$, 3 to a string in the form of $\mathsf{00}$, 4 to a string $\mathsf{01}$ etc. In general, having some string $b_1b_2b_3\ldots b_k$, we prepend $\mathsf{1}$, i.e. we have string $1b_1b_2\ldots b_k$. String $1b_1b_2\ldots b_k$ represents some number m using standard binary notation. We get appropriate dyadic number by taking value $m-1$.

This procedure can also be inverted to get an appropriate dyadic string from a number. Having some integer m, we write down a binary representation of $m+1$ and to get the dyadic notation we substract the leading $\mathsf{1}$.

Having this representation, we can naturally use standard arithmetical operations on strings - we can add, substract or compare them. For example having $x = \mathsf{10011}$, $y = \mathsf{0110}$ we know that x corresponds to number 50 and y corresponds to 22. So $x + y = 72$ which is $\mathsf{001001}$ in dyadic, $y \leq x$ etc.

Functions. We also need objects of the first order logic representing k-ary functions applied to variables $x_1, x_2, \ldots, x_k$. For these functions we require

- to be polynomial time computable in the secutiry parameter $\boldsymbol{n}$
- to take strings as inputs
- to output strings

For each such function we have a function symbol, i.e. variable, which represents such function. That allows to quantify - we quantify over these function variables. Using function variables we will be able to denote adversaries to a cryptosystem or to some security property.

We also find usefull next properties - a composition of functions, an application of arithmetical operations. We define the composition of functions using a special type of recursion. This will be described later.

Counting terms. In cryptography we often use terms of probality theory and their properties. To have the possibility to use such terms, we need one more type of objects, the so-called *counting terms*. A counting term denotes integer determining the number of elements of some set of strings. By the statement

$$\#\left(|x_1| = m_1 \wedge |x_2| = m_2 \wedge \cdots \wedge |x_k| = m_k\right) \varphi(x_1, \ldots, x_k)$$

we denote the number of vectors $\vec{x} = (x_1, \ldots, x_k)$ (with lengths of coordinates $m_1, \ldots, m_k$) satisfying formula $\varphi(x_1, \ldots, x_k)$. The exact calculation of this value may not be trivial, even it may not be polynomial time computable. Because of that, we suppose that a counting integer is not an input to functions. And how can we use this term for probabilities? As we know, according to the classic definition of probability, we calculate the probability of some event as the number of cases favorable for the event divided by the total number of all possible cases of event. As we said, calculation of both values may not be trivial at all. But counting integers belong to the primary objects of our system, therefore the fraction $\frac{\#(|x|=t)\varphi(x)}{\#(|x|=t)(x=x)}$ (which represents the probability that x satisfies $\varphi(x)$) is a permitted calculation.

1.3 The language

Constants. We assume the existence of the following constants:

- 0 corresponds to an empty string ϵ
- 1 corresponds to a string $\mathsf{0}$ in the dyadic notation
- 2 corresponds to a string $\mathsf{1}$ in the dyadic notation
- $\boldsymbol{n} \in \mathbb{N}$ is the security parameter
- string $\mathbf{s} \in \{0,1\}^*$ such that $|\mathbf{s}| = \boldsymbol{n}$

Variables. In the system we have two sorts of variables. We have function variables, representing k-ary functions. We will usually use symbols $f^k, g^k, h^k, \ldots$ for function variables. The second sort of variables is for integers and strings. As we said, using dyadic notation we can represent integers by strings and vice versa. So we can use just one variable for a couple *string* $\sim$ *integer*. We will usually use symbols such $x, y, z, i, j, k, \ldots$ for integers/strings.

Operations. We allow the following operations in our theory.

- The *addition* $x + y$ defined for integers. Having the unique correspondation between integers and strings, we can also view this operation as an operation on strings.
- The *multiplication* $x \cdot y$, similarly as in the previous case we can view this operation as multiplying of strings.

- The *length function* $|\boldsymbol{x}|$ representing the length of string x.
- The *smash* function $x \otimes y$ defined as $2^{|x|\cdot|y|}$.
- $\lfloor \frac{\boldsymbol{x} \dot{-} \boldsymbol{y}}{\boldsymbol{2}} \rfloor$, where $x \dot{-} y = \max\{x - y, 0\}$.
- The operations $x \leq y$, $x = y$ defined for integers. Again, we can view this as an operation on strings.
- $\mathbf{A}_{\boldsymbol{p}}^{\boldsymbol{k}}$ for $k \geq 1$ stands for an application of k-ary function. Formally we denote it as $\mathrm{A}_p^k(f^k, t_1, \ldots, t_k)$ meaning the aplication of k-ary function f^k to an inputs $t_1, \ldots, t_k$. Mostly we will use more usual notation $f(t_1, \ldots, t_k)$.

1.4 Terms and formulas

Now let us describe terms and formulas, which we will use later. We define terms from objects of the theory using operations defined in the previous pararaph. Using operations from first-order logic we then define also formulas in our theory.

Terms.

- $0,\ 1,\ 2,\ \boldsymbol{s},\ \boldsymbol{n}$ are terms.
- Every variable representing string is a term.
- Let $t_1, \ldots, t_k$ are terms, f^k a function variable. Then $\mathrm{A}p^k(f^k, t_1, \ldots, t_k)$ is a term.
- Let $r,\ s$ are two terms. Then $r + s$, $r \cdot s$, $r \otimes s$, $|r|$, $\lfloor \frac{r \dot{-} s}{2} \rfloor$ are terms.

Formulas. We define formulas from terms.

- Let $r,\ s$ are terms. Then $r \leq s,\ r = s$ are formulas.
- If φ, ψ are formulas, then $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \to \psi$, $\varphi \equiv \psi$ are formulas.
- If φ is a formula and x is a string variable, then $\forall x\varphi$ a $\exists x\varphi$ are formulas.
- If φ is a formula and f is a function variable, then $\forall f\varphi$ a $\exists f\varphi$ are formulas.

Counting terms. Counting terms represent a number of x's of length t satisfying a formula φ. We denote it as $\#(|x| = |t|)\varphi$, eventually

$$\#\begin{pmatrix} |x_1| = |t_1| \\ \cdots \\ |x_k| = |t_k| \end{pmatrix} \varphi$$

Counting terms are defined recursively - φ can contain another counting term. Counting term is a term.

1.5 Axioms of the theory

We divide axioms into several categories.

Axioms of the security parameter. In the paragraph describing objects of the theory we introduced the security parameter $\mathbf{s}$ - a string of length $\boldsymbol{n}$ such that each asymptotically true statement is true in $\boldsymbol{n}$. We can axiomatize this parameter as follows. Denote by k term in the form of $\underbrace{1+1+\cdots+1}_{k}$. Then for each $k \geq 0$ there exists the axiom Sk in the form $|\mathbf{s}| \geq k$. In other words, length $\boldsymbol{n}$ of the security parameter $\mathbf{s}$ is greater than all $k \geq 0$.

Basic Axioms. *Basic axioms* is set of basic rules defining how we can work with variables and integers in our theory using the operations described above. They introduce properties, which are obvious, as well as properties, which may not be clear on the first view. Complete list of them can be found in the appendix, here we show just those which will be useful later.

- B10: $|2x+1| = |x|+1 \wedge |2x+2| = |x|+1$. To clearly understand this axiom, it is sufficient to realize how we defined the correspondance between strings and integers. Let for example have a string $x = \mathsf{1001}$, wich corresponds to a number 24. Then $2x+1 = 49$, $2x+2 = 50$. These values correspond to strings $\mathsf{10010}$ a $\mathsf{10011}$, i.e. the string x extended by one bit from the right. Therefore $2x+1$ is a string $\mathsf{x0}$ and $2x+2$ is a string $\mathsf{x1}$. This axiom is very important and we will use it later in definitions of functions.
- B12: $(x \neq 0 \wedge y \neq 0) \rightarrow |x \otimes y| = |x| \cdot |y|$. We have to just realize how we defined the operation $\otimes$ and then use the correspondance *strings* $\leftrightarrow$ *integers*. We have $x \otimes y = 2^{|x|\cdot|y|}$. This value has in binary representation $|x| \cdot |y| + 1$ bits. Therefore using the dyadic notation the resulting string will have $|x| \cdot |y|$ bits.
- B32: $x \neq 0 \rightarrow |x| = |\lfloor \frac{x \dot{-} 1}{2} \rfloor| + 1$. What does this axiom say? Let have x in the form of $x = b_1 b_2 \ldots b_k$. Then $\lfloor \frac{x \dot{-} 1}{2} \rfloor$ is a string in the form of $b_1 b_2 \ldots b_{k-1}$, i.e. string of length $k-1$. Thus this axiom says that by adding one bit the length of a string increases by 1.
- B33: $y \neq 0 \rightarrow \Big(x = \lfloor \frac{y \dot{-} 1}{2} \rfloor \equiv (2x+1 = y \vee 2x+2 = y)\Big)$ This axiom specifies the axiom B32. It says that, having two strings x,y of lengths $k-1$ and k which agree on first $k-1$ bits, then y is made from x by adding $\mathsf{0}$ or $\mathsf{1}$ to x. Again, very important axiom which will be used later for function definitions.

Function axioms. For the possibility to work with polynomial functions, we need to have polynomial functions as values of function variables. The intended range of function variables is the class of non-uniform polynomial time functions. This can not be enforced by a first order theory, so we need to add enough axioms which will enforce some closure properties of this class. Let us now formulate these axioms.

- We define the *axiom of projection* (denote it $\mathsf{PROJ}(k,i)$), which guarantees

existence of a function that returns projection to a given coordinate i:

$$\forall k\ \forall i\ \exists f^k\ \forall(x_1,\ldots,x_k)\ (f(x_1,\ldots,x_k)=x_i)$$

- We shall have the existence of the *zero function* - function, which returns the constant value 0, no matter on the input:

$$\forall k\ \exists f^k\ \forall(x_1,\ldots,x_k)\ (f(x_1,\ldots,x_k)=0)$$

- We then need to define the *axiom of composition of functions* (denote as $\mathsf{COMP}(k,l)$). For each k,l we have

$$\forall g^{k+1}\ \forall h^{k+l}\ \exists f^{k+l}\ \forall\vec{x}\ \forall\vec{y}\ \ (f(\vec{x},\vec{y})=g\,(\vec{x},h(\vec{x},\vec{y})))$$

  Thanks to this axiom we get the closure of polynomial function variables under the composition.

- The following axiom will also be very helpful in further text. It describes *limited recursion on notation* (we denote it as LRN axiom) and gives us an excelent way how to easily define many kinds of functions. We define it as follows:

$$\forall g^k\ \forall h_1^{k+2}\ \forall h_2^{k+2}\ \forall b^{k+1}\ \exists f^{k+1}\ \forall\vec{x}\ \forall y($$

  $f(\vec{x},0)=g(\vec{x})\wedge$

  $|h_1(\vec{x},y,f(\vec{x},y)))|\leq|b(\vec{x},2y+1)|\rightarrow(f(\vec{x},2y+1)=h_1(\vec{x},y,f(\vec{x},y))))\wedge$

  $|h_1(\vec{x},y,f(\vec{x},y)))|>|b(\vec{x},2y+1)|\rightarrow(f(\vec{x},2y+1)=b(\vec{x},2y+1))\wedge$

  $|h_2(\vec{x},y,f(\vec{x},y)))|\leq|b(\vec{x},2y+2)|\rightarrow(f(\vec{x},2y+2)=h_2(\vec{x},y,f(\vec{x},y))))\wedge$

  $|h_2(\vec{x},y,f(\vec{x},y)))|>|b(\vec{x},2y+2)|\rightarrow(f(\vec{x},2y+2)=b(\vec{x},2y+2)))$

  The axiom gives us a procedure how to expand strings by adding bits. We have some bounding function b, which gives us a value that cannot be exceeded. Each time we try to add one bit to actual string ($\mathsf{0}$ or $\mathsf{1}$ to the string, denote that bit by a) and if it is possible (i.e. the value of function h_a does not exceed b), we can use the recursion rule. Otherwise we use the bounding function b.

