Verification of Multi-Party Ping-Pong Protocols via Program Transformation*

Antonina Nepeivoda

Program Systems Institute, Pereslavl-Zalessky, Russia
a_nevod@mail.ru

Abstract

The paper describes a verification technique based on program transformation with unfolding. The technique allows to find short attacks on multi-party ping-pong protocols in the Dolev–Yao intruder model. Protocols are modelled by prefix grammars, and questions of model optimization and complexity are considered. Examples of model programs for protocols were written in a functional language and were analyzed by a general-purpose supercompiler.

1 Introduction

It is known that, even in the case when algorithms of message encryption themselves are considered as completely secure, some existing cryptographic protocols that use them may be insecure. Ideed, some vulnerabilities of the protocols are due to the fact that an intruder has access to the communication channel used by the legal participants of the interaction. Although the problem of automatic verification of such interactions is undecidable in general [1], some models of protocols have been described for which the verification task has a decision procedure. In particular, Dolev and Yao presented a ping-pong model of cryptographic protocols [4]. In a ping-pong protocol a message is a single data item encrypted by a sequence of keys. Two principals can apply a finite number of operations to the message. Dolev, Yao, and Karp showed that the safety of this this protocol can be verified in the case that an intruder can listen tot the channel and can put messages on the channel at every phase of the interaction [3]. The verification procedure was generalized for multi-party ping-pong protocols in the paper [5].

In our previous work [9] we showed a way to construct Dolev–Yao models of ping-pong protocols via the language of prefix grammars and described how to verify the models by a general purpose program transformation technique with unfolding. The methods presented in [9] can be also applied to the case of Dolev–Yao models with more than two legal parties but the computational complexity of the verification process grows exponentially with the number of parties [5]. In this paper we generalize the results from the paper [9] and show some methods to decrease the complexity of the verification process.

Our contributions are the following:

1. We refine the method described in the paper [9] in such a way that the notion of time indexing becomes unnecessary in the model.
2. We show how to model multi-party ping-pong protocols by prefix grammars and how to represent the prefix grammars by programs in a functional language. Unfolding a semantic tree of these program models constructs a set of all short attack models. We discuss some ways to decrease the computational complexity of the unfolding process.

*The reported study was partially supported by RFBR, research project No. 14-07-00133 a, and by Program No. 16 for Basic Research of Presidium of Russian Academy of Sciences.

3. We show that a short attack length has an exponential upper bound over the number of rules in the corresponding prefix grammar.

The examples presented in the paper were tested by the supercompiler SCP4 [8].

2 Multi-Party Ping-Pong Protocols

A cryptographic protocol is a set of rules that determine the behavior of the parties of a message exchange in a network (for instance, the message exchange may be an authentication process, or a transmission of secure data). Let us describe a model of a cryptographic protocol formally.

Consider a data exchange process between several participants via a common network. Transmitted messages are represented by strings in a finite alphabet; the set of the strings is denoted by $\mathbb{S}$. The set of all users of the network is denoted by U. Let Σ_A be a vocabulary of some participant A — i.e. the set of functions from $\mathbb{S} \to \mathbb{S}$ that are available to a participant A; let Λ be the identity operator (that has the meaning of doing nothing); and let the set of all possible users of the network be denoted by U. A composition of two functions $f_A \in \Sigma_A$ and $g_B \in \Sigma_B$ is denoted by $f_A g_B$ (note that this means g_B is applied before f_A). Let x be a variable from U; a var-operator f_x from $U \times \mathbb{S} \to \mathbb{S}$ is a function form which becomes f_A if x takes some A as a value and $f_A \in \Sigma_A$.

Definition 1. *A var-operator f_x consists of an operator form and a variable $x \in U$. The set of all var-operators is denoted F.*

Some standard examples of operators from Σ_x are: E_x — an encryption of a message by a public key of x; D_x — a decryption of a message by a key of x. D_x is inverse to E_x (that can be expressed as a conjunction of $D_x E_x = \Lambda$ and $E_x D_x = \Lambda$). a_x denotes concatenation of the name of a user x to a message, d_x denotes deletion of the name of x from a message; d_x is a left inverse to a_x (such that $d_x a_x = \Lambda$). It is assumed that a user can apply the encryption key of every other user and can add and remove the name of every other user, but can apply only his own decryption key, more formally $\forall x \in U \forall y \in U(a_x \in \Sigma_y \,\&\, d_x \in \Sigma_y \,\&\, E_x \in \Sigma_y)$ and $\forall x \in U(D_x \in \Sigma_x)$ but $\forall y \in U(y \neq x \Rightarrow D_x \notin \Sigma_y)$.

There may be other operators in Σ_x. In the original work [5] the set of operator forms is restricted to $\{E_x, D_x, a_x, d_x, f_x, g_x\}$ where f_x and g_x are permutation operators with the following properties: $f_x g_x = g_x f_x = \Lambda$ and $\forall x, y(f_x \in \Sigma_y \,\&\, g_x \in \Sigma_y)$. In accordance to the terminology of [5], we call an operator algebra generated by Σ_x an algebra with a freeness assumption if elements of Σ_x satisfy no word equations other than equations of the form $A_1 A_2 = \Lambda$. In this case, A_1 also is denoted as A_2^{-1} (and vice versa). The freeness assumption is an important stipulation which facilitates the construction of a model of a protocol. In the paper below we do not take the freeness assumption for granted but explicitly underline all the cases where this assumption allows to construct a simpler model.

Some operators from Σ_x, such as an encryption, a decryption, an appending of a single letter, etc, are elementary; actions that correspond to these operators are not decomposable, and the operators are denoted by single letters. Other operators from Σ_x can be presented as compositions of elementary operators that are not present in Σ_x. This can happen, for example, if a participant of an interaction is a user of some specific cryptographic program, and this program does not allow the user to apply the encryption algorithm without adding his/her personal information to a message to be encrypted. The composite actions from Σ_x are represented as sequences of letters denoting corresponding elementary operators.

Definition 2. *A p-party ping-pong protocol $P[x_1, \ldots, x_p]$ (over F) is a sequence of pairs $((y_1, \alpha_1[x_1, \ldots, x_p]), (y_2, \alpha_2[x_1, \ldots, x_p]), \ldots, (y_l, \alpha_l[x_1, \ldots, x_p]))$ where y_i is a variable with the range of values U and $\alpha_i[x_1, \ldots, x_p]$ is a sequence of var-operators in the vocabulary of y_i.*

This definition can be explained as follows. Let several participants $U_1, U_2, \ldots, U_n$ exchange some data using an open network. Let us denote an initial secure data item to be transmitted as M. If U_1, U_2, $\ldots$, U_n use a multi-party protocol $P[x_1, \ldots, x_p]$ (where $p \leq n$) then they organize their interaction as follows.

