Security proofs of cryptographic protocols

Eva Jencusova
Department of Computer Science
University of P.J.Safarik
Jesenna 5, 040 01 Kosice, Slovakia
jencusova@yahoo.com

1 Introduction and motivation

We live in time of computers and scientific and technological progress but this time is also marked with many attacks on big Internet companies. So with expand of Internet people start to protect information that they exchange through Internet. In the Internet world it is necessary for systems as Internet banking, Internet shopping or e-commerce to secure private information that are transferred via Internet. This necessity leads to design security improvements of existing protocols and to design and implement cryptographic algorithms (for example protocols as L2TP, Microsoft PPTP, IPSec, SSH or algorithms as DES, Blowfish and others).

Consider that we have two banks, bank “Alice” and bank “Bob” and these two banks for business purposes need the private cipher for their communication. They need this cipher to do secret bank operation. So they need the secure way, how to arrange the cipher. For this purpose the designers designed the communication protocols. But the both banks “Alice” and “Bob” must be sure that no one can achieve this cipher by negotiation and both must be sure that they talk with the cipher one with the other. And such proofs about communicating protocols can be done only with the security analysis.

So it is not enough to design any “security” protocol or cryptographic algorithm, more important is to prove that protocol is really secure. In the world of informaticians and designers it must be developed new methods and formalisms that give us exact proofs of correctness and soundness of security protocols and cryptographic algorithms.

2 Database and logic abduction method (DLA)

The main goal of our work is to show another possible approach to analysis of security and cryptographic protocols. We also want to show where are weak places of some formal methods based on logic and we propose an alternative solution.

2.1 A Combination of top-down and bottom-up reasoning

We deal with models based on logic, but our point of view is more common with databases point of view and the model is independent on used crypto-system in analyzing protocol. Our model **DLA** (=database and logic abduction) is different from others that mostly used top-down reasoning. In our model we combine *top-down* and *bottom-up* reasoning. Shortly, first we describe the knowledge of participants as relations in database model. Then we translate the protocol into the rules. In our model we have three types of rules – protocol independent cryptographic rules, protocol specification rules and relational rules, that is another difference from other models. Protocol independent cryptographic rules are rules that are for every protocol the same. This type of rules describes general cryptographic functions used in security protocol as encryption, decryption, hash, handshake and operations with words as concatenation and projection. Protocol specification rules are used for description of analyzed protocol. And the last group of rules is the relational rules that are used to distinguish type of words in main relation. We used all types of rules to make the closure of the knowledge relations. In this moment we apply bottom-up reasoning. If we are not successful to show that there exists knowledge to lead to an attack we use the logic abduction (top-down reasoning) to describe eventually weak places in the protocol that would lead to an attack by using some additional information. First of all we describe the environment of our model.

2.2 DLA Environment

By every design before it will be designed new model it must be described the environment of that model. In the world of security and cryptographic protocols there are three important issues that describe environment:

1. Description of the adversary (intruder or attacker).
2. Defining of an attack.
3. Description of protocol.

2.2.1 Description of the adversary

The role of the adversary varies considerably across the many of the developed protocol analysis tools. In the first of models there were no adversary explicitly. This was because the goal of these analyses was to draw conclusions about the beliefs of the legitimate participants. This hold especially about most of analyses based on logic of belief. And some others approaches modeled the adversary as nearly the same sort of participant as the legitimate participants. These approaches do not find right way how to express the range of the adversary's possible actions.

Our adversary is near to adversary in Dexter's model. The adversary has complete control over the network. It may read, block, delay or alter any message sent. It also may create new messages. Of cause, the adversary has all abilities of a legitimate participant. That means adversary can also initiate legitimate communication. It knows identifiers of all legitimate participants, it may share a key with the key server.

The goal of the adversary is to try to acquire illegitimately some privilege or information of a legitimate participant. Simply, the adversary may try to learn some information not intended for it (violating *secrecy*), or it may try to masquerade as another legitimate participant (violating *authenticity*).

2.2.2 Defining of an attack

Defining of an attack is one the most important thing by the analysis. The interpretation of result of analysis depends on the description of the attacker and of the attack. Here we will deal about different approaches of attack in literature and at the end we define an attack in our analysis.

With defining of an attack it is similar as with description of the adversary: it is very variable across the literature. In BAN logic modeling there is present no description of the attack. Instead, the BAN logic analysis presents that it's necessary to have additional assumptions about the protocol, because results whether or not there are flaws in the protocol depend on these assumptions. In algebraic approach of Dolev and Yao it looks that an attacks is defined as the ability of the adversary to learn plain text of encrypted message. But Meadows does not give satisfaction with such definition of the attacker. In her work she extends this definition with the ability of an adversary to achieve that a bogus key will be distributed. Description of an attack corresponds in her work with description of the *insecure* states in state machine. The attack is reached if it exists a path to these *insecure* states.

The definition of an attack depends on ability of the designers to describe possible flaws. There are different types of flaws of the protocols and there is also taxonomy of known flaws. *Carlsen* in [4] presents the *taxonomy* of flaw categories. He presents these flaws:

- *elementary* – unsuitable use of protocol or cryptography
- *password-guessing* – depending on inadequate password policies
- *freshless* – if it isn't possible to determine whether received message was created recently

- *oracle* – or design flaws mean that the adversary is able from protocol evaluation to learn secret information not intended to him
- *type* – these allow to adversary to exploit a principal's inability to determine the type (key, nonce, etc.) of particular message
- *internal* – or implementation flaws mean that the adversary is not able to perform some necessary internal operations as for example authentication or verification
- *crypto-system related* – implementation flaws that were created by use of a particular crypto-system.