- We need to add one more axiom which is not explicitely presented in the original theory [IK03] but it is used there. This was first noticed by Emil Jeřábek in [J05]. The original theory does not involve functions that return a constant value, i.e. function $c_x=x$. Thus we add also the axiom $\mathsf{CONST}(x)$ in the form

$$\forall x\exists f\ \ f(0)=x$$

  This axiom also causes that the range of function variables is not the class of uniform polynomial time functions, but the class of non-uniform polynomial time functions. That is because it allows to substitute parameters into functions. The axiom will be very useful in further study, because it will allow us to replace in the formalization the class of adversaries that are randomized polynomial time algorithms by a larger class of non-uniform polynomial time algorithms (i.e.circuits).

Induction axiom. To have the possibility of using an induction proof technique, we need to define suitable axioms. We admit induction just for open formulas without quantifiers. We define induction using the length of dyadic notation of given induction variable. Denote induction axioms by a symbol LIND and define it for each open formula $\varphi(x)$ and for each polynomial p as follows

$$(\varphi(0) \wedge (\forall x < p(|y|)\ (\varphi(x) \rightarrow \varphi(x+1)))) \rightarrow \forall x \leq p(|y|)\ \varphi(x).$$

Counting axioms. We now formulate several axioms for using counting terms. First three of them are some sort of modification of the Kolmogorov axioms of probability. Kolmogorov's probabilistic approach says that for discrete probability space Ω we have at most countably many elements $\omega \in \Omega$. So the Kolmogorov axiom of normality which says that $\Pr[\Omega] = 1$, can be translated into our theory as

$$(C1) \quad \#(|x| = |y|)(\varphi(x) \vee \neg\varphi(x)) = y \otimes 1.$$

Next Kolmogorov axiom tells us that $\Pr[\omega] \geq 0$ pro $\forall\omega \in \Omega$. In our theory we can rewrite it as

$$(C2) \quad \#(|\vec{x}| = |\vec{y}|)\varphi(x) \geq 0.$$

For the next counting axiom let us realize that for any formula φ it holds that $\{\vec{x};\ \varphi(\vec{x})\} \cap \{\vec{x};\ \neg\varphi(\vec{x})\} = \emptyset$. So we can interpret the Kolmogorov axiom of aditivity as

$$(C3) \quad \#(|\vec{x}| = |\vec{y}|)\varphi(x) = \#(|\vec{x}| = |\vec{y}|)(\varphi(x) \wedge \psi(x)) + \#(|\vec{x}| = |\vec{y}|)(\varphi(x) \wedge \neg\psi(x)).$$

We have some straightforward corrolaries of these axioms. In classic probability theory we can derive from Kolmogorov axioms that for any event $\omega \in \Omega$ it holds $\Pr[\omega] \in [0, 1]$. Similarly from (C1), (C2) it follows that

$$\#(|\vec{x}| = |\vec{y}|)\varphi(x) \geq 0 \wedge \#(|\vec{x}| = |\vec{y}|)\varphi(x) \leq y \otimes 1.$$

As a corrolary of the Kolmogorov axioms is often stated the monotony property. It means for two events ω_1, ω_2 such that $\omega_1 \subseteq \omega_2$ it holds that $\Pr[\omega_1] \leq \Pr[\omega_2]$. For an interpretation in our theory let denote $R := \{\vec{x}; \varphi(x)\}$, $S := \{\vec{x}; \psi(x)\}$. We would like to have an analogous property ideally in the form $\#(|\vec{x}| = |\vec{r}|)\ \varphi \leq \#(|\vec{x}| = |\vec{r}|)\ \psi$ if $R \subseteq S$. But to make it correct, we need to take into account two things:

- there has to exist the graph of an injective map M between $\vec{x} \in R$ and $\vec{y} \in S$. This can be done within defining function r such that $(x, y) \in M \Leftrightarrow r(\vec{x}, \vec{y}) = 0$. If such function exists, then we know that there exists a relation between $\vec{x}$ and $\vec{y}$.
- this relation (between values from R and values from S) has to be injective

We can formalize this by the formula $\mathsf{INJ}(\vec{u}, \vec{v}, r)$, where r is a function variable, saying that r is an injective mapping.

$$(\forall|\vec{x}| = |\vec{u}|)\ (\forall|\vec{z}| = |\vec{u}|)\ (\forall|\vec{y}| = |\vec{v}|)\ (r(\vec{x}, \vec{y}) = 0 \wedge r(\vec{z}, \vec{y}) = 0 \rightarrow \vec{x} = \vec{y}).$$

Then we can formulate the monotony property in our theory as follows:

$$(C4)\ \exists r(\mathsf{INJ}(\vec{u},\vec{v},r) \wedge (\forall|\vec{x}| = |\vec{u}|)\ (\varphi(\vec{x}) \rightarrow (\exists|\vec{y}| = |\vec{v}|)\ (\psi(\vec{y}) \wedge r(\vec{x},\vec{y}) = 0))) \\ \rightarrow \#(|\vec{x}| = |\vec{u}|)\ \varphi(x) \leq \#(|\vec{y}| = |\vec{v}|)\ \psi(y).$$

Encoding theory into first order logic. Let us make one important diggresion now. We would like to encode T into first-order logic. How can we do that? Let us realize that we can index polynomial functions of arity k using clocked Turing machine. Having that we can view application of k-ary function as a function $ap^k([i,k], s_1, \ldots, s_k)$. Such function then represents an application of Turing machine i with bound n^k to inputs $s_1, \ldots, s_k$. In this way we can translate all function variables from T. Counting terms can be translated as follows. For every formula φ and for every $k > 0$ we can take a function $ct^k_\varphi(s_1, \ldots, s_k)$ representing the number of tuples $[x_1, \ldots, x_k]$ with lengths $|x_i| \leq |s_i|$ which satisfies φ.

1.6 Properties of counting terms

We have presented basic counting axioms needed to think about probabilities in our model. Let us make one abbreviation now. By a symbol T we mean the theory we just presented, i.e. objects along with the language, terms, formulas and axioms.

For the next part we need the notion *deriving in T*. For formulas φ, ψ in T we denote $T, \varphi \vdash \psi$ a statement that ψ can be derived from φ using basic, function, induction and counting axioms of T along with the rules and operations of T.

Now we present some other properties of counting terms, which can be derived in T. We will use the following abbreviations. Let C be a counting term in the form of $\#(|\vec{x}| = |\vec{t}|)\varphi$. By a term $\#(|\vec{y}| = |\vec{u}|)C$ we mean the term $\#(|\vec{y}| = |\vec{u}| \wedge |\vec{x}| = |\vec{t}|)\varphi$. By a symbol $\varphi[t/x]$ we mean the substitution by t for all free occurances of x in a formula φ.

Lemma 1.1. ([IK03]). $T \vdash |x| = 0 \rightarrow x = 0$.

Proof: Let suppose for contradiction that $x \neq 0$. According to (B32) we have $|x| = ||\lfloor\frac{x \dot{-} 1}{2}\rfloor| + 1$. Denote $k := ||\lfloor\frac{x \dot{-} 1}{2}\rfloor|$. Then $|x| = k + 1$. We also have $|x| = 0$. Hence $k + 1 = 0$. By (B3) is $k + 1 = 0 \leq k$, therefore $k + 1 \leq k$. By (B6) is $k + 1 \leq k \ \vee \ k \leq k + 1$. If $k > k + 1$, then we would get contradiction with (B1), because $k \leq k$. Hence $k \leq k + 1$. Thus using (B7) we get $y = y + 1$, which leads to contradiction with (B2). □

Lemma 1.2. (C5)**([IK03]).** *For any counting term C it holds that*

$$T \vdash \#(|x| = 0)C = C(0).$$

Proof: Let take a term $C = \#(|\vec{x}| = |\vec{t}|)\varphi$. Then we have $\#(|x| = 0)C = \#(|x| = 0 \wedge |\vec{x}| = |\vec{t}|)\varphi$, $C(0) = \#(|\vec{x}| = |\vec{t}|)\varphi[0/x]$. For any $x, \vec{x}$ such that $|x| = 0 \wedge |\vec{x}| = |\vec{t}| \wedge \varphi$ we have from Lemma 1.1 $x = 0$, so $|\vec{x}|$ satisfies $\varphi[0/x]$. Therefore from (C4) follows, that $\#(|x| = 0)C \leq C\,[0/x]$.

It also holds reverse inequality, because by (B9) is $|0| = 0$ and we get $(\forall|\vec{x}| = |\vec{t}|)(\varphi[0/x] \to (\exists|y| = |0|)C(y))$. Tedy $\#(|\vec{x}| = |\vec{t}|)\varphi[0/x] \leq \#(|y| = 0)C$, which we wanted. □

Lemma 1.3. (C6)**([IK03]).** *For any counting term C it holds that*

$$T \vdash \#(|x| = |y| + 1)C = \#(|x| = |y|)C[2x + 1/x] + \#(|x| = |y|)C[2x + 2/x]$$

Proof: Let take C in the form $\#(|\vec{x}| = |\vec{t}|)\varphi$. We show at first the inequality "$\geq$", in other words

$$\#(|x| = |y|)C[2x+1/x]+\#(|x| = |y|)C[2x+2/x] \leq \#(|x| = |y|+1 \wedge |\vec{x}| = |\vec{y}|)\varphi.$$

Suppose that $|x| = |y| \wedge |\vec{x}| = |\vec{t}| \wedge \varphi[2x + 1/x]$, $|x| = |y| \wedge |\vec{x}| = |\vec{y}| \wedge \varphi[2x + 2/x]$. Let take $z_1 = 2x + 1, z_2 = 2x + 2$. Then

$$|z_1| = |y| + 1 \wedge |\vec{x}| = |\vec{y}| \wedge \varphi[z_1/x] \wedge z_1 = 2 \cdot \left\lfloor \frac{z_1 \dot{-} 1}{2} \right\rfloor + 1$$

$$|z_2| = |y| + 1 \wedge |\vec{x}| = |\vec{y}| \wedge \varphi[z_2/x] \wedge z_2 = 2 \cdot \left\lfloor \frac{z_2 \dot{-} 1}{2} \right\rfloor + 1.$$

We then take

$$r_1(\{x, \vec{x}\}, \{z_1, \vec{y}\}) = 0 \Leftrightarrow z_1 = 2x + 1 \wedge \vec{x} = \vec{y}$$
$$r_2(\{x, \vec{x}\}, \{z_2, \vec{y}\}) = 0 \Leftrightarrow z_2 = 2x + 2 \wedge \vec{x} = \vec{y}.$$

From axiom (C4) (using functions r_1, r_2) we get that

$$\#(|x| = |y|)C[2x + 1/x] \leq \#(|x| = |y| + 1 \wedge |\vec{x}| = |\vec{y}|) \left(\varphi \wedge x = 2 \cdot \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1 \right)$$

$$\#(|x| = |y|)C[2x+2/x] \leq \#(|x| = |y|+1 \wedge |\vec{x}| = |\vec{y}|) \left(\varphi \wedge x = 2 \cdot \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 2 \right).$$

Using axiom (B33) we get

$$\#(|x| = |y|)C[2x+2/x] \leq \#(|x| = |y|+1 \wedge |\vec{x}| = |\vec{y}|) \left(\varphi \wedge \neg \left(x = 2 \cdot \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1 \right) \right).$$

The requested inequality then follows directly from axiom (C3).

