Journal of Information & Communication Technology
Vol. 5, No. 1, (Spring 2011) 01-11

Design of Authentication and Key Distribution Protocols with Routing Functionality

Ocenasek Pavel*
Hranac Jakub*
Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno University of Technology, Czech Republic

ABSTRACT

This paper proposed the possible implementations of the regression-based method for the design of authentication and key distribution protocols. The automation of this method is presented as well as the basic routing functionality that allows a designer to define participants who are trustworthy enough to transfer information between other two participants without existence of a direct channel.

INSPEC Classification : B61, C60

Keywords : Authentication Protocol, Key Distribution, Regression Logic, Design

1. INTRODUCTION

This journal article is an extended version of the conference paper (Ocenasek, Hranac, 2011) and is based primarily upon authors' previous research (Ocenasek, Hranac, 2010; Hranac, 2010).

Logic which we are building on was first introduced in (Buttyán et al., 1998). It serves as means for generating authentication protocols (a subset of security protocols (Schneider et al., 2000)). While creating automatic synthesis using this logic as its basis a few unexpected problems were discovered. Some could be solved by extending the set of heuristic rules to cover them (freshness of formulas for example) but one had to be solved by extending the original logic.

*The material presented by the authors does not necessarily portray the viewpoint of the editors and the management of the Institute of Business and Technology (IBT) or Brno University of Technology, Czech Republic.

*Pavel Ocenasek: ocenaspa@fit.vutbr.cz
*Jakub Hranac: xhrana00@stu d.fit.vutbr.cz

First section describes the process of an automated protocol synthesis based on designer specified inputs (Ocenasek, 2010).

Second section then proposes an extension to the used logic which fixes the problem with message owners (who on the start of communication knows which messages?) and introduces an idea of routing messages (allowing principals to act as 'routers' and deliver messages even when there is no direct or public channel connecting the source and the destination).

1.1. BRIEF DESCRIPTION OF THE LOGIC

For full description of the logic please refer to (Buttyán et al., 1998). This section describes only enough to understand the following article.

Logic relies on existence of communication channels. It does not define how channel is realized, only describes its parameters. Depending on those, different encryption methods have to be used. In this article we will use only two types of channels. The direct channel (connects two principals and is secure) and the public channel (connects several principals and is insecure).

Several terms will be used often (Schneider et al., 2000) :
Freshness - Almost exclusively related to nonces. It allows the principal to determine whether a message/nonce was generated 'recently' (in this run of the protocol).
Assumption - Assumption is 'The Truth'. No matter how it's stated or what it says, synthesis expects it to be true and does not check if it is or not or if there is another assumption saying otherwise.
Heuristic rule - By using a heuristic rule you can 'decompose' a goal into 'smaller' goals or assumptions which have to be met in order to complete the original goal.

2. AUTOMATION OF THE SYNTHESIS PROCESS

The goal of creating automated synthesis (Zhou, 2003) is to allow a protocol designer to use a library function to read inputs (set of heuristic rules, assumptions and goals) and get an object which can then be called to get an ordered set of communication goals or information that the synthesis is not possible.

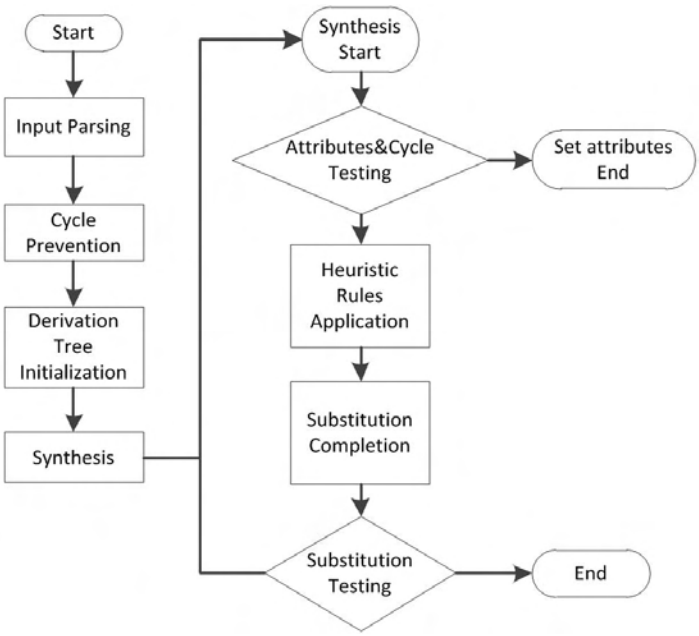
Input consists of declarations for channels, principals, formulas and 'fixed messages'[1], followed by a list of heuristic rules or assumptions and goals. Any 'unknown' token is handled as a universal message token.