But there exist others taxonomies, for example Syverson in [34] focused only on replay attacks. This taxonomy is based on the protocol run of origin for a message. *Syverson's taxonomy* follows:

1. Run external attacks (replay of messages from outside the current run of the protocol)
   (a) Interleavings (requiring contemporaneous protocol runs)
      i. Deflections (message is directed to other than the intended recipient)
         A. Reflections (message is sent back to sender)
         B. Deflections to a third party
      ii. Straight replays (intended principal receives message, but message is delayed)
   (b) Classic replays (runs need not be contemporaneous)
      i. Deflections (message is directed to other than the intended recipient)
         A. Reflections (message is sent back to sender)
         B. Deflections to a third party
      ii. Straight replays (intended principal receives message, but message is delayed)
2. Run internal attacks (replay of messages from inside the current run of the protocol)
   (a) Deflections (message is directed to other than the intended recipient)
      i. Reflections (message is sent back to sender)
      ii. Deflections to a third party
   (b) Straight replays (intended principal receives message, but message is delayed)

This categorization is hierarchical and each level in the hierarchy forms a partition of the preceding level. The taxonomy is trivially complete: all replay attacks (in the sense just given) can be classified as falling into one of the categories.

From our point of view for analysis of protocols we do not need search in protocol for all types of flaws. It is simply impossible to find by formal analysis the implementation flaws (as for example bad handling with interrupts and others). So in our model we focused on flaws that can arise in the design stage. So we restrict our attention to fresh and oracle flaws from Carlsen taxonomy. These types of attacks allow to the attacker to achieve messages from one protocol evaluation and then attacker can perform the crypto-analysis or can simply use intercepted data by next protocol evaluation to try to masquerade or to learn session key.

We use following possibilities to try to find these types of flaws. We try to determine whether the adversary is able to produce all protocol messages generated by one of the legitimate participants (oracle flaws) and we try to show whether the adversary is able to learn the content of the encrypted message at the end of the protocol. We also try to find whether it is possible to do an denial of services attack, where one of the participants believe that he is sharing session key with the other one, but this doesn't share any session key with him. The adversary does not have any priori knowledge of session-dependent information.

2.2.3 Description of protocol

If we want to analyze any protocol we need to find suitable language to express it in form that can be used for formal analysis. As a lot of others researchers we use in our model the language of logic programming to express the evaluation of the protocol.

In our model we describe the protocol as a set of inference rules that express the sending and receiving of the messages and also the order of sending of the messages. We show that description of the protocol and the whole environment can be done also in different language as the language of logic programming is. We present in next sections how it is possible to express the protocol and its environment in language of database modeling.

2.3 Specification of DLA method

In next two sections we show how we use the language of logic programming and database modeling to express environment of evaluation and functions of the protocol. We also show how works our program *"Converter"* that help us by translation of a protocol into language of logic programming.

2.3.1 DLA Language

First of all we describe the knowledge set and its attributes. The core of whole model is relation *Messages* that contains knowledge of all participants. *Messages* is binary relation.

Messages(Agent*: Dagents ,* Word*: Dmessages)* – means agent *Agent* knows message *Word*

The domains of attributes and variables are:

- *Dagents* – is set that contains identifiers of all participants
- $Dmessages = (Dnonces \cup Dagents \cup Dkeys \cup Dtimestamps \cup Dwords)^n$ it means that any message can consists of nonces, identifiers of participants, cryptographic keys, timestamps and arbitrary data. Arbitrary data can be data exchanged after evaluation of protocol. In the messages can also be new words that are created by evaluation using

functions as concatenation, projection, handshake, cryptographic hash and key encryption.

We use in this work these notations:

- Nonces *n, m* - unpredictable random numbers generated during evaluation of protocol to provide a rough agent specific sense of freshness.
- Agents *a, b, c, i, s* – identifiers of participants
- Keys: *ka, kb* – keys shared by participants *a* and *b* with server *s*, *kab* – session key shared between participant *a* and *b*
- Others:
  - handshake *f(X)* – function that participants allows to permute message. Often is implemented as the decrement function. It is easily computed and easily converted function.
  - cryptographic hash *h(M,K)* – is function that allows participants to compute secure “message digests” that are used by authentication. Any participant can compute hash of message *M* if he knows the key *K*.
  - key encryption *e(M,K)* – is function that encrypts message *M* using the key *K*.
  - concatenation of messages $m_1,...,m_n$ is denoted as [$m_1,...,m_n$]
  - arbitrary data are denoted as *p,q,...*

So in the model are these relation schemes:

- *Messages(* Agent: *Dagents* , Word: *Dmessages)*
- *Nonce(* Nonce*: Dnonces)* – says that “Nonce” is a nonce.
- *Key(* Key*: Dkeys)* – says that “Key” is a key.
- *Timestamp(*Time*: Dtimestamps)* - says that “Time” is a timestamp.
- *Agent(*Agent*: Dagents)* – says that “Agent” is a identifier of participant
- *Nonces(*Agent*: Dagents;* Nonce*: Dnonces)* – says that participant “Agent” knows nonce “Nonce”.
- *Keys(*Agent*: Dagents;* Key*: Dkeys)* – says that participant “Agent” knows key “Key”.
- *Timestamps(*Agent*: Dagents;* Time*: Dtimestamps)* – says that participant “Agent” knows timestamp “Time”.
- *Agents(*Agent*: Dagents;* Identifier*: Dagents)* – says that participant “Agent” knows identifier of participant “Identifier”.
- *Com_Nonces(*Agent1, Agent2*: Dagents;* Nonce*: Dnonces)* – says that nonce “Nonce” uses participant “Agent1” by communication with participant “Agent2”.
- *Com_Keys(*Agent1, Agent2*: Dagents;* Key*: Dkeys)* – says that key “Key” is the session key between participants “Agent1” and “Agent2”.
- *Com_Timestamps(*Agent1, Agent2*: Dagents;* Time*: Dtimestamps)* – says that timestamp “Time” is used by communication between participant “Agent1” and participant “Agent2”.

