ISSN 0146-4116, Automatic Control and Computer Sciences, 2019, Vol. 53, No. 7, pp. 573–583. © Allerton Press, Inc., 2019.
Russian Text © The Author(s), 2018, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2018, No. 6, pp. 589–606.

Even Simple π-Calculus Processes are Difficult to Analyze

M. M. Abbas[a], * and V. A. Zakharov[b, c], **

[a] *Moscow State University, Moscow, 119991 Russia*

[b] *National Research University University Higher School of Economy, Moscow, 101000 Russia*

[c] *Ivannikov Institute of System Programming, Russian Academy of Sciences (ISP RAS), Moscow, 109004 Russia*

**e-mail: unlock96@gmail.com*

***e-mail: zakh@cs.msu.ru*

Received September 15, 2018; revised October 30, 2018; accepted November 10, 2018

Abstract—Mathematical models of distributed computations, based on calculus of mobile processes (π-calculus) are widely used for checking information security properties of cryptographic protocols. Since π-calculus is a Turing-complete computation model, this problem is unsolvable in the general case. Therefore, its study is carried out only for some special classes of π-calculus processes with
1 restricted computational capabilities, for example, for nonrecursive processes with all runs limited in length, for processes with a limited number of parallel components, etc. However, even in these cases the proposed checking procedures are very time consuming. We assume that this is due to the very nature of the π-calculus processes. The goal of this paper is to show that even for the weakest passive adversary model and for relatively simple protocols that make use of only basic π-calculus operations, the checking of the information security properties of these protocols is a co-NP-complete problem.

Keywords: π-calculus, protocol, security, passive adversary, verification, complexity, NP-completeness

DOI: 10.3103/S0146411619070022

INTRODUCTION

Aiming to extend the expressive capacities of Calculus of Communicating Systems (CCS), Rober Milner and his colleagues introduced in [25] a new mathematical model of distributed computation environments (DCEs) called mobile process calculi or, in short, π-calculus. Its two distinct features are the pattern (ν-operator) of generating new names and the opportunity to transfer communication channel names along communication channels. These qualities enable π-calculus processes to change the communicative environment by introducing new communication channels in the course of computations and thuswise model the migration of processes. As shown by Milner in article [26], π-calculus processes can simulate computations of λ-calculus terms. This is why, unlike CCS, π-calculus is a Turing-complete computation model. The Turing-completeness of π-calculus stems from the combination of the two specified features with the replication operator inherited from CCS; should any of the three factors be absent, π-calculus ceases being a universal computational model.

Within a relatively short period following the publication of [25] it was found out that, in addition to describing the behavior of mobile process systems, π could be used with success to formally describe patterns of handling objects in object-oriented programming [33] and model business processes [30] and biochemical reactions [28]. However, the greatest interest was sparked by the fact that the authors of [1] revealed that π-calculus could be used for constructing formal models of cryptographic protocols.

In their seminal work [19], D. Dolev and E. Yao propose to divide the check of cryptographic protocol security properties in two subtasks: 1) prove the security properties (confidentiality, integrity, etc.) for basic functions (encryptions, hashing, etc.) used in cryptographic protocols; 2) check cryptographic protocols for resistance under the assumption that all the cryptographic primitives used in them meet necessary security requirements. That said, the Dolev–Yao model endows an adversary with powerful capabilities, including the one to intercept, form, and send messages along open communication channels. The spi-calculus proposed in [1] for modeling cryptographic protocols has appeared an efficient model in the framework of which the security check of cryptographic protocols in the Dolev–Yao model can be reduced (or, more formally, defined by reduction) to checking various kinds of equivalence of spi-calculus processes or to checking specially distinguished *vulnerability states* of processes for reachability. The paper

[2] proposes one more expansion of π-calculus—so called applied π-calculus, which allows constructing complex terms and describing their algebraic properties through equations. Applied π-calculus can also be enriched with auxiliary memory with shared and confidential cells [5, 9]; this memory makes it convenient to describe authentication and key distribution protocols, etc.

There are a lot of works about analyzing the behavior of processes in calculi of cryptographic protocols based on π-calculus. Since π-calculus is an algorithmically complete computation model, all the specified tasks are algorithmically undecidable in the general case. This is why, they are studied only for some special classes of processes with limited computation capabilities, e.g., for nonrecursive processes with all runs limited in length, for processes with a limited number of parallel components, etc. The major achievements of exploring the tasks of analyzing the behavior of spi-calculus processes are exposed below. The reachabilty check has been proven a decidable task for nonrecursive spi-calculus processes [3, 23] and is NP-complete [29]. The most useful kinds of process equivalence for cryptographic apps are test [1] and visible equivalence [2]. The decidability of checking nonrecursive processes for test equivalence was established in [20]; however, the proposed decision procedure has high computational complexity. The algorithms of checking visible equivalence for some classes of spi-calculus processes and applied π-calculus were presented and corrected in [10, 15, 16, 18]. As also shown in [16], the visible equivalence check for nonrecursive applied π-calculus processes is an NP-complete task. In addition to visible processes equivalence, more primitive bi-simulation relationships were examined as well [8]. According to [32], the check of the open bi-simulation of nonrecursive spi-calculus processes is a decidable task. In addition, it was explored in [7] whether spi-calculus processes could be verified by static analysis. The strength of this approach is that it can be applied to any processes, recursive processes included. The proposed formal techniques of analyzing cryptographic protocols with the help of π-calculus were applied in practice and make it possible to detect and eliminate a vulnerability in the authenticated routing protocol (ARAN) [22].