The core of synthesis process (Zhou, 2003) lies in comparing syntax trees and ability to create substitutions between those trees. When a goal is being processed it (its syntax tree) is compared to assumptions (their syntax trees) and heuristic rules. Matching trees are found and substitutions generated in order to fully define subgoals.

Even when the supplied heuristic rules are unlikely to be modified, synthesis allows the designer to input almost any set of heuristic rules (which pass the cycle prevention test). Figure 1 shows a simple overview of the whole syntesis process. Each part will be explained in more detail later in this section.

[1] Fixed message was originally a nonce declaration; however it was necessary to include more than just nonces in this list which will be made clear in next section. For now you can expect any fixed message to behave just like a nonce.

Figure 1
Overview of preparation and synthesis process



2.1. Parsing, Syntax Trees, Cycle Prevention

Parsing can be done using push down automaton (PDA) since the logic can be described by using a CF grammar. The only problem lies in determining which tokens hold type of a principal and which are messages. After implementing extension described in next section it was decided to simplify the whole process and principals are the only token which can have two declarations: one as a principal, one as a fixed message. Which one of those two it will be in the end is needed to achieve by type checks (principals have attributes, believes, can say messages; messages can only be said or believed to be fresh).

Construction of syntax tree was done by using PDA again and reducing expression in post fix notation into a single token (type-checks are enough to test valid input).

Cycle prevention is a phase where syntax trees for heuristic rules and assumptions are tested for patterns that indicate possible built-in cycles, or just a common nonsense. It's not complete, there still might be some cycles that slipped through, but those will be detected later on.

Patterns that are used should include at least detection and removal/warning about following expressions which are common sources of generating cycles[2]:

$$\begin{aligned} &\alpha \rightarrow \alpha \\ &P \mid\equiv (P \mid\equiv \alpha) \rightarrow \alpha \end{aligned} \tag{1}$$

[2] Unlike normal cycles when we get expression A from A by applying different transformations, generating cycles generate expressions containing A which can be processed as A by a heuristic rule and generate even bigger and bigger expressions.

Cycles where non-generating loop appears are detected later during synthesis and mark the whole derivation branch as invalid.

in later iterations), its attributes and if needed (and possible) the information about possible use of heuristic rules on this goal and set of new subgoals (new derivation trees). It has to remember used heuristic rule, processed and not yet processed subgoals, and generated substitutions for each branch. Process of generating branches will be described later in this section. For now imagine a structure:

```
class DerivationTree:
   DerivationTree* parent
   Expression goal
   Boolean isAssumption
   Boolean isProtocolStep
   List<Derivation> childNodes

class Derivation:
   HeuristicRule usedRule
   SubstitutionTable substitutions
   List<DerivationTree> subGoals
```

Expression is a class representing a syntax tree for one expression, *HeuristicRule* contains a copy of a heuristic rule[3] and *SubstitionTable* is set of three[4] lists of pairs of expressions (an old and a new one).

2.3. Synthesis

Synthesis can be run in parallel since goals can be processed independently. First iteration spawns one thread for each goal; further iterations use one thread for each generated subgoal.

Cycle detection. Before any further analysis of current goal can be done the goal is tested for cycles. Test is simple syntax tree absolute comparison between current goal and all its parents. If a match is detected, cycle appeared and current branch is marked as invalid. When the control is returned to the parent thread (all child branches are analyzed) all invalid branches are removed.

Attribute checks. After cycle detection attributes are checked. This decides if the current goal is a protocol step or an assumption or neither. In those cases it's considered final and heuristic rules are not applied.

Heuristic rule application. When the processed goal is not final, it is necessary to get a set of applicable heuristic rules. This is done by comparing the syntax tree of the goal to syntax trees of base expressions in all heuristic rules. At this point it is not necessary to generate any substitutions yet. We just create a branch for each applicable heuristic rule, copy the rule to this branch and initialize an empty substitution table and empty set of subgoals.