2.3.2 Conversion of protocol into DLA language rules

The relations *Nonce, Key* and *Timestamp* are unary relations that are not changed during evaluation of protocol, because they only define the types of words in protocol messages. The relations *Nonces*, *Keys* and *Timestamps* are binary relations that describe some type of knowledge of participants and they can change during evaluation of protocol by using following rules:

[R.1] *Nonces(X, N):-Messages(X, N)&Nonce(N).*
[R.2] *Keys(X, K):-Messages(X, K)&Key(K).*
[R.3] *Timestamps(X, T):-Messages(X, T)&Timestamp(T).*
[R.4] *Agents(X, A):-Messages(X, A)&Agent(A).*

Fig. 2.1. Relational inference rules

From database view the rules look like:

```
[R.1]     INSERT INTO Nonces
          VALUES (SELECT M.Agent, N.Nonce FROM Messages M, Nonce N
                  WHERE M.Word=N.Nonce)
[R.2]     INSERT INTO Keys
          VALUES (SELECT M.Agent, K.Key FROM Messages M, Key K
                  WHERE M.Word=K.Key)
[R.3]     INSERT INTO Timestamps
          VALUES (SELECT M.Agent, T.Time FROM Messages M, Timestamp T
                  WHERE M.Word=T.Time)
[R.4]     INSERT INTO Agents
          VALUES (SELECT M.Agent, A.Agent FROM Messages M, Agent A
                  WHERE M.Word=A.Agent)
```

Except these rules we also need another rules to interpret standard functions as encryption, decryption, hash, handshake, concatenation and projection. These type of rules are called protocol-independent cryptographic rules because are most the same for every protocol.

[I'.1] *Messages(X, T):-Messages(X, e(T, L))&Keys(X, L).* - decryption
[I'.2] *Messages(X, e(T, L)):-Messages(X, T)&Keys(X, L).* - encryption
[I'.3] *Messages(X, f(T)):-Messages(X, T).* - handshake
[I'.4] *Messages(X, T):-Messages(X, f(T)).*
[I'.5] *Messages(X, T1):-Messages(X, [T1, T2]).* - projection
Messages(X, T2):-Messages(X, [T1, T2]).
[I'.6] *Messages(X, [T1, T2]):-Messages(X, T1)&Messages(X, T2).* – concatenation
[I'.7] *Messages(X, h(T, L)):-Messages(X, T)&Keys(X, L).* - hash

Fig. 2.2. Protocol-independent cryptographic rules

For example:

Rule *Messages(X, T):-Messages(X, e(T, L))&Keys(X, L)* means that any agent *X* can learn message *T* from encrypted message *e(T,L)* if he has key *L* and encrypted message *e(T,L)*.

Variables: X is from domain *Dagents*.
L is from domain *Dkeys*.
T, T1 and T2 are arbitrary words (from domain *Dwords*).

And the same rules in database view:

```
[I’.1]      INSERT INTO Messages
            VALUES (SELECT M.Agent, e-1(M.Word,K.Key)
                    FROM Messages M, Keys K WHERE M.Agent=K.Agent)
[I.’2]      INSERT INTO Messages
            VALUES (SELECT M.Agent, e(M.Word,K.Key)
                    FROM Messages M, Keys K WHERE M.Agent=K.Agent)
[I’.3]      INSERT INTO Messages
            VALUES (SELECT M.Agent, f(M.Word) FROM Messages M)
[I’.4]      INSERT INTO Messages
            VALUES (SELECT M.Agent, f-1(M.Word) FROM Messages M)
[I’.5]      INSERT INTO Messages
            VALUES (SELECT M.Agent, T1) FROM Messages M
                    WHERE M.Word = T1+T2)
            INSERT INTO Messages
            VALUES (SELECT M.Agent, T2 FROM Messages M
                   WHERE M.Word=T1+T2)
[I’.6]      INSERT INTO Messages
            VALUES (SELECT M1.Agent, M1.Word+M2.Word
                    FROM Messages M1, Messages M2
                   WHERE M1.Agent=M2.Agent)
```

At the beginning every relation has some initial data. Initial data are some base data that have any participants as identifiers of all participants, private key shared with key server, information that legal the adversary knows from previous legitimate evaluation of the protocol with the others participants and others.

So we have environment of the model and protocol-independent cryptographic rules and relational rules that rise set of knowledge. We also need to have the rules that represent protocol. At the beginning we present as an example the transformation of Needham-Schroeder protocol.

1. $a \rightarrow s : a, b, n$
2. $s \rightarrow a : \{n, b, k, \{kab, a\}_{kb}\}_{ka}$
3. $a \rightarrow b : \{k, a\}_{kb}$
4. $b \rightarrow a : \{m\}_{kab}$
5. $a \rightarrow b : \{m-1\}_{kab}$

6a. $a \rightarrow b : \{p\}_{kab}$

6b. $b \rightarrow a : \{q\}_{kab}$

Fig. 2.3. Needham-Schroeder protocol

In the first message, one of the participants, *a*, sends a message to the key server *s*, indicating that *a* whishes to initiate communication with participant *b*. Agent *a* includes a nonce n_i, which a uses to identify the response to its request. Server s then responds with a message encrypted under the key, that a and s share. This message contains nonce of agent a, the new key k_i, and a message encrypted under the key, that b and s share. Agent a will forward this message to b. Next step is that a forwards this message to b, which contains the new key and an indication that it is to be used for communication with a. The next two messages are "confirmation" that the participants have received the new key. Agent b sends a new nonce m_i to a, encrypted under new key. Agent a then decrypts the message and replies with a handshake. Then the agents proceed with their session. We assume that the session key will be used at some point by either or both agents to encrypt information, that is to be kept secret from the adversary. We model this secret information explicitly as p and q. Messages 6a and 6b are added for analysis and are not part of the original version of the protocol.