Even when nonrecursive spi-calculus process P has only a finite set of computations, it is not a simple task to analyze the behavior of P in interaction with the outside environment. The reason is that in the Dolev–Yao model the outside environment (adversary) is presented as an infinite family of processes $\mathscr{A}$ and the task of checking information security properties consists in analyzing the behavior of all $P|R$ processes, where $R \in \mathscr{A}$. Generally speaking, the processes from $\mathscr{A}$ are capable of generating as complex messages as possible, which is why the system of processes $\{P|R : R \in \mathscr{A}\}$ can have an infinitely large number of states and computations. The techniques proposed and developed in works [10, 15, 16, 18, 20, 32] allow distinguishing (sometimes indirectly) such finite subset of processes $\mathscr{A}'$, $\mathscr{A}' \subseteq \mathscr{A}$, which describes the outside environment, that in the Dolev–Yao model the security check of protocol P requires only analyzing a finite number of $P|R$ processes, where $R \in \mathscr{A}'$. To say the least, the size of set of processes $\mathscr{A}'$ appears exponentially dependent on the size of process P, which is why the algorithms of checking properties of spi-calculus or applied π-calculus for security, proposed in the indicated works, are not efficient.

The papers [3, 4] were the first ones, where the task of checking insecurity properties of cryptographic protocols with a limited number of sessions (nonrecursive protocols) was declared NP-complete. However, those works considered only model cryptographic protocols with simple (atomic) encryption keys. In addition, the authors of [4] provide only outline the estimation of the complexity of the decision procedure they describe. For the full proof of the theorem of NP-completeness of checking insecurity properties of nonrecursive cryptographic protocols with atomic encryption keys see publications [21, 31]. The paper [29] supported that result by showing that the insecurity property checking task also remained NP-complete for nonrecursive cryptographic protocols with composite encryption keys. The proof that this task belongs to the complexity class NP relies on the proposition that any unsecure protocol in the Dolev–Yao model can be compromised by an active adversary able to generate messages the size of which polynomially depends on the protocol size. In subsequent articles [11–14] this approach to designing insecure checking procedures and estimating their complexity was applied to protocols with more intricate cryptographic primitives. However, as shown in article [24], the proof of the proposition about the existence of the minimal polynomial adversary model, which is exposed in article [29], contains a mistake corrected in [24].

In all the works mentioned above the authors explored the task of checking insecurity properties for cryptographic protocols with an ever growing diversity of computational and communicative actions, and the main emphasis was to prove that the problem under consideration is in NP. The NP-completeness of the task in question was proven by reducing the satisfiability problem for 3-CNF to that task. Naturally, the reduction would be simplified with the growing complexity of the cryptographic protocols model. Various means accepted in spi-calculus were used for the purpose, including encryption/decryption functions, pair functions, branching operators, etc. We assume, however, that the difficulties with analyzing the security of cryptographic protocols modeled by various extensions of π-calculus are conditioned, first

of all, by the algorithmic difficulties typical of π-calculus proper as the basic model of distributed mobile computation systems. We consider the task of checking for security of protocols, described by the basic means of π-calculus, in the passive adversary model. The analyzed protocols are a parallel composition of processes $P_1|P_2|\cdots|P_n$, each of which is a successive composition of message sending and receiving $P_i = act_1.act_2.\cdots.act_m$. Only one among all the names used as messages is confidential; all the others are considered either commonly known or random names used only once. The passive adversary can intercept (eavesdrop) messages transferred along communication channels the names of which he knows. The captured names expand the adversary's knowledge and can be used in subsequent interceptions. A protocol is considered secure when, whatever its execution, confidential names will not be part of the adversary's knowledge.

In this article we show that even basic π-calculus tools will suffice for constructing for any 3-CNF φ such $Proc_\varphi = P_1|P_2|\cdots|P_n$ process of the specified kind that will resist the passive adversary if and only if a 3-CNF φ is unsatisfiable. That said, the size of $Proc_\varphi$ linearly depends on the size of formula φ, and all the runs of this process are completed in a normal manner, without getting caught in a deadlock. In light of the results on the NP-completeness of checking the insecurity properties and cryptographic protocols, that are obtained in articles [3, 4, 11–14, 21, 24, 29, 31], the main theorem of this article shows that the main factor determinative to the complexity of the considered task is the limited computating length of cryptographic protocols; the influence of other factors such as diversity of cryptographic primitives and protocol computing control tools used in the protocols, structure and size of transferred messages, etc., is secondary.

The article is structured as exposed below. Section 2 describes the syntaxis and semantics of π-calculus. Section 3 defines the passive adversary model and formulates the task of checking π-calculus processes for security in the passive adversary model. Section 4 describes the structure of $Proc_\varphi$, correspondent to arbitrary 3-CNF φ, and shows that this process is characterized by normal termination—all of its computations are completed in one and the same empty process. It has also been found out that all the computations in $Proc_\varphi$ are safe if and only if 3-CNF φ is unsatisfiable. It follows hereforth that the satisfiability of 3-CNF can be reduced to checking the insecurity properties of π-calculus processes in the passive adversary model. In the concluding section we discuss the significance of the results.

Syntaxis and Semantics of π-Calculus

We shall confine only to considering the basic recursive fragment of synchronous monadic calculus of mobile processes. Assume there is some infinite number of objects $\mathcal{N}$, that will be referred to as *names*. They serve to indicate communication channels and also data transferred along such. Names will be recorded in lowercase letters as $a, b, \ldots, x, y, z$.

Elementary synchronous communicative *action* E is any expression $\overline{x}\langle y\rangle$ (message y sent along communication channel x) or $x(y)$ (reception of a message, binding name y, along communication channel x). A π-calculus process is any expression made up according to the following rules:

$$\begin{aligned} P,Q ::= \quad & \mathbf{0} && \text{(complete the process)}\\ \mid\ & E.P && \text{(take action } E \text{ and proceed to executing process } P)\\ \mid\ & P|Q && \text{(execute processes } P \text{ and } Q \text{ in parallel)}\\ \mid\ & (\nu x)\,P && \text{(enter new name } x \text{ and proceed to executing } P). \end{aligned}$$

The set of processes thus defined will be referred to as $\mathcal{P}$.

The occurrences of names in a process are divided in *free* and *bound*.

Set $fn(P)$ of the free names of P is defined with respect to the process structure according to the following rules:

1. $fn(\mathbf{0}) = \emptyset$,
2. $fn(\overline{x}\langle y\rangle.P) = fn(P) \cup \{x, y\}$, $fn(x(y).P) = fn(P) \cup \{x\})\backslash\{y\}$,
3. $fn((\nu x)P) = fn(P)\backslash\{x\}$,
4. $fn(P|Q) = fn(P) \cup fn(Q)$.