Comparing syntax trees is done recursively. Separate tokens are accepted based on two conditions: if token in the pattern (heuristic rule) is a formula token, target (goal) must NOT be any of the following: message, channel, principal. In all other cases type of the pattern must be exactly the same as type of the target. Comparison is then called on all child tokens recursively. If any mismatch is detected, rule is not applicable.

[3] Heuristic rules must be preprocessed to remove any used ORs in set of subgoals. Such rules must be cloned for each possible alternative where all ORs are removed.

[4] There are three levels of substitution. They will be described in more detail later on.

Substitution generation. As stated before, substitution is three layered. The first layer can be called 'heuristic substitutions'. Those are generated to convert the base expression in the used heuristic rule into the currently analyzed goal. This level of substitution will never return multiple possibilities. Searching is similar to comparing statements but it records which tokens were compared. This record is in fact a substitution table. After the first layer of substitution is generated, subgoals in the heuristic rule (let's call those 'incomplete subgoals') are checked if this one substitution table can complete them (substitute all tokens). If so, they are moved to the list of subgoals (let's call those 'complete subgoals'). If there are no more incomplete subgoals synthesis can recursively call itself on the set of complete subgoals.

However, if there are still incomplete subgoals, two new layers of substitution will have to be generated. Those are based on comparing incomplete subgoals to the set of assumptions to check if any matches. All incomplete subgoals $A \mid\equiv B \mid\sim A, Na$ are compared with each substitution to cover all combinations. Searching is again similar to comparing statements, but this time it is important to record substitutions TO assumptions and substitutions FROM assumptions. Assumptions can contain formulas or template messages (not fixed messages) and those must be substituted and recorded as above mentioned substitutions 'from assumptions'. It is demonstrated in the following example.

Example: *Imagine we are processing a subgoal where A, B are principals and A, Na are fixed messages (principal name message and a nonce). One of heuristic rules can be defined as:*

$$P \mid\equiv \alpha$$
$$\mapsto P \mid\equiv \beta$$
$$\mapsto P \mid\equiv \beta \rightarrow \alpha$$

Base heuristic substitution would be $\{(P, A), (\alpha, B \mid\sim A, Na)\}$. *After application of this substitution it is still impossible to get any complete subgoals and all stay incomplete (there is no substitution defined for* β*). This is where two new layers of substitution come in. By searching the set of assumptions synthesis discovered assumption that says* $A \mid\equiv (S \mid\equiv B \mid\sim X) \rightarrow (B \mid\sim X)$. *A believes that if S believes that B said some message, then B really said it.*
Two new substitution entries are generated: 'to assumption' $\{(\beta, S \mid\equiv B \mid\sim X)\}$ *and 'from assumption'* $\{(X, A, Na)\}$.

When applying those substitutions it is vital to apply them in the correct order (heuristic substitutions, to assumption substitutions, from assumption substitutions). Those two new layers of substitution generate not a single result, but usually more (depending on the set of assumptions, one for each fitting assumption). Each possibility must clone the current branch in the derivation tree with all so far known attributes (partial substitution table, complete and incomplete subgoals) and finish the substitution table with generated results. When there are still some incomplete subgoals whole processes of searching for substitutions is repeated. If no fitting assumption is found, branch is marked as invalid and immediately deleted because it can never be completed given the structure of the principals net and current set of believes.

Validation and recursive synthesis call. Substitutions are validated by a simple check if there is not a double substitution (two different tokens substitute into a same expression could cause generating cycles). After this the whole process is repeated in parallel for all subgoals in all generated branches.

3. MESSAGE OWNERSHIP AND ROUTING EXTENSION

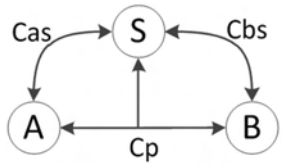
When implementing and testing the solution, one problem with message delivery was

discovered. Core of the problem lays in the 'Seeing rule' defined in (Buttyán et al., 1998) which states:
If a principal P receives a message X via a channel C, and can read this channel, then recognizes that the message has arrived on C and P can see the message.

$$\frac{P \triangleleft C(X), P \in r(C)}{P \mid\equiv (P \triangleleft X \mid C), P \triangleleft X} \tag{3}$$

While on the first look this rule may seem correct and sufficient, it does not say anything about someone being able to write the channel C. Imagine the following situation. There is a net of principals as described in Figure 2.

