ISSN 0032-9460, Problems of Information Transmission, 2009, Vol. 45, No. 2, pp. 168–183. © Pleiades Publishing, Inc., 2009.
Original Russian Text © A.A. Kozhevnikov, S.I. Nikolenko, 2009, published in Problemy Peredachi Informatsii, 2009, Vol. 45, No. 2, pp. 101–118.

LARGE SYSTEMS

On Complete One-Way Functions[1]

A. A. Kozhevnikov and S. I. Nikolenko

Steklov Mathematical Institute, St. Petersburg
arist@pdmi.ras.ru sergey@logic.pdmi.ras.ru

Received May 19, 2008; in final form, January 20, 2009

Abstract—Complete constructions play an important role in theoretical computer science. However, in cryptography complete constructions have so far been either absent or purely theoretical. In 2003, L.A. Levin presented the idea of a combinatorial complete one-way function. In this paper, we present two new one-way functions based on semi-Thue string rewriting systems and a version of the Post correspondence problem. We also present the properties of a combinatorial problem that allow a complete one-way function to be based on this problem. The paper also gives an alternative proof of Levin's result.

DOI: 10.1134/S0032946009020082

1. INTRODUCTION

Problems arising in theoretical computer science are often presented as problems of recognizing certain *languages*, sets of strings over a certain alphabet (usually in the binary alphabet $\{0,1\}$). These languages, in turn, may be divided into *complexity classes*. For example, the complexity class P consists of all languages that are accepted by a deterministic polynomial Turing machine, while the complexity class NP consists of all languages accepted by a nondeterministic polynomial Turing machine.

Complexity theory started with studying the complexity of individual problems. However, it became evident that techniques that allow to study complexity classes as a whole were much more useful than results concerning individual problems. One of the first major breakthroughs in complexity theory was the *Cook–Levin theorem* [1, 2], which allowed to develop the theory of NP-complete problems [3, 4]. Complete problems for other complexity classes followed. For example, complete problems for average-case complexity are presented in [5–8]; see also general references on complexity theory [9–11].

Theoretical computer science holds in high esteem the possibilities that arise when a complete problem is found for a particular complexity class. A complete problem allows to shift the analysis from the whole class (where, in most cases, nothing can really be proved) to this certain, well-specified complete problem. For example, SAT, the satisfiability problem for Boolean formulas, is NP-complete, and even exponential algorithms for SAT are of great interest [12, 13]. Similarly, the DistNP complexity class (DistNP consists of NP problems with polynomial-time computable distributions), which is more relevant for this work, has complete problems, in particular, Post correspondence and matrix transformation problems [6, 8, 14–16]. Though complete problems do not exist for some classes [17], they remain one of the most useful objects in complexity theory.

However, not all complete problems are actually useful for theoretical and practical applications. While such problems as satisfiability or graph coloring allow for combinatorial approaches, there are problems that are undoubtedly complete for their complexity classes but do not actually cause such

[1] Supported in part by the Russian Foundation for Basic Research, project nos. 05-01-00932, 06-01-00502, 08-01-00640-a, 08-01-00649, 09-01-00784-a, and INTAS, YSF fellowship 05-109-5565.

a nice concept shift because they are hardly easier to analyze than the class itself. Such problems usually come from diagonalization procedures and require enumeration of all Turing machines or all problems of a certain complexity class.

Our results lie in the field of cryptography. For a long time, little has been known about complete problems in cryptography. While "conventional" complexity classes got their complete representatives relatively soon, it took thirty years since the definition of a public-key cryptosystem [18] to present a complete problem for the class of all public-key cryptosystems (with bounded decoding error) [19, 20]. Moreover, this complete problem is of the "bad" kind, since it requires enumerating all Turing machines and can hardly be put to any use, be it practical implementation or theoretical complexity analysis.

Before tackling public-key cryptosystems, it is natural to ask the same question about a seemingly simpler object: one-way functions (public-key cryptography is equivalent to the existence of a trapdoor function, a particular case of a one-way function). The first big step towards "useful" complete one-way functions was taken by L.A. Levin, who provided a construction of the first known complete one-way function [21] (see also [22–24]). Completeness is understood here in the following sense.

Definition 1. Assume that a function $f\colon \{0,1\}^* \to \{0,1\}^*$ has the following property: if there exists a strongly (weakly) one-way function $g\colon \{0,1\}^* \to \{0,1\}^*$, then f is also strongly (weakly) one-way. Then f is called a *complete* strongly (weakly) one-way function.

In complexity theory, completeness usually implies that a problem is complete with respect to certain *reductions*. However, in the theory of one-way functions, reductions are not yet known; in [24], Levin stated this problem as one of the most important problems of the field. Thus, in the present work we use the "classical" Levin's definition of a complete one-way function.

The first known construction of a (weakly) one-way function, developed by Levin, is called a *universal* one-way function. It uses a universal Turing machine U to compute the following function:

$$f_{\text{uni}}(\text{desc}(M), x) = (\text{desc}(M), M(x)),$$

where $\text{desc}(M)$ is the description of a Turing machine M, and $M(x)$ is the output of M on input x after $\le |x|^2$ steps. If there are one-way functions among Turing machines M, then f_{uni} is itself a weakly one-way function. Since it is easy to show that if there are one-way functions, then there are one-way functions working for quadratic time [22], f_{uni} is a complete weakly one-way function.

Note that this complete one-way function is of the "useless" kind of complete problems. It can hardly be put to practice. Naturally, Levin asked whether it is possible to find "combinatorial" complete one-way functions, functions that would not depend on enumerating Turing machines or giving their descriptions as input. For 15 years, the problem remained open and then was resolved by Levin himself [24]. Levin devised a smart trick of completely forbidding indeterministic choice in the computation of the function itself, allowing it only for the inverse function (we describe this construction in detail below).

Also, Levin in [24] formulated the problem of finding other combinatorial complete one-way function. In this paper we show how, using ideas similar to [24], one can get a complete one-way function from string rewriting systems. In [25], the bounded accessibility problem for string rewriting systems was shown to be average-case complete. A slightly more complex construction allows a complete one-way function to be based on the Post correspondence problem. Besides, we discuss the general properties that a combinatorial problem should possess in order to contain a complete one-way function by similar arguments. This paper describes and extends the results of [23].