The occurrence of a name x in P is considered free if it is not ocurred in any subprocess $(\nu x)P'$ or $y(x).P'$ of P. We write $P\{x/y\}$ to indicate a process derived from P by synchronously replacing all the free occurrences of name y with name x. Substitution $\{x/y\}$ is called correct for process P when any free occurrence of y is not found in any subprocess $(\nu x)P'$ of P.

The operation semantics of mobile processes is determined by structural congruence relation $\equiv_\pi$ and reduction relation $\to_\pi$. The relation $\equiv_\pi$ is the smallest congruence relation in set of processes $\mathcal{P}$, that satisfies the following equalities:

1. $\mathbf{0}|P \equiv_\pi P$, $P|Q \equiv_\pi Q|P$, $P|(Q|R) \equiv_\pi (P|Q)|R$, i.e., system $(\mathcal{P}, |, \mathbf{0})$ is a commutative semigroup;

2. $(\nu y)P \equiv_\pi (\nu x)P\{x/y\}$ for any name $x, x \notin fn(P)$, and correct for the process P substitution $\{x/y\}$;

3. $((\nu x)P)|Q \equiv_\pi (\nu x)(P|Q)$ for any name $x, x \notin fn(Q)$;

4. $(\nu x).\mathbf{0} \equiv_\pi \mathbf{0}$, $(\nu x)((\nu y)P) \equiv_\pi (\nu y)((\nu x)P)$.

It is easy to see that if $P \equiv_\pi Q$ then $fn(P) = fn(Q)$.

The reduction relation is the smallest relationship $\to_\pi \subseteq \mathcal{P} \times \mathcal{N} \times \mathcal{N} \times \mathcal{P}$ which for any processes P, Q, P', Q' and names x, y, z meets the following requirements (the satisfiability of $\to_\pi$ for a quadruplet P, x, y, Q is traditionally denoted as $P \xrightarrow{x(y)}_\pi Q$):

1. $(\overline{x}\langle y\rangle.P)|(x(z).Q) \xrightarrow{x(y)}_\pi P|Q\{y/z\}$;

2. $P|Q \xrightarrow{x(y)}_\pi P'|Q$ at $P \xrightarrow{x(y)}_\pi P'$;

3. $(\nu z).P \xrightarrow{x(y)}_\pi (\nu z).P'$ at $P \xrightarrow{x(y)}_\pi P'$;

4. $Q \xrightarrow{x(y)}_\pi Q'$ at $P \equiv_\pi Q$, $P \xrightarrow{x(y)}_\pi P'$ and $P' \equiv_\pi Q'$.

Quadruplets $P \xrightarrow{x(y)}_\pi Q$ meeting the process reduction relation will be referred to as *reductions* (in other words, communications along communication channels x). A channel x, along which it is possible to reduce process P, is called *active in process* P.

The *run* of a process P_0 is any sequence of reductions recorded as

$$P_0 \xrightarrow{x_1(y_1)}_\pi P_1 \xrightarrow{x_2(y_2)}_\pi \cdots \xrightarrow{x_{n-1}(y_{n-1})}_\pi P_{n-1} \xrightarrow{x_n(y_n)}_\pi P_n. \quad (1)$$

A process not admitting any kind of reduction is called *deadlock*. The run (1) is called complete if P_n is a deadlock. If $P_n \equiv_\pi \mathbf{0}$ then the run completion is *normal*. The obvious proposition we shall use in analyzing process calculations is exposed below.

Proposition 1. *If a communication channel x is active in a process P_0, then the communication along x occurs in any complete run* (1).

In other words, a message will be eventually sent along each active channel in P_0.

Passive Adversary Model

An outside observer (adversary) relative to π-calculus process P is regarded as some process R that can be engaged in interaction with P. This interaction is determined by parallel composition $P|R$. All the names of P that are connected by the operator ν can be interpreted as one-time used data generated by the process. This is why the communication of P and R can initially be executed only along the communication channels from the set $fn(P)$. Then, however, the adversary gets (intercepts) the messages communicated along these channels and thus finds out new names he can use for subsequently communicating with P. Passive adversary R is able only to eavesdrop messages sent by P along the communication channels known to the adversary in the course of sending and receiving messages. This ability of the passive adversary is modeled by a pair of successively executed actions $x(y).\overline{x}\langle y\rangle$. The messages thus eavesdropped are not lost or replaced. The passive adversary can be represented by any process R emerging as a successive composition of pairs of reception and sending of one and the same message along one and the same communication channel. Unlike the passive adversary, the active adversary is able not only to eavesdrop messages, but, also to construct new messages and substitute intercepted messages; the active adversary can be represented by any π-calculus process R.

The information security properties of process P displays itself in its interaction with an arbitrary of a particular kind. The abstract adversary model usually used to avoid the need for considering the behavior of all possible $P|R$ compositions is determined by the state of its knowledge K defined as the set of names known to the adversary. This set may change while process P performs actions. If a process P interacts with the passive adversary, state (P, K) of the interacting system can be transformed to (P', K') only through a certain transition of the process P. If an adversary is active, (P, K) can be transformed to (P', K') by both, making a certain reduction of the process P and performing some action $x(y)$ or $\overline{x}\langle y\rangle$ of the process P on condition that the state of the adversary's knowledge K allows him to form pairwise action $\overline{x}\langle z\rangle$ or $x(z)$. The adversary's purpose is to achieve the state of knowledge from some distinguished family S that poses a security threat.

In this article we confine ourselves to considering the interaction of π-calculus processes with the passive adversary. He is characterized by the list of entries about the names he knows. To form the entries we shall introduce new operator κ and use expression (κx), where $x \in \mathcal{N}$, to indicate the entry of name x in the adversary's database. The *passive adversary model* is any list of entries (database) A that can be constructed according to the following rules:

$$\begin{aligned} A ::= \ & \mathbf{0} && \text{(empty database)} \\ & \mid (\kappa x).A && \text{(database with the entry of name } x\text{)}. \end{aligned}$$

Unlike the operator ν, the entry operator κ does not bind names, which is why the set $fn(A)$ of the free names of an adversary A consists of all the names included in expression A.