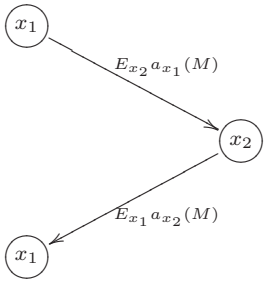
U_1 starts a transmission by sending U_2 the message $\alpha_1(M)$ where $\alpha_1 \in \Sigma^*_{U_1}$. Then U_2 sends the message $\alpha_2\alpha_1(M)$ (where $\alpha_2 \in \Sigma^*_{U_2}$) to U_3 and so on until the transformed message reaches U_p. After that, U_p applies the sequence of operators α_p ($\alpha_p \in \Sigma^*_{U_p}$) to $\alpha_{p-1} \ldots \alpha_1(M)$ and transmits the result to all of the users U_1, U_2, $\ldots$, U_n.

Example 1. *Consider the following 2-party ping-pong protocol, which describes a safe protocol from [4] in the terms of Definition 2*

$$P_2[x_1, x_2] = ((x_1, E_{x_2} a_{x_1}), (x_2, E_{x_1} a_{x_2} d_{x_1} D_{x_2}))$$

Recall that $d_x a_x = \Lambda$ and $D_x E_x = E_x D_x = \Lambda$. So x_2 first cancels the operations applied by x_1 to the initial message and then applies $E_{x_1} a_{x_2}$ to the result.

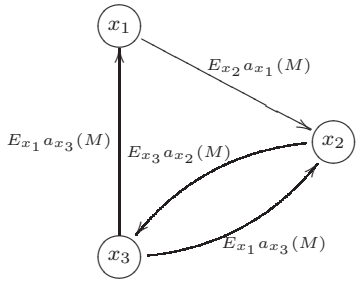
Informally $P_2[x_1, x_2]$ may be presented as the following graph, where figures in the circles denote user variables and arrows are marked by strings of var-operators that are applied to M (var-operators which are canceled are not written):



Now consider a 3-party analogue of the protocol $P_2[x_1, x_2]$.

$$P_3[x_1, x_2, x_3] = ((x_1, E_{x_2} a_{x_1}), (x_2, E_{x_3} a_{x_2} d_{x_1} D_{x_2}), (x_3, E_{x_1} a_{x_3} d_{x_2} D_{x_3}))$$

Informally $P_3[x_1, x_2, x_3]$ can be presented as follows:



Is $P_3[x_1, x_2, x_3]$ as safe as $P_2[x_1, x_2]$? To answer this question we need to formalize an intruder's behavior in the case of multi-party message exchange.

To formalize the notion of an attack on a protocol we must take into account the fact that the protocol can be played many times in parallel by the same set of users (or sets of users with a non-empty intersection). So every user playing a protocol is associated by the two entities: his/her unique identifier, which is constant for all interactions, and a temporary "party role", which is assigned in accordance with the protocol and restricts actions of the user as the protocol party.

Definition 3. *Let $u = (U_1, U_2, \ldots, U_p)$ be a sequence of p elements in U, and $P[x]$ be a p-party ping-pong protocol. A u-instance of the protocol $P[x_1, \ldots, x_p]$, denoted $P[U_1, \ldots, U_p]$, is the result of substituting the variable x_j in $P[x_1, \ldots, x_p]$ by the user U_j for every $1 \leq j \leq p$. The operator word $\alpha_j[U_1, \ldots, U_p]$ is called a u-instance of the protocol word $\alpha_j[x_1, \ldots, x_p]$.*

A u-instance of a p-party ping-pong protocol is called proper if u consists of p distinct elements in U.

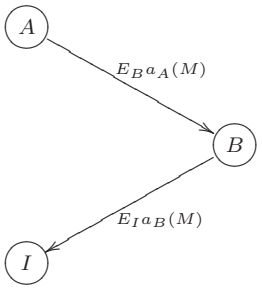
Definition 4. *Let $P[x_1, \ldots, x_p]$ be a p-party protocol consisting of the words $\alpha_1[x_1, \ldots, x_p], \alpha_2[x_1, \ldots, x_p], \ldots, \alpha_l[x_1, \ldots, x_p]$. Let $u = (U_1, U_2, \ldots, U_p)$ be an arbitrary sequence of p distinct users and U' be the set of these users.*

For every J, $J \subseteq U$, let Σ_J denote the union of the vocabularies of users in J; that is, $\Sigma_J = \bigcup_{j \in J} \Sigma_j$.

For every J, $J \subseteq U$, let $INST(P, J)$ denote the set of all proper instances of protocol words of P in which the users are from J.

Protocol $P[x_1, \ldots, x_p]$ is strongly insecure in the Dolev–Yao intruder model if there exists a set $S \subseteq U \setminus U'$ and an operator string $\xi \in (\Sigma_S \cup INST(P, U))^*$ such that $\xi\alpha_1[U_1, \ldots, U_p] \equiv \Lambda$.

Example 2. *$P_2[x_1, x_2]$ is safe in the Dolev–Yao intruder model, and it may look plausible that $P_3[x_1, x_2, x_3]$ is also safe. But consider the following attack scheme (the index I stands for an intruder, A and B stand for the principals). The intruder plays the role of the third participant x_3, and B encrypts the message by the key $E_{x_3} = E_I$ (not knowing that x_3 is an intruder).*



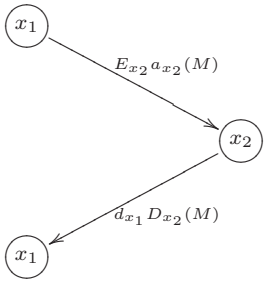
The intruder's vocabulary contains d_B and D_I, and I is able to construct Λ from $E_I a_B$. $P_3[x_1, x_2, x_3]$ appears to be strongly insecure.

The authors of [5] also define a notion of weak insecurity. The weak insecurity admits not only proper instances in an attack but all instances (thus, an instance can contain substitutions of a single user to different roles in the single var-operator string $\alpha_i[x]$). If a protocol is insecure in the weak sense this does not imply that the protocol is insecure in the strong sense. This is shown in Example 3 below from the paper [5].

Example 3. *Consider the following 2-party protocol*

$$P_{WI}[x_1, x_2] = ((x_1, E_{x_2} a_{x_2}), (x_2, d_{x_1} D_{x_2}))$$

which can be represented by the following diagram



This protocol is secure in the strong sense since the value of the var-operator $d_{x_1} a_{x_2}$ is an error. But $P_{WI}[x_1, x_2]$ is insecure in the weak sense: if the user substituted in x_2 coincides with the user substituted in x_1 in $d_{x_1} D_{x_2}$ then the result of $\alpha_2[U_2, U_2]\alpha_1[U_1, U_2]$ is Λ.