2. DISTRIBUTIONAL ACCESSIBILITY PROBLEM FOR SEMI-THUE SYSTEMS

Consider a finite alphabet $\mathcal{A}$. An ordered pair of strings $\langle g, h\rangle$ over $\mathcal{A}$ is called a *rewriting rule* (sometimes also called a *production*). We write these rules as $g \to h$ and interpret them as rules for making substitutions in other strings. Formally, for two strings $u, v \in \mathcal{A}^*$ we write $u \Rightarrow_{g\to h} v$ if $u = agb$ and $v = ahb$ for some $a, b \in \mathcal{A}^*$. A set of rewriting rules is called a *semi-Thue system*. For a semi-Thue system R, we write $u \Rightarrow_R v$ if $u \Rightarrow_{g\to h} v$ for some rewriting rule $\langle g, h\rangle \in R$. We write $u \Rightarrow_R^* v$ if there exists a finite sequence of rewriting rules $\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle \in R$ such that

$$u = u_0 \Rightarrow_{g_1\to h_1} u_1 \Rightarrow_{g_2\to h_2} u_2 \Rightarrow \ldots \Rightarrow_{g_m\to h_m} u_m = v$$

(i.e., instead of the original definition of $\Rightarrow_R$ we take its transitive reflexive closure). Also, we define a restricted version of $\Rightarrow_R^*$: we write $u \Rightarrow_R^n v$ if there exists a finite sequence of rewriting rules $\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle \in R$ such that

$$u = u_0 \Rightarrow_{g_1\to h_1} u_1 \Rightarrow_{g_2\to h_2} u_2 \Rightarrow \ldots \Rightarrow_{g_m\to h_m} u_m = v, \quad \text{where} \quad m \le n;$$

i.e., one can get v from u by the rules of R in no more than n steps.

We can now define the distributional accessibility problem for semi-Thue systems.

Instance. A semi-Thue system $R = \{\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle\}$, two binary strings u and v, a positive integer n in unary.[2] The input size,[3] therefore, is $n + |u| + |v| + \sum_1^m (|g_i| + |h_i|)$.

Question. Is $u \Rightarrow_R^n v$?

Distribution. Randomly and independently choose positive integers n and m and binary strings u and v. Then randomly and independently choose binary strings $g_1, h_1, \ldots, g_m, h_m$. Integers and strings are chosen with the default uniform probability distribution, namely, the distribution proportional to $\dfrac{1}{n^2}$ for integers, and to $\dfrac{2^{-|u|}}{|u|^2}$, for binary strings.

We do not describe in detail the properties of semi-Thue systems; they are discussed, for example, in [26–28]. We only note a relevant property: in [29] it was shown that the distributional accessibility problem for semi-Thue systems is complete for the DistNP complexity class. The proof relies on the fact that semi-Thue systems can model Turing machines. This fact attracted attention to semi-Thue systems in the first place: their first application was the proof of undecidability of the accessibility problem. The proof went by modeling the Turing machine halting problem in semi-Thue systems [27, 30].

However, in what follows we also need another notion of accessibility in semi-Thue systems. Namely, for a semi-Thue system R we write $u \overset{!}{\Rightarrow}_R v$ if $u = agb$ and $v = ahb$ for some $\langle g, h\rangle \in R$ and strings $a, b \in \mathcal{A}^*$ and, moreover, there does not exist another rewriting rule $\langle g', h'\rangle \in R$ such that $u = a'g'b'$ and $v = a'h'b'$ for some $a', b' \in \mathcal{A}^*$. Similarly to $\Rightarrow_R$, we extend $\overset{!}{\Rightarrow}_R$ to its transitive reflexive closure $\overset{!}{\Rightarrow}_R^*$ and introduce a restricted version: $u \overset{!}{\Rightarrow}_R^n v$ if $u \overset{!}{\Rightarrow}_R^* v$ and the corresponding rewriting sequence consists of no more than n steps.

In other words, $u \overset{!}{\Rightarrow}_R^* v$ if $u \Rightarrow_R^* v$ and on each step of this derivation there was only one applicable rewriting rule. This uniqueness (or, better to say, determinism) is crucial to perform Levin's trick in Section 6.

[2] We use unary notation for n, so that algorithms can be considered polynomial in input length, not in some abstract "security parameter" that may turn out to be exponentially larger than the input length.

[3] Hereafter, we code g_i, h_i, u, and v with prefix codes that allow unambiguous decoding.

3. POST CORRESPONDENCE PROBLEM

The following problem was proved to be complete for DistNP in [6] (see also Remark 2 in [14]).

Instance. A positive integer m, pairs

$$\Gamma = \{\langle u_1, v_1\rangle, \ldots, \langle u_m, v_m\rangle\},$$

a binary string x, a positive integer n. The size of the instance is $n + |x| + \sum_1^m (|u_i| + |v_i|)$.

Question. Is $u_{i_1} \ldots u_{i_k} = x v_{i_1} \ldots v_{i_k}$ for some $k \le n$?

Distribution. Randomly and independently choose positive integers n and m and binary strings u and v. Then randomly and independently choose binary strings $g_1, h_1, \ldots, g_m, h_m$. Integers and strings are chosen with the default uniform probability distribution, namely, the distribution proportional to $\frac{1}{n^2}$ for integers, and to $\frac{2^{-|u|}}{|u|^2}$, for binary strings.

In order to build a complete one-way function, we need a modification of this problem. Namely, we pose the question as follows: does

$$u_{i_1} \ldots u_{i_k} y = x v_{i_1} \ldots v_{i_k}$$

hold for some y?

Properties of the Post correspondence problem are similar to properties of semi-Thue systems. The Post correspondence problem was also formulated as a combinatorial version of an undecidable problem and attracted attention as one of the simplest and, therefore, most useful undecidable problems [31–34]. Our version of the Post correspondence is also undecidable if we remove the restriction on n; the bounded version is also complete for DistNP, as is shown in [6].

Let us elaborate on the analogy with semi-Thue systems. The function based on modified Post correspondence can be naturally viewed as a derivation with certain inference rules. Namely, for some nonempty set

$$\Gamma = (\langle u_1, v_1\rangle, \ldots, \langle u_m, v_m\rangle),$$

we say that a string x *yields* a string y *in one step* and write $x \vdash_\Gamma y$ if there is a pair $\langle u, v\rangle \in \Gamma$ such that $uy = xv$. The "*yields*" relation $\vdash_\Gamma^*$ is defined as the transitive closure of the "yields-in-one-step" relation: a binary string x *yields* a binary string y if there exists a sequence of pairs $\langle u_1, v_1\rangle, \ldots, \langle u_m, v_m\rangle \in \Gamma$ and a sequence of strings $x = x_1, x_2 \ldots, x_m, x_{m+1} = y$ such that $u_i x_{i+1} = x_i v_i$ for each $1 \le i \le m$.

To perform Levin's trick, we need to get rid of the indeterminism. This time, the description of a deterministic version $\vdash^!$ of the relation $\vdash$ is more complicated than in the case of semi-Thue systems. If we simply required it to be deterministic, we would not be able to move the head of the Turing machine to the left (see Section 7). To solve this problem, we have to look one step ahead: if there are two applicable rules and one of the two branches leads to a dead end on the very next step, we consider the choice deterministic.