The passive adversary may follow π-calculus runs; we shall formally describe this phenomenon through the parallel composition operator.

We shall use the term "monitoring of a process P by an adversary A" to refer to any expression M that can be derived according to the following rules:

$$\begin{aligned} M ::= \ & P|A && \text{(adversary } A \text{ follows process } P\text{)} \\ & \mid (\nu x).M && \text{(monitoring } M \text{ is performed in the region of determining name } x \text{ used once)}. \end{aligned}$$

We shall indicate the set of all possible monitorings as $\mathcal{M}$, and the set of free names of a monitoring M as $fn(M)$, in which we are especially interested in free names included in adversary entries (κx). We shall use expression $open(M)$ to denote the subset of all free names of the specified kind.

Similarly to the semantics of π-calculus processes, the operation semantics of monitorings is determined by structural congruence relation ≡ and transition relation →. The relation ≡ is the smallest congruence relation on the set of monitorings $\mathcal{M}$ that includes structural congruence relation $\equiv_\pi$ on the set of processes and meets the following equalities:

1. $(\nu y).\ M \equiv (\nu x).\ M\{x/y\}$ for any name x, $x \notin fn(M)$, and a correct for monitoring M substitution $\{x/y\}$;

2. $(\kappa x).(\kappa y).A \equiv (\kappa y).(\kappa x).A$ for any pair of names x, y and an adversary A;

3. $(\kappa x).(\kappa x).A \equiv (\kappa x).A$ for any names x and an adversary A.

The first of these identities means that all of the adversary's data about one-time names are valid only for the observed process whereas the other two identities allow considering the adversary's knowledge base as an unordered set of entries. Taking account of the latter circumstance, we shall settle with $A(X)$ to denote adversary $(\kappa x_1).(\kappa x_2).\ldots.(\kappa x_n).\mathbf{0}$ for an arbitrary set of names.

Transition relation → is the smallest binary relation on the set of monitorings $\mathcal{M}$, that meets the following requirements for any processes P, Q, P', Q', model adversary A, monitorings M, N, M', N', and names x, y:

1. $P|A \to P'|A$ at $P \xrightarrow{x(y)}_\pi P'$;

2. $P|(\kappa x).A \to P'|(\kappa y).(\kappa x).A$ at $P \xrightarrow{x(y)}_\pi P'$;

3. $(\nu x).M \to (\nu x).M'$ at $M \to M'$;

4. $N \to N'$ at $M \equiv N$, $M \to M'$, and $M' \equiv N'$.

According to the second of the herein cited rules, if an observed process communicates certain data along a communication channel, the name of which is known to the adversary, the communicated data will also

be known to the adversary. We shall refer to the reflective transitive closure of the transition relation as $\rightarrow^*$. If $M \rightarrow^* M'$ holds, monitoring M' is considered reachable from monitoring M.

Case 1. Assume that there is a process

$$P = ((\nu x).(\nu y).\overline{ch}\langle x\rangle.x(y).\mathbf{0}) \mid ((\nu z).ch(z).\overline{z}\langle secret\rangle.\mathbf{0})$$

and a passive adversary model $A = (\kappa\ ch).\mathbf{0}$. Then the monitoring $M = P|A$ generates the following sequence of transitions:

$$M$$
$$\downarrow$$
$$(\nu x).(\nu z).(((\nu y).x(y).0) \mid (\overline{x}\langle secret\rangle.0) \mid (\kappa x).(\kappa\ ch).\mathbf{0})$$
$$\downarrow$$
$$(\nu x).(\nu y).(\nu z).((\kappa\ secret).(\kappa x).(\kappa\ ch).\mathbf{0}).$$

The adversary with certain a priori data about the process aims to find out certain confidential data handled in the process. This is why finite name sets X and Y are both the attack and the threat relative to which the process security requirement is formulated. We shall say that a process P secure with respect to the threat Y in carrying out the attack X (in short, P is (X,Y)-secure) if $Y \subseteq open(M)$ is not true for any monitoring M reachablee from the initial monitoring $P|A(X)$, i.e., none of the runs of the process P allows a passive adversary, knowing only name set X in the beginning, eavesdrop the process communications so as to form set Y from the eavesdropped names. In the passive adversary model the security checking problem for processes is to find out whether process P is (X,Y)-resistant for arbitrary process P, attack X, and threat Y. In the aforementioned example process P is not secure w.r.t the threat $\{secret\}$ in carrying out the attack $\{ch\}$.

Complexity of the Security Checking Problem for Processes

Theorem 1. *In the passive adversary model the security checking problem for processes from the set $\mathcal{P}$ is a co-NP-complete.*

Proof. Since the processes from set $\mathcal{P}$ contain no replication operator ! or other means of describing computations recursively, the length of each run of process a P from set $\mathcal{P}$ does not exceed the size of the process P. In addition, as seen from the definition of transition relation $\rightarrow$, the number of pairwise incongruent descendants (images) M' for each monitoring M by the relation $\rightarrow$ does not exceed the squared size of M. It follows hereforth that the security checking problem for processes from the set $\mathcal{P}$ is in co-NP.

To provee the co-NP-completeness of the considered problem, we shall show that the problem of checking 3-CNF for unsatisfiability is *log-space reducible to it*. Process $Proc_\varphi$ we shall construct for an arbitrarily specified 3-CNF φ is able to model the computation of the value of formula φ on all sets of variable values and allows an adversary to eavesdrop confidential name *secret* transmitted along open communication channel *ch* iff φ is satisfiable.

Assume that an arbitrary 3-CNF $\varphi = D_1 \wedge D_2 \wedge \ldots \wedge D_N$ depends on variables $x_1, x_2, \ldots, x_n$; each clause D_i, $1 \le i \le N$, in 3-CNF is recorded as $\ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$, where $\ell_{1i}, \ell_{2i}, \ell_{3i}$ are literals that are variables or their negations. We shall distinguish the name of literal ℓ from the name of the variable this literal is based on. Each literal ℓ will be denoted as x^σ, where $\sigma = 1$ if $\ell = x$ and $\sigma = 0$ if $\ell = \neg x$. It can be reckoned without loss of generality that each literal x_i^σ, $1 \le i \le n$, $\sigma \in \{0,1\}$, is included in 3-CNF φ. In addition, we for-literal ℓ we shall write ℓ^* to denote the contrary literal of opposite polarity and write m_ℓ to denote the total number of occurences of literal ℓ in the formula φ.