The authors point out that the notion of the weak insecurity is unnatural as opposed to the notion of the strong insecurity [5]. Surely, Example 3 gives us an example of an "unnatural" attack: the second party of the protocol, who is meant to be a principal, substitutes themselves to x_1 (or this attack can be considered as "sending a message from one to himself").

If we narrow the definition of weak attack to the one that admits only instances that:

1. do not substitute an active principal to any other position, and
2. do not substitute any other party to the position of an active principal

the notion of a weak attack becomes more relevant to practical needs. In particular, this weak model of an attack coincides with the model of an attack on a classical 2-party ping-pong protocol. Like the generic notion of a weak attack, this weak model allows to reduce the set of intruders to a singleton (and the reduction allows to construct models of protocols with significantly less number of rules). And this notion is still strictly wider than the notion of a strong attack.

Example 4. *Consider the multi-party ping-pong protocol $P_D[x_1, x_2, x_3, x_4, x_5]$ with the following var-operator sequences.*

$$\alpha_1 = E_{x_2} a_{x_3} a_{x_4} a_{x_4} a_{x_1} D_{x_1}$$

$$\alpha_2 = E_{x_3} a_{x_2} a_{x_4} a_{x_5} a_{x_1} E_{x_1} d_{x_1} d_{x_5} d_{x_4} d_{x_3} D_{x_2}$$

$$\alpha_3 = E_{x_1} d_{x_1} d_{x_5} d_{x_4} d_{x_2} D_{x_3}$$

In the strong attack model $P_D[x_1, x_2, x_3, x_4, x_5]$ is safe but if $u^1 = (U_1, U_2, U_3, U_4, U_5)$ and $u^2 = (U_1, U_3, U_2, U_4, U_5)$ then $\alpha_3[u^2]\alpha_1[u^1] = \Lambda$. This is an attack on $P_D[x_1, x_2, x_3, x_4, x_5]$ in the weak attack model with no self-substitutions of an active user.

In the sequel we use the term "a weak attack model" in the restricted sense described above. An attack model plays a significant role in the algorithm of modelling a multi-party protocol by a prefix grammar.

In can be noticed that in some cryptographic protocols the initial data item M is not the only secure information. For example, in the Needham–Shroeder protocol $P_{NS}[x_1, x_2]$ (which also can be verified by supercompilation [2]) the fact of the intruder receiving the initial random number generated by x_1 does not point to an attack on the protocol. In $P_{NS}[x_1, x_2]$, an attack is a sequence of transmissions that allows an intruder to get the random number generated

by x_2 in response to x_1. So there appears a natural generalization of an attack on a ping-pong protocol: having a protocol $P[x_1, \ldots, x_p]$, an intruder I and an insecurity set $INSEC$, which consists of operator words from $\{\Sigma_{x_1} \cup \Sigma_{x_2} \cup \cdots \cup \Sigma_{x_p}\}^*$, a generalized weak attack on $P[x_1, \ldots, x_p]$ exists if there exist $\xi \in (\Sigma_I \cup INST(P, I \cup U))^*$ and $\beta \in INSEC$ such that $\xi\alpha_1[U_1, \ldots, U_p] \equiv \beta[U_1, \ldots, U_p]$.

3 Prefix Grammars and Empty Word Problem Solution

It was shown in [10] how to model a two-party ping-pong protocol by a prefix grammar if we want to verify safety of the protocol by program transformation. The case of multi-party ping-pong protocols can be handled in a similar way. We now briefly introduce the notion of a prefix grammar and informally show how a protocol can be modelled by a prefix grammar.

Let us denote letters of an alphabet Υ by the small Latin letters $a, b, c, \ldots, p, q, r$ and the capital Latin letters $A, B, C, D, E, F, S, T, U$ (maybe with subscripts or superscripts), variables by x, y, z, w; and let us denote words from Υ^* by the Greek capitals $\Gamma, \Delta, \Phi, \Psi, \Theta$.

Definition 5. *Consider a tuple $\langle \Upsilon, \mathbf{R}, \Gamma_0 \rangle$, where Υ is an alphabet, $\Gamma_0 \in \Upsilon^*$ is an initial word and $\mathbf{R} \subset \Upsilon^* \times \Upsilon^*$ is a set of rewrite rules. If the rewrite rules are applied only to word prefixes $\dfrac{R : \Phi \longrightarrow \Psi}{\Phi\Theta \xrightarrow{R} \Psi\Theta}$ then the tuple $\langle \Sigma, \mathbf{R}, \Gamma_0 \rangle$ is a prefix rewriting grammar.*

We call a trace of a prefix rewriting grammar $\mathbf{G} = \langle \Upsilon, \mathbf{R}, \Gamma_0 \rangle$ a sequence $\{\Phi_i\}$ (finite or infinite) s.t. $\Phi_1 = \Gamma_0$ and $\forall i \exists R (R : R_l \to R_r \ \& \ R \in \mathbf{R} \ \& \ \Phi_i = R_l\Theta \ \& \ \Phi_{i+1} = R_r\Theta)$.

Example 5. *Consider the following 3-party protocol $P_{3a}[x_1, x_2, x_3]$, which is a slight modification of the insecure protocol from Example 1.*

$\alpha_1[x_1, x_2, x_3] = E_{x_2} a_{x_3} a_{x_1}$
$\alpha_2[x_1, x_2, x_3] = E_{x_3} a_{x_1} d_{x_1} d_{x_3} D_{x_2}$
$\alpha_3[x_1, x_2, x_3] = E_{x_1} d_{x_1} D_{x_3}$

The set of insecure operators $INSEC = \{\Lambda\}$.

Let us try to model some rules of this protocol (and the intruder behavior) by a prefix grammar. In the weak attack model we can use $\{A, B, C, I\}$ as the set U of all network users without loss of generality (the strong attack model requires not one but at least seven intruders [5]). We place Λ in the left-hand side of a rule if the rule is applicable to any word and does not change letters of the word to which it is applied.

First we fix some initial instance $P_{3a}[A, B, C]$ and substitute it into the protocol words. Then the initial word becomes $E_B a_C a_A$. The rewrite rules that corresponds to a legal interaction look like

$R_1 : E_B a_C a_A \to E_C a_A$
$R_2 : E_C a_A \to E_A$

If the participants of the protocol initiate another interaction with another party distribution then there appear more rules such that

$R_3 : E_A a_C a_B \to E_C a_B$
$R_4 : E_C a_I a_I \to E_I a_I$

and, generally, all the rules that are generated by instances containing substitutions of the set $\{A, B, C, I\}$ in the positions x_1, x_2, and x_3 (excluding ones where an active participant substitutes themselves, e.g. $E_C a_C a_A \to E_C a_A$).