Formally speaking, we say that x yields y in one step ($x \vdash^!_\Gamma y$) if there are no more than two pairs $\langle p, s\rangle, \langle p', s'\rangle \in \Gamma$ such that $py = xs$ and $p'y' = xs'$ for some strings y and y' (where $y \ne y'$, but p may equal p': two possible different applications of the same rule are still nondeterministic) and, moreover, we cannot apply any rule in Γ to y'. As above, we denote by $x \vdash^{!*}_\Gamma y$ the transitive reflexive closure of this relation and write $u \vdash^{!n}_\Gamma v$ if $u \vdash^!_\Gamma v$ in no more than n steps.

4. TURING MACHINES AND ONE-WAY FUNCTIONS

Proofs of completeness for one-way functions rely upon modeling Turing machines with these functions. Therefore, we need to fix basic definitions. In defining a Turing machine we follow the classical textbook [35].

Definition 2. A (one-tape) *Turing machine* is an ordered 7-tuple

$$M = \langle Q, \Gamma, B, \Sigma, \pi, s, H \rangle, \quad \text{where:}$$

- Q is a finite set of *states*;
- Γ is a finite set of *tape symbols*;
- $B \in \Gamma$ is the blank symbol (the only symbol allowed to occur on the tape infinitely often at any step during the computation);
- $\Sigma \subseteq \Gamma \setminus \{B\}$ is the set (subalphabet) of input symbols;
- $\pi\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *transition function*: L denotes a left shift of the machine's head, R denotes a right shift (there exist modifications that allow the machine's head to remain at the same place);
- $s \in Q$ is the *initial state*;
- $H \subseteq Q$ is the set of *final* (or *accepting*) *states*.

We are only interested in Turing machines with a single final state $H = \{h\}$. Informally speaking, a Turing machine consists of an infinite tape divided into a countable number of cells and a head, which is located above one of these cells at any step of the computation. In the initial state s, the tape contains the input to the Turing machine, which is a string over the alphabet Σ. The machine makes transitions from state to state according to the function π, which takes as inputs the current state $q \in Q$ and the symbol $\alpha \in \Gamma$ that is currently under the machine's head. At each step, the head moves to the right or to the left according to the output of π.

We say that a Turing maching M *computes* a function $f\colon \Sigma^* \to (\Gamma^* \setminus \{B\})^*$ if, after starting M with input $x \in \Sigma^*$, it finishes the computation (i.e., reaches the final state h), leaving the string $f(x)$ written on the tape. For simplicity, in what follows we do not distinguish the alphabets Σ and $\Gamma \setminus \{B\}$. By $t_M(x)$, where $t_M\colon \Sigma^* \to \mathbb{N}$, we denote the number of steps that a Turing machine M requires to reach the final state h on input x. We say that M *works for time* $T\colon \mathbb{N} \to \mathbb{N}$ if

$$\forall n \in \mathbb{N} \qquad \max_{x\colon |x|=n} t_M(x) \le T(n),$$

where $|x|$ is the length of the string x.

We will also need nondeterministic and probabilistic Turing machines. We do not give standard definitions but rather consider a nondeterministic Turing machine as a regular deterministic machine with an additional tape that contains the "witness"; a nondeterministic Turing machine M *computes* a function $f\colon \Sigma^* \to \Sigma^*$ if for every input x there exists a "witness" $y \in \Sigma^*$ such that $M(x, y) = f(x)$, and for no witness the machine terminates with a wrong answer (however, M is allowed to terminate without giving any answer at all).

A probabilistic Turing machine is just a nondeterministic one, the only thing that changes is the interpretation. This time, the witness is interpreted as random symbols, and we say that a probabilistic Turing maching *computes* a function f on input x with probability p if

$$\Pr_{y \in U_m} [M(x, y) = f(x)] = p,$$

where m is the number of random symbols from Σ, U_m is the uniform distribution on strings of length m over alphabet Σ, and $y \in U_m$ means "y is taken over the distribution U_m". In what follows, we usually set $\Sigma = \{0, 1\}$.

To make the paper self-contained, we also give the definition of a one-way function [22].

Definition 3. A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is called *strongly one-way* if the following conditions hold:

1. There exists a deterministic Turing machine M with input alphabet $\{0,1\}$ that computes f in polynomial time;
2. For every probabilistic Turing machine M', every polynomial p, and every sufficiently large n,

$$\Pr_{M',x\in U_n}\left[M'(f(x),1^n)\in f^{-1}(f(x))\right] < \frac{1}{p(n)},$$

where the probability is taken over the random bits of M' and input x (both distributions are uniform).

In other words, a function is strongly one-way if no adversary can invert it on any significant fraction of inputs.

Definition 4. A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is called *weakly one-way* if the following conditions hold:

1. There exists a deterministic Turing machine M with input alphabet $\{0,1\}$ that computes f in polynomial time;
2. There exists a polynomial p such that for every probabilistic Turing machine M' and for every sufficiently large n,

$$\Pr_{M',x\in U_n}\left[M'(f(x),1^n)\notin f^{-1}(f(x))\right] > \frac{1}{p(n)},$$

where the probability is taken over the random bits of M' and input x (both distributions are uniform).

In other words, a function is weakly one-way if for every polynomial adversary there is a significant fraction of inputs where this adversary fails to invert this function.

In what follows, we need the following facts; see [22] for proofs.

Proposition 1. 1. *If there exists a weakly one-way function, then there exists a strongly one-way function (that is, the existence of weakly and strongly one-way functions is equivalent).*

2. *If there exists a weakly (strongly) one-way function, then there exists a weakly (strongly) one-way function computable by a Turing machine that works for no more than n^2 steps on inputs of length n.*

5. THE COMPLETE ONE-WAY TILING FUNCTION

Before presenting our own construction, we recall Levin's complete one-way function from [24]. In this section, we slightly modify Levin's construction and present an alternative Turing machine modeling technique based on ideas from [36]. The difference from the original Levin's construction is that he considered a tiling function for tiles with marked corners, namely, the corners of tiles, instead of edges, are marked with symbols. In Levin's tiling of an $n\times n$ square, symbols on touching corners of adjacent tiles should match.

In our construction, a *tile* is a square whose edges are marked with symbols from a finite alphabet $\mathcal{A}$. We assume infinite supply of tiles of each kind. A *tiling* of an $n\times n$ square is a set of n^2 tiles covering an $n\times n$ grid such that symbols on adjacent sides match.

It will be convenient for us to consider the tiling function as a string transformation system similar to a semi-Thue system (and based on the same semigroup ideas). Fix a finite set of tiles T. We say that T *transforms* a string x over alphabet $\mathcal{A}$ to y, where $|x|=|y|$, if there is a tiling of an $|x|\times|x|$ square such that symbols on the bottom of the lowest row of squares form the string x, and symbols on the top of the uppermost row form y. We write $x \longrightarrow_T y$ in this case.