Figure 2
Principals and channels connecting them each with S and one public channel Cp



Now, the problem lies in composing communication steps for the goal:

$$A \mid\equiv B \mid\sim A, Na \tag{4}$$

There is a set of assumptions and heuristic rules:

Heuristic rules

$$\begin{aligned} & P \triangleleft X \\ & \mapsto P \triangleleft C(X) \\ & \mapsto P \in r(C) \end{aligned} \tag{H1}$$

$$\begin{aligned} & P \mid\equiv Q \mid\sim X \\ & \mapsto P \mid\equiv Q \mid\sim X \\ & \mapsto P \mid\equiv \#(X) \end{aligned} \tag{H2}$$

$$\begin{aligned} & P \mid\equiv Q \mid\sim X \\ & \mapsto P \triangleleft C(X) \\ & \mapsto P \in r(C) \\ & \mapsto P \mid\equiv (w(C) = \{Q\}) \mid P \mid\equiv (w(C) = \{P, Q\}) \\ & \mapsto Q \triangleleft X \end{aligned} \tag{H3}$$

$$\begin{aligned} & P \mid\equiv Q \mid\sim \alpha \\ & \mapsto P \triangleleft C(\alpha) \\ & \mapsto P \in r(C) \\ & \mapsto P \mid\equiv (w(C) = \{Q\}) \mid P \mid\equiv (w(C) = \{P, Q\}) \\ & \mapsto Q \mid\equiv \alpha \\ & \mapsto P \mid\equiv ((Q \mid\sim \alpha) \rightarrow (Q \mid\equiv \alpha)) \end{aligned} \tag{H4}$$

$$P \mid\equiv \alpha \\ \mapsto P \mid\equiv \beta \\ \mapsto P \mid\equiv \beta \rightarrow \alpha \tag{H5}$$

Assumptions

$$A \mid\equiv (w(Cas) = \{A,S\}); S \mid\equiv (w(Cas) = \{A,S\}); \\ B \mid\equiv (w(Cbs) = \{B,S\}); S \mid\equiv (w(Cbs) = \{B,S\}); \\ S \in r(Cas); A \in r(Cas); S \in r(Cbs); B \in r(Cbs); \\ A \in r(Cp); A \in w(Cp); B \in r(Cp); B \in w(Cp); \\ S \in r(Cp); S \in w(Cp); A \mid\equiv \#(Na); \\ A \mid\equiv ((S \mid\sim \alpha) \rightarrow (S \mid\equiv \alpha)); \\ A \mid\equiv ((S \mid\equiv (B \mid\sim X)) \rightarrow (B \mid\sim X)); \tag{5}$$

By applying those we get to the point where we have to get done the goal:

$$B \triangleleft A, Na \tag{6}$$

This goal can be completed by applying the heuristic rule[5] (H1) (introduced in (Buttyán et al., 1998) and based on (3)). At this point we generate one protocol step and we need to have defined which channels B can actually read. If it can read both (Cbs and Cp) then the correct (read 'correct in the logic') answer is to get two variants as solution for this goal:

$$B \triangleleft Cp(A, Na) \\ B \triangleleft Cbs(A, Na) \tag{7}$$

The first one is correct, however the second one can be correct only if S (who is the only one who can write to Cbs except for B itself) already knows (A, Na). If this is not the case, then we have successfully generated a faulty protocol.

Even worse, when the channel Cp is removed then only the second variant from (8) gets generated and the protocol will never work.

First goal will be to eliminate generating of invalid rules. To solve this problem we introduced an attributed called 'message ownership'.