The only word that corresponds to an instance of an insecure operator is Λ. To model an intruder behavior we also need rules that append (and cancel) operators from Σ_I. The rules of the sort look as $R_5 : E_I \to \Lambda$, $R_6 : \Lambda \to E_I$, and so on.

When a prefix grammar model of a protocol is built the question of verification of this protocol is easily formulated. Namely, to verify the prefix grammar model of the protocol in the Dolev–Yao intruder model is to find out whether a trace starting from the initial word and ending by some Φ that models an insecure operator from $INSEC$ exists (the similar proposition for the equivalent models by finite automata is considered in [4]). If $INSEC = \{\Lambda\}$, these traces represent sequences of transmissions corresponding to the protocol that allow an intruder to get an initial message M transmitted by the first party of the protocol. Generally, if there exists at least one such trace then there is an infinite set of such traces but most of them are uninteresting from the practical point of view. For instance, in Example 5 we can apply the rules R_6 and R_5 consequently to any word without any effect on the word to which they are applied.

Definition 6. *An attack model is a trace generated by a prefix grammar that ends with a word from a chosen set $INSW$.*

A short attack model is a trace starting with Γ_0 and ending with a word Φ from $INSW$, such that no subsequence of the rule sequence generating Φ from Γ_0 also generates Φ from Γ_0.

So to verify a protocol model in a prefix grammar $\mathbf{G}$ it is enough to find out whether words from $INSW$ belong to the language generated by $\mathbf{G}$.

If $INSW$ is finite then the set of short attack models is finite (this is an auxiliary consequence of Proposition 2). Our algorithm aims on constructing this set. The algorithm is a part of the supercompilation technique and is based on the following observations.

Definition 7. *A prefix rewriting grammar $\mathbf{G}$ is called annotated if every pair of rules either have the same right-hand side or have no letters shared by their right-hand sides.*

Definition 8. *Let $\mathbf{G}$ be an annotated prefix grammar with an alphabet Υ. Let us say that $\Gamma \in \Upsilon^*$ is redundant iff for some $a \in \Upsilon$ the number of occurrences of a in Γ is greater than the number of a occurrences in the left-hand sides of rewrite rules of the grammar $\mathbf{G}$. For every $a \in \Upsilon$ the number of occurrences of a in the left-hand sides is called **an erasing limit of** a (denoted by $EL(a)$).*

Example 6. *Let us consider $\mathbf{G_{2EXP}} = \langle\{a, b, c, A, B, C, e\}, \mathbf{R_{2EXP}}, e\rangle$ with the following $\mathbf{R_{2EXP}}$:*

$R^{[1]} : e \to aA$ $R^{[5]} : AA \to \Lambda$ $R^{[9]} : Ba \to bB$
$R^{[2]} : \Lambda \to aA$ $R^{[6]} : BB \to \Lambda$ $R^{[10]} : Cb \to cC$
$R^{[3]} : \Lambda \to bB$ $R^{[7]} : CC \to \Lambda$
$R^{[4]} : \Lambda \to cC$ $R^{[8]} : c \to \Lambda$

The grammar is annotated since every two right-hand sides either coincide or share no letters. The erasing limit $EL(e) = 1$; also $EL(a) = EL(b) = EL(c) = 1$, so the words $aAaA$ and $cCcC$ are redundant. The word cCC is not redundant since $EL(C) = 3$.

Proposition 1. *Let $\mathbf{G}$ be a finite annotated prefix grammar. Every infinite trace generated by $\mathbf{G}$ either contains some Γ and Δ such that $\Gamma = \Delta$, or contains a redundant word.*

Proof. In an infinite trace a word length is either bounded by some N or grows infinitely. In the first case in the trace there are two words Γ and Δ such that $\Gamma = \Delta$. In the second case some word Γ reaches the length $|\Upsilon| * M + 1$, where M denotes maximal number of occurrences of a same letter in the left-hand sides $R_l^{[i]}$ of rewrite rules. Then some letter in Γ must have more occurrences than its erasing limit.

□

Proposition 2. *Let* $\mathbf{G}$ *be an arbitrary finite annotated prefix grammar. All short attack models generated by* $\mathbf{G}$ *contain no* Γ *and* Δ *such that* $\Gamma = \Delta$ *or* Γ *is redundant.*

Proof. The case when there is a pair of two equal words in a trace is obvious.

Let some short attack model contain a redundant Γ. Let a be a letter that is to be erased at least twice by the same rule. Consider the words where a is erased by a same rule. They look as $\hat{R}_l a\Psi a\Theta_0$ and $\hat{R}_l a\Theta_0$, where $\hat{R}_l$ denotes a prefix of the left-hand side of some rule R and Θ_0 is never modified on the segment from $\hat{R}_l a\Psi a\Theta_0$ to $\hat{R}_l a\Theta_0$. Now consider the words in which a were generated. They look as $\hat{R}'_r a\Theta_0$ and $\hat{R}'_r a\Psi a\Theta_0$ respectively where $\hat{R}'_r$ denotes a prefix of the right-hand side of some rule R' and Θ_0 is the same as in the previous case.

All the considered words together form the following sequence.

$\ldots$
$\hat{R}'_r a\Theta_0$
$\ldots$
$\hat{R}'_r a\Psi a\Theta_0$
$\ldots$
$\hat{R}_l a\Psi a\Theta_0$
$\ldots$
$\hat{R}_l a\Theta_0$
$\ldots$
Φ

We now can transform $\hat{R}'_r a\Theta_0$ to $\hat{R}_l a\Theta_0$ by the rule sequence that transforms $\hat{R}'_r a\Psi a\Theta_0$ to $\hat{R}_l a\Psi\Theta_0$ and get a shorter attack model.

□

Proposition 2 together with Proposition 1 give a sound criterion of terminating a trace unfolding if we want to find all the paths that correspond to short attack models.

But to apply these propositions, we must precisely formulate how to build a sound model of a ping-pong protocol in the terms of a prefix grammar. The example below shows some difficulties on this way.

Example 7. *Let* $T_{\langle x,y\rangle}$ *be a secret key that is known only to* x *and* y, $U_{\langle x,y\rangle}$ *be a universal secret decryption algorithm for* $T_{\langle x,y\rangle}$ *that is known only to* x, $T_{\langle x,y\rangle}U_{\langle x,y\rangle} = \Lambda$, *and* $U_{\langle x,y\rangle}T_{\langle x,y\rangle} = \Lambda$, $E_x U_{\langle x,y\rangle} = \Lambda$, $U_{\langle x,y\rangle}E_x = \Lambda$.