Let us describe the structure of a process $Proc_\varphi$. The only two free names it contains are *ch* and *secret*. The former denotes an open communication channel (exposed to eavesdropping) and constitutes an attack. The name *secret* denotes confidential data and constitutes a threat. All the other names occurred in $Proc_\varphi$ are bounded by operators ν. This set of names consists of names of variables $x_1, \ldots, x_n$, names $\ell_1, \ldots, \ell_n, \ell_1^*, \ldots, \ell_n^*$ of positive and negative literals corresponding to these variables, clause names

$d_1,\ldots,d_N$, and also three special names g, h, and r. The process $Proc_\varphi$ is a composition of parallel subprocesses meant to play the roles exposed below.

1. A subprocess *Init* is meant to activate for each variable x_i, $1 \le i \le n$, precisely one of literals x_i or $\neg x_i$ by sending for each x_i a message that can be received either by process S_{ℓ_i} or by process $S_{\ell_i^*}$:

$$Init = \overline{x_1}\langle z\rangle.\overline{x_2}\langle z\rangle.\ldots.\overline{x_n}\langle z\rangle.\mathbf{0}.$$

2. For each of $2n$ literals $\ell = x_i^\sigma$, $1 \le i \le n$, and $\sigma \in \{0,1\}$ a subprocess S_ℓ is meant to activate all the name ℓ communication channels used in the process $Proc_\varphi$:

$$S_\ell = x_i(z).\underbrace{\overline{\ell}\langle z\rangle.\overline{\ell}\langle z\rangle.\ldots.\overline{\ell}\langle z\rangle}_{m_\ell + m_{\ell^*}\ @\text{раз}}.\mathbf{0}.$$

3. A subprocess $Check_{\varphi=0}$ is meant to check that formula φ takes the value 0 on the set of variable values correspondent to activated literals:

$$Check_{\varphi=0} = Check_{D_1=0} \mid Check_{D_2=0} \mid \ldots \mid Check_{D_N=0},$$

where a subprocess $Check_{D_i=0}$ is recorded for each clause $D_i = \ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$ as

$$Check_{D_i=0} = \ell^*_{1i}(z).\ell^*_{2i}(z).\ell^*_{3i}(z).\overline{r}\langle ch\rangle.\mathbf{0}.$$

Thus, if the literals contrary to literals $\ell_{1i}, \ell_{2i}, \ell_{3i}$ are activated, each process $Check_{D_i=0}$ sends the name of the open channel *ch* along the communication channel r.

4. A subprocess $Check_{\varphi=1}$ is meant to check that the formula φ takes the value 1 on the set of variable values correspondent to activated literals:

$$Check_{\varphi=1} = Check_{D_1=1} \mid Check_{D_2=1} \mid \ldots \mid Check_{D_N=1} \mid CheckAll,$$

where a subprocess $Check_{D_i=1}$ for each clause $D_i = \ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$ is recorded as

$$Check_{D_i=1} = (\ell_{1i}(z).\overline{d_i}\langle z\rangle.\mathbf{0}) \mid (\ell_{2i}(z).\overline{d_i}\langle z\rangle.\mathbf{0}) \mid (\ell_{3i}(z).\overline{d_i}\langle z\rangle.\mathbf{0}),$$

and a subprocess *CheckAll* as

$$CheckAll = d_1(z).d_2(z).\ldots.d_N(z).\overline{r}\langle secret\rangle.\mathbf{0}.$$

Thus, if at least one literal ℓ_{ji}, $1 \le j \le 3$ included in each clause $D_i = \ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$, $1 \le i \le N$, is activated, then the process $Check_{\varphi=1}$ sends the secret name *secret* along the communication channel r.

5. A subprocess *OpenCh* is meant to ensure communication along the open communication channel *ch* after checking the value of φ and launch the garbage collection process that will allow executing all unfinished actions of the process $Proc_\varphi$:

$$OpenCh = (r(y).\overline{ch}\langle y\rangle.\overline{g}\langle z\rangle.\overline{h}\langle z\rangle.\mathbf{0}) \mid (ch(x).\mathbf{0}).$$

The collection of garbage is launched by message sending actions $\overline{g}\langle z\rangle$ and $\overline{h}\langle z\rangle$.

6. Garbage collection subprocess *Garbage* is meant to activate all the

literals not activated by the subprocess *Init* and force to execute all the actions of subprocesses $Check_{\varphi=0}$ and $Check_{\varphi=1}$ that remained untaken after the first activation of the literals by the process *Init*:

$$Garbage = Final \mid Collect,$$

where

$$Final = g(z).\overline{x_1}\langle z\rangle.\overline{x_2}\langle z\rangle.\ldots.\overline{x_n}\langle z\rangle.\mathbf{0},$$

$$Collect = h(z).d_1(z).d_1(z).d_2(z).d_2(z).\ldots.d_N(z).d_N(z).r(y_1).r(y_2).\ldots.r(y_N).\mathbf{0}.$$

We shall denote by *names* the set of all nemes, except for *ch* and *secret*, appear in the aforemetioned processes. The process $Proc_\varphi$ is the parallel composition of all those subprocesses. In this composition all the names from the list *names* are bounded by the operator ν as

$$Proc_{\varphi} = (\nu\ names).\Big(Init\,|S_{\ell_1}|\,S_{\ell_1^*}|...|\,S_{\ell_n}|\,S_{\ell_n^*}|Check_{\varphi=0}|Check_{\varphi=1}|OpenCh|Garbage\Big).$$

It is easy to see that, with CNF φ available, $Proc_{\varphi}$ can be constructed in logarithmic space.