At the beginning we assign this attribute to principals who 'own' a fixed message. That means they know this message from the start of the communication. Now, it is important to own only fixed messages, because those are substituted strictly to 'itself' during synthesis while template messages can be any message at all (it is the same problem as with freshness test on nonces. If a nonce would not be a fixed message, principal would accept any message as fresh).
To use the owner attribute it is necessary to modify the 'Seeing rule' as follows:

$$\frac{P \triangleleft C(X), P \in r(C), Q \in w(C), Q \in owns(X)}{P \mid\equiv (P \triangleleft X \mid C), P \triangleleft X} \tag{8}$$

[5] Optional route is to split it into two goals when each would be B seeing message A or Na. Then (H1) would have to be applied on both of those goals.

Meaning: P can see message X if it receives it via a channel it can read and there is someone else who can write to this channel and knows (owns) the message.

This modification of the logic removes problem with expecting messages on channels where no principal knows them. It works well in 'routed' environment where there always exists a direct[6] channel (at least insecure channel) between two principals.

3.1. Routing

When there is basis of message ownership in the logic it is possible to build on this and set up a basic routing for messages[7]. Imagine the situation form Figure 2 when the common channel (*Cp*) gets removed and the goal stays the same. There still exists a route for message (*A, Na*) however it must be retransmitted by the principal *S*.

Plus, when there would be more principals like S (who connect *A* and *B*) it may be useful to allow protocol designer to specify which principals can send what messages, from which sources to which principals. This defines a basic trust levels between principals. **Note:** Only messages that would normally be transmitted via insecure channels should be allowed to be retransmitted by any principals.

To cover this functionality we defined three new attributes. *CanResend* and *canResendFrom* when each of those takes a message (fixed or template) as one parameter and optionally a principal as other parameter. The last attribute is *canOwn* which is a meta-attribute. It must not be set in assumptions and is only used in heuristic rules. It encapsulates routing rights.

Heuristic rules need to be modified to reflect those changes, so from (H1) we get those new rules:

$$
\begin{aligned}
& P \triangleleft X \\
& \mapsto P \triangleleft C(X) \\
& \mapsto P \in r(C) \\
& \mapsto Q \in w(C) \\
& \mapsto Q \in canOwn(X) \mid Q \in owns(X)
\end{aligned}
\qquad \text{(H1A)}
$$

$$
\begin{aligned}
& P \in canOwn(X) \\
& \mapsto P \triangleleft C(X) \\
& \mapsto P \in r(C) \\
& \mapsto Q \in w(C) \\
& \mapsto P \in canResend(X) \mid P \in canResendFrom(X, Q) \\
& \mapsto Q \in canOwn(X) \mid Q \in owns(X)
\end{aligned}
\qquad \text{(H1B)}
$$

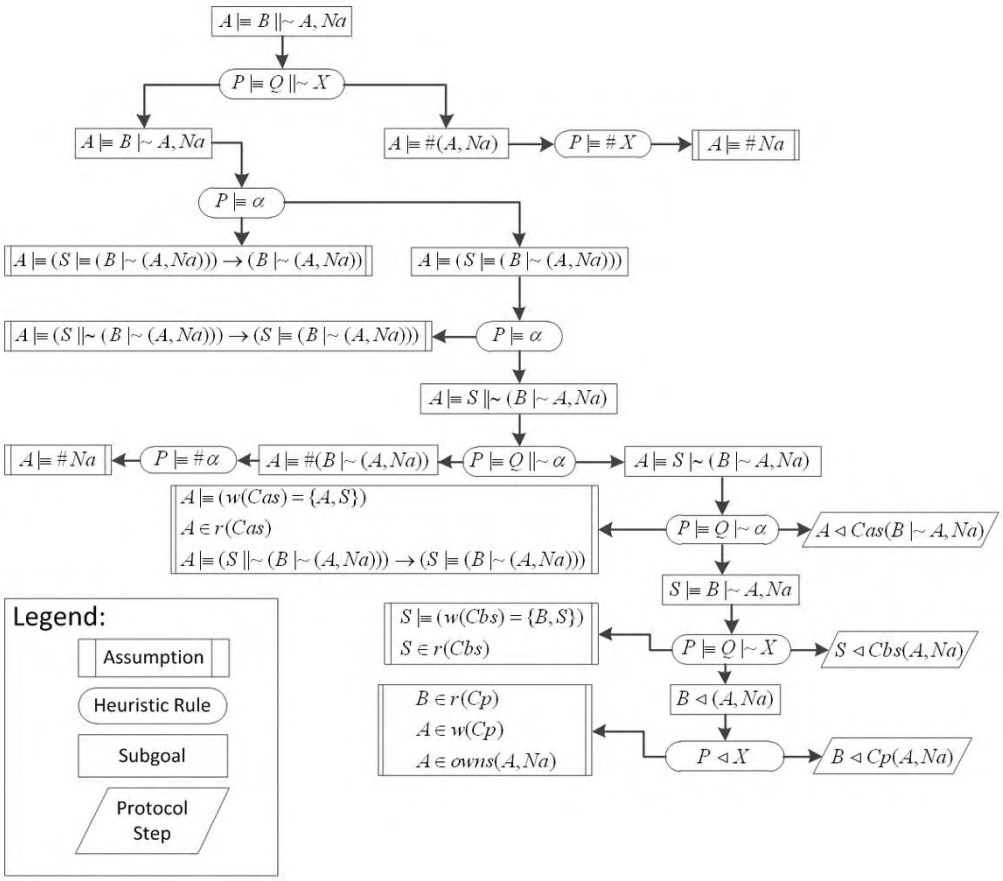
Note: Notice that trust is set in a backwards manner. It is not set on route from source (setting whom we entrust with a message) but rather from destination to source (setting who is reliable source). Power of this reasoning stays the same.