By axiom (C3) we have

$$\begin{aligned} \#(|x| = |y| + 1)C \leq & \#(|x| = |y| + 1 \wedge |\vec{x}| \leq |\vec{y}|) \left(\varphi \wedge x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1 \right) + \\ & \#(|x| = |y| + 1 \wedge |\vec{x}| \leq |\vec{y}|) \left(\varphi \wedge \neg \left(x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1 \right) \right). \end{aligned}$$

Using (C4) we have

$$\#(|x| = |y| + 1 \wedge |\vec{x}| \leq |\vec{y}|) \left(\varphi \wedge x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1 \right) \leq C[2x + 1/x].$$

And finally, accorging to (B33) and (C4), we get

$$\#(|x| = |y| + 1 \wedge |\vec{x}| \leq |\vec{y}|) \left(\varphi \wedge \neg \left(x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1 \right) \right) \leq C[2x + 2/x].$$

That is what we wanted. □

Lemma 1.4. (C7)**([IK03]).** *For every counting term C which contains x but not as a free variable it holds*

$$\#(|x| = |y|)C = (y \otimes 1) \cdot C.$$

Proof: We prove this by induction using the axiom LIND. At first by property (C5) we have $\#(|x| = 0)C = C[0/x]$. By assumption x is not free in C, hence $C[0/x] = C = (0 \otimes 1)C$.

For the induction step let suppose that for $|y| = k$ it holds that $\#(|x| = |y|)C = (y \otimes 1) \cdot C$. Then by (C6) we have

$$\#(|x| = |2y + 1|)C = \#(|x| = |y|)C[2x + 1/x] + \#(|x| = |y|)C[2x + 2/x].$$

But x is not free in C, so we can write

$$\#(|x| = |y|)C[2x + 1/x] + \#(|x| = |y|)C[2x + 2/x] = \#(|x| = |y|)C + \\ + \#(|x| = |y|)C.$$

By induction assumption

$$\#(|x| = |y|)C + \#(|x| = |y|)C = (y \otimes 1) \cdot C + (y \otimes 1)C = 2(y \otimes 1) \cdot C.$$

By axiom (B14) we can write

$$2(y \otimes 1) \cdot C = ((2y + 1) \otimes 1) \cdot C.$$

Lemma then directly follows by LIND axiom. □

Lemma 1.5. (C8)

$$T \vdash |z| = |y| \to \#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi) = \#(|\vec{x}| = |\vec{y}|)\varphi[z/x].$$

Proof: It follows from (C4) that for any z

$$\#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi) \leq \#(|\vec{x}| = |\vec{y}|)\varphi[z/x].$$

From (C4) we also have

$$|z| = |y| \to \#(|\vec{x}| = |\vec{y}|)\varphi[z/x] \leq \#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi),$$

which gives us the requested equality. □

1.7 Soundness of the theory

The main reason for introducing the theory is to have some formal system for verifying the soundness of cryptographic constructions and protocols based on some known cryptographic primitive. In the last chapter we will also present an example how to use this theory for a specific cryptographic construction. Now let us describe the way how we can verify constructions in general.

Let us suppose that we have some cryptographic primitive, which can be expressed as a polynomial function f. Let us also suppose that we have another polynomial function f', which is defined from f. Our goal is to prove some cryptographic property of f'.

For example, f describes security of some well-known cryptographic primitive or cryptographic protocol, which is considered secure, and f' then describes analogous security in a protocol that we are constructing.

Let formula φ_2 with inputs f, f' (occuring as free variables) describes a relation between f and f'. Suppose next that a formula φ_1 formalizes some cryptographic property of f. Usually it will be some cryptographic property of the primitive, for example that no polynomial time adversary on f has the probability better than negligible of breaking f. We can represent adversary as a polynomial function g. Then we can write that security property as a formula $\forall g \forall \vec{z} \varphi_1(f, g, z, \mathbf{s})$, where $\vec{z}$ are some (random) strings. We then want to prove that function f' satisfies a cryptographic property expressed by a formula ψ. Usually it will be some cryptographical property in the constructed protocol similar to φ_1. For example, it could be again security of f', meaning that no polynomial adverory attacking on f' has probability better than negligible of breaking f'. We can again write down this property as a formula $\forall g \forall z \psi(f', g, z, \mathbf{s})$.

If we can derive in T formula ψ from φ_1, φ_2 with rules of T, then function f' has the required property.

That is the main idea of the soundness theorem. Before we present a formal formulation, we need few more definitions.

Definition 1.1. *We say that formula φ is bounded if it contains only bounded quantifiers, it means quantifiers in the form $\forall x(|x| \leq |y| \rightarrow \varphi)$ or $\exists x(|x| \leq |y| \wedge \varphi)$, where x is a string(or integer) variable, and no function quantifiers.*

Definition 1.2. *Let have formula $\varphi(\vec{f}, \vec{z}, x)$ with free variables f, z, x. For any sequence of non-uniform polynomial functions $\vec{\alpha}$ we say that φ holds asymptotically in $\mathbb{N}$ if for all sequences of polynomials $\vec{p}$*

$$\exists n_0 \in \mathbb{N}\ \forall \vec{s}, t \in \mathbb{N}:\ |t| > |n_0|\ \wedge |s_i| = p_i(|x_i|) \wedge \Big(\varphi[\vec{\alpha}/\vec{f}, \vec{s}/\vec{z}, t/x]\ \text{ holds in } \mathbb{N}\Big).$$

We now formulate the Soundness theorem. But the problem is that the theorem does not hold as stated in [IK03] with the new axiom CONST. The reason is that the original theorem holds only for class of uniform functions, but we have the possibility of non-uniform functions with CONST axiom. To make the Soundness theorem holds, we need to change the formulation little bit. The following

formulation comes from Emil Jeřábek's work [J05]. In this work the theorem is formulated for circuits, not for algorithms. But a polynomial sized circuit is equivalent to a non-uniform polynomial time algorithm. So we can talk about circuits instead of non-uniform polynomial time algorithms. In the following $|C|$ denotes the size of circuit C.

Theorem 1.6. (Soundness theorem)([J05]). *Let φ and ψ be bounded formulas. Assume that*

$$T \vdash \forall g \forall \vec{z} \varphi(\vec{f}, g, \vec{z}, \boldsymbol{s}) \to \forall g \forall \vec{z} \psi(\vec{f}, g, \vec{z}, \boldsymbol{s}).$$

Lef $\vec{f} \in$ FP/poly. If for every polynomial $q(n)$, the formula $\varphi(\vec{f}, C, \vec{z}, x)$ holds for all sufficiently large x and all $|\vec{z}|$, $|C| \leq q(|x|)$, then the same is true of ψ.

The proof of the Soundness theorem is based on model theory by defining a little bit changed theory, for which there exists some model. It then uses properties of this model which implies that the theorem holds. Complete proof of the original theorem can be found in [IK03]. The proof of the modified theorem as we stated here can be found in [J05].

2. The Goldreich-Levin theorem

In the previous chapter we presented the formal logic system, theory T, for formal reasoning about cryptographic constructions. We also promised to show an application of the theory for some well-known cryptographic concept. In the rest of this work we will try to formalize the Goldreich-Levin theorem and its proof in theory T. The theorem deals with classic cryptographic concepts such as hard-core predicates or one-way functions. Proof of the theorem uses some other concepts and terms from cryptography and probability theory. Let us start with a quick introduction into this field of cryptology.

2.1 Hard-Core predicate

One of the most important concepts in constructing pseudorandom generators and secure cryptographical protocols are *hard-core predicates* and *one-way functions*.

Consider at first a function $\varepsilon : \mathbb{N} \to \mathbb{N}$. This function is called *negligible* if and only if

$$\forall k \geq 1 \; \exists n_k \; \forall n \geq n_k \; \epsilon(n) < \frac{1}{n^k}.$$

For instance functions $2^{-n}, n^{-\log n}$ are negligible functions, but n^{-100} is not.

Next consider a function $f : \{0,1\}^n \to \{0,1\}^m$. We call it *one-way* if $f(x)$, for $x \in \{0,1\}^n$, can be computed by a polynomial time algorithm but for every randomized algorithm A, that runs in time polynomial in n, there exists a negligible function $\varepsilon(n)$ such that

$$\forall n \in \mathbb{N} \; Pr_{x \in U_n} \left[A(1^n, f(x)) \in f^{-1}(f(x)) \right] \leq \varepsilon(n),$$

where U_n is the uniform distribution on inputs of length n.

In other words, function f is one-way if we can compute it easily, but given an y from the range of f it is hard to compute value x such that $f(x) = y$.

We can state some examples: *the factoring problem* (given a value pq determine one of the values p, q for chosen primes p, q satisfying some condition), *the discrete logarithm problem* (determine x from given values $\{p, g, g^x \mod p\}$, where g is a generator of Z_p^*).

It is also fair to say that these functions are just *assumed to be one-way*. In fact, we don't even know whether any one-way function exists or not. But we know that their existence relates with the well-known P vs. NP problem. In particular, we know that the existence of a one-way function implies $P \neq NP$. It can be showed by a contradiction argument. If $P = NP$ then the class of deterministic polynomial time functions equals the class of nondeterministic polynomial time

functions. But there exists a nondeterministic polynomial time algorithm that inverts any polynomial-time computable function. It implies that no one-way function exists. However, we can not prove the opposite direction (that the existence of a one way function follows from $P \neq NP$).

One could expect that if some function is hard to invert there should be some specific bits in input x, which are hard to compute given value $f(x)$. But consider this example. Let us have a one-way function f and for any $i = 0, \ldots, n$, define f_i as $f_i(x) = x_i \| f(x_0 \ldots x_{i-1} x_{i+1} \ldots x_{n-1})$. We see that f_i is also one-way for each i, but we can easily determine i-th bit of the input. So after n iterations we can determine the whole input.

Thus there is an obvious question: Is there some bit that can be extracted from x and is hard to compute given $f(x)$? We formalize this as follows:

Definition 2.1. *(Hard-Core bit)*

Let us have a one-way function $f : \{0,1\}^* \to \{0,1\}^*$ and a predicate $b : \{0,1\}^* \to \{0,1\}$. We say that b is a hard-core predicate for the function f if for every probabilistic polynomial algorithm A

$$Pr_x \left[A(f(x)) = b(x) \right] \leq \frac{1}{2} + \varepsilon(n)$$

where $\varepsilon(n)$ is a negligible function and $x \in U_n$ is taken uniformly at random.

At the end of this paragraph let us see one interesting theorem. See, for example, [P09] for a proof.

Theorem 2.1. ([G01a]). *Let f is a one-way permutation and b is a hard-core predicate of f. Then the function $g : \{0,1\}^n \to \{0,1\}^{n+1}$ defined as $g(x) = f(x) \| b(x)$ is a pseudorandom generator.*

So it gives us a manual how to find a proper pseudorandom generator. Of course, only under the assumption that we have some one-way permutation.

2.2 The Goldreich-Levin theorem

Another question which can come to mind is if there exists a hard-core bit for every one-way function. Answer is again unknown. But the situation is not so bad since Oded Goldreich and Leonid Levin have proved that every one-way function f can be transformed into another function g such that g is still one way function and g has a hard-core bit. We denote $\langle a, b \rangle$ the inner product modulo 2.

Theorem 2.2. ([GL89]). *Let f is an one-way function. Let us define function g as $g(x, r) = f(x) \| r$ for $(x, r) \in \{0,1\}^n \times \{0,1\}^n$. Then function b defined as $b(x, r) = \langle x, r \rangle$ is a hard-core predicate for the function g.*

2.3 Adaptation in our theory

2.3.1 Cryptographic terms

In the previous paragraphs we showed the main ideas and concepts leading to the Goldreich-Levin theorem. We presented definitions in classic complexity theory. But we would like to use these concepts in automated systems. So we will now try to adapt these constructions in our formal logic system presented in the previous chapter.