We shall show first that any complete run of $Proc_{\varphi}$ recorded as

$$Proc_{\varphi} = P_0 \to_{\pi} P_1 \to_{\pi} \cdots \to_{\pi} P_{m-1} \to_{\pi} P_m \tag{2}$$

finishes in a normal way, i.e., $P_m = \mathbf{0}$.

First of all, note that none of names u being the argument of any message reception action $v(u)$ is the name of any communication channel. This means that in the course of the run (2) the names of communication channels are unchanged (only renaming acceptable for congruence relation $\equiv_{\pi}$ is possible). In addition, for every name u of a channel the number of message sendings along this channel in the process $Proc_{\varphi}$ is equal to the number of message receptions from the same channel. This is why, to prove the normal completion of the run (2), it is enough to show that all the message sending actions of the process $Proc_{\varphi}$ are executed in this run.

Note that in $Proc_{\varphi}$ the channel x_1 is active, whereas the process P_m is deadlock. This means that, by Proposition 1, the communication along the channel x_1 was carried out during the run (2). The receptions of messages from this channel are observed only in literal activation subprocesses S_{x_1} and $S_{\neg x_1}$. We shall consider one of those subprocesses, in which the communication along the channel x_1 was carried out earliest in the run (2); assume that this subprocess is meant to activate a literal $\ell_1 = x_1^{\sigma_1}$. Then, after the first communication along the channel x_1 is executed, the communication channel ℓ_1 is activated. According to the description of the subprocess S_{ℓ}, the activity of the channel ℓ_1 will be maintained during the run until all the possible actions aimed at receiving messages from this channel are completed.

After the communication is executed along x_1 (in either *Init* or *Final*), the channel x_2 also becomes active, and reasonings similar to the ones provided above for the channel x_1 will hold. Extending these reasonings to all communication channels $x_1, x_2, ..., x_n$, we conclude that in the run (2) communication channels $\ell_1 = x_1^{\sigma_1}$, $l_2 = x_2^{\sigma_2}$, ..., $\ell_n = x_n^{\sigma_n}$ appear activated for certain binary tuple $\alpha = (\sigma_1, \sigma_1, ..., \sigma_n)$; these channels will be maintained as active during the run until all possible actions aimed at receiving messages from these channels are completed.

Then we need to consider two cases, depending on the value taken on by the formula φ on the set of variable values α.

If $\varphi(\alpha) = 0$ then CNF φ contains a clause $D_j = \neg\ell_{i_1} \vee \neg\ell_{i_2} \vee \neg\ell_{i_3}$ for a certain triplet of above distinguished literals $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$. According to the description of the subprocess $Check_{\varphi=0}$, one of the components of its parallel compositions is a subprocess

$$Check_{D_j=0} = \ell_{i_1}(z).\ell_{i_2}(z).\ell_{i_3}(z).\overline{r}\,\langle ch\rangle.\mathbf{0}.$$

Since, as it was shown above, the subprocesses S_{ℓ} sustain the activity of paths $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$ until all possible actions aimed at receiving messages from these channels are performed, the communication in the run (2) is executed along the channels $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$, involving the message receiving actions of the subprocess $Check_{D_j=0}$. After the last of the three reductions is made, the communication channel r becomes active, because the reception of messages from this channel launches one of the components of the parallel composition of the subprocess *OpenCh*. If $\varphi(\alpha) = 1$, then each clause D_j, $1 \le j \le N$,contains one of above distinguished literals $\ell_1, \ell_2, ..., \ell_n$. Therefore, according to the description of the subprocess $Check_{\varphi=1}$ for each j, $1 \le j \le N$, the parallel composition of this subprocess contains the component $\ell_i(z).\overline{d_j}\,\langle z\rangle.\mathbf{0}$, where ℓ_i is one of the literals distinguished above that is the name of an activated communication channel. Since the *literals* channels are maintained as active until all possible actions aimed at receiving messages from these channels are performed, reductions are made in the run (2) involving all the message receiving actions occurred in the beginning of the indicated components. After these reductions are executed, communication channels with names $d_1, d_2, ..., d_N$ are activated in sequence. The activation of these channels is conditioned by the fact that the receptions of messages from these channels appear in the beginning of

the sequential composition of actions forming subprocess *CheckAll*. Then, according to Proposition 1, the run (2) involves communications along the channels $d_1, d_2, \ldots, d_N$.

If all the mentioned message exchanges along $d_1, d_2, \ldots, d_N$ include the message receiving actions from the subprocess *CheckAll*, the communication channel r is activated upon their execution. If at least one of the mentioned communications includes an act of message reception from the subprocess *Collect*, this will be possible according to the description of this subprocess only upon a communication along the channel h. However, according to the description of the subprocess *OpenCh*, a message can be transmitted along this channel only upon a communication along the channel r.

Thus the channel r in the run (2) is activated irrespective of the value of $\varphi(\alpha)$. So, according to Proposition 1, an act of communication takes place along the channel r in the run (2). Let us consider the very first of such reductions. It cannot be made via message receptions from the subprocess *Collect*; as indicated above, these actions can be used only upon message exchange along the communication channel h, which can be activated only after at least one act of communication along r. Therefore, the first communication along r in the run (2) involves the reception of a message along this channel from the subprocess *OpenCh*.

According to the description of this subprocess, after the first act of message reception from the channel r is executed that is contained in the subprocess, the channels ch, g, and h are activated in sequence. This is why, according to Proposition (1), the communications in the run (2) take place along the indicated channels. After these reductions all channels $x_1, x_2, \ldots, x_n$ appear reactivated. After the acts of communication are executed along these channels through the acts of message receptions from subprocesses S_ℓ, all the channels correspondent to all CNF φ literals are activated. After the communication along all the activated literal communication channels through the acts of message reception from the subprocesses $Check_{\varphi=0}$ and $Check_{\varphi=1}$, all channels $d_1, d_2, \ldots, d_N$ as well as the channel r appear activated. The communications along these paths are executed through the acts of message reception from the subprocess *Collect* as well as *CheckAll* in case of $\varphi(\alpha) = 0$. The process P_m appeared in the run (2) as a result of these actions contains no actions at all; i.e., $P_m \equiv \mathbf{0}$.