[6] In terms of this logic all channels are direct. It does not say anything about how is the channel physically realized.

[7] Note that routing is never used for formulas because only a formula principal believes in can be transmitted which means principal either builds its own formula or it's already received from somewhere else.

Note: It is not important to worry about cycles since those are already tested for during cycle detection. So message will never travel through same principal twice. In complex networks where everyone is connected with everyone except for two principals who wish to exchange a message there will be as many protocols as there are routes in an oriented graph where nodes (principals) are connected by using channels.

Figure 3
Synthesis output

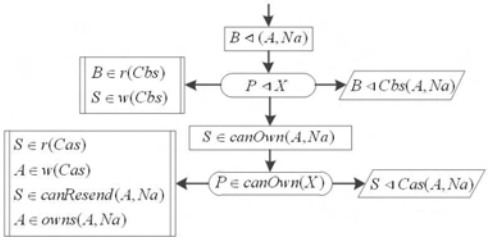


Let's see how this will work by demonstrating functionality on the goal (4) and net defined by Figure 2 using assumptions (5) and rules (H2-H5, H1A, H1B). First, nobody has the *canResend* attribute set (see Figure 3.).

Synthesis has successfully filtered out problematic protocol step $B \lhd Cbs(A, Na)$. Next demonstration will show how routing can be done by giving S the attribute *canResend* for the message (*A*, *Na*). To reduce amount of found protocols we remove the channel Cp. Only part from $B \lhd (A, Na)$ gets extended as shown in Figure 4.

The message gets properly routed through S and reproduced to *B* so it can then send it back. Although this example is not very complex, it demonstrates well how routing works. Same can be done with multiple principals routing messages, however amount of generated protocols gets higher with each possible route from *A* to *B* in the principals net.

Figure 4
Modified part of synthesis with routing enabled



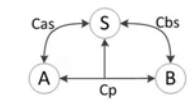
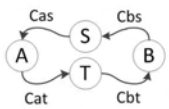
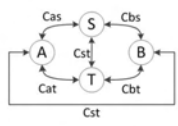
3.2 Benchmarking

Message owner modification only adds one attribute which should not affect performance too much. Routing can increase synthesis time especially with complex net of principals where there are many routes for a message to travel between two principals. Following test uses rules (H2-H5, H1A, H1B) and always has one goal:

$$A \mid\equiv B \mid\sim A, Na \tag{9}$$

Set of assumptions is defined by a principal net (for channel read/write permissions, channels connecting two principals only are considered secure). Message resend attribute is noted in each case separately as well as any other assumptions which are not channel related.