For this purpose (transformation protocol into rules) we designed program *"Converter"* that converts protocol given in form in Figure 2.3 into rules similar given in Figure 2.4. The program is written in Delphi version 3.0 and the input protocol must be written in text file. The program read the protocol from input file for verification also shows what was read and then it is possible to do the translation. To have the same rules as in Figure 2.4 we need to make some modification. These modifications deal with using variables in rules. So the process of transformation is not fully automated yet.

[P'.1] *Messages(X, [a, b, n]):- X=a v X=b v X=i v X=s.*
[P'.2] *Messages(X, e([N, Z,K, e([K, Y], KZ)], KY)):-Messages(s,[Y, Z, N])&*
&Com_Nonces(Y,Z,N)&Com_keys(s,Y,KY)&Com_Keys(s,Z,KZ)
&Com_Keys(Z,Y,K).
[P'.3] *Messages(X, S):-Keys(Z, L)&Messages(Z, e([n, b, K, S],L)).*
[P'.4] *Messages(X, e(m, K)):-Keys(Z, L)&Messages(Z, e([K, U], L)).*
[P'.5] *Messages(X, e(f(M), K)):-Keys(Z, K)&Keys(Z, L)&*
&Messages(Z, e([n, b, K, S], L))&Messages(Z, e(M, K)).
[P'.6] *Messages(X, e(P, K)):- Keys(Z, K)&Keys(Z, L)&*
&Messages(Z, e([n, b, K, S], L))&Messages(Z, e(M, K)).
[P'.7] *Messages(X, e(Q, K)):-Keys(Z, K)&Keys(Z, L)&*
&Messages(Z, e(f(m), K))&Messages(Z, e([K, U], L)).

Fig. 2.4. Protocol specification inference rules (Needham-Schroeder protocol)

Constants: a, b, i, s - from domain *Dagents*
m, n – nonces from *Dnonces*
kab – session key between agents a ,b from domain *Dkeys*
Variables: X, Y, Z, U from domain *Dagents*
K, KZ, KY, L from domain *Dkeys*
P, Q, S arbitrary word from domain *Dwords*
M, N from domain *Dnonces*

And the as the example the rule [P'.2] in database view:

```
[P'.2]     INSERT INTO Messages
           VALUES (SELECT M.Agent, e( [N.Nonce,K.Agent1,
                     K.Key,e([K.Key,N.agent1],K2.Key),],
  K1.Key )
FROM Messages M, Com_Keys K1,
     Com_Keys K2, Com_Keys K,
     Com_Nonces N
WHERE M.Agent=s AND
      M.Messages[1]=K1.Agent2 AND
      M.Messages[1]=K.Agent2 AND
      M.Messages[1]=N.Agent1 AND
      M.Messages[2]=N.Agent2 AND
      M.Messages[2]=K2.Agent2 AND
      M.Messages[2]=K.Agent1 AND
      K1.Agent1=s AND K2.agent1=s AND
      M.Messages[3]=N.Nonce)
```

In Fig. 2.4 there is description of protocol that doesn't allow to alter messages for an adversary. If adversary can also modify messages, then adversary *i* first of all participants knows all sent messages and can inquire and modify them. So we must modify the protocol rules in Fig.2.4 into rules in Fig. 2.5 and by modification we say what words in massages can be changed by the adversary and how. In next section if we will deal with evaluation of the protocol we will always deal with the protocol rules where the adversary can change messages.

[P.1] *Messages(X, [a, b, n]):- X=a v X=i.*
Messages(X, [Y, Z,N]):- Agents(i, Y)&Agents(i, Z)&Nonces(i, N)&Messages(i,[a,b,n]).
[P.2] *Messages(i, e([N, Z, K, e([K, Y], KZ)], KY)):-Messages(s,[Y, Z, N])&*
&Com_Nonces(Y, Z, N)&Com_keys(s, Y, KY)&Com_Keys(s, Z, KZ)&Com_Keys(Z, Y, K).
Messages(X, e([M, U, k, e([k, V], K(U))], K(V))):- Messages(i, e([N, Z, k, e([k, Y], K(Z))], K(Y)))&
&Messages(i, e([M, U, k, e([k, V], K(U))], K(V))&.&Nonce(M)&Agent(U)&Agent(V).
[P.3] *Messages(i, S):-Keys(Z, L)&Messages(Z, e([n, b, K, S],L)).*
Messages(X, T):-Messages(i, S)& Messages(i, T).
[P.4] *Messages(i, e(m, K)):-Keys(Z, L)&Messages(Z, e([K, U], L)).*
Messages(X, e(M, L)):-Messages(i, e(m, K))&Messages(i,e(M, L))& Nonce(M)&Key(L).
[P.5] *Messages(i, e(f(M), K)):-Keys(Z, K)&Keys(Z, L)&Messages(Z, e([n, b, K, S], L))&*
&Messages(Z, e(M, K)).
Messages(X, e(f(N), L)):-Messages(i, e(f(M), K))& Messages(i, e(f(N), L))& Nonce(N)&Key(L).
[P.6] *Messages(i, e(p, K)):- Keys(Z, K)&Keys(Z, L)&Messages(Z, e([n, b, K, S], L))&*
&Messages(Z, e(M, K)).
Messages(X, e(Q,, L)):-Messages(i, e(p, K))& Messages(i, e(Q,, L))&Key(L).
[P.7] *Messages(i, e(q, K)):-Keys(Z, K)&Keys(Z, L)&Messages(Z, e(f(m), K))&*
&Messages(Z, e([K, U], L)).
Messages(X, e(P, L)):-Messages(i, e(q, K))& Messages(i, e(P, L))&Key(L).