We will start with a negligible function. Recall that we have a formal logic system with a special constant $\boldsymbol{n} \in \mathbb{N}$, which we call *the security parameter*. We also have strings which can be viewed as integers using dyadic notation and function symbols for $k-$ary functions, where $k > 0$. Because of the basic property of the security parameter that each asymptotically true statement is true in $\boldsymbol{n}$ and our intuition that (and some models have this property) all strings have the length bounded by a polynomial in $\boldsymbol{n}$, we can transform the classic definition of a negligible function into the following one:

Definition 2.2. *Function* $q : \{0,1\}^n \to \{0,1\}^n$ *is negligible if and only if*

$$\forall z \ \forall x \text{ with } |x| = \boldsymbol{n}, \ q(x) \leq \frac{1}{|z|}.$$

For the definition of a one-way fuction we need one more object from our theory - a counting integer $\#(|x| = k)\varphi(x)$. It denotes the number of strings x with length k satisfying $\varphi(x)$. As we stated before, using this object we can reason about probabilities. We will use this idea right now.

Definition 2.3. *We say that function* $f : \{0,1\}^* \to \{0,1\}^*$ *is one-way if* $\forall A : \{0,1\}^* \to \{0,1\}^* \ \forall z \ \forall k \in \mathbb{N}$:

$$\frac{\#(|x| = k)(f(A(1^k, f(x))) = f(x))}{\#(|x| = k)(x = x)} \leq \frac{1}{|z|},$$

where function variable A ranges non-uniform polynomial time algortihms (thanks to the CONST axiom).

Similarily we can transform the definition of the hard-core predicate.

Definition 2.4. *Let* f *be a one-way function. Function* $b : \{0,1\}^* \to \{0,1\}$ *is a hard core predicate for the function* f *if* $\forall A : \{0,1\}^* \to \{0,1\}^* \ \forall z \ \forall k \in \mathbb{N}$:

$$\frac{\#(|x| = k)(A(1^k, f(x)) = b(x))}{\#(|x| = k)(x = x)} \leq \frac{1}{2} + \frac{1}{|z|}.$$

2.3.2 Inner product

Our current goal is to present an adaptation of the Goldreich-Levin theorem in our formal system. Theorem shows us how to find a hard-core predicate for a

specific type of one-way function. It says that a hard-core predicate has the form of the inner product modulo 2.

So our first task is to find an interpretation of a function calculating the inner product modulo 2. Let us describe what does it really mean in our system. Ordinarily the inner product of two k-coordinated tuples is defined as the sum of products of the corresponding coordinates. Binary inner product modulo 2 is defined identically using arithmetic modulo 2. In our theory we do not have binary representation, but dyadic representation. However, we will use the same logic. By the *dyadic inner product* we mean the number of ones modulo 2, which are common in both dyadic representation of the two numbers. In paricular, if we have two numbers x, y with dyadic representations $x = a_1a_2 \dots a_k$, $y = b_1b_2 \dots b_l$, we define their dyadic inner product as $\sum_{i=1}^{\min\{k,l\}} a_{k-i+1} \cdot b_{l-i+1} \mod 2$.

How can we do that? Let us have numbers 0,0 which corresponds to ε, ε. Obviously the resulting inner product is ε. The same is for the inner product of 0 and any other number. Let now have strings $b_1b_2 \dots b_r$ and $c_1c_2 \dots c_r$. We denote their inner product as d. Then the inner product of $b_1b_2 \dots b_rb_{r+1}$ and $c_1c_2 \dots c_rc_{r+1}$ is equal to $d + 1 \mod 2$ if and only if $b_{r+1} = c_{r+1} = 1$. In other cases the inner product will be equal to d.

In other words we can compute the inner product recursively by induction on the notation of two numbers. If we have two numbers k, l, represented by strings $k_1k_2 \dots k_r$, $l_1l_2 \dots l_s$ in dyadic notation, we can compute their inner product as inner product of $\left\lfloor \frac{k \dot{-} 1}{2} \right\rfloor$ and $\left\lfloor \frac{l \dot{-} 1}{2} \right\rfloor$ added by 1 if $k_r = l_r = 1$ and by 0 in other cases. We can continue in this way until we end up with the inner product of 0 and any number which is, as we noticed, 0.

We can summarize this procedure in the following definition.

Definition 2.5. *Let k, l are numbers in our theory in the form of $k = 2x + c$, $l = 2y + d$, where $c, d \in \{1, 2\}$. Then we define their inner product (denoted as $k \odot l$ or $IP(k, l)$) as follows:*

$$0 \odot 0 = 0$$

$$0 \odot y = 0$$

$$y \odot 0 = 0$$

$$2x + 1 \odot 2y + 1 = x \odot y$$

$$2x + 1 \odot 2y + 2 = x \odot y$$

$$2x + 2 \odot 2y + 1 = x \odot y$$

$$2x + 2 \odot 2y + 2 = x \odot y + 1$$

Example: Let us show this definition on some example. For instance we take

$k = 14$, $l = 12$. Then

$$\begin{aligned} k \odot l &= (2 \cdot 6 + 2) \odot (2 \cdot 5 + 2) = 6 \odot 5 + 1 \\ &= (2 \cdot 2 + 2) \odot (2 \cdot 2 + 1) + 1 = 2 \odot 2 + 1 \\ &= (2 \cdot 0 + 2) \odot (2 \cdot 0 + 2) + 1 = 0 \odot 0 + 1 + 1 \\ &= 0 + 2 = 2 \end{aligned}$$

The definition holds also for numbers which have dyadic representations with different lengths. Then their inner product is the inner product on their common part. In particular, it means that if we have $k = k_1 k_2 \dots k_r$, $l = l_1 l_2 \dots l_s$, where $r < s$, then $IP(k, l) = IP(k_1 k_2 \dots k_r, l_{s-r+1} \dots l_s)$. That is because we count it recursively starting at the last bit. For example $10 \odot 29 = (2 \cdot 4 + 2) \odot (2 \cdot 14 + 1) = 4 \odot 14 = (2 \cdot 1 + 2) \odot (2 \cdot 6 + 2) = 1 \odot 6 + 1 = (2 \cdot 0 + 1) \odot (2 \cdot 2 + 2) + 1 = 0 \odot 2 + 1 = 1$.

Note: Let us realize some properties of inner product:

- $IP(k, l) \geq 0$. Moreover $IP(k, l) \geq 1 \Leftrightarrow$ k and l are both even.
- $IP(k, k)$ denotes number of ones in dyadic representation of k.

Now we are able to compute the inner product of arbitrary two numbers. This value is a number between 0 and a length of the smaller of these two numbers. But we would like to simulate calculation of inner product modulo 2 - to use it later for defining a hard-core predicate. So we need to restrict range of inner product just to numbers $2, 1$ (in dyadic bits 1,0). To achieve this, we construct function which makes from number m value ($m \mod 2$).

Definition 2.6. *Using LRN axiom we define function symbol drop as follows:*

$$\begin{gathered} \mathsf{drop}(x, 0) = x \\ \mathsf{drop}(x, 2y + 1) = \lfloor \tfrac{\mathsf{drop}(x,y) \dot{-} 1}{2} \rfloor \\ \mathsf{drop}(x, 2y + 2) = \lfloor \tfrac{\mathsf{drop}(x,y) \dot{-} 1}{2} \rfloor \\ |\mathsf{drop}(x, y)| \leq |x|. \end{gathered}$$

What does drop exactly do? Denote r the number of bits in dyadic notation of y. Function $\mathsf{drop}(x, y)$ extracts last r bits from the dyadic representation of x and returns the remaining part. If y has in dyadic more bits than x, then $\mathsf{drop}(x, y)$ returns ε.

Example: Let us show on an example how this function works. Let $k = 12$, $l = 3$. Then

$$\begin{aligned} \mathsf{drop}(k, l) &= \mathsf{drop}(12, 3) = \mathsf{drop}(12, 2 \cdot 1 + 1) \\ &= \left\lfloor \frac{\mathsf{drop}(12, 1) - 1}{2} \right\rfloor = \left\lfloor \frac{\mathsf{drop}(12, 2 \cdot 0 + 1) - 1}{2} \right\rfloor \\ &= \left\lfloor \frac{\left\lfloor \frac{\mathsf{drop}(12,0) - 1}{2} \right\rfloor - 1}{2} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{12 - 1}{2} \right\rfloor - 1}{2} \right\rfloor \\ &= \left\lfloor \frac{4}{2} \right\rfloor = 2 \end{aligned}$$

In dyadic $k = 101$. Length of l is 2, so $\mathsf{drop}(12, 3)$ should return string 1 which fits.

Note: We can see that $\mathsf{drop}(x, y)$ does not depend on value y. The only thing on which $\mathsf{drop}(x, y)$ depends is the number of bits in dyadic representation of y, not its concrete value. It means that for instance $\mathsf{drop}(12, 3) = \mathsf{drop}(12, 4) = \mathsf{drop}(12, 5) = \mathsf{drop}(12, 6) = 2$.

Now we would like to define a function which returns the last bit in the dyadic representation of some number. This operation can be defined using the drop function. At first define the inverse function to drop.

Definition 2.7. *Function lv is defined as follows:*

$$\mathsf{lv}(x, y) = x \dot{-} \mathsf{drop}(x, y) \cdot (1 \otimes y).$$

Example: Function lv is inverse of the function drop. For example let us take $x = 20$, $y = 11$. Then $\mathsf{drop}(x, y) = 1$. Meaning that if we extract $11 \sim 100$ from $20 \sim 0101$, the remainings is $1 \sim 0$. We also have

$$\begin{aligned} \mathsf{lv}(x, y) = \mathsf{lv}(20, 11) &= 20 \dot{-} \mathsf{drop}(20, 11) \cdot (1 \otimes 11) \\ &= 20 - 1 \cdot 2^3 = 12 \sim 101, \end{aligned}$$

which is exactly that part of x, which we extracted using drop.

Using these two functions we are now able to find any particular part of the dyadic representation of arbitrary number. If we have string x with length k, then we denote

$$y_{\{1 \ldots i\}} = \begin{cases} x & \text{if } i \geq k \\ \mathsf{drop}(x, (k - i) \otimes 1) & \text{otherwise} \end{cases}$$

$$y_{\{i \ldots k\}} = \begin{cases} 0 & \text{if } i > k \\ \mathsf{lv}(x, (k - (i - 1)) \otimes 1) & \text{otherwise} \end{cases}$$

Our goal is to extract the last bit from the dyadic representation which can be easily done using previous notation:

$$x_k = \mathsf{lv}(x, (k - (k - 1)) \otimes 1 = \mathsf{lv}(x, 1 \otimes 1) = \mathsf{lv}(x, 2).$$

Now we are able to determine whether the last bit of some string is 0 or 1. We need this to compute the inner product mod 2. The problem is that using the above calculation for inner product we get just some string. If we would compute last bit of its dyadic representation, we get 0 or 1, but this would not be value, which we wanted to get. To get mod 2 value, we need to determine the last bit of binary representation not dyadic representation. So our task now is to find a way how to get the binary representation of some string. For example, for $x = 3$, which corresponds to 00 in dyadic, we would like to get binary representation of x, i.e. 11, which corresponds to 6. The following table specifies the situation.

number	string	binary	number
0	ε	0	1
1	0	1	2
2	1	10	5
3	00	11	6
4	01	100	11
5	10	101	12
6	11	110	13
7	000	111	14
8	001	1000	23
9	010	1001	24
10	011	1010	25

Definition 2.8. *We define function* lb *as follows:*

$$\mathsf{lb}(x) = x + (2x \otimes 1) - 1.$$

Such a function does exactly what we would like to achieve. For example for $x = 8$ we have $\mathsf{lb}(x) = 8 + (2^{|2\cdot 8|}) - 1 = 8 + 2^4 - 1 = 23$. 8 corresponds to 001 and 23 corresponds to 1000, which is 8 in binary.

Now we can sumarize the whole calculation of inner product mod 2.

Definition 2.9. *We define function symbol* IP_2 *(also denoted* $\odot$*) as follows:*

$$\mathsf{IP}_2(x,y) = \begin{cases} 0 & \text{if } \mathsf{lv}(\mathsf{lb}(\mathsf{IP}(x,y)), 2 \otimes 1) = 0 \\ 1 & \text{if } \mathsf{lv}(\mathsf{lb}(\mathsf{IP}(x,y)), 2 \otimes 1) = 1 \end{cases}$$

2.3.3 The theorem

To formalize the Golreich-Levin theorem we need one more function.