Table 1
Performance with and without using routing extension[8]

Principal net scheme	Results for message owner extension	Results for message routing extension[9]
 $S \in canResend(A, Na)$; $A \mid\equiv \#(Na)$; $A \mid\equiv ((S \mid\sim \alpha) \rightarrow (S \mid\equiv \alpha))$; $A \mid\equiv ((S \mid\equiv (B \mid\sim X)) \rightarrow (B \mid\sim X$	Results: 1 Average time: 41ms Solution: $B \triangleleft Cp(A, Na)$ $S \triangleleft Cbs(A, Na)$ $A \triangleleft Cas(B \mid\sim A, Na)$	Results: 1+4 Average time: 76ms One of solutions: $S \triangleleft Cas(A, Na)$ $B \triangleleft Cbs(A, Na)$ $S \triangleleft Cbs(A, Na)$ $A \triangleleft Cas(B \mid\sim A, Na)$
 $S, T \in canResend(A, Na)$; $A \mid\equiv \#(Na)$; $A \mid\equiv ((S \mid\sim \alpha) \rightarrow (S \mid\equiv \alpha))$; $A \mid\equiv ((S \mid\equiv (B \mid\sim X)) \rightarrow (B \mid\sim X))$;	Results: 0 Average time: 30ms No solution was found.	Results: 0+1 Average time: 61ms Solution: $T \triangleleft Cat(A, Na)$ $B \triangleleft Cbt(A, Na)$ $S \triangleleft Cbs(A, Na)$ $A \triangleleft Cas(B \mid\sim A, Na)$
 $S, T \in canResend(A, Na)$; $A \mid\equiv \#(Na)$;	Results: 1 Average time: 55ms Solution: $B \triangleleft Cab(A, Na)$ $A \triangleleft Cab(A, Na)$	Results: 1+4 Average time: 160ms One of solutions: $T \triangleleft Cat(A, Na)$ $S \triangleleft Cst(A, Na)$ $B \triangleleft Cbs(A, Na)$ $A \triangleleft Cab(A, Na)$

[8] Tested on Intel® Core™ 2 Duo @ 2.66GHz, 64-bit OS; time is an average time calculated from 1000 tries.

[9] Results for routing extension always include the original ones as well so the amount of available communication schemes is numbered as x+y where x is the number of original solutions and y amount of new ones.

4. CONCLUSION

While routing has almost double increase in resources needed to find valid protocol, it has a chance of creating several new correct protocols. However it is necessary to avoid assigning canResend permission to everyone (use only reliable principals and those which you actually need). Dense principal nets make synthesis to take much longer to complete when routing is allowed to multiple nodes and synthesis returns rather large amount of possible protocols.

5. ACKNOWLEDGEMENT

The research has been supported by Technology Agency of the Czech Republic (TACR) in frame of the project SCADA system for control and monitoring RT processes, TA01010632. The research has also been supported by the Czech Ministry of Education in frame of the Research Intention MSM 0021630528: Security-Oriented Research in Information Technology, MSM 0021630503 MIKROSYN: New Trends in Microelectronic Systems and Nanotechnologies, by the the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070 and projects FIT-S-11-1 and FIT-S-11-2.

6. REFERENCES

Buttyán, L., Staamann, S., Wilhelm, U. "A Simple Logic for Authentication Protocol Design", In: 11th IEEE Computer Security Foundations Workshop, 1998, 153.

Hranac, J. "Methods of the Security Protocols Design" (MSc thesis, Brno University of Technology, 2010).

Ocenasek, P., Hranac, J. "Routing Functionality in the Logic Approach for Authentication Protocol Design", in Human Interface and the Management of Information. Interacting with Information, Berlin Heidelberg, DE, Springer, 2011, pp. 366-373, ISBN 978-3-642-21792-0, ISSN 0302-9743.

Ocenasek, P., Hranac, J. "Regression Based Logic for Authentication Protocol Design", in 2010 International Conference on Communication and Vehicular Technology, Chengdu, CN, IEEE, 2010, pp. 89-92, ISBN 978-1-4244-9674-7.

Ocenasek, P. "Automated Design of Authentication and Key Distribution Protocols" (PhD diss., Brno University of Technology, 2010).

Schneider, S.A., Ryan, P.Y.A. "Modelling and Analysis of Security Protocols". Addison Wesley, Boston, 2000, ISBN 0-201-67471-8.

Zhou, H., Foley, S. N."Fast automatic synthesis of security protocols using backward search", In: Proceedings of the 2003 ACM workshop on Formal methods in security engineering, October 30, 2003, Washington, D.C., 1-10.