Fig. 2.5. Protocol specification inference rules (Needham-Schroeder protocol)

So if we want to express an attack (for example that adversary is able to learn message *m*) we represent it by the formula *Messages(i,m)* or in databases as "`SELECT count(agent) FROM Messages WHERE agent=i AND word=m`".

3 Theoretical results

We assume the reader is familiar with the basics of Logic Programming (as in [28]) and Databases and Knowledge-Base Systems (as in [36]).

3.1 Evaluation of analysis

Now we have all what we need to start to analyze some security or cryptographic protocol. As it was said, the analysis consists of two steps:

1. To make the closure of knowledge and to try to find an attack with the closure (to ask whether or not is possible an attack)
2. And if no attack is possible then to state conditions when it should be possible to make an attack. To find additional knowledge needed to make an attack.

Step 1: In first step we use *datalog* evaluation This is done with *rules* and *facts* defined with logic program *F*. Facts are the input data, and rules can be used to derive more facts, and hopefully, the solution of the given problem. The declarative programming language datalog, which is known for being a convenient tool for knowledge representation. Datalog is the deductive database language and can therefore be seen as a way to query data from databases. Datalog is strictly more powerful than for example SQL (everything that can be done with the core SQL language can also be done with datalog, and more), but it is also often described as a system for answer set programming (ASP). This is a powerful new paradigm from the area of "Nonmonotonic Reasoning" which allows to formulate even very complicated problems in a straightforward and highly declarative way. One may call this paradigm even more declarative than classical logic.

After preparing rules and initial data for protocol we start evaluation in datalog to generate the closure of relations in our model. After this evaluations we have in relations all knowledge that can participants learn from traffic. In this point we can ask on some knowledge. The most interesting knowledge is the adversary's knowledge. There are five most important questions:

a) Does adversary know the session key *kab* between participants *a* and *b*?
b) Does adversary know the secret information ***p*** sent by participant ***a*** to participant ***b***?
c) Does adversary know the secret information ***q*** sent by participant ***b*** to participant ***a***?
d) Does the participant ***a*** know the word ***q*** sent by the participant ***b*** with new secret session key?
e) Does the participant ***b*** know the word ***p*** sent by the participant ***a*** with new secret session key?

Example:

Question: Does the adversary *i* have (the same as knows) the session key *kab* that server produced for secure communication between participants *a* and *b*?

These questions can be written in different formalisms:

1. From database point of view :
```
SELECT count(agent)
FROM Messages M, Keys K
WHERE M.agent=i AND M.agent=K.agent AND
      M.word=K.Key AND K.key=kab”
```
2. From relational algebraic point of view: (i,kab) $\in$ Messages^{+}

There exist two possible answers – “*yes*” or “*no*” on these questions. In formalism 1 answer “*yes*” means that select returns at least one row (it depends on structure of table *Messages*) and in formalism 2 it means that ordered pair *(i,kab)* is member of relation *Messages*.

If answer is “*no*” we start with second step of our evaluation, because it is useful to know in what cases can answer be true.

Step 2: In this moment we use *abduction* to search conditions that must hold to answer should be “*yes*”. Abduction is an important form of non-monotonic reasoning allowing one to find explanations for certain symptoms or observations. Abduction is the process of reasoning to explanations for a given observation according to a general theory that describes the problem domain of the application. The problem is represented by an abductive theory. In **A**bductive **L**ogic **P**rogramming (**ALP**) an abductive theory is defined in chapter 3.2.2. In our case observations are questions on security properties of protocol that we try to analyze. The background knowledge (or assumptions) is evaluated closure of relation *Messages* and rules for abduction are the same rules used in first step by evaluation of closure of relation *Messages*.

3.2 Procedural and declarative semantics

3.2.1 Classical Logic Programming approach

We describe the adversary’s reasoning using a set of formulae F_C. The set F_C is logic program, that represent evaluation of communication protocol *C*. The set F_C consists of four parts $F_C=F_1 \cup F_2 \cup F_3 \cup F_4$. The set of facts F_1 represents initial knowledge of participants, the set of rules F_2 represents relation rules [R.1]-[R.4] (or additional, it depends on protocol), the set of rules F_3 represents the protocol-independent cryptographic rules [I’.1]-[I’.7] and finally the set of rules F_4 represents the protocol specification rules (exactly description of protocol's evaluation).

Definition 1 (L-computed): Let F_C be a logic program that interprets evaluation of a communication protocol C, then a substitution Θ is said to be a **L-computed answer** for logic program F_C and query ?-*Messages(X,Y)* if there is a sequence $(G_0,Q_0),\dots,(G_n,Q_n)$ such that $G_0=Messages(X,Y)$ and $G_n=ð$, $\Theta=\Theta_0\dots\Theta_n|Var(G_0)$ and (G_{i+1},Q_{i+1}) is derived from (G_i,Q_i) by one of the inference rules of logic program F_C.

Definition 2 (L-correct): Let F_C be a logic program that interprets evaluation of a communication protocol C, a substitution Θ is an **L-correct answer** for the logic program F_C and query ?-*Messages(X,Y),* if in every model M of F_C holds: $M \models (Messages(X,Y))\,\Theta$.

Theorem 1 (soundness, completeness): Let F_C be a logic program that interprets evaluation of a communication protocol C, then the following holds:
Every L-computed answer is L-correct answer for logic program F_C and query ?-*Messages(X,Y)* (soundness).
Every L-correct answer is L-computed answer for logic program F_C and query ?-*Messages(X,Y)* (completeness).
Proof: Standard Logic Programming (**LP**) theory [28].