Definition 2.10. *Function* $\circ$ *is defined using LRN as follows:*

$$x \circ 0 = x$$
$$x \circ (2y+1) = 2 \cdot (x \circ y) + 1$$
$$x \circ (2y+2) = 2 \cdot (x \circ y) + 2$$
$$|x \circ y| \leq |x| + |y|$$

This function returns the concatenation of strings x and y. For example let $x = 9$, $y = 13$. Then

$$\begin{aligned} x \circ y &= 9 \circ (2 \cdot 6 + 1) = 2 \cdot (9 \circ 6) + 1 = 2 \cdot (9 \circ (2 \cdot 2 + 2)) + 1 \\ &= 2 \cdot (2 \cdot (9 \circ 2) + 2) + 1 = 2 \cdot (2 \cdot (9 \circ (2 \cdot 0 + 2)) + 2) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot (9 \circ 0) + 2) + 2) + 1 = 85 \end{aligned}$$

We have $x =$010 and $y =$110. Concatenation of x and y is 010110 which corresponds to 85.

Now we can present a formalization of the Goldreich-Levin theorem in our logic system.

Theorem 2.3. *(The Goldreich-Levin theorem)*

Let f be a one-way function. Define function g as

$$g(x, r) = f(x) \circ r \text{ for } (x, r) \in \{0,1\}^n \times \{0,1\}^n,$$

and define function b as

$$b(x, r) = \mathit{IP}_2(x, r).$$

Then $b(x, r)$ is a hard-core predicate for the one-way function g.

We now have the formalization of the statement of the Golreich-Levin theorem in our theory. Our next task is to show that such statement can be proved in the theory as well. This will be the subject of the next chapter.

3. Proof of the Goldreich-Levin theorem

In this chapter we will try to prove the Goldreich-Levin theorem in our formal system of logic. The proof will be based on the classic proof of the Goldreich-Levin theorem as presented in [G01b], but adapted to our system.

3.1 Idea of the original proof

Let us recall the original proof first. We assume that there exists $u \in \{0,1\}^n$ and function $h : \{0,1\}^n \to \{0,1\}$ such that $\mathrm{Pr}_r[\mathrm{h(r)}{=}\langle u, r\rangle] \geq \frac{1}{2} + \varepsilon$, where r is taken uniformly at random. We then want to construct a probabilistic polynomial time (in (ε^{-1}, n)) algorithm (called the Goldreich-Levin algorithm), which uses the funcion h as an oracle and its task is to return u with probability at least $\Omega(\frac{\varepsilon^2}{n})$. Then we can easily show that if such an algorithm exists, the Goldreich-Levin theorem holds.

The main idea of constructing the algorithm can be described as follows. At first suppose the most simple situation when $\mathrm{Pr}_r[\mathrm{h(r)}{=}\langle u, r\rangle] = 1$. In this case we can just ask for values $h(e_i)$, $i = 1 \ldots n$, where e_i is a vector, which has 1 on $i-$th place and 0 otherwise. Then $h(e_i) = u_i$. Next suppose that $\mathrm{Pr}_r[\mathrm{h(r)}{=}\langle u, r\rangle] > \frac{3}{4} + \varepsilon$. In this case we choose r and ask for $h(r) \oplus h(r \oplus e_i)$. If $h(r) = \langle u, r\rangle$ and $h(r \oplus e_i) = \langle u, r \oplus e_i\rangle$, then $h(r) \oplus h(r \oplus e_i) = \langle x, e_i\rangle = x_i$. We also have

$$Pr_r[h(r) = \langle x, r\rangle \wedge h(r \oplus e_i) = \langle x, r \oplus e_i\rangle] \geq 1 - 2 \cdot \left(\frac{1}{4} - \varepsilon\right) = \frac{1}{2} + 2\varepsilon.$$

We can now repeate this procedure sufficiently many times for random r to receive u_i with high enough probability - by taking majority of the resulting values.

This procedure is convenient for this case but not for the original assumption $\mathrm{Pr}_r[\mathrm{h(r)}{=}\langle u, r\rangle] \geq \frac{1}{2} + \varepsilon$. If we compute such probability as in the previous case, we get

$$Pr_r[h(r) = \langle x, r\rangle \wedge h(r \oplus e_i) = \langle x, r \oplus e_i\rangle] \geq 1 - 2 \cdot \left(\frac{1}{2} - \varepsilon\right) = 2\varepsilon.$$

We can not use the same argument as before because amplifying this process will not give us better probability. The reason for that is that we double the probability of error during each turn.

A convenient solution is to ask just once for a value of h during each turn. We will ask just for $h(r \oplus e_i)$ and we will guess values $\langle u, r\rangle$. Denote these guesses as b_i. We do this for some particular number of strings and take the majority value of results. At the end we will get the succes probability at least $1 - \frac{1}{4n}$.

Combining this with the probability that all our guesses are right, we get sufficiently good probability that the algorithm outputs u. We will give more details later in this chapter. Now our task is to transform this proof to our system.

3.2 Idea of the formalized proof

In the first chapter we presented the Soundness theorem for proving the correctness of cryptographical constructions. Now we would like to use this theorem in a proof of the Goldreich-Levin theorem. Let us describe the situation. We have a cryptographic primitive as an assumption for the theorem - a one way permutation. Denote it f. We would like to obtain another cryptographic primitive f'- the hard core predicate for a function f'' constructed from f. Theorem also explicitly tells us how to construct the hard core predicate.

The Soundness theorem then shows us how to formalize this situation. We want to show

$$T, \varphi_2(f, f', f''), \forall g \forall \vec{z} \varphi_1(f, g, \vec{z}, \mathbf{s}) \vdash \forall h \forall \vec{z} \psi(f', f'', h, \vec{z}, \mathbf{s})$$

where f denotes the one way permutation, f' denotes a function describing the hard-core predicate for function f'', both made from f. Formula φ_1 formalizes security properties of the one-way permutation, ψ describes security of the hard-core predicate and φ_2 shows us how to obtain f', f'' from f.

The previous statement can be also viewed as follows:

$$T, \varphi_2(f, f', f''), \exists h \exists \vec{z} \neg \psi(f', f'', h, \vec{z}, \mathbf{s}) \rightarrow \exists g \exists \vec{z} \neg \varphi_1(f, g, \vec{z}, \mathbf{s}).$$

In other words, having the transformations $f \rightarrow f'$, $f \rightarrow f''$ symbolised by φ_2 if there exists an adversary (represented by function h) which can break ψ (i.e. he can predict the inner product) then there exists an adversary g who testifies that f is not a one-way permutation.

We would like to use just open formulas. Therefore we rewrite our statement as

$$\varphi_2(f, f', f''), \neg \psi(f', f'', h, \vec{z}, \mathbf{s}) \rightarrow \exists GL \exists w \neg \varphi_1(f, GL, w, \mathbf{s}).$$

Here GL is a function variable (intended for the Goldreich-Levin algorithm). By Cobham's theorem we know that there exists a way how to define in our theory probabilistic polynomial function GL from function h. We define this GL function to be the Goldreich-Levin algorithm which uses h. Denote it GL_h.

3.3 Steps of the proof

We will divide the proof into several steps:

1. We will formalize the formulas $\varphi_1, \varphi_2, \psi$ in our formal system.
2. Using limited recursion on notation and other axioms from our theory we will give an explicit definition of the Goldreich-Levin algorithm GL_h
3. Finaly we will show $T \vdash \varphi_2(f, f', f''), \neg \psi(f', f'', h, z, \mathbf{s}) \rightarrow \exists w \neg \varphi_1(f, GL_h, w, \mathbf{s})$.

3.4 Formalization of φ_1

We want to find a formula $\varphi_1(f, g, \vec{z}, \mathbf{s})$ which formalizes cryptographic security of the function f, i.e. one-way permutation, given any adversary g and a vector $\vec{z}$. So we have to find formulas which describe the requirements we have:

1. f is permutation,
2. f is one-way.

Note: According to the security parameter we have we can make one more reduction. We can work in a vector space of dimension of $\boldsymbol{n}$. If the proof would hold in this vector space it will then hold asymptotically. We will use this reduction in the whole proof.

Ad 1.

We have to present properties which say that f transforms an input of length $\boldsymbol{n}$ to an output of length $\boldsymbol{n}$ and f is a bijection (i.e. f is surjective and injective). The first property can be written as

$$\forall(|x| = \boldsymbol{n})|Ap^n(f, x)| = \boldsymbol{n}$$

We will describe the injective mapping as follows:

$$(\forall|x| = \boldsymbol{n})\,(\forall|y| = \boldsymbol{n})\,(f\,(x) = f(y)) \Rightarrow (x = y)$$

Surjectivity then follows from the two previous properties (injectivity and the length of outputs from f).

Ad 2.

Now we want to describe one-wayness of the function f - it means that there is just a negligible probability that an adversary can compute x given $f(x)$. We presented a definition of a one-way function in the previous chapter, so we will just modify it to the actual parameters. That leads to a formula:

$$\frac{\#(|x| = \boldsymbol{n})(f(g(f(x))) = f(x))}{\#(|x| = \boldsymbol{n})(x = x)} \leq \frac{1}{|z|}$$

So we have the formula $\forall g \forall z \varphi_1(f, g, z, \mathbf{s})$ in the form of

$$\begin{aligned}
&\forall g \forall z(\\
&\quad \forall(|x| = \boldsymbol{n})|Ap^n(f, x)| = \boldsymbol{n} \wedge \\
&\quad (\forall|x| = \boldsymbol{n})\,(\forall|y| = \boldsymbol{n})\,(f\,(x) = f(y)) \Rightarrow (x = y) \wedge \\
&\quad \tfrac{\#(|x|=\boldsymbol{n})(f(g(f(x)))=f(x))}{\#(|x|=\boldsymbol{n})(x=x)} \leq \tfrac{1}{|z|})
\end{aligned}$$

3.5 Formalization of φ_2

The formula φ_2 should formalize the definition of f' and f'' in terms of f. It tells us how to, from the one-way permutation f, obtain the hard-core predicate f' for some function f''. Now our goal is to give the definition of f',f'' and afterwards we will be able to formalize the security of f' through the formula ψ.

The Goldreich-Levin theorem gives us a manual how to define f' and f''. Given the one-way permutation f, the Golreich-Levin theorem says that a function $p(x,r) = f(x) \circ r$ is also one-way and a function $b(x,r) = x \odot r$ is a hard-core predicate for p.

Therefore, we define functions f', f'' and formula $\varphi_2(f, f', f'')$ as follows. Put:

$$\begin{aligned} f'(x,r) =& x \odot r \\ f''(x,r) =& f(x) \circ r \end{aligned}$$

and define formula $\varphi_2(f, f', f'')$ in the form of $\varphi_2'(f, f') \wedge \varphi_2''(f, f'')$ where:

- $\varphi_2'(f, f')$ is the formula in the form $\forall r \ f'(x,r) = x \odot r$,
- $\varphi_2''(f, f'')$ is the formula in the form $\forall r \ f''(x,r) = f(x) \circ r$.

Functions $\odot$ and $\circ$ were defined in Section 2.3.

3.6 Formalization of ψ

As we already mentioned in the previous paragraphs, formula ψ formalizes the security of a hard-core predicate. Given formula φ_2 we have functions f' and f'' describing how to obtain a hard-core predicate from a one way permutation.

We have to formalize that f' is a hard-core predicate for one-way function f''.

We also know by the theorem in the previous chapter that having f' the hard-core predicate for the function f'', function defined as $f'' \circ f'$ is a pseudorandom generator with stretch 1. The theorem could be then formulated in a different way talking about the resulting pseudorandom generator. But we will concentrate on the original Theorem 2.3.