By a *tiling process* we mean completion of a partially tiled square by one tile at the time. Similarly to semi-Thue systems, we define $x \overset{!}{\longrightarrow}_T y$ if and only if $x \longrightarrow_T y$ with an additional restriction: we permit the extension of a partially tiled square only if the possible extension is unique (there is precisely one suitable tile in T). Note that in this approach, for each string x, either there exists precisely one string y such that $x \overset{!}{\longrightarrow}_T y$ or there is no string y with this property.[4]

Let us fix a certain encoding of the set of labels with binary strings that encodes each label by a string of length no more than $\log|\mathcal{A}| + O(1)$. We do not distinguish a label and its encoding in this section; it is not important for tiling, but will play a significant role in Section 6.

Definition 5. The *tiling simulating function* (Tiling) is a function $f\colon \{0,1\}^* \to \{0,1\}^*$ defined as follows:

- If the input has the form (T, x) for a finite set of tiles T and a string x, and $x \overset{!}{\longrightarrow}_T y$, then $f(T, x) = (T, y)$;
- Otherwise, $f(T, x) = (T, x)$.

Theorem 1. *If one-way functions exist, then Tiling is a weakly one-way function.*

Proof. The proof closely follows the completeness proof for the universal one-way function given in [22] (and introduced in [21]). Let g be a length-preserving (i.e., $|g(x)| = |x|$ for every x) one-way function such that there exists a Turing machine that computes g and works for no more than n^2 steps on inputs of length n (by Proposition 1, if one-way functions exist, there exists a one-way function like this).

Consider a Turing machine f. We prove that every Turing machine can be modeled as a tiling problem.

Lemma 1. *For every deterministic Turing machine M with tape alphabet $\Gamma = \{0, 1, B\}$ working for no more than n^2 steps on inputs of length n, there exists a set of tiles T_M such that its set of labels $\mathcal{A}$ includes Γ, and $M(x) = y$, $|x| = |y|$, if and only if*

$$\$sxB^{n(n-1)}\# \overset{!}{\longrightarrow}_{T_M} \$hyB^{n(n-1)}\#,$$

where s is the initial state of M, h is its final state, B is its blank symbol, and $\$, \# \notin \Gamma$ are additional symbols marking the beginning and end of a string.

Proof. Let Q_M be the set of states of a Turing machine M, s be the initial state of M, h be the final state, π be the transition function of M, and $\{0, 1, B\}$ be the tape symbols. By \$ we denote the begin marker, and by #, the end marker. We also introduce a new symbol for each pair from $Q \times \{0, 1, B\}$. We are now ready to present a construction of a tileset T_M.

1. For each tape symbol $a \in \{0, 1, B\}$, we add

2. For each $a, b, c \in \{0, 1, B\}$, $q \in Q \setminus \{h\}$, $p \in Q$, if

$$\pi_M(q, a) = (p, b, R),$$

[4] There also exists a different tiling task where the question is posed as follows: can we tile the whole plane with a given set of tiles? This problem is also undecidable (for similar reasons); it was introduced in 1961 by Hao Wang and led to several interesting results [37–41]. However, we only consider a "bounded" version of tiling in this paper.

we add

3. For each $a, b, c \in \{0, 1, B\}$, $q \in Q \setminus \{h\}$, $p \in Q$, if

$$\pi_M(q, a) = (p, b, L),$$

we add

4. Finally, for \$ and # we add

It is now easy to see that each computation of M now corresponds to a proper tiling of the square. Thus, for each computation on the deterministic Turing machine M on input x that goes on for no more than n^2 steps, there exists a corresponding tiling of the $|x|^2 \times |x|^2$ square in which bottom labels of the lowest row form the input string (padded with symbols B to length n), and top labels of the uppermost row form the output string (padded with symbols B to length n). If the computation halts after less than n^2 steps, we can still finish the tiling with tiles from item 1. $\triangle$

Lemma 1 guarantees that there exists a finite set of tiles T_M such that

$$\$sxB^{n(n-1)}\# \longrightarrow^*_{T_M} \$hyB^{n(n-1)}\#$$

is equivalent to $g(x) = y$. Therefore, with constant probability (equal to the probability of the event that the input string begins with the description of T_M), inverting Tiling is equivalent to inverting g. $\triangle$

Thus, Tiling is a complete one-way function; it is a weakly one-way function if and only if there exists a weakly one-way function. By Proposition 1, one can build a strongly one-way function with the same property.

6. A COMPLETE ONE-WAY FUNCTION BASED ON SEMI-THUE SYSTEMS

The first new complete one-way function that we present in this paper is based upon the distributional accessibility problem for semi-Thue systems. First, we need to make this decision problem a function and then add Levin's trick, always keeping in mind length preservation.

Definition 6. The *semi-Thue accessibility function* (STAF) is a function $f\colon \mathcal{A}^* \to \mathcal{A}^*$ defined as follows:

- If the input has the form $(\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle, x)$, and in the semi-Thue system

$$\Gamma = (\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle)$$

we have $x \stackrel{!}{\Rightarrow}{}^{t}_{\Gamma} y$, $t = |x|^2 + 4|x| + 2$, there are no rewriting rules in Γ applicable to y, and $|y| = |x|$, then

$$f(\Gamma, x) = (\Gamma, y);$$

- Otherwise, $f(\Gamma, x) = (\Gamma, x)$.

Theorem 2. *If one-way functions exist, then STAF is a weakly one-way function.*

Proof. The proof of this theorem, as well as of Theorem 1, follows the ideas of [22].

First, note that STAF is easy to compute: one simply needs to use the first part of the input as a semi-Thue system (if this is impossible, return the input) and apply its rules until either there are two rules that apply, or we have worked for $|x|^2 + 4|x| + 2$ steps, or some other string y has been reached and no other rules can be applied. In the first two cases, return the input. In the third case, check that $|y| = |x|$ and return (Γ, y) if so, or the input, otherwise.

Then, note that with constant probability (for the uniform distribution, this probabilty is proportional to $\frac{1}{|R|^2 2^{|R|}}$), the description of any given semi-Thue system (which is simply a binary string) appears as the first part of STAF's input. Consider a length-preserving one-way function g and a Turing machine M_g that computes g and runs in quadratic time (they exist by the assumption and by Proposition 1).

To continue the proof, we need to encode Turing machines into a semi-Thue system. We need an encoding that is efficient enough (so that the distributions on inputs and encoded outputs would be polynomially equivalent). This can be done with a so-called *dynamic binary coding scheme* developed by Yu. Gurevich and used in [6, 29, 36].