Presence of an attack can be expressed as follows (the adversary *i* knows the message *p*):

$$F_C \models Messages(i,p).$$

Definition 3 (L-Safety): Let F_C be a logic program that interprets evaluation of a communication protocol C. Let *p,q* the secret messages that the participants *a,b* of the protocol want to exchange. Let *kab* be a new session key between *a* and *b*.Let *i* be the adversary. Then the protocol *C* represented by logic program F_C is **L-computionaly safe**, if there is no computed answer for logic program F_C and queries:

a) ?-*Messages(i,kab)*
b) ?-*Messages(i,p)*
c) ?-*Messages(i,q)*

and if there exists computed answer for logic program F_C and queries:

a) ?-*Messages(a,q)*
b) ?-*Messages(b,p).*

3.2.2 Introduction of function Depth_ into classical LP

Now we have whole environment but there are some weaknesses. If we will have protocol-independent rules in form as in Fig. 1 by reasoning it will lead to infinite cycle, especially rules [I'.2], [I'.3] and [I'.7]. It is caused by every new word it is possible to use one of these rules and we get new word and on this new word we can again use one of these rules and so on.... To prevent this infinite cycle we use for description of every protocol constant *MAX* (positive integer) that states the maximal relevant depth of used encryption, hash and handshake in protocol messages plus one. The purpose of that constant is to reduce the infinite search space. It means that at the beginning we don't have only set *F* but also the constant *MAX*, and this constant changes the protocol-independent rules into new form:

[I.2] *Messages(X, e(T, L)):-Messages(X, T)&Keys(X, L)&DepthE(T)<Max*
[I.3] *Messages(X, f(T)):-Messages(X, T) &DepthF(T)<Max*
[I.7] *Messages(X, h(T, L)):-Messages(X, T)&Keys(X, L) &DepthH(T)<Max*

Fig. 3.1. Changed protocol-independent inference rules

The function *Depth_(X)* expresses how many times are in the word *X* used function *e, f* or *h.* For example:

X= *e(m,k)* *DepthE(X)* = 1
X= *e([A,B,N,k,e(A,kb)],ka)* *DepthE(X)* = 2

With using this restriction we have a new model and the adversary's reasoning we now describe using a set of formulae H_C. The set H_C consists of four parts $H_C = F_1 \cup F_2 \cup H_3 \cup F_4$. H_3 represents the modified protocol-independent rules [I.1]-[I.7] restricted with constant *MAX*. The sets F_1,F_2 and F_4 remain unchanged. Presence of an attack can be expressed as:

$$H_C \models Messages(i,p).$$

Definition 4 (LDepth-computed answer): Let H_C to be a logic program, that interprets evaluation of communicating protocol *C*, $H_C=F_1 \cup F_2 \cup H_3 \cup F_4$ reduced by constant $MAX=n$, $n \in N$, the substitution Θ is said to be a **LDepth-computed answer** for logic program H_C and query ?-*Messages(X,Y)* if there is a sequence $(G_0,Q_0),\ldots,(G_n,Q_n)$ such that $G_0=Messages(X,Y)$ and G_n=ð, $\Theta=\Theta_0\ldots\Theta_n|Var(G_0)$.

Know we need to prove that this reduction of using protocol-independent rules doesn't change the set of results.

***Theorem 2*:** Consider $F_C= F_1 \cup F_2 \cup F_3 \cup F_4$ and $H_C = F_1 \cup F_2 \cup H_3 \cup F_4$

1. If Θ is LDepth-computed answer for H_C and query ?-*Messages(X,Y)* then Θ is also L-computed answer for F_C and query ?-*Messages(X,Y).*

2. If Θ is L-computed answer for F_C and query ?-*Messages(X,Y)* then Θ is also LDepth-computed answer for H_C and query ?-*Messages(X,Y).*

Proof: $H_C = F_1 \cup F_2 \cup H_3 \cup F_4$ and $F=F_1 \cup F_2 \cup F_3 \cup F_4$

1. So it is clear that if we have a substitution Θ the LDepth-computed answer for query ?-*Messages(X,Y)* in reduced program H_C then Θ is also L-computed answer for this query in F_C, because only distinction between F_C and H_C is in parts H_3 and F_3. So if in evaluation the step from G_i to G_{i+1} is done with the rule from H_3 then it is also possible with the rule from F_3.(the body of such rule is smaller)

Conclusion: Every LDepth-computed answer in H_C is also L-computed answer in F_C.

2. Now we need to show that there don't exist such enumeration that is in F_C and is not in H_C. We try it to prove by confrontation.

   Consider, we have computation $G=G_0,...,G_n$ that exists in F_C and doesn't exist in H_C. Computation G is a path in computation tree of logic program F_C that describes reasoning from initial knowledge to an attack and from G_i to G_{i+1} was used rule r_i from logic program F_C.

   If G is in F_C and is not in H_C then it must in the path $r_1....r_m$ exists the index j such that rule r_j that can be used by evaluation in F_C but cannot be used in G_C.