Security property for a hard-core predicate can just be modified from what we presented in previous chapter. We formalize that f' is a hard-core predicate for f'' by a formula $\forall g \forall z \psi(f', g, z, \mathbf{s})$ in the form of

$$\forall g \forall z \Big(\frac{\#(|x| = \boldsymbol{n} \wedge |r| = \boldsymbol{n})(g(f''(x,r)) = f'(x,r))}{\#(|x| = \boldsymbol{n} \wedge |r| = \boldsymbol{n})(x = x)} \leq \frac{1}{2} + \frac{1}{|z|} \Big)$$

3.7 Construction of the Goldreich-Levin algorithm

3.7.1 Preparation

Now we would like to define the Goldreich-Levin algorithm. We will assume, similarly as in the original proof of the Goldreich-Levin theorem, that we have some function that returns the inner product with high probability. Denote such function h. We will construct the Golreich-Levin algorithm as a function to be denoted GL_h.

Let us give some more details about the problem. We assume that there exist some u and a function h such that $\Pr_r[h(u,r) = u \odot r] \geq \frac{1}{2} + \varepsilon$, where $\varepsilon := \frac{1}{|z|}$ represents the probability of an error. We will use both notations interchangeably. Given the function h we will construct the function GL_h which returns u with probability at least $\Omega(\frac{\varepsilon^2}{n})$ and runs in time $\text{poly}(\boldsymbol{n}, \varepsilon^{-1})$. We will define this function in our theory by the limited recursion on notation.

First we have to define some other functions that will help us with defining the algorithm GL_h.

Definition 3.1. *Function symbol $c(\vec{x}, k)$ for each $k > 0$ is defined using LRN as follows:*

$$c(\vec{x}, 0) = \epsilon$$
$$c(\vec{x}, i+1) = c(\vec{x}, i) \circ x_{i+1}$$
$$|c(\vec{x}, i)| \leq |\vec{x}|$$

Function c returns the concatenation of all coordinates in vector $\vec{x}$. Starting from the last coordinate the function concatenates this coordinate with the rest of $\vec{x}$. Definition of function symbol $\circ$ was presented in the previous chapter. For example $c(\{1,2,3\},3) = c(\{1,2,3\},2)\circ 3 = c(\{1,2,3\},1)\circ 2\circ 3 = c(\{1,2,3\},0)\circ 1\circ 2\circ 3 = 0\circ 1\circ 2\circ 3 = \epsilon\circ \mathsf{0}\circ \mathsf{1}\circ \mathsf{00} = \mathsf{0100}$. Let us also realize that manipulation with the i-th element of $\vec{x}$ is guaranteed due to the axiom $\mathsf{PROJ}(n,i)$. The function c will be very helpful in further study - it gives us a way how to view vector as one string of bits which can be used as a corresponding number (using dyadic notation).

Definition 3.2. *Function maj is defined as follows:*

$$\mathsf{maj}(x) = \begin{cases} \mathsf{0} & \text{if } s'(x,x) < x \\ \mathsf{1} & \text{if } s'(x,x) > x \\ \epsilon & \text{if } s'(x,x) = x \end{cases},$$

where function s' is defined using LRN as:

$$s'(x,0) = x$$
$$s'(x,2y+1) = s'(x,y)\dot{-}1$$
$$s'(x,2y+2) = s'(x,y) + 1$$

$$|s'(x,y)| \leq x + |x|$$

Function maj returns the majority value in the dyadic representation of x, i.e. the value occuring most frequently in x. If x has an equal number of ones and zeros, it returns the empty word ϵ. The idea behind this function is that it takes value of x (using dyadic conversion) as some *base value* and, starting from the last bit, it adds 1 to that value if that bit is 1 or it substracts 1 if the current last bit is 0. It is well defined - $s'(x,x)$ will never be negative - x is always at least $|x|$.

The next operation which will be useful is the so-called XOR. It means operation on bits (standardly denoted as $\oplus$), which is defined as: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$.

We can easily extend the definition of $\oplus$ to strings of arbitrary length using LRN. We keep the symbol $\oplus$ as well for the extended function, because by restricting the extended function to the original inputs (i.e. bits 0,1) we get the original function.

Definition 3.3. *We define function $\oplus$ for x, y of length k as follows:*

$$0 \oplus 0 = 0$$

$$0 \oplus x = x \oplus 0 = 0$$

$$2x + 1 \oplus 2y + 1 = 2 \cdot (x \oplus y) + 1$$

$$2x + 1 \oplus 2y + 2 = 2 \cdot (x \oplus y) + 2$$

$$2x + 2 \oplus 2y + 1 = 2 \cdot (x \oplus y) + 2$$

$$2x + 2 \oplus 2y + 2 = 2 \cdot (x \oplus y) + 1$$

$$|x \oplus y| \leq |x|$$

It is important to notice that this function $\oplus$ is a bit-wise XOR of two strings x, y.

3.7.2 The algorithm

We can now formulate the Golreich-Levin algorithm, adapted to our theory. After that we will formulate it into the function GL_h.

1. We choose $k <\boldsymbol{n}$ such that $2^k = c \cdot \boldsymbol{n} \cdot \varepsilon^{-2}$ for some value c. Such k certainly exists and can be found in polynomial time since $k = \log(c \cdot \boldsymbol{n} \cdot \varepsilon^{-2})$. Take t any string of length k.
2. We choose uniformly at random vectors $\vec{s_1}, \ldots, \vec{s_k}$ of length $\boldsymbol{n}$ and bits $b_1, \ldots, b_k$.
3. For each non-empty subset $J \subseteq \{1, 2, \ldots, k\}$ we want to compute

$$\vec{r_J} := \bigoplus_{j \in J} \vec{s_j}$$

$$b_J := \bigoplus_{j \in J} b_j$$

We can simulate this calculation in our theory as follows. Let us realize first that the number of non-empty subsets of $\{1, 2, \ldots, k\}$ is $2^k - 1 = t \otimes 1 - 1$. This is exactly the number of strings in dyadic notation with length equal to k except string $\mathsf{00}\ldots\mathsf{0}$. These strings correspond to numbers $2^k, 2^k + 1, \ldots, 2^{k+1} - 2$ (or equally $t \otimes 1, t \otimes 1 + 1 \ldots, 2 \cdot (t \otimes 1) - 2$). The desired sums (values r_J, b_J) can be obtained from these strings as well. We can view the last bit of such a string as s_1 (resp. b_1), previous bit as s_2 (resp. b_2) etc. Then we can just make XOR of those elements, which have 1 on corresponding position in string. Running this for all possible strings we get all posible combinations. For example, let us have $k = 4$ and a string $\mathsf{1010}$ in dyadic. It corresponds to a subset $J = \{s_2, s_4\}$ and using our logic we get $s_2 \oplus s_4$. Equivalently we get $b_2 \oplus b_4$. To formalize this, we define function symbols b, r as follows:

$$b(x, 0) = 0$$
$$b(x, 2y + 1) = b(x, y)$$
$$b(x, 2y + 2) = b(x, y) \oplus b_{|x|-|y|}$$
$$|b(x, y)| \leq |b_1|$$

$$r(x, 0) = 0$$
$$r(x, 2y + 1) = r(x, y)$$
$$r(x, 2y + 2) = r(x, y) \oplus \vec{s}_{|x|-|y|}$$
$$|r(x, y)| \leq |\vec{s_1}|$$

4. For each $i = 1, \ldots, \boldsymbol{n}$ we define

$$a_{i,J} := b_J \oplus h(\vec{r_J} \oplus e_i) \text{ for all } J$$
$$a_i := \mathsf{maj}_J\{a_{i,J}\}$$

In our theory we can do this by defining the function symbol a as:

$$a(i, j) = b(j, j) \oplus h(r(j, j) \oplus e_i)$$

We then define function v which gives us concatenation of all $a_{i,J}$ for all J. That will be usefull to get the majority of the values and compute a_i. Let us remind that we want to have these values only for $j = 2^k, \ldots, 2^{k+1} - 2$ (which correspond to non-empty subsets J with length k).

$$v(i, k, t \otimes 1) = a(i, t \otimes 1)$$
$$v(i, k, j + 1) = v(i, k, j) \circ a(i, j + 1)$$
$$|v(i, k, j)| \leq (t \otimes 1) - 1$$

5. Return the concatenation of all a_i.

Using the algorithm and functions that we have just presented we can define desired function GL_h which simulates the whole Goldreich-Levin algorithm, as was just presented.

Definition 3.4. *Let us suppose, that there exists a vector $\vec{u}$ and a function h such that $Pr_r[h(r) = u \odot r] \geq \frac{1}{2} + \varepsilon$. Then we define for $k = \log(c \cdot \boldsymbol{n} \cdot \varepsilon^{-2})$, for each randomly chosen vectors $s_1, \ldots, s_k$ and for randomly chosen $b_1, \ldots, b_k$, where $b_i \in \{0, 1\} \forall i$, function GL_h as*

$$GL_h(1) = \mathit{maj}(v(1, k, 2 \cdot (t \otimes 1) - 2))$$

$$GL_h(i+1) = GL_h(i) \circ \mathit{maj}(v(i, k, 2 \cdot (t \otimes 1) - 2))$$

$$|GL_h(i)| \leq \boldsymbol{n},$$

where we use functions $v(i, k, j), r(j, j), b(j, j)$ as defined in the previous paragraph.

3.7.3 The correctness and the success probability of the algorithm

We have to verify the correctness of this algorithm and the probability of success of finding u. The following will be based on the standard analysis of the Goldreich-Levin algorithm. At first let us make one abbreviation.

Definition 3.5. *Let $\varphi(x)$ be a formula without quantifiers in our theory and z some number. We define symbol $\mathit{Prob}^*_{|x|=z}$ as*

$$\mathit{Prob}^*_{|x|=z}[\varphi(x)] := \frac{\#(|x| = z)\,\varphi(x)}{\#(|x| = z)\,(x = x)}$$

Let us notice that by a symbol $\mathsf{Prob}^*[\varphi(x)]$ we mean the probability of $\varphi(x)$ in the sense of above definition taken over the possible choices of the inputs into the algorithm, i.e. over the values $s_1, \ldots, s_k$ of length $\boldsymbol{n}$ and $b_1, \ldots, b_k$ each of length 1.

Lemma 3.1. $\mathit{Prob}^*_{|\vec{b}|=k}[b_1 = s_1 \odot u \wedge b_2 = s_2 \odot u \wedge \ldots b_k = s_k \odot u] = \frac{1}{t \otimes 1}$.

Proof: We will prove this lemma by induction by k. This type of proof is allowed due to LIND axioms. For $k = 1$ we have

$$\mathsf{Prob}^*_{|\vec{b}|=1}[b_1 = s_1 \odot u] = \frac{\#(|b_1| = 1)(b_1 = s_1 \odot u)}{\#(|b| = 1)(b_1 = b_1)} = \frac{1}{2}.$$

For the iduction step denote $\varphi(k+1) = b_1 = s_1 \odot u \wedge \cdots \wedge b_k = s_k \odot u \wedge b_{k+1} = s_{k+1} \odot u$ and $|\vec{b}| = |b_1| = 1 \wedge \cdots \wedge |b_{k+1}| = 1$. Then

$$\mathsf{Prob}^*_{|\vec{b}|=k+1}[\varphi(k+1)] = \frac{\#\left(|\vec{b}| = k+1\right)(\varphi(k+1))}{\#(|\vec{b}| = k+1)(b_1 = b_1 \wedge \cdots \wedge b_{k+1} = b_{k+1})} = \frac{1}{(t \otimes 1) \cdot 2}.$$

□

Before we state the next lemma, we need one property of the inner product. This is well known for the classic inner product but we have to show that it holds also in our theory.