Proposition 2. *For any finite alphabet $\mathcal{A}$ with $|\mathcal{A}| > 2$ and any pair of binary strings x and y beginning with ones, there exists a* dynamic binary coding scheme *of $\mathcal{A}$ over the alphabet $\{0, 1\}$ with the following properties.*

1. *All codes (binary codes of symbols of $\mathcal{A}$) have the same length $\ell = 2\log|x| + O(1)$;*
2. *Both strings x and y are distinguishable from every code; i.e., no code is a substring of x or y;*
3. *If a nonempty suffix z of a code u is a prefix of a code v, then $z = u = v$ (one can always distinguish where a code ends and another code begins);*
4. *Strings x and y can be written as a unique concatenation of binary strings* 1, 10, 000, *and* 100, *which are not prefixes of any code.*

Now we can prove that semi-Thue systems are capable of modeling deterministic Turing machines.

Lemma 2. *For every deterministic Turing machine M, there exists a semi-Thue system R_M such that $M(x) = y$ if and only if*

$$\underline{s}x\underline{\$} \overset{!}{\Rightarrow}{}^{t}_{R_M} \underline{hy}\underline{\$},$$

where $t = T + 2|x| + 2|y| + 2$, and T is the time that M works for on input x.

Proof. Let us define the semi-Thue system R_M that corresponds to a Turing machine M. The rewriting rules are divided into three parts:

$$R_M = R_1 \cup R_2 \cup R_3.$$

Let us denote $\mathcal{B} = \{1, 10, 100, 000\}$, fix a dynamic binary coding scheme for $\mathcal{A}$, x, and y, and denote by $\underline{w}$ the encoding of w in this scheme.

The set R_1 consists of the following rules for each $u \in \mathcal{B}$:

$$\begin{array}{rcl} \underline{s}u & \to & \underline{\$us_1}, \\ \underline{s_1}u & \to & \underline{us_1}, \\ \underline{us_1\$} & \to & \underline{s_2u\$}, \\ \underline{us_2} & \to & \underline{s_2u}, \\ \underline{\$s_2} & \to & \underline{\$s}, \end{array}$$

where s, s_1, s_2, and \$ are auxiliary symbols. These rules are necessary to rewrite the initial string $\underline{s}x\underline{\$}$ into $\underline{\$sx\$}$. Since x can be uniquely written as $u_1 \dots u_m$ for some $u_i \in \mathcal{B}$, this transformation can be carried out in $2m + 1 \le 2|x| + 1$ steps.

The set R_2 consists of rewriting rules corresponding to Turing machine instructions.

1. For each state $q \in Q \setminus \{h\}$, $p \in Q$, $a, b, c \in \{0, 1, B\}$,

$$\pi_M(q, a) = (p, b, R) \quad \Rightarrow \quad qac \to bpc, qa\$ \to bpB\$ \in R_2;$$

2. For each state $q \in Q \setminus \{h\}$, $p \in Q$, $a, b, d \in \{0, 1, B\}$, and $c \in \{0, 1, \$\}$,

$$\pi_M(q, a) = (p, b, L) \quad \Rightarrow \quad dqac \to pdbc, dqB\$ \to pdbB\$ \in R_2$$

for $a \neq B$, $c \neq \$$, or $b \neq B$.

Note that R_1 and R_2 are completely similar to the construction presented in [36], where it is used to prove average-case completeness for the word equivalence problem in a semi-Thue system. By properties of a dynamic binary coding scheme, the strings used in the system can be decoded unambiguously, and the rewriting rules correspond to Turing machine instructions. The third part of this construction is supposed to reduce the result from $\underline{\$sy\$}$, where y is the result of the Turing machine computation, to the protocol of the Turing machine that is necessary to prove that nondeterministic semi-Thue systems are DistNP-hard.

This time we have to deviate from [36]: we need a different set of rules because we actually need the output of the machine, not the protocol. Thus, our version of R_3 looks as follows:

$$\begin{array}{rcl}
\underline{\$hu} & \to & \underline{\$u}\underline{s_3}, \\
\underline{s_3u} & \to & u\underline{s_3}, \\
\underline{s_3u\$} & \to & u\underline{s_4\$}, \\
\underline{us_4} & \to & \underline{s_4}u, \\
\underline{\$s_4} & \to & \underline{h},
\end{array}$$

where s_3 and s_4 are auxiliary symbols. This transformation can be done in $2|y| + 1$ steps.

These rules simply translate $\underline{y}$ back into the original y and add h in the beginning of the output, thus achieving the actual output configuration of the original Turing machine M. $\triangle$

By Lemma 2, there exists a semi-Thue system R_M such that $\underline{s}x\underline{\$} \Rightarrow^{*,t}_{R_M} \underline{hy\$}$ is equivalent to $g(x) = y$. Therefore, on a constant fraction of inputs, inverting STAF is equivalent to inverting g. $\triangle$

7. A COMPLETE ONE-WAY FUNCTION BASED ON POST CORRESPONDENCE

In this section, we describe a one-way function based on the Post correspondence problem and prove that it is complete. We recall that in Section 3 we have defined a deterministic version of Post correspondence with the following "backtracking": if there are two applicable rules, and one of them immediately leads to a dead end (there is no applicable rule at the next step), we choose the other and consider this choice deterministic.

The function is defined as follows.

Definition 7. The *Post transformation function* (PTF) is a function $f\colon \mathcal{A}^* \to \mathcal{A}^*$ defined as follows:

- If the input has the form $(\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle, x)$, and for the rewriting rules

$$\Gamma = (\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle)$$

we have $x \xrightarrow[\Gamma]{!}{}^{n^4} y$, there are no rules in Γ applicable to y, and $|y| = |x|$, then

$$f(\Gamma, x) = (\Gamma, y);$$

- Otherwise, $f(\Gamma, x) = (\Gamma, x)$.

Theorem 3. *If one-way functions exist, then PTF is a weakly one-way function.*

Proof. The proof of this theorem is similar to the proofs of Theorems 1 and 2. Assume that g is a length-preserving one-way function, and M is the Turing machine computing this function that works for no more than n^2 steps on inputs of length n.

We have to reduce Turing machine computations to Post correspondence; a version of this reduction is described in [6].

First we define the system of rewriting rules Γ_M corresponding to a given Turing machine M. As usual, let Q be the set of states of a Turing machine M; s be the initial state of M; h be the halting state; π_M be the transition function of M; and 0, 1, and B be the tape symbols. For all symbols we use the dynamic binary coding scheme described in Section 6. The derivation set Γ_M consists of the following inference rules.