Consider the ping-pong protocol $P_{SEC}[x_1, x_2]$ *with the following protocol words:*

$\alpha_1[x_1, x_2] = E_{x_2}T_{\langle x_2,x_1\rangle}$
$\alpha_2[x_1, x_2] = E_{x_1}a_{x_2}U_{\langle x_2,x_1\rangle}U_{\langle x_2,x_1\rangle}$
$\alpha_3[x_1, x_2] = T_{\langle x_2,x_1\rangle}d_{x_2}D_{x_1}$

$INSEC = \{\Lambda\}$. *The meaning of this protocol is the following.* x_1 *wants to transmit some secret information to the computer database* x_2 *(which possesses* $U_{\langle x_2,x_1\rangle}$*) and sends an initial* M *to* x_2*, applying both public and secret keys.* x_2 *applies the universal decryption key twice, stores the message* M*, and returns it to* x_1*, signing it by* x_2*'s name and using the public key* E_{x_1}*. After that,* x_1 *automatically sends* x_2 *a confirmation about the successful interaction.*

Let us construct a prefix grammar corresponding to the legal actions of the protocol and an intruder's behavior. The legal participants are denoted by A *and* B*. An intruder is* I*.*

$\Gamma_0 = E_B T_{\langle B,A\rangle}$. *Some rules of the grammar are straightforward:*

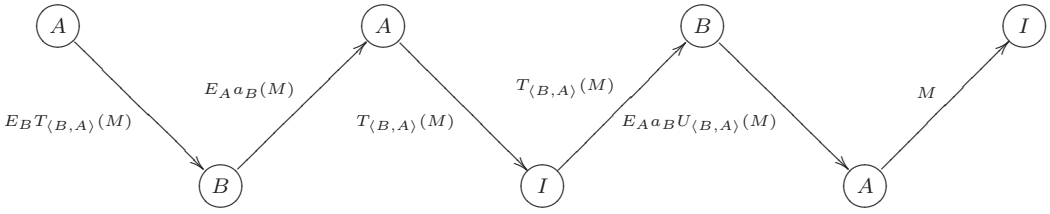
$R_1 : E_B T_{\langle B,A\rangle} \to E_A a_B$
$R_2 : E_A a_B \to T_{\langle B,A\rangle}$

Some more rules describe an intruder's possible interactions with the principals:

$R_{I1} : E_B T_{\langle B,I\rangle} \to E_I a_B$
$R_{I2} : E_A a_I \to T_{\langle I,A\rangle}$

And some rules correspond to the actions of the intruder that correspond to their operator alphabet $\Sigma_I = \{E_A, E_B, E_I, a_A, a_B, a_I, T_{\langle I,A\rangle}, T_{\langle A,I\rangle}, T_{\langle I,B\rangle}, T_{\langle B,I\rangle}, D_I, d_A, d_B, d_I, U_{\langle I,B\rangle}, U_{\langle I,A\rangle}\}$ *(for every operator* $f \in \Sigma_I$ *there is a rule* $\Lambda \to f$ *if either* f *is not a left inverse (e.g.* f *is* a_x*) or* f *is a full inverse for some* g *(e.g.* f *is* E_x*, or* D_x*, or* T*, or* U*), and* $g \to \Lambda$ *if* f *is a left inverse and* $fg \to \Lambda$ *(e.g.* f *is* d_x*)).*

Are there enough rules? The language generated by this grammar does not contain Λ*. But consider the following interaction:*



An intruder tackles $T_{\langle B,A\rangle}(M)$ *and sends it to the database, where* $U_{\langle B,A\rangle}(M)$ *(which looks like a nonsense) is stored. Then the database sends a confirmation to A, and A automatically replies by the initial message M which corresponds to* Λ*.*

So the model prefix grammar is unsound with respect to the protocol $P_{SEC}[x_1, x_2]$*.*

In Section 4 we describe how to construct consistent models of the protocols by prefix grammars. In Section 5 we briefly describe how to represent prefix grammar models by programs in a functional language in a way that makes the verification process more efficient.

4 Modeling Ping-Pong Protocols by Prefix Grammars

Given a protocol P, let every instance of an elementary operator be a letter in the initial alphabet of the model prefix grammar. It is shown in [9] that every word $c_1c_2c_3\ldots c_n$ corresponding to an operator string from $INST(P,U)^*$ can be modeled by the set of rewrite rules corresponding to representations of all possible cancellations caused by an application of $c_1c_2c_3\ldots c_n$.

$$
\begin{array}{lcl}
R^{[1]} : \Lambda & \to & c_1c_2\ldots c_n \\
R^{[2]} : c_n^{-1} & \to & c_1c_2\ldots c_{n-1} \\
\ldots & & \ldots \\
R^{[n-1]} : c_n^{-1}\ldots c_2^{-1} & \to & c_1 \\
R^{[n]} : c_n^{-1}c_{n-1}^{-1}\ldots c_1^{-1} & \to & \Lambda
\end{array}
$$

In the case of multi-party ping-pong protocols, the described modeling algorithm cannot be applied as is, since the cardinality of the set $INST(P, S \cup U)$ grows exponentially over the number of parties. In particular, the model does not take into account the Church–Rosser property of the ping-pong protocols over the var-operator algebra with the freeness assumption: in the model presented in [9] an application of a single operator string corresponds to a set consisting of numerous rules, most of which are redundant. E.g. if Γ_0 is $c_n^{-1}c_{n-1}^{-1}\ldots c_2^{-1}$ and the operator word $c_1c_2\ldots c_n$ is applied to Λ then only an application of R_{n-1} can appear in a

short attack model and all trace segments ending with Λ derived from the applications of R_1 – R_{n-2} will contain redundant rule applications.

There are several possible ways to take the Church–Rosser property into account. One of them is the following. If the var-operator algebra contains operators with no left inverses (e.g. d_x) then in every short attack model applications of the operators correspond to cancellations. E.g. if $\alpha_k = a_1 a_2 \dots d_i a_{i+1} \dots a_{n_k}$ and $d_i a_i = \Lambda$ but $a_i d_i \neq \Lambda$ then in a short attack model α_k can only be applied to words with the prefix $a_{n_k}^{-1} \dots a_{i+1}^{-1} a_i$ (for the proof see [5]). If we take this observation into account the number of rules in the model can be noticeably decreased.

Let us illustrate the above idea and model $P_{3a}[x_1, x_2, x_3]$ from Example 5 by a prefix grammar.