Thus, any run (2) of the process $Proc_\varphi$ is completed in a normal manner.

Then we should note that the only names allowed by process $Proc_\varphi$ for transmission along the channels by r are ch and *secret*. We shall show that a reduction $P_i \xrightarrow{ch(secret)}_\pi P_{i+1}$ will be possible in any run (2) of $Proc_\varphi$ if and only if 3-CNF φ is satisfiable.

The only act of message transmission along the open communication channel ch occurs in the subprocess *OpenCh* structured so that in the (2) the reduction $P_i \xrightarrow{ch(secret)}_\pi P_{i+1}$ shall be preceded by the reduction $P_j \xrightarrow{r(secret)}_\pi P_{j+1}$ and cannot be preceded by any communication with acts from the subprocess *Garbage*.

All the acts of message sending along the channel r appear only in the subprocesses $Check_{\varphi=0}$ and $Check_{\varphi=1}$. However, in $Check_{\varphi=0}$ these actions are meant to send the name ch along the channel r and the name *secret* can be sent along the channel r only in the subprocess *CheckAll*. In this subprocess, such an action is preceded by the reception of messages from the communication channels $d_1, d_2, \ldots, d_N$. Therefore, in the run (2), the reduction $P_j \xrightarrow{r(secret)}_\pi P_{j+1}$ can be made if and only if this action is preceded by the communications along the paths $d_1, d_2, \ldots, d_N$.

The acts of message sending along the indicated channels are contained only in the subprocesses $Check_{D_k=1}$, $1 \le k \le N$. In the description of each of subprocesses $Check_{D_k=1}$ the activation of communication channel d_k is preceded by a communication along one of three channels $\ell_{1k}, \ell_{2k}, \ell_{3k}$. Denote by ℓ_{i_k} the name of the channel ℓ_{1k}, ℓ_{2k}, or ℓ_{3k} the communication along which in the run (2) preceded the activation of the channel d_k.

Let us consider set of singled out literals $L = \{\ell_{i_k} : 1 \le k \le N\}$. According to the description of the subprocesses $Check_{D_k=1}$, $1 \le k \le N$, each clause D_k of CNF φ contains one of the literals of the considered set. In addition, one can see that set L has no complementary pairs. Actually, if L had complementary pair of literals $\ell = x_{i_k}^0$ and $\ell^* = x_{i_k}^1$, that would mean that in the run (2) communication channels ℓ and

ℓ^* would have been activated before the reduction $P_i \xrightarrow{ch(secret)}_\pi P_{i+1}$. According to the description of the subprocesses S_ℓ and S_{ℓ^*}, both indicated channels can be activated only after two communications along the channel x_m are executed. The acts of message sending along the channel x_m occur in subprocesses *Garbage* and *Init*; however, in the section of the run (2), preceding the execution of the reduction $P_i \xrightarrow{ch(secret)}_\pi P_{i+1}$, all the actions of the subprocess *Garbage* are still blocked and the subprocess *Init* has only one act of message sending along the channel x_m. Hence, only one of two channels ℓ and ℓ^* can be activated before the reduction $P_i \xrightarrow{ch(secret)}_\pi P_{i+1}$ in the run (2).

The existence of a noncontradictory set of literals with at least one literal in common with each clause in CNF $Proc_\varphi$ means that CNF φ is satisfiable. Thus, if the reduction $P_i \xrightarrow{ch(secret)}_\pi P_{i+1}$ is executed in some runcalculation of the process $Proc_\varphi$, CNF φ is satisfiable. On the contrary, if CNF φ is satisfiable, it is easy to make the run of $Proc_\varphi$ in which the name *secret* is transmitted along the open communication channel *ch*. Therefore, $Proc_\varphi$ is secure with respect to the threat {*secret*} in carrying out the attack {*ch*} if and only if CNF φ is unsatisfiable. Hence, the unsatisfiability of 3 CNF is *log-space* reducible to checking the security of processes from set $\mathcal{P}$ in the passive adversary model.

CONCLUSION

We should note once again that the result of the co-NP-completeness of checking nonrecursive cryptographic protocol models for security is not an essential novelty. The novelty of the result is that it has been obtained for perhaps the most basic computational model in which it is possible to formulate the task of checking information security properties. Theorem 1 shows that, even in the simplest setting, when there are no cryptographic primitives in protocols being checked and the adversary is passive this problem is intractable.

The passive adversary model and the new understanding of monitoring, which expands the expressive capabilities of mobile process calculi as a means for specifying cryptographic protocols, worth special attention. As far as we know, the adversary and its interactions with the protocol were earlier modeled beyond the limits of the rigid model of π-calculus. We are convinced that the notion of monitoring will allow developing the general active adversary model correspondent to the Dolev–Yao concept. The purpose of our further research is to create this model and obtain for it results on the complexity of checking the resistance of protocols similar to the ones determined in articles [4, 11, 21, 24, 29, 31].

Since the check of nonrecursive π-calculus processes appears to be a difficult problem, it is interesting to find out for which classes of processes this problem is decidable in polynomial time. According to the proof of theorem 1, this problem is closely related to the check of π-calculus processes for normal termination. The latter is a topical problem of checking the behavior of systems of interacting processes for correctness. One of the topics we also consider for further research is that of finding efficiently verifiable sufficient conditions for normal termination of π-calculus processes.

ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research, projects nos. 18-01-00854 517.9 10.18255/1818-1015 Even Simple Processes of Π-Calculus are Hard for Analysis and 16-01-00714

REFERENCES

1. Abadi, M. and Gordon, A.D., A calculus for cryptographic protocols: The spi calculus, *Inf. Comput.,* 1999, vol. 148, no. 1, pp. 1–70.
2. Abadi, M. and Fournet, C., Mobile values, new names, and secure communication, *Proceedings of the 28-th ACM Symposium on Principles of Programming Languages,* 2001, pp. 104–115.
3. Amadio, M.R. and Lugiez, D., On the reachability problem for cryptographic protocols, *Proceedings of the 11-th International Conference on Concurrency Theory,* 2000, pp. 380–394.
4. Amadio, M.R., Lugiez, D., and Vanackere, V., On the symbolic reduction of processes with cryptographic functions, *Theor. Comput. Sci.,* 2003, vol. 290, no. 1, pp. 695–740.
5. Arapinis, M., Liu, J., Ritter, E., and Ryan, M., Stateful applied pi calculus, *Proceedings of the Principles of Security and Trust—Third International Conference,* 2014, pp. 22–41.