1. For every tape symbol u,
$$\langle \underline{u}, \underline{u} \rangle;$$
2. For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a, b \in \{0, 1\}$, and rule $\pi_M(q, a) = (p, b, R)$,
$$\langle \underline{qa}, \underline{bp} \rangle;$$
3. For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a \in \{0, 1\}$, and rule $\pi_M(q, B) = (p, a, R)$,
$$\langle \underline{qB}, \underline{bpB} \rangle;$$
4. For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a, b, c \in \{0, 1\}$, and rule $\pi_M(q, a) = (p, b, L)$,
$$\langle \underline{cqa}, \underline{pcb} \rangle;$$
5. For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a \in \{0, 1\}$, and rule $\pi_M(q, B) = (p, a, L)$,
$$\langle \underline{cqB}, \underline{pcbB} \rangle.$$

Lemma 3. *For a deterministic Turing machine M with running time at most n^2 and its corresponding Post transformation system Γ_M,*

$$M(x) = y \quad \textit{if and only if} \quad \underline{sxB} \vdash_{\Gamma_M}^{*,n^4} \underline{hyB}.$$

Proof. The configuration of M after t steps of computation is represented by a string xqy, where q is the current state of M, x is the tape before the head, and y is the tape from the head to the first blank symbol. Simulation of one step of M's computation from a configuration xqy consists of no more than $|x|$ applications of rule 1, followed by one application of one of the rules 2–5, then followed by $|y| - 1$ applications of rule 1.

Note that before applying a rule that moves M's head to the left, one could also apply rule 1. If the Turing machine M is deterministic, then this "wrong" application leads to a situation where no rule from Γ_M is applicable. Thus, our version of deterministic Post correspondence can cope with this choice. $\triangle$

By Lemma 3, there exists a finite system of pairs Γ_M such that

$$\underline{sxB} \vdash_{R_M}^{*,n^4} \underline{hyB}$$

is equivalent to $g(x) = y$. Therefore, with constant probability, solving PTF is equivalent to inverting g. $\triangle$

Remark. Note a slight change in distributions on inputs and outputs: PTF accepts as input $\underline{x}$ and outputs $\underline{y}$, while the emulated machine g accepts x and outputs y. Such nice distinctions often hold the devil of average-case reasoning. Fortunately, distributions on x and $\underline{x}$ can be transformed from one to another by a polynomial algorithm, so PTF is still a weak one-way function (see [22,42] for details).

8. COMPLETE ONE-WAY FUNCTIONS AND DistNP-HARD COMBINATORIAL PROBLEMS

Both of our constructions of a complete one-way function look very similar to the construction on the Tiling complete one-way function. This naturally leads to the question: in what other combinatorial settings can one apply the same reasoning to find a complete one-way function?

All functions considered above were based on combinatorial problems that also can be modified to give DistNP-hard problems [6,8]. However, the construction of complete one-way functions is not quite straightforward: one needs to apply Levin's trick to prove completeness. The whole point of this proof is to keep the function both length-preserving and easily computable. Obvious functions fall into one of two classes.

1. *Easily computable but not length-preserving.* For any DistNP-hard problem, one can construct a hard-to-invert function f that transfers *protocols* of this problem into its results. This function is hard to invert; moreover, it is hard to invert on average. However, it does not preserve length (the protocol is usually much longer than the input), and thus it is impossible to translate a uniform distribution on the *outputs* of f into a reasonable distribution on its *inputs*. The reader is welcome to think of a reasonably uniform distribution on *proper* tilings (i.e., tilings with matching symbols on neighboring tile sides) that would result in a "reasonably uniform" distribution on their upper rows. We believe that constructing such a distribution is either impossible or requires a major new insight.
2. *Length-preserving but hard to compute.* Take a DistNP-hard problem and consider the function that sends its input into its output (e.g., the lowest row of the tiling into its uppermost row). This function is hard to invert and length-preserving, so there is no problem with distributions. But, unfortunately, it is also hard to compute, because one needs to solve Tiling to compute it; therefore, it is hardly a good candidate for a one-way function.

Following Levin, we get around these obstacles by having a *deterministic* version of a DistNP-hard problem. This time, a Tiling problem produces nontrivial results only if there is always *only one* proper tile to attach. Thus, we get a function described above as the second type, but this time it is easy to compute, too. Similarly, in Section 6 we demanded that only one rewriting rule was applicable on each step (we introduced $\Rightarrow^*$ for this very purpose). And in Section 7 we have slightly generalized this idea of "deterministic choice", allowing fixed-length deterministic backtrack.

We conclude that a combinatorial problem should have two properties in order to hold a complete one-way function.

1. It should have a deterministic restricted version that can be computed in polynomial time.
2. Its deterministic version should be expressive enough to simulate a deterministic Turing machine. For example, natural deterministic Post correspondence (without any backtrack) is, of course, easy to formulate, but does not seem to be powerful enough (fails to model moving the Turing machine's head to the left).

These ideas look so well-defined that one is tempted to formulate them as a mathematical statement. We give such a formalization in the next section.

9. GENERALIZING COMPLETE ONE-WAY FUNCTIONS

First of all, we need to define a combinatorial problem. Note that all our examples contained a finite set of rules that transformed one configuration of the problem into another; for example, under a rewriting rule, a semi-Thue system transformed one binary string into another. This intuition underlies the following definition.

Definition 8. A *combinatorial problem* S is a tuple

$$S = \langle X, f_1, \ldots, f_m \rangle,$$

where X is some fixed set (a set of configurations), and $f_i \colon X \to X$ are partial functions.

The definition should be understood as follows: if a configuration $x \in X$ lies in the domain of a function f_i, it means that the rule f_i can transform it into $f_i(x)$.

Example. For the Post correspondence problem defined by pairs of strings

$$(u_1, v_1), \ldots, (u_m, v_m),$$

$X = \{0,1\}^* \times \{0,1\}^*$, and for each $1 \le i \le m$, the function $f_i \colon \{0,1\}^* \to \{0,1\}^*$ transforms a pair of strings (a, b) into $(u_i a, b v_i)$.

Then, we need to define what it means for a combinatorial problem to "simulate a Turing machine." In our definitions, it is also easy to define. We denote by $t_M \colon \Gamma^* \times Q \times \mathbb{N} \to \Gamma^*$ (here t stands for "tape") the function that reflects how the Turing machine M transforms its tape. The string $t_M(\gamma, q, \mathrm{pos})$ coincides with γ everywhere except for the position with index pos; the symbol in this position is substituted by the second component of the transition function $\pi(q, \gamma_{\mathrm{pos}})$.