Example 8. *The initial non-optimized prefix grammar that models $P_{3a}[x_1, x_2, x_3]$ is*

$\Gamma_0 = E_B a_C a_A$

$R_{1-1} : \Lambda \to E_C a_A d_A d_C D_B$ $R_{1-2} : \Lambda \to E_B a_C d_C d_B D_A$ $\dots$ $R_{1-24} : \Lambda \to E_A a_I d_I d_A D_C$
$R_{2-1} : E_B \to E_C a_A d_A d_C$ $R_{2-2} : E_A \to E_B a_C d_C d_B$ $\dots$ $R_{2-24} : E_C \to E_A a_I d_I d_A$
$\dots$
$R_{4-1} : E_B a_C a_A \to E_C a_A$ $R_{4-2} : E_A a_B a_C \to E_B a_C$ $\dots$ $R_{4-24} : E_C a_A a_I \to E_A a_I$
$R_{5-1} : \Lambda \to E_A d_A D_C$ $R_{5-2} : \Lambda \to E_C d_C D_B$ $\dots$ $R_{5-12} : \Lambda \to E_I d_I D_A$
$R_{6-1} : E_C \to E_A d_A$ $R_{6-2} : E_B \to E_C d_C$ $\dots$ $R_{6-12} : E_A \to E_I d_I$
$\dots$
$R_{8-1} : E_C a_A D_A \to \Lambda$ $R_{8-2} : E_B a_C D_C \to \Lambda$ $\dots$ $R_{8-12} : E_A a_I D_I \to \Lambda$
$R_{9-1} : \Lambda \to E_A$ $R_{9-2} : \Lambda \to E_B$ $R_{9-3} : \Lambda \to E_C$ $R_{9-4} : \Lambda \to E_I$
$R_{10-1} : \Lambda \to a_A$ $R_{10-2} : \Lambda \to a_B$ $R_{10-3} : \Lambda \to a_C$ $R_{10-4} : \Lambda \to a_I$
$R_{11-1} : \Lambda \to d_A$ $R_{11-2} : \Lambda \to d_B$ $R_{11-3} : \Lambda \to d_C$ $R_{11-4} : \Lambda \to d_I$
$R_{12-1} : D_A \to \Lambda$ $R_{12-2} : D_B \to \Lambda$ $R_{12-3} : D_C \to \Lambda$ $R_{12-4} : D_I \to \Lambda$
$R_{13-1} : a_A \to \Lambda$ $R_{13-2} : a_B \to \Lambda$ $R_{13-3} : a_C \to \Lambda$ $R_{13-4} : a_I \to \Lambda$
$R_{14} : \Lambda \to D_I$ $R_{15} : E_I \to \Lambda$

If the non-existence of left inverses of d_x is taken into account, the rewrite rules R_{1-1} – R_{1-24}, R_{2-1} – R_{2-24}, R_{3-1} – R_{3-24}, R_{5-1} – R_{5-12}, R_{6-1} – R_{6-12}, and R_{11-1} – R_{11-4} disappear. All the rules from the blocks R_{4-1} – R_{4-24}, R_{7-1} – R_{7-12}, R_{8-1} – R_{8-12} with E_I in the left-hand side or D_I in the right-hand side are also redundant since their applications are the compositions of the applications of the rules from the blocks R_9–R_{13} and the rules R_{14}–R_{15}.

If the participants A, B, C cannot play any role in the protocol then the number of rules in the grammar also decreases. For instance, if A is a user who only can initiate an interaction but cannot reply, and B and C cannot initiate (but can reply), then the model grammar $\mathbf{G_{3a}}$ *for $P_{3a}[x_1, x_2, x_3]$ becomes as follows:*

$\Gamma_0 = E_B a_C a_A$

$R_{1a} : E_B a_C a_A \to E_C a_A$ $R_{1b} : E_B a_A a_C \to E_A a_C$ $R_{1c} : E_B a_I a_A \to E_I a_A$ $R_{1d} : E_B a_A a_I \to E_A a_I$
$R_{1e} : E_B a_C a_I \to E_C a_I$ $R_{1f} : E_B a_I a_C \to E_I a_C$ $R_{1g} : E_C a_B a_A \to E_B a_A$ $R_{1h} : E_C a_A a_B \to E_A a_B$
$R_{1i} : E_C a_I a_A \to E_I a_A$ $R_{1j} : E_C a_A a_I \to E_A a_I$ $R_{1k} : E_C a_I a_B \to E_I a_B$ $R_{1l} : E_C a_B a_I \to E_B a_I$
$R_{2a} : E_C a_A D_A \to \Lambda$ $R_{2b} : E_C a_I D_I \to \Lambda$ $R_{2c} : E_C a_B D_B \to \Lambda$ $R_{2d} : E_B a_A D_A \to \Lambda$
$R_{2e} : E_B a_C D_C \to \Lambda$ $R_{2f} : E_B a_I D_I \to \Lambda$
$R_{3a} : E_C a_A \to E_A$ $R_{3b} : E_C a_I \to E_I$ $R_{3c} : E_C a_B \to E_B$ $R_{3d} : E_B a_A \to E_A$
$R_{3e} : E_B a_C \to E_C$ $R_{3f} : E_B a_I \to E_I$
$R_{4a} : \Lambda \to E_A$ $R_{4b} : \Lambda \to E_B$ $R_{4c} : \Lambda \to E_C$ $R_{4d} : \Lambda \to E_I$
$R_{5a} : \Lambda \to a_A$ $R_{5b} : \Lambda \to a_B$ $R_{5c} : \Lambda \to a_C$ $R_{5d} : \Lambda \to a_I$
$R_{6a} : D_A \to \Lambda$ $R_{6b} : D_B \to \Lambda$ $R_{6c} : D_C \to \Lambda$ $R_{6d} : D_I \to \Lambda$
$R_{7a} : a_A \to \Lambda$ $R_{7b} : a_B \to \Lambda$ $R_{7c} : a_C \to \Lambda$ $R_{7d} : a_I \to \Lambda$
$R_8 : \Lambda \to D_I$ $R_9 : E_I \to \Lambda$

Note that if we want to use Proposition 2 then the same letters generated by different rewrite rules must be counted as distinct ones. But a_C in Γ_0 is equal to, e.g., a_C generated by R_{5c}, so the generated grammar is not annotated. Annotating, i.e. forbidding a program transformation tool to treat these letters as equal, is made in the corresponding program model.

5 Constructing a Program Model for a Prefix Grammar

Having a grammar that models a protocol, a program to verify the protocol is constructed as follows. Every rewrite rule $R_l^{[i]} \to R_r^{[i]}$ is supplied by a unique identifier i. Every letter c

generated by $R^{[i]}$ is represented as a pair $\langle c, i\rangle$. $\Phi[last]$ denotes the last letter of the word Φ. The program line that corresponds to an application of $R_l^{[i]} \to R_r^{[i]}$ looks as

$f(\langle R_l^{[i]}[1], x_1\rangle\langle R_l^{[i]}[2], x_2\rangle \ldots \langle R_l^{[i]}[last], x_{[last]}\rangle\Psi, cons(i, Hist), EL_1) =$

$f(\langle R_r^{[i]}[1], i\rangle\langle R_r^{[i]}[2], i\rangle \ldots \langle R_r^{[i]}[last], i\rangle\Psi, Hist, EL_2)$

The second parameter `Hist` (a history) and the initial word Γ_0 (with all letters supplied by the index 0) are input parameters of `f`. The history is added to the model to make the program deterministic — this method was successfully used in verification of cache-coherence protocols via supercompilation [6].