Lemma 3.2. *For any a, b, c with length k:*

$$(a \odot c) \oplus (b \odot c) = (a \oplus b) \odot c.$$

Proof: We prove this property within induction by length of a, b, c. For $k = 1$ the only possibilities for a, b, c are 0,1 in dyadic. So we have the following combination of values:

a	b	c	$(a \odot c) \oplus (b \odot c)$	$(a \oplus b) \odot c$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

For the induction step let us realize that a, b, c are in the form of $a = 2x + 1$ or $a = 2x + 2$, $b = 2y + 1$ or $b = 2y + 2$, $c = 2z + 1$ or $c = 2z + 2$. We have again just eight combination of threesomes. Using definitions of functions $\oplus, \odot$ we have, for example:

$$(2x + 2 \odot 2z + 1) \oplus (2y + 1 \odot 2z + 1) = (x \odot z) \oplus (y \odot z) = (x \oplus y) \odot z.$$

The last equation follows from the induction assumption. We also have

$$(2x + 2 \oplus 2y + 1) \odot 2z + 1 = 2(x \oplus y) + 2 \odot 2z + 1 = (x \oplus y) \odot z.$$

Other possibilities can be done analogously. Finaly we get

a	b	c	$(a \odot c) \oplus (b \odot c)$	$(a \oplus b) \odot c$
2x+1	2y+1	2z+1	$(x \oplus y) \odot z$	$(x \oplus y) \odot z$
2x+1	2y+1	2z+2	$(x \oplus y) \odot z$	$(x \oplus y) \odot z$
2x+1	2y+2	2z+1	$(x \oplus y) \odot z$	$(x \oplus y) \odot z$
2x+1	2y+2	2z+2	$(x \oplus y) \odot z + 1$	$(x \oplus y) \odot z + 1$
2x+2	2y+1	2z+1	$(x \oplus y) \odot z$	$(x \oplus y) \odot z$
2x+2	2y+1	2z+2	$(x \oplus y) \odot z + 1$	$(x \oplus y) \odot z + 1$
2x+2	2y+2	2z+1	$(x \oplus y) \odot z$	$(x \oplus y) \odot z$
2x+2	2y+2	2z+2	$(x \oplus y) \odot z$	$(x \oplus y) \odot z$

□

Lemma 3.3. *Let $b_1 = s_1 \odot u \wedge b_2 = s_2 \odot u \wedge \cdots \wedge b_k = s_k \odot u$. Then $\forall\ j = 2^k, \ldots, 2^{k+1} - 2$ it holds: $b(j, j) = r(j, j) \odot u$.*

Proof: We prove this lemma using LIND axiom again with induction by k. For $k = 1$ we have just $j = 2$. Then $r(j, j) \odot u = s_1 \odot u = b_1 = b(j, j)$.

For the iduction step denote $B_{k+1} = b_1 \oplus b_2 \oplus \cdots \oplus b_{k+1}$, $S_{k+1} = s_1 \oplus s_2 \oplus \cdots \oplus s_{k+1}$. Without loss of generality we can show induction step just for the *maximal* value 2^{k+1}, which corresponds to subset $\{1, 2, \ldots, k + 1\}$. For other values (i.e. other

subsets) it holds by the induction assumption. We have

$$\begin{aligned} b(2^{k+2}-2,2^{k+2}-2) =& B_{k+1} = (s_1 \odot u) \oplus (s_2 \odot u) \oplus \cdots \oplus (s_{k+1} \odot u) \\ =& r(2^{k+1}-2,2^{k+1}-2) \oplus (s_{k+1} \odot u) \\ =& (S_k \odot u) \oplus (s_{k+1} \odot u) \\ =& (S_k \oplus s_{k+1}) \odot u = r(2^{k+2}-2,2^{k+2}-2) \odot u. \end{aligned}$$

□

Before we state the most important observation we need to define a few concepts and notation known from classic probability theory.

Definition 3.6. *Let $\varphi(x)$ be a formula in our theory. We define symbol $\mathit{Exp}^*[\varphi(x)]$ as*

$$\mathit{Exp}^*[\varphi(x)] = \sum_{|x|=\boldsymbol{n}} \mathit{Prob}^*_x[\varphi(x)] \cdot x,$$

where the operation $\sum$ represents sum of values and can be easily defined using LRN and the sum is taken over all possible integers x. We then define symbol $\mathit{Var}^*[\varphi(x)]$ as

$$\mathit{Var}^*[\varphi(x)] = \mathit{Exp}^*\left[\varphi(x) - (\mathit{Exp}^*[\varphi(x)])^2\right].$$

Later we will need also the following property of these symbols. In classic theory it is known as the Chebysev inequality.

Lemma 3.4.

$$\forall r \leq \boldsymbol{n}^k : \quad \mathit{Prob}^*_{|x|=\boldsymbol{n}}\left[\varphi(x) - \mathit{Exp}^*\left[\varphi(x)\right] \geq r\right] < \frac{\mathit{Var}^*[\varphi(x)]}{r^2}$$

Proof: Let define set S to be the set of all x satisfying $\varphi(x) - \mathsf{Exp}^*\varphi(x) \geq r$. Then we have

$$\begin{aligned} \mathsf{Var}^*\left[\varphi(x)\right] =& \sum_{x \leq \boldsymbol{n}} \mathsf{Prob}^*_{|x|=\boldsymbol{n}}(\varphi(x))(\varphi(x) - \mathsf{Exp}^*\left[\varphi(x)\right])^2 \\ =& \sum_{x \in S} \mathsf{Prob}^*_x(\varphi(x))(\varphi(x) - \mathsf{Exp}^*\left[\varphi(x)\right])^2 + \\ &+ \sum_{x \notin S} \mathsf{Prob}^*_x(\varphi(x))(\varphi(x) - \mathsf{Exp}^*\left[\varphi(x)\right])^2 \\ \overset{\text{(i)}}{\geq}& \sum_{x \in S} \mathsf{Prob}^*_x(\varphi(x))(\varphi(x) - \mathsf{Exp}^*\left[\varphi(x)\right])^2 \\ \overset{\text{(ii)}}{\geq}& \sum_{x \in S} \mathsf{Prob}^*(\varphi(x)) \cdot r^2 \\ =& r^2 \cdot \mathsf{Prob}^*(\varphi(x) - \mathsf{Exp}^*\left[\varphi(x)\right] \geq r). \end{aligned}$$

By the symbol $\sum$ we mean the sum of values. The inequality (i) follows from $(\varphi(x) - \mathsf{Exp}^*[\varphi(x)])^2 \geq 0$. The inequality (ii) holds because for all $x \in S$ we have $\varphi(x) - \mathsf{Exp}^*[\varphi(x)] \geq r$. □

Now we will show the most important lemma which gives us an estimation on the probability of an error during our algorithm. Recall that we can view non-empty subsets of $\{1, 2, \ldots, k\}$ as numbers from the range $2^k, 2^k + 1, \ldots, 2^{k+1} - 2$ (or equally strings of length k except $\mathsf{00}\ldots\mathsf{0}$). Therefore instead of quantifiying over subsets $J \subseteq \{1, 2, \ldots, k\}$, we can quantify over integers $j = 2^k, 2^k + 1, \ldots, 2^{k+1} - 2$.

Lemma 3.5. ([G01b]). *For $i = 1, \ldots, n$*

$$\mathit{Prob}^*[\mathit{maj}_{j=2^k,\ldots,2^{k+1}-2}\{u \odot b(j,j) \oplus h(r(j,j) \oplus e_i)\} = u_i] \geq 1 - \frac{1}{4n},$$

where $\mathit{maj}_{j=2^k,\ldots,2^{k+1}-2}\{u \odot b(j,j) \oplus h(r(j,j) \oplus e_i)\}$ stands for the maj function applied to a string composed of the values $u \odot b(j,j) \oplus h(r(j,j) \oplus e_i)$ for all $j = 2^k, \ldots, 2^{k+1} - 2$.

Proof: For each j we define variable ρ_j as

$$\rho_j = \begin{cases} 1 & \text{if } (u \odot b(j,j)) \oplus h(r(j,j) \oplus e_i) = u_i \\ 0 & \text{otherwise} \end{cases}$$

We need to show that for all j, ρ_j's are pairwise independent. This immediately follows from the fact that $r(j,j)$'s for all j are pairwise independent. Let show this fact. Let take some $j \leq k$ and values j_0, k_0 such that $j_0 \leq |j|$ and $j_{|j|-j_0+1} = 1$, resp. $k_0 \leq |k|$ and $k_{|k|-k_0+1} = 1$. Let suppose, without loss of generality, that $j_0 \neq k_0$. Then for every strings a, b of length n we have

$$\begin{aligned} \mathsf{Prob}^*[r(k,k) = b \mid r(j,j) = a] =& \mathsf{Prob}^*[s_{k_0} = b \mid s_{j_0} = a] \\ =& \frac{\mathsf{Prob}^*[s_{k_0} = b] \cdot \mathsf{Prob}^*[s_{j_0} = a]}{\mathsf{Prob}^*[s_{j_0} = a]} \\ =& \mathsf{Prob}^*[s_{k_0} = b] = \mathsf{Prob}^*[r(k,k) = b] \end{aligned}$$

We then have $\mathsf{Prob}^*_j[\rho_j = 1] = \mathsf{Prob}^*_r[h(r) = u \odot r] = \frac{1}{2} + \varepsilon$. It follows that $\mathsf{Exp}^*[\rho_j] = \mathsf{Prob}^*_{|x|=1}[\rho_j = 1] \cdot 1 + \mathsf{Prob}^*_{|x|=1}[\rho_j = 0] \cdot 0 = \frac{1}{2} + \varepsilon$ and $\mathsf{Var}^*[\rho_j] = (\frac{1}{2} + \varepsilon) - \left(\frac{1}{2} + \varepsilon\right)^2$. Denote $m = 2^k - 1$. Let also have counting term C defined as $\#(|j| = \boldsymbol{n})(\rho_j = 1)$. Using variable ρ_j we have

$$\begin{aligned} \mathsf{Prob}^*_j\left[C \leq \frac{1}{2}m\right] =& \mathsf{Prob}^*_j\left[C \dot{-} m \cdot \mathsf{Exp}^*[\rho_j] \leq \frac{1}{2}m \dot{-} m \cdot \mathsf{Exp}^*[\rho_j]\right] \\ \leq& \mathsf{Prob}^*_j\left[\left|C \dot{-} m(\frac{1}{2} + \varepsilon)\right| \geq \varepsilon m\right], \end{aligned}$$

where $|x - y|$ denotes $\max\{x \dot{-} y, y \dot{-} x\}$. Using Lemma 3.4 we get

$$\mathsf{Prob}^*_j\left[\left|C \dot{-} m(\frac{1}{2} + \varepsilon)\right| \geq \varepsilon m\right] < \frac{\mathsf{Var}^*[C]}{\varepsilon^2 m^2}$$

Now we use the property, that ρ_j for all j are pairwise independent. Therefore

$$
\begin{aligned}
\mathsf{Var}^*[C] =& \mathsf{Exp}^* \left[\left(\sum_{j=2^k}^{2^{k+1}-2} \rho_j \right)^2 \right] - \left(\mathsf{Exp}^* \left[\sum_{j=2^k}^{2^{k+1}-2} \rho_j \right] \right)^2 \\
=& \sum_{j_1,j_2} \mathsf{Exp}^* \left[\rho_{j_1}\rho_{j_2}\right] - \sum_{j_1,j_2} \mathsf{Exp}^* \left[\rho_{j_1}\right] \mathsf{Exp}^* \left[\rho_{j_2}\right] \\
=& \sum_{j=2^k}^{2^{k+1}-2} \mathsf{Exp}^*[\rho_j^2] + \sum_{j_1 \neq j_2} \mathsf{Exp}^*[\rho_{j_1}\rho_{j_2}] - \sum_{j=2^k}^{2^{k+1}-2} \mathsf{Exp}^*[\rho_j^2]^2 - \sum_{j_1 \neq j_2} \mathsf{Exp}^*[\rho_{j_1}]\mathsf{Exp}^*[\rho_{j_2}] \\
\overset{\text{(i)}}{=}& \sum_{j=2^k}^{2^{k+1}-2} \mathsf{Var}^*[\rho_j],
\end{aligned}
$$

where function $\sum$ represents the sum of values which can be easily defined by LRN. The equality (i) follows from pairwise independance of ρ_j for all j.