Definition 9. A family of combinatorial problems $\mathcal{S}$ *simulates deterministic Turing machines* if for every Turing machine

$$M = \langle Q, \Gamma, B, \Sigma, \pi, s, H \rangle$$

there exists a combinatorial problem $S = \langle X, f_1, \ldots, f_m \rangle \in \mathcal{S}$ and a mapping $\sigma \colon Q \times \Gamma^* \to X$ such that

$$\pi(q, \alpha) = (q', \alpha', \mathrm{move}), \quad \text{where} \quad \mathrm{move} \in \{L, R\},$$

if and only if for every string $\gamma \in \Gamma^*$ and every position $\mathrm{pos} \in \mathbb{N}$ such that $\gamma_{\mathrm{pos}} = \alpha$ there exists a unique i, $1 \le i \le m$, such that $\sigma(q, \gamma) \in \operatorname{dom} f_i$ and

$$f_i(\sigma(q, \gamma)) = \sigma(q', t_M(\gamma, q, \mathrm{pos})).$$

In other words, the configurations of a combinatorial problem should be able to encode (the definition denotes this encoding by σ) every state of a Turing machine plus the string that is currently on the tape. Moreover, images of these configurations under functions f_i should be consistent with the Turing machine's transition function.

Now let us introduce the notion of derivation for a combinatorial problem $S = \langle X, f_1, \ldots, f_m \rangle$. For two configurations $x, y \in X$, we say that y follows from x in one step and write $x \Rightarrow_S y$ if for some index i, $1 \le i \le m$, $x \in \operatorname{dom} f_i$ and $f_i(x) = y$. We also need its "deterministic version" $\Rightarrow_S^*$: $x \Rightarrow_S^* y$ in one step if there exists a unique index i, $1 \le i \le m$, such that $x \in \operatorname{dom} f_i$ and $f_i(x) = y$. As above, we take transitive reflexive closures of $\Rightarrow_S$ and $\Rightarrow_S^*$. We also write $x \Rightarrow_S^{*,n} y$ if $x \Rightarrow_S^* y$ in n steps at most.

To get a complete one-way function, we now only need to ensure that all these functions and simulations can be performed in polynomial time.

Definition 10. A family of combinatorial problems $\mathcal{S}$ is *efficiently encoded* if there exists a map $\rho \colon \mathcal{S} \to \{0,1\}^*$, called an *efficient encoding*, a polynomial p, and a Turing machine $M_{\mathcal{S}}$ such that for every problem $S = \langle X, f_1, \ldots, f_m \rangle \in \mathcal{S}$ the length of the code $|\rho(S)| \le p(m)$, and the Turing machine $M_{\mathcal{S}}$ for every $i = 1, \ldots, m$ and for every configuration $x \in \operatorname{dom} f_i$ computes the function

$$\rho(x) \mapsto \rho(f_i(x))$$

in polynomial time.

For example, for the family of semi-Thue systems we can get an efficient encoding by simply coding left- and right-hand sides of each rewriting rule. The machine M in this case is supposed to read a string, find an occurrence of some left-hand side, and make the substitution.

Definition 11. For a family of combinatorial problems $\mathcal{S}$ that simulates deterministic Turing machines and is efficiently encoded by an encoding $\rho_{\mathcal{S}}$, we call by its *Turing machine simulating function* (TMSF) a function $f\colon \{0,1\}^* \to \{0,1\}^*$ defined as follows:

- If the input is a pair $(\rho_{\mathcal{S}}(S), x)$ for some $S \in \mathcal{S}$, and, moreover, $x \Rightarrow_S^{*,n^2} y$ for some y and $y \notin \operatorname{dom} f_i$ for any $i = 1, \ldots, m$, then $f(\rho_{\mathcal{S}}(S), x) = (\rho_{\mathcal{S}}(S), y)$;
- Otherwise, f outputs its input.

Theorem 4. *If one-way functions exist, then the TMSF of every efficiently encoded family of combinatorial problems $\mathcal{S}$ that models deterministic Turing machines is a one-way function* (*more precisely, a weakly one-way function*).

Proof. With constant probability (for the uniform distribution this probability is proportional to $\frac{1}{|\rho(S)|^2 2^{|\rho(S)|}}$), the efficient encoding of every combinatorial problem $S \in \mathcal{S}$ shows up as a prefix of the input. Consider a length-preserving one-way function g and a Turing machine M_g that computes g in quadratic time (it exists by Proposition 1). Then, by Definition 9, there exists a combinatorial problem $S_g \in \mathcal{S}$ that simulates M_g. Thus, with constant probability, inverting TMSF of $\mathcal{S}$ is as hard as inverting g. $\triangle$

10. CONCLUSION AND OPEN PROBLEMS

In this paper, we have considered several new complete one-way functions based on combinatorial problems, discussed possibilities for other combinatorial settings to hold complete one-way functions, and proved a sufficient condition for this by introducing formal definitions generalizing these problems. These functions are combinatorial in nature and represent a step towards easy-to-analyze complete cryptographic objects, much like the satisfiability problem is a perfect complete problem for NP.

However, we are still not quite there. Basically, in all examples discussed above we sample a Turing machine at random and hope to find precisely the hard one. This is, of course, hardly a practical way to ensure security. We believe that constructing a *practical* complete one-way function requires a major new insight, and such a construction represents one of the most important challenges in modern theoretical cryptography.

Another direction would be to study different reductions between one-way functions. These investigations might lead to having a complete one-way function in the usual sense of "completeness," with respect to some class of polynomial reductions rather than in the sense of Definition 1.

The authors thank D.Yu. Grigoriev, Yu.V. Matiyasevich, I.N. Ponomarenko, and an anonymous referee for numerous fruitful discussions, important comments, and suggestions.

REFERENCES

1. Cook, S.A., The Complexity of Theorem-Proving Procedures, in *Proc. 3rd Annual ACM Sympos. on Theory of Computing, Shaker Heights, Ohio, USA, 1971*, pp. 151–158.

2. Levin, L.A., Universal Sequential Search Problems, *Probl. Peredachi Inf.*, 1973, vol. 9, no. 3, pp. 115–116 [*Probl. Inf. Trans.* (Engl. Transl.), 1973, vol. 9, no. 3, pp. 265–266].

3. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco: Freeman, 1979.

4. Karp, R.M., Reducibility Among Combinatorial Problems, *Complexity of Computer Computations (Proc. Sympos. on the Complexity of Computer Computations, Yorktown Heights, USA, 1972)*, Miller, R.E. and Thatcher, J.W., Eds., New York: Plenum, 1972, pp. 85–103.

5. Bogdanov, A. and Trevisan, L., Average-Case Complexity, *Found. Trends Theor. Comput. Sci.*, 2006, vol. 2, no. 1, pp. 1–106.

6. Gurevich, Yu., Average Case Completeness, *J. Comput. System Sci.*, 1991, vol. 42, no. 3, pp. 346–398.

7. Levin, L.A., Average Case Complete Problems, *SIAM J. Comput.*, 1986, vol. 15, no. 1, pp. 285–286.

8. Wang, J., Average-Case Intractable NP Problems, *Advances in Algorithms, Languages, and Complexity*, Du, D.-Z. and Ko, K.-I, Eds., Dordrecht: Kluwer, 1997, pp. 313–378.