The third parameter in $f(\Phi, Hist, EL)$, EL, lists the number of occurrences of every c from Υ in the first parameter Φ (i.e. in the current word in the trace). This parameter is used to provide termination with accordance to Proposition 2. If some counter in the list EL reaches the erasing limit of the corresponding letter, then the trace is terminated independently of the value of $Hist$.

Example 9. *Let us construct a prefix grammar and the corresponding modeling program for* $P_2[x_1, x_2] = ((x_1, E_{x_2}a_{x_1}), (x_2, E_{x_1}a_{x_2}d_{x_1}D_{x_2}))$.

The prefix grammar $\mathbf{G_2}$ *is the following.*

$\Gamma_0 = E_B a_A$.

$R_1 : E_B a_A \to E_A a_B$	$R_2 : E_A a_B \to E_B a_A$	$R_3 : E_B a_I \to E_I a_B$	$R_4 : E_A a_I \to E_I a_A$
$R_5 : \Lambda \to E_A$	$R_6 : \Lambda \to E_B$	$R_7 : \Lambda \to E_I$	
$R_8 : \Lambda \to a_A$	$R_9 : \Lambda \to a_B$	$R_{10} : \Lambda \to a_I$	
$R_{11} : a_A \to \Lambda$	$R_{12} : a_B \to \Lambda$	$R_{13} : a_I \to \Lambda$	$R_{14} : E_I \to \Lambda$

Then let us assign erasing limits. $EL(E_A) = EL(E_B) = EL(a_A) = EL(a_B) = 2$; $EL(E_I) = 1$; $EL(a_I) = 3$.

Now we are ready to construct a model program. In the program, all the names starting with `x` *denote variables and all the other names (e.g. R4, a, etc.) denote constant data.* `xL_occ` *stands for the number of occurrences of the letter L in the current word. The constant a denotes* E_A, *b denotes* E_B, *and i denotes* E_i; *similarly, A denotes* a_A, *B denotes* a_B, *and I denotes* a_i. *To make the program to operate annotated rules we represent letters in the rules by the pairs* (`Letter of the initial grammar`, `Number of the rule applied`).

```
f(Nil,xHist,xa_occ,xb_occ,xi_occ,xA_occ,xB_occ,xI_occ)          = An attack found;

f(xWord, xHist, 3, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)      = No short attack;
f(xWord, xHist, xa_occ, 3, xi_occ, xA_occ, xB_occ, xI_occ)      = No short attack;
f(xWord, xHist, xa_occ, xb_occ, xi_occ, 3, xB_occ, xI_occ)      = No short attack;
f(xWord, xHist, xa_occ, xb_occ, xi_occ, xA_occ, 3, xI_occ)      = No short attack;
f(xWord, xHist, xa_occ, xb_occ, 2, xA_occ, xB_occ, xI_occ)      = No short attack;
f(xWord, xHist, xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, 4)      = No short attack;

f(cons((b, x1),cons((A,x2), xWord))),
     cons(R1, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
            = f(cons((a,1),cons(B,1), xWord)),
                    xHist, xa_occ+1, xb_occ-1, xi_occ, xA_occ-1, xB_occ+1, xI_occ);

f(cons((a,x1), cons((B,x2), xWord))),
     cons(R2, xHist), (xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
            = f(cons((b,2),cons((A,2), xWord)),
                    xHist, xa_occ-1, xb_occ+1, xi_occ, xA_occ+1, xB_occ-1, xI_occ);

f(cons((b,x1), cons((I,x2), xWord))),
```

```
      cons(R3, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons((i,3),cons((B,3), xWord)),
                      xHist, xa_occ, xb_occ-1, xi_occ+1, xA_occ, xB_occ+1, xI_occ-1);

f(cons((a,x1), cons((I,x2), xWord))),
      cons(R4, xHist), (xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons((i,4),cons((A,4), xWord)),
                      xHist, xa_occ-1, xb_occ, xi_occ+1, xA_occ+1, xB_occ, xI_occ-1);

f(xWord, cons(R5, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons((a,5), xWord),
                      xHist, (xa_occ+1, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ);

f(xWord, cons(R6, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons((b,6), xWord),
                      xHist, xa_occ, xb_occ+1, xi_occ, xA_occ, xB_occ, xI_occ);

f(xWord, cons(R7, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons((i,7), xWord),
                      xHist, xa_occ, xb_occ, xi_occ+1, xA_occ, xB_occ, xI_occ);

f(xWord, cons(R8, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons((A,8), xWord),
                      xHist, xa_occ, xb_occ, xi_occ, xA_occ+1, xB_occ, xI_occ);

f(xWord, cons(R9, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f(cons(B,9), xWord),
                      xHist, xa_occ, xb_occ, xi_occ, xA_occ, xB_occ+1, xI_occ);

f(xWord, cons(R10, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f (cons(I,10), xWord),
                      xHist, xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ+1);

f(cons((A,x1), xWord),
      cons(R11, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f (xWord, xHist, xa_occ, xb_occ, xi_occ, xA_occ-1, xB_occ, xI_occ);

f(cons((B,x1), xWord),
      cons(R12, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f (xWord, xHist, xa_occ, xb_occ, xi_occ, xA_occ, xB_occ-1, xI_occ);

f(cons((I,x1),xWord),
      cons(R13, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f (xWord, xHist, xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ-1);

f(cons((i,x1),xWord),
      cons(R14, xHist), xa_occ, xb_occ, xi_occ, xA_occ, xB_occ, xI_occ)
              = f (xWord, xHist, xa_occ, xb_occ, xi_occ-1, xA_occ, xB_occ, xI_occ);
```

Note that we do not count letters generated by different rules separately. Theoretically this can lead to a loss of a trace ending with Λ*, but practically we did not see the cases where the supercompilation results of the program with the common counters differed from the results*

provided by the algorithm of Dolev, Yao, and Karp from the paper [4].