Hence we have

$$
\frac{\mathsf{Var}^*\left[C\right]}{\varepsilon^2 m^2} = \frac{\sum_{j=2^k}^{2^{k+1}-2} \mathsf{Var}^*[\rho_j]}{\varepsilon^2 m^2} \leq \frac{m}{\varepsilon^2 m^2} = \frac{1}{c\boldsymbol{n}}.
$$

So we can set $c := 4$ and the lemma follows.

□

We showed that if $b_i = s_i \odot u$ for all i, function GL_h returns u with probability at least $1 - \frac{1}{4\boldsymbol{n}} \geq \frac{3}{4}$. By Lemma 3.1 we know that $b_i = s_i \odot u$ for all i holds with probability $\frac{1}{t \otimes 1} = \frac{\varepsilon^2}{c\boldsymbol{n}} = \frac{\varepsilon^2}{4\boldsymbol{n}}$. So the overall probability of retrieving u is at least $\frac{3}{4} \cdot \frac{\varepsilon^2}{4\boldsymbol{n}}$, which is what we want.

3.8 Final step

In previous paragraphs we showed how to recover the value u if we have some function h with $\Pr_r[h(r) = u \odot r] \geq \frac{1}{2} + \varepsilon$ for some u. Now our last goal is to show how to use such algorithm GL_h to break a one-way permutation. I.e. if we have some witness who can predict the inner product there exists another witness which testifies that f is not one-way. Formally we want to show

$$
\varphi_2(f, f', f''), \neg\psi(f', f'', h, \vec{z}, s) \rightarrow \exists GL_h \exists w \neg\varphi_1(f, GL_h, w).
$$

First let us formalize the way how to obtain the preimage of $f(u)$ for some u. We will use the function GL_h presented in previous paragraphs. We assume to have some function, which testifies $\neg\psi$, it means it predicts inner product. Denote this function h_u. So we have $\mathsf{Prob}^*_r[h_u(f(u), r) = u \odot r] \geq \frac{1}{2} + \varepsilon$, where ε is non-negligible. For inverting function f, we take input $r = f(u)$ and run function GL_h with $h(r) := h_u(f(u), r)$ as an oracle function used in the algorithm. Formally we define formula $\mathsf{INV}(GL_h, h_u, f)$ as

$$\forall r(\quad h(r) = h_u(f(u), r) \wedge$$
$$GL_h(i+1) = GL_h(i) \circ \mathsf{maj}(v(i, k, 2^{k+1} - 2)) \wedge$$
$$GL_h(1) = \mathsf{maj}(v(1, k, 2^{k+1} - 2) \wedge$$
$$|GL_h(i)| \leq \boldsymbol{n} \wedge$$
$$u = GL_h(\boldsymbol{n}))$$

In other words having some input $f(u)$, we get preimage u as the output from GL_h which uses function h_u that computes inner product with high probability.

For deriving the probability of success of inverting f we can observe following lemma.

Lemma 3.6. ([G01b]).

$$\mathit{Prob}^*_{|u|=\boldsymbol{n}}\left[\mathit{Prob}^*_{|r|=\boldsymbol{n}}[h_u(f(u), r) = u \odot r] \geq \frac{1}{2} + \varepsilon\right] \geq \frac{\varepsilon}{2}.$$

So we have probability of inverting f at least $\frac{\varepsilon}{2} \cdot \frac{3\varepsilon^2}{16\boldsymbol{n}} = \frac{3\varepsilon^3}{32\boldsymbol{n}}$, which is what we wanted.

Let us derive the probability of success of inverting f formally. Using formula INV, we want, in fact, to prove following:

$$\mathsf{INV}(GL_h, h, f), \varphi_2(f, f', f''), \neg\psi(f', f'', h, \vec{z}, s) \to \exists w \neg\varphi_1(f, GL_h, w).$$

We assume $\neg\psi$, it means that there exists h_u such that $\mathsf{Prob}^*_{|r|=\boldsymbol{n}}[h_u(f(u), r) = u \odot r] \geq \frac{1}{2} + \varepsilon$, where ε is non-negligible. We want to show that there exists effective adversary GL_h and there exists w such that $\neg\varphi_1(f, GL_h, w)$. Using our definitions we want to show $\mathsf{Prob}^*_{|u|=\boldsymbol{n}}[GL_{h_u}(f(u)) = u] > \frac{1}{|w|}$. By chosing $f(u)$ and by application of Lemma 3.6, Lemma 3.1 and Lemma 3.5 we have

$$\begin{aligned}\mathsf{Prob}^*_{|u|=\boldsymbol{n}}[GL_{h_u}(f(u)) = u] =& \mathsf{Prob}^*_u\left[\mathsf{Prob}^*_r[h_u(f(u), r) = u \odot r] \geq \frac{1}{2} + \varepsilon\right] \\ & \cdot \mathsf{Prob}^*_j[b(j, j) = u \oplus r(j, j)] \\ & \cdot \mathsf{Prob}^*_u[\mathsf{maj}_j\{(u \odot r(j, j)) \oplus h_u(f(u), r(j, j) \oplus e_i)\}] \\ > & 2^{-k} \cdot \frac{\varepsilon}{2} \cdot \left(1 - \frac{1}{4\boldsymbol{n}}\right).\end{aligned}$$

In other words

$$\mathsf{Prob}^*_u[GL_{h_u}(f(u)) = u] > \frac{\varepsilon^3}{8\boldsymbol{n}}\left(1 - \frac{1}{4\boldsymbol{n}}\right).$$

Therefore we can take w such that $|w| > \frac{8\boldsymbol{n}}{\varepsilon^3\left(1 - \frac{1}{4\boldsymbol{n}}\right)} = \frac{8\boldsymbol{n}|z|^3}{1 - \frac{1}{4\boldsymbol{n}}}$ and the statement follows and so does (by the Soudness theorem 1.6) the whole Goldreich-Levin theorem.

Conclusion

We presented a way how we can formaly verify in principle the correctness of cryptographic protocols. It is based on the logical theory T. We have objects (function, strings and integers with a special kind of notation), operations to work with objects, axioms, terms and formulas. We also have a special type of terms, called *the counting terms*, that denote the number of elements satisfying some formula. We have also a special integer called *the security parameter*. We denote it by $\boldsymbol{n}$. It has a property that asymptotical properties need to be checked just for $\boldsymbol{n}$.

All these objects and rules allow us to describe cryptogryphic properties. We showed how to formalize one-way functions, negligible functions, hard-core predicates. The theory also brings a formal way how to prove these properties - *the Soundness theorem*. It tells us that having some property of f described by some formula φ and having a transformation between function f and some other function f', the soundness of f' (described by a formula ψ) holds.

We used the Soundness Theorem for a proof of the Golreich-Levin theorem. This theorem tells that there exists a hard-core predicate for some special kind of one-way permutation. It also gives us an exact manual how to find this hard-core bit. We presented the formalization of this theorem in our formal theory as well as the formalization of required cryptographic primitives into our theory. We then proved the theorem using objects, rules and axioms from the theory.

This showed us that the theory is strong enough to be used for verifying cryptographic protocols. We saw that standard cryptographic terms can be transformed into the theory. We also saw that with counting terms we can use expressions of the probability theory and we can view adversaries on a cryptographic construction as functions. Therefore we can formulate the security properties.

The purpose of this work was to show that we can use some formal logic theory for protocol verifying. I think that such a theory has quite a big potential in the future. On the presented proof of the Golreich Levin theorem we saw that it would be possible to easily implement used terms and functions into some program. This could lead to automated proof that some constructed protocol is correct and secure which would be quite a big step for the field of cryptography.

Bibliography

[A01] Arora, Sanjeev. One-way functions and Goldreich-Levin Theorem. In *Lecture notes on Princeton University*, 2001.

[G01a] Goldreich, Oded. The Foundations of Cryptography - Volume 1 (Basic Techniques) Cambridge University Press, UK, 2001.

[G01b] Goldreich, Oded. Lecture Notes on Pseudorandomness - Part I Weizmann Institute of Science , Israel, 2001.

[GL89] Goldreich, Oded - Levin, Leonid. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*. New York: ACM, 1989, p. 25-32.

[IK03] Impagliazzo, Russell - Kapron, Bruce M. Logics for reasoning about cryptographic constructions In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. Washington: IEEE Computer Society Press, 2003.

[J05] Jeřábek, Emil. Theories with explicit counting. In *Weak pigeonhole principle, and randomized computation*. Ph.D. thesis on Faculty of Mathematics and Physics at Charles University in Prague, 2005, p. 81-99.

[P06] Pass, Rafael. The Goldreich-Levin Theorem. In *Lecture notes on Cornell University, Introduction to Cryptography*, 2006.

[P09] Pass, Rafael. Hard-Core Predicates. In *Lecture notes on Cornell University, Theory of Computing*, 2009.

Appendix

Basic Axioms

In the first chapter we presented the formal theory for verifying protocols. We stated objects of the theory, operations, terms, formulas and axioms. We described some interesting basic axioms and we promised to present all Basic axioms in appendix. The following list of axioms is based on [IK03].

B1: $x \leq y \rightarrow x \leq y + 1$

B2: $x \neq x + 1$

B3: $0 \leq x$

B4: $(x \leq y \wedge x \neq y) \rightarrow x + 1 \leq y$

B5: $x \neq 0 \rightarrow 2x \neq 0$

B6: $x \leq y \vee y \leq x$

B7: $(x \leq y \wedge y \leq x) \rightarrow x = y$

B8: $(x \leq y \wedge y \leq z) \rightarrow x \leq z$

B9: $|0| = 0$

B10: $|2x + 1| = |x| + 1 \wedge |2x + 2| = |x| + 1$

B11: $x \leq y|x| \leq |y|$

B12: $(x \neq 0 \wedge y \neq 0) \rightarrow |x \otimes y| = |x| \cdot |y|$

B13: $0 \otimes x = 1$

B14: $(1 \otimes 2x + 1) = 2 \cdot (1 \otimes x) \wedge (1 \otimes 2x + 2) = 2 \cdot (1 \otimes x)$

B15: $x \otimes y = y \otimes x$

B16: $|x| = |y| \rightarrow x \otimes z = y \otimes z$

B17: $|x| = |y| + |z| \rightarrow x \otimes d = (y \otimes d) \cdot (z \otimes d)$

B18: $x \leq x + y$

B19: $(x \wedge x \neq y) \rightarrow (2x + 2 \leq 2y + 1 \wedge 2x + 2 \neq 2y + 1)$

B20: $x + y = y + x$

B21: $x + 0 = x$

B22: $x + (y + 1) = (x + y) + 1$

B23: $(x + y) + z = x + (y + z)$

B24: $x + y \leq x + z \rightarrow y \leq z$

B25: $x \cdot 0 = 0$

B26: $x \cdot (y+1) = x \cdot y + x$

B27: $x \cdot y = y \cdot x$

B28: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

B29: $x \cdot (y+z) = (x \cdot y) + (x \cdot z)$

B30: $1 \leq x \rightarrow ((x \cdot y \leq x \cdot z) \Leftrightarrow (y \leq z))$

B31: $\lfloor \frac{0 \dot{-} 1}{2} \rfloor 0$

B32: $x \neq 0 \rightarrow |x| = \left| \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor \right| + 1$

B33: $y \neq 0 \rightarrow (x = \lfloor \frac{y \dot{-} 1}{2} \Leftrightarrow (2x+1) = y \vee 2x + 2 = y)$

B34: $y \leq z \rightarrow y \dot{-} z = 0$

B35: $z \leq y \rightarrow (x = y \dot{-} z \Leftrightarrow y = x + z)$

B36: $\boldsymbol{n} = |s|$.