6. Blanchet, B. and Smith, B., Automated reasoning for equivalences in the applied pi calculus with barriers, *Proceedings of the 29-th IEEE Computer Security Foundations Symposium,* 2014, pp. 310–324.
7. Bodei, C., Degano, P., Nielson, F., and Nielson, H.R., Static analysis for the pi-calculus with applications to security, *Inf. Comput.,* 2001, vol. 168, no. 1, pp. 68–92.
8. Borgstrom, J. and Nestmann, U., On bisimulations for the spi calculus, *Math. Struct. Comput. Sci.,* 2005, vol. 15, no. 3, pp. 487–552.
9. Bruni, A., Modersheim, S., Nielson, F., and Nielson, H.R., Set-pi: Set membership pi-calculus, *Proceedings of the 28-th IEEE Computer Security Foundations Symposium,* 2015, pp. 185–198.
10. Chadha, R., Cheval, V., Ciobaca, S., and Kremer, S., Automated verification of equivalence properties of cryptographic protocols, *ACM Trans. Comput. Logic,* 2016, vol. 17, no. 4, pp. 1–32.
11. Chevalier, Y., Kusters, R., Rusinowitch, M., and Turuani, M., Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents, *Proceedings of the 23-rd Annual Conference on the Foundations of Software Technology and Theoretical Computer Science,* 2003, pp. 124–135.
12. Chevalier, Y., Kusters, R., Rusinowitch, M., and Turuani, M., An NP decision procedure for protocol insecurity with XOR, *Theor. Comput. Sci.,* 2005, vol. 338, nos. 1–3, pp. 247–274.
13. Chevalier, Y., Kusters, R., Rusinowitch, M., and Turuani, M., Deciding the security of protocols with commuting public key encryption, *Electron. Notes Theor. Comput. Sci.,* 2005, vol. 125, no. 1, pp. 55–66.
14. Chevalier, Y., Kusters, R., Rusinowitch, M., and Turuani, M., Complexity results for security protocols with Diffie-Hellman exponentiation and commuting public key encryption, *ACM Trans. Comput. Logic,* 2008, vol. 9, no. 4, pp. 1–52.
15. Chretien, R., Cortier, V., and Delaune, S., Decidability of trace equivalence for protocols with nonces, *Proceedings of the 28-th IEEE Computer Security Foundations Symposium,* 2015, pp. 170–184.
16. Cortier, V. and Delaune, S., A method for proving observational equivalence, *Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium,* 2009, pp. 266–276.
17. Curti, M., Degano, P., Priami, C., and Balardi, C.T., Modelling biochemical pathways through enhanced pi-calculus, *Theor. Comput. Sci.,* 2004, vol. 325, no. 1, pp. 111–140.
18. Delaune, S., Ryan, M., and Smyth, B., Automatic verification of privacy properties in the applied pi calculus, *Trust Manage. II,* 2008, vol. 263, pp. 263–278.
19. Dolev, D. and Yao, A., On the security of public key protocols, *IEEE Trans. Inf. Theory,* 1983, vol. 29, no. 2, pp. 198–208.
20. Durante, L., Sisto, R., and Valenzano, A., Automatic testing equivalence verification of spi calculus specifications, *ACM Trans. Software Eng. Methodol.,* 2003, vol. 12, no. 2, pp. 222–284.
21. Durgin, N.A., Lincoln, P., and Mitchell, J.C., Multiset rewriting and the complexity of bounded security protocols, *J. Comput. Secur.,* 2004, vol. 12, no. 2, pp. 247–311.
22. Godskesen, J.C., Formal verification of the ARAN protocol using the applied pi-calculus, *Proceedings of the Sixth International IFIP WG 1.7 Workshop on Issues in the Theory of Security,* 2006, pp. 99–113.
23. Huima, A., Efficient infinite state analysis of security protocols, *Proceedings of the Workshop on Formal Methods and Security Protocols,* 1999.
24. Liang, Z. and Verma, R.M., Correcting and improving the NP proof for cryptographic protocol insecurity, *Proceedings of the 5-th International Conference on Information Systems Security,* 2009, pp. 101–116.
25. Milner, R., Parrow, J., and Walker, D., A calculus of mobile processes, I and II, *Inf. Comput.,* 1992, vol. 100, no. 1, pp. 1–77.
26. Milner, R., Functions as processes, *Math. Struct. Comput. Sci.,* 1992, vol. 2, pp. 119–141.
27. Milner, R., *Communicating and Mobile Systems—The Pi-Calculus,* MIT Press, 1999.
28. Regev, A., Representation and simulation of biochemical processes using the pi-calculus process algebra, *Proceedings of the 6-th Pacific Symposium on Biocomputing,* 2001, pp. 459–470.
29. Rusinowitch, M. and Turuani, M., Protocol insecurity with finite number of sessions is NP-complete, *Theor. Comput. Sci.,* 2003, vol. 299, nos. 1–3, pp. 451–475.
30. Smith, H. and Fingar, P., *Business Process Management: The Third Wave,* Meghan-Kiffer Press Tampa, 2003.
31. Tiplea, F.L., Enea, C., and Birjoveanu, C.V., Decidability and complexity results for security protocols, in *Verification of Infinite-State Systems with Applications to Security,* Amsterdam: IOS Press, 2006, pp. 185–211.
32. Tiu, A. and Dawson, J., Automating open bisimulation checking for the spi calculus, *Proceedings of the 23rd IEEE Computer Security Foundations Symposium,* 2010, pp. 307–321.
33. Walker, D., Objects in the π-calculus, *Inf. Comput.,* 1995, vol. 116, no. 4, pp. 253–271.

Translated by S. Kuznetsov

SPELL: 1. OK