   What a rule can be the rule r_j?

   a) $r_j \in F_2$ – is not true because in this case r_j will be evaluated also in H_C, so it holds original theorem.

   b) $r_j \in F_3$ – Consider it's true. Some of the rules from F_3 aren't in H_C because of reduction in H_C. If there is not any evaluation in H_C, it is because of reduction. That means that by reasoning the path of attack we used the rules for encryption, hash or handshake more then *MAX* times (*MAX* = the maximal relevant depth of used encryption, hash and handshake in protocol messages plus one). We have the enumeration, it means that we have an attack so if the adversary is able to learn the content of any encrypted message or session key than he learns that all only from the protocol messages. But in the protocol messages there are not used words with *Depth_* more than *MAX*. There are two way how can words with depth more than *MAX* came into being and than the adversary obtained it:

      1. Any of the legitimate users used some of the rules from F_3 more than *MAX* times and then sent it in some of the protocol messages. – it is not true because of in the protocol messages there are not used words with *Depth_* more than *MAX*.
      2. The adversary itself generated this word. It means that he obtained at the beginning of the generation the original word (*Depth_=0*) or word that *Depth_* is less than *MAX*. By the attack it is important to obtain original word not word ciphered by hash, handshake or encryption. So if we have the enumeration and it leads to an attack. It does not have sense by the adversary to generate from obtained word new word with *Depth_* more than *MAX*. Such word can not lead to attack. So it is not true that in enumeration was used some of reduced rules more than *MAX* times.

      Conclusion is that $r_j \in F_3$ is not true.

   c) $r_j \in F_4$ – is not true because in this case r_j will be evaluated also in H_C, so it holds original theorem.

Result: $r_j \notin F_1$ and $r_j \notin F_2$ and $r_j \notin F_3$ so it doesn't exist such rule r_j and it means that we have contradiction to our assumption of existence of the computed answer that is in F_C and is not in H_C.

End of the proof.

3.2.3 Abductive approach

As it was mentioned abduction is a reasoning method (beside deduction and induction) used in artificial intelligence and diagnostic methods. In the step two of our DLA model we used the abduction for setting the conditions of possible attack on the protocol. Let's look how it works. We assume that the reader is common with Abductive Logic Programming [12]

***Definition 5* (*Abductive Theory*):** An abudctive theory in ALP (abductive logic programming) is a triple $<F_C,A,IC>$ where F_C is a logic program (representing evaluation of communication protocol C), A is a set of predicate symbols, called abducibles (in our case predicates about knowledge of keys or key material), which are not defined (or are partially defined) in F_C, and IC is a set of first order closed formulae, called integrity constraints.

In an abductive theory $< F_C,A,IC>$, the program F_C models the basic structure of the problem (in our case describe the evaluation of the protocol C), the abducibles play the role of the answer-holders, for the solutions to particular tasks (goals) in the problem, and the integrity constraints IC represent the validity requirements that any solution must respect. A goal G is a logic programming goal. A solution to a goal G is an abductive explanation of G defined as follows.

***Definition 6* (*A-computed answer*):** A set of instances of abducible predicates Δ is **A-computed answer** (abductive explanation) for abductive theory $<F_C,A,IC>$ and observation $G=Messages(X,Y)$ if there exists a LP computation $G_0 \ldots G_n$ such that $G_0=G$ and $G_n=\Delta\subseteq A$.

***Definition 7* (*A-correct answer*):** An abductive explanation or **A-correct answer** for a goal G is a set Δ of ground abducible formulae which when added to the program F_C imply the goal G and satisfy the integrity constraints in IC, ie.

$$F_C \cup \Delta \models_{\mathrm{lp}} G \quad \text{and } F_C \cup \Delta \models_{\mathrm{lp}} IC$$

where $\models_{\mathrm{lp}}$ is the underlying semantics of Logic Programming.

Theorem 3 (soundness, completeness): In an abductive theory $<F_C,A,IC>$, F_C is a logic program that interprets evaluation of a communication protocol C, then the following holds:
Every A-computed answer is A-correct answer in $<F_C,A,IC>$ and for observation ?-*Messages(X,Y)* (soundness).
Every A-correct answer is A-computed answer in $<F_C,A,IC>$ and for observation ?-*Messages(X,Y)* (completeness).
Proof: Standard Abductive Logic theory [12].

***Definition 8 (ADepth-computed answer)**:* A set of instances of abducible predicates Δ is **ADepth-computed answer** (abductive explanation) for abductive theory $<H_C,A,IC>$, H_C is a logic program, that interprets evaluation of communicating protocol C, $H_C=F_1\cup F_2 \cup H_3 \cup F_4$

reduced by constant $MAX=n$, $n\in N$, and observation $G=Messages(X,Y)$ if there exists a LP computation $G_0... G_n$ such that $G_0=G$ and $G_n=\Delta\subseteq A$.

***Theorem 4(soundness, completeness)*:** Consider $F_C= F_1\cup F_2 \cup F_3 \cup F_4$ and $H_C = F_1\cup F_2 \cup H_3 \cup F_4$

1. If a set of abducible predicates Δ is ADepth-computed answer for $<H_C,A,IC>$ and observation ?-*Messages(X,Y)* then Δ is also A-computed answer for $<F_C,A,IC>$ and observation ?-*Messages(X,Y).*

2. If a set of abducible predicates Δ is A-computed answer for $<F_C,A,IC>$ and observation ?-*Messages(X,Y)* then Δ is also ADepth-computed answer for $<H_C,A,IC>$ and observation ?-*Messages(X,Y).*

Proof: Analogous as proof of theorem 2.

The computational process for deriving the abductive solution (explanation) consists of two interleaving phases, called the abductive and consistency phases. In an abductive phase, hypotheses on the abducible predicates are generated, by reducing the goals through model of the problem in *F*, thus forming a possible solution set. A consistency phase checks whether these hypotheses are consistent with respect to the integrity constraints. During a consistency phase it is possible for new goals to be generated, if these are needed in order to ensure that the hypotheses so far can indeed satisfy the integrity constraints. In turn these new goals can generate further abducible assumptions, to be added to the solution set. It is also possible that the consistency phase refines the solution set of assumptions generated originally – by setting constraints on the existential variables involved in the abducible assumptions – when this restriction can help ensure the satisfaction of (some of) the integrity constraints. In our case the integrity constraints is the empty set.