To find out whether the program model generates Λ *we must run a supercompilation of the program on the input* $\texttt{f}(\texttt{cons}((\texttt{B},\texttt{0}),\texttt{cons}((\texttt{a},\texttt{0}),\texttt{Nil}))),\texttt{xHist},\texttt{1},\texttt{0},\texttt{0},\texttt{0},\texttt{1},\texttt{0})$*, which yields an unfolding process on the indefinite parameter* `xHist`*. If the residual program generates the message* ***`An attack found;`*** *then the protocol model is unsafe; otherwise the protocol model is very likely safe, but to guarantee this we must add distinct occurence counters for the same letters that are generated by distinct rules (e.g.* $\texttt{xaR1}_{\texttt{occ}}$ *and* $\texttt{xaR5}_{\texttt{occ}}$ *for* `a`*).*

Supercompilation of the example extended by distinct counters, as well as supercompilation of a model program for $P_{3a}[x_1, x_2, x_3]$ *with distinct counters, generates no attack models; for* $P_3[x_1, x_2, x_3]$ *several short attacks are explicitly constructed by the supercompiler.*

It can be noticed that the program and the grammar in Example 9 can be constructed in the reverse order: in this case the initial word is Λ (`Nil` in the program), the set $INSEC$ is $\{E_B a_A\}$ (`cons((b,x1), cons((A,x2),Nil))` in the program), and every rule of the form $R_l \to R_r$ is written in the form $R_r \to R_l$ (the similar method of reverse representation was described in [7]). The reverse trace from Λ to $E_B a_A$ also satisfies Proposition 2, and in most cases a supercompilation of a reverse program is faster than a supercompilation of the initial program. The cause is that the intruder's alphabet contains more appending operations (that correspond to the rules of the form $\Lambda \to x$) than deleting operations (reverses of which correspond to rules of the form $\Lambda \to x$), and therefore the semantic tree of the reverse program has less branching.

6 On Complexity of the Verification Process

In this section we analyze bounds on the attack length. It is not difficult to apply the results [10] together with Proposition 2 to find a rough upper bound for the maximal attack length on a ping-pong protocol.

Proposition 3. *Let* $\mathbf{G}$ *be an arbitrary finite annotated prefix rewriting grammar. Then the maximal length of a short attack model is bounded from above by an exponential under card*$(\mathbf{R})$ *function.*

Proof. For every rule $R : R_l \to R_r$ let us add to $\mathbf{G}$ $EL(R_r[last]) - 1$ copies of the rule ($EL(x)$ is an erasing counter of x). Let the classical subsequence relation $\trianglelefteq$ distinguish applications of all these rules and then unfold a computation applying exactly such a copy of a rule that matches the corresponding set of letter occurrences. The maximal bad sequence length in this grammar is not more than $\frac{k^{N+1}-1}{k-1}$ where k is the length of the maximal right-hand side and N is the total number of rules (with all the copies) [10]. And now we have $N \leq EL_{max} * N_0$, where N_0 is the number of rules in the initial grammar $\mathbf{G}$.

□

To build a grammar with a long shortest possible trace ending by Λ, we use the same idea of the "ladder" construction as in [10].

Proposition 4. *There exists a grammar such that the minimal attack length is exponential in the number of rules.*

Proof. Consider the following grammar $\mathbf{G_{NEXP}}$ with the initial word $a_1 A_1$.

$\mathbf{G_{NEXP}}$:

$R_1^W : \Lambda \to a_1 A_1$	$R_1^T : A_2 a_1 \to a_2 A_2$	$R_1^D : A_1 \to \Lambda$
.....	.....	.....
$R_i^W : \Lambda \to a_i A_i$	$R_i^T : A_{i+1} A_i \to a_{i+1} A_{i+1}$	$R_i^D : A_i \to \Lambda$
.....	.....	.....
$R_N^W : \Lambda \to a_N A_N$	$R_N^T : a_N \to \Lambda$	$R_N^D : A_N \to \Lambda$

In the grammar $\mathbf{G_{1EXP}}$ the shortest trace ending by Λ (we call these traces attacks for the sake of brevity) is $a_1 A_1 \to A_1 \to \Lambda$ and has the length 3.

Let us denote the minimal attack length in the grammar $\mathbf{G_{NEXP}}$ as $F(N)$. Now let us make a transition to the grammar $\mathbf{G_{N+1EXP}}$. To erase a_1 from the initial $a_1 A_1$ we must come to $A_2 a_1 A_1$, so we must build an attack in the grammar $\mathbf{G_{NEXP}}$ (modulo index renaming) without the last step of erasing A_2. Then we apply the rule R_1^T and get the word $a_2 A_2 A_1$. Now again we must repeat an attack on $\mathbf{G_{NEXP}}$ (modulo index renaming) to erase $a_2 A_2$ and at last erase A_1 by R_1^D. The total number of steps in the attack is not less than $2 * F(N) + 1$. So the formula for the minimal attack length for $\mathbf{G_{NEXP}}$ is $2^{N+1} - 1$.

□

7 Conclusion

In this paper, we have shown that the method presented in [9] for 2-party ping-pong protocols is also applicable for multi-party ping-pong protocols; moreover, also this extension can be implemented with the help of the supercompilation program SCP4. We showed that the initial condition with erasing distinction can be replaced in the verification algorithm with a simpler condition, that makes the algorithm applicable not only with SCP4 but also with other program transformation tools. Finally, we analyzed the worst-case length of a trace that can model an attack.

Our method is widely applicable (it allows one to verify not only a single protocol, but also several protocols that share some operators); however, every special case of its application requires extra modeling efforts due to the high computational complexity of the semantic tree unfolding. In the paper we presented some methods for building more efficient models of ping-pong protocols; in spite of existence of successful applications of the verification method on existing and newly proposed protocol models, it is yet unclear whether the method can be further refined, so that it can solve verification tasks for protocols with a large number of rewrite rules and party permutations within a reasonable run time.

8 Acknowledgments

The author is grateful to A.P. Nemytykh who encouraged and directed the study and to anonymous referees who helped to improve the presentation of the study.

References

[1] M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5:267–303, 1998.

[2] A. Ahmed, A. Lisitsa, and A. Nemytykh. Cryptographic protocol verification via supercompilation (a case study). In Alexei Lisitsa and Andrei Nemytykh, editors, *VPT 2013*, volume 16 of *EPiC Series*, pages 16–29. EasyChair, 2013.

[3] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55:57–68, 1982.

[4] D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, 29:198–208, 1983.

[5] S. Even and O. Goldrich. On the security of multi-party ping-pong protocols. Technical Report, 1985.

[6] A. Lisitsa and A. P. Nemytykh. Reachability analysis in verification via supercompilation. *International Journal of Foundations of Computer Science*, 19(4):953–970, 2008.

[7] A.P. Lisitsa and A.P. Nemytykh. On one application of computations with oracle. *Programming and Computer Software*, 36(3):157–165, 2010.

[8] A. P. Nemytykh. *The Supercompiler Scp4: General Structure*. URSS, Moscow, 2007.

[9] A. Nepeivoda. Ping-pong protocols as prefix grammars and turchin's relation. In *VPT 2013. First International Workshop on Verification and Program Transformation*, volume 16, pages 74–87. EPiC Series, EasyChair, 2013.

[10] A. Nepeivoda. A refinement of higman embedding for loop approximation. [unpublished], 2013. `http://refal.botik.ru/preprints/Antonina_Nepeivoda-On_Turchin_Theorem-06042013v1.pdf`.