9. Arora, S. and Barak, B., *Complexity Theory: A Modern Approach*, Cambridge: Cambridge Univ. Press, 2009.

10. *Handbook of Theoretical Computer Science*, vol. A: *Algorithms and Complexity*, van Leeuwen, J., Ed., Amsterdam: Elsevier; Cambridge: MIT Press, 1990.

11. Papadimitriou, C.H., *Computational Complexity*, Reading: Addison Wesley, 1994.

12. Gu, J., Purdom, P.W., Franco, J., and Wah, B.W., *Algorithms for the Satisfiability Problem*, Cambridge: Cambridge Univ. Press, 2000.

13. Vsemirnov, M.A., Hirsch, E.A., Dantsin, E.Ya., and Ivanov, S.V., Propositional Satisfiability Algorithms and Upper Bounds for Their Complexity, *Zap. Nauchn. Sem. S.-Peterburg. Otdel. Mat. Inst. Steklov. (POMI)*, 2001, pp. 14–46 [*J. Math. Sci. (N.Y.)* (Engl. Transl.), 2003, vol. 118, no. 2, pp. 4948–4962].

14. Blass, A. and Gurevich, Yu., Matrix Transformation is Complete for the Average Case, *SIAM J. Comput*, 1995, vol. 24, no. 1, pp. 3–29.

15. Gurevich, Yu., Complete and Incomplete Randomized NP Problems, in *Proc. 28th Annual Sympos. on Foundations of Computer Science, Los Angeles, USA, 1987*, Wasington: IEEE Comp. Soc., 1988, pp. 111–117.

16. Venkatesan, R. and Rajagopalan, S., Average Case Intractability of Matrix and Diophantine Problems, in *Proc. 24th Annual ACM Sympos. on Theory of Computing, Victoria, Canada, 1992*, New York: ACM, pp. 632–642.

17. Sipser, M., On Relativization and the Existence of Complete Sets, *Proc. 9th Colloq. on Automata, Languages and Programming (ICALP'82), Aarhus, Denmark, 1982*, Nielsen, M. and Schmidt, E.M., Eds., Lect. Notes Comp. Sci., vol. 140, Berlin: Springer, 1982, pp. 523–531.

18. Diffie, W. and Hellman, M., New Directions in Cryptography, *IEEE Trans. Inform. Theory*, 1976, vol. 22, no. 6, pp. 644–654.

19. Grigoriev, D., Hirsch, E.A., and Pervyshev, K., A Complete Public-Key Cryptosystem, *Groups–Complexity–Cryptology*, 2009, vol. 1, no. 1, pp. 1–12.

20. Harnik, D., Kilian, J., Naor, M., Reingold, O., and Rosen, A., On Robust Combiners for Oblivious Transfers and Other Primitives, *Advances in Cryptology—EUROCRYPT 2005. Proc. 24th Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark*, Cramer, R., Ed., Lect. Notes Comp. Sci., vol. 3494, Berlin: Springer, 2005, pp. 96–113.

21. Levin, L.A., One-Way Functions and Pseudorandom Generators, *Combinatorica*, 1987, vol. 7, no. 4, pp. 357–363.

22. Goldreich, O., *Foundations of Cryptography. Basic Tools*, Cambridge: Cambridge Univ. Press, 2001.

23. Kojevnikov, A.A. and Nikolenko, S.I., New Combinatorial Complete One-Way Functions, in *Proc. 25th Sympos. on Theoretical Aspects of Computer Science, Bordeaux, France, 2008*, pp. 457–466.

24. Levin, L.A., The Tale of One-Way Functions, *Probl. Peredachi Inf.*, 2003, vol. 39, no. 1, pp. 103–117 [*Probl. Inf. Trans.* (Engl. Transl.), 2003, vol. 39, no. 1, pp. 92–103].

25. Wang, J., Random Instances of Bounded String Rewriting Are Hard, *J. Comput. Inf.*, 1995, vol. 1, no. 2, pp. 11–23.

26. Book, R.V. and Otto, F., *String-Rewriting Systems*, New York: Springer, 1993.

27. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*, Davis, M., Ed., New York: Raven, 1965.

28. Gorbatov, V.A., *Fundamental'nye osnovy diskretnoi matematiki. Informatsionnaya matematika* (Fundamentals of Discrete Mathematics. Informational Mathematics), Moscow: Nauka, 2000.

29. Wang, J. and Belanger, J., On the NP-Isomorphism Problem with Respect to Random Instances, *J. Comput. System Sci.*, 1995, vol. 50, no. 1, pp. 151–164.

30. Post, E., Recursive Unsolvability of a Problem of Thue, *J. Symbolic Logic*, 1947, vol. 12, pp. 1–11.

31. Gurari, E.M., *An Introduction to the Theory of Computation*, Ohio State Univ.: Computer Science Press, 1989.

32. Post, E., A Variant of a Recursively Unsolvable Problem, *Bull. Amer. Math. Soc.*, 1946, vol. 52, pp. 264–268.

33. Ruohonen, K., On Some Variants of Post's Correspondence Problem, *Acta Inform.*, 1983, vol. 19, no. 4, pp. 357–367.

34. Sipser, M., *Introduction to the Theory of Computation*, Boston: Thomson Course Technology, 2006.

35. Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages and Computation*, Reading: Addison-Wesley, 1979.

36. Wang, J., Distributional Word Problem for Groups, *SIAM J. Comput.*, 1999, vol. 28, no. 4, pp. 1264–1283.

37. Berger, R., The Undecidability of the Domino Problem, *Mem. Amer. Math. Soc.*, 1966, vol. 66, pp. 1–72.

38. Culik, K., An Aperiodic Set of 13 Wang Tiles, *Discrete Math.*, 1996, vol. 160, no. 1–3, pp. 245–251.

39. Culik, K. and Kari, J., An Aperiodic Set of Wang Cubes, *J. Universal Comp. Sci.*, 1995, vol. 1, no. 10, pp. 675–686.

40. Kari, J., A Small Aperiodic Set of Wang Tiles, *Discrete Math.*, 1996, vol. 160, no. 1–3, pp. 259–264.

41. Wang, H., Proving Theorems by Pattern Recognition II, *Bell Syst. Techn. J.*, 1961, vol. 40, no. 1, pp. 1–41.

42. Ben-David, S., Chor, B., and Goldreich, O., On the Theory of Average Case Complexity, in *Proc. 21st Annual ACM Sympos. on Theory of Computing (STOC'89), Seattle, USA*, New York: ACM, 1989, pp. 204–216.