3.3 Soundness and completeness DLA method

As it was said the DLA method is based on the combination of logic based data model called ***datalog*** and ***abduction*.** We deal with abduction in previous chapter, now we will talk about datalog. The name datalog was coined to suggest a version of Prolog suitable for database systems. The underlying mathematical model of data for datalog is essentially that of relational model. Predicate symbols in datalog denote relations. The classical dalalog doesn't allow function symbols.

3.3.1 Classical datalog

Datalog programs are built form *atomic formulae*, which are predicate symbols with a list of arguments, e.g., $p(A_1,...,A_n)$, where *p* is the predicate symbol. An argument in datalog can be either a variable or a constant.

In the logical view on databases, computing the answer to a query q from database means to find all objects o for which formula $q \neg\ o$ is true, detailed, such that the rule $r_0 \neg r_1 \& ... \& r_n$ is true. The usual approach to answer such queries in classical deductive databases is following (for details see [36]):

1. rectification of rules. Instead of $r_0(c_1,...,c_n) \neg\ r_1 \& ... \& r_n$ work with the rule $r_0(X_1,...,X_n) \neg X_1=c_1 \& \quad \& X_n=c_n \& r_1 \& ... \& r_n$ – means after that in the head of rule cannot be constants only variables.
2. Having such rule and relations $R_1,...,R_n$ interpreting predicates $r_1,...,r_n$ we use $X_1=c_1 \& \quad \& X_n=c_n$ as selection condition and the body of rule is then interpreted by the join $><(s_c(R_1),...,s_c(R_n))$ and the head is just projection to attributes of r_0.
3. Multiple rectified rules $r_0^i \leftarrow r_1^i,...,r_{n_i}^i$ have the same head and hence their simultaneous representation is union of them.

$$R_0 = \bigcup_{i=1}^{K} (\prod_X (>< (s_c(R_1^i),..., s_c(R_{n_i}^i))))$$

4. In such way we get m relational equations with m unknown relations. These m unknown relations are intensional predicates. Extensional relations are those appearing only in bodies. The system of equations

$$R_0=f_0(R_0, R_1,...,R_m)$$
$$R_1=f_1(R_0, R_1,..., R_i,...,R_m)$$
$$R_m=f_m(R_0, R_1,...,R_m)$$

is solved using the smallest fixpoint of the production operator. Of cause this fixpoint should be computable.

***Definition 9 (T_P operator)*:** Operator defined by solution the system equations

$$R_0=f_0(R_0, R_1,...,R_m)$$
$$R_1=f_1(R_0, R_1,..., R_i,...,R_m)$$
$$R_m=f_m(R_0, R_1,...,R_m)$$

is called T_P **operator.**

***Definition 10 (D-correct answer)*:** Assume $R_1,...,R_n$ are interpretation of predicate symbols $r_1,...,r_n$ and F_C is a logic program, then the relation R is a **D-correct answer** for $R_1,...,R_n$ and query r if for all tuples $(a_1,...,a_n) \in R$ the substitution $\Theta=\{x_1/a_1,..., x_n/a_n\}$ is an L-correct answer for the program F_C extended by facts $R_1,...,R_n$ and query $r_0(x_1,...,x_n)$.

***Definition 11 (D-computed answer)*:** Assume I is the smallest fixpoint of the operator T_P than $T_P(I)(r_0)$ is a **D-computed answer**.

***Theorem 5 (soundness, completeness)*:** Let P to be a datalog program, then the following holds:

Every D-computed answer is D-correct answer for datalog program P (soundness).

Every D-correct answer is D-computed answer for datalog program P (completeness).

Proof: Standard Datalog Programming (**LP**) theory [36].

3.3.2 DLA – Datalog with function symbols

In our DLA method we need to extend the classical datalog, because the classical datalog doesn't use the function symbols. This extension could leads to infinite relations. To prevent this we use the function symbols only limited. The restriction in the rules with the constant *MAX* and functions *Depth_* is done to limited relations before infinity. The restriction is the same as in chapter 4.1. This change means, that we change the classical datalog production operator.

Definition 12 (DLA-computed answer): Let H_C to be a extended datalog program that interprets evaluation of a communication protocol *C*, the fixpoint (closure) of relation *Messages* is said to be a **DLA-computed answer** for extended datalog program H_C .

Definition 13 (DLA-correct answer): Let H_C to be a extended datalog program (the same as logic program restricted with constant *MAX=n*, *n* is positive integer) that interprets evaluation of a communication protocol *C*. Assume $R_1,...,R_n$ are interpretation of predicate symbols $r_1,...,r_n$. Then the relation *R* is a **DLA-correct answer** for $R_1,...,R_n$ and query *r* if for all tuples $(a_1,...,a_n)\in R$ the substitution $\Theta=\{x_1/a_1,..., x_n/a_n\}$ is an LDepth-correct answer for the program H_C extended by facts $R_1,...,R_n$ and query $r_0(x_1,...,x_n)$.

***Theorem 6 (soundness, completeness)*:** Let H_C to be a logic program that interprets evaluation of a communication protocol *C* (H_C has a function symbols and is restricted with constant *MAX=n*, *n* is positive integer), then the following holds:
Every DLA-computed answer is DLA-correct answer for extended datalog program H_C (soundness).
Every DLA-correct answer is DLA-computed answer for extended datalog program F_C (completeness).
Proof: Intuitively. The only distinction between classical datalog and our extended datalog is defining the function symbols. As it was mentioned this can cause by evaluation of relations that the relation can be infinite. Shortly that there doesn't exist the fixpoint. But we also make the restriction of using such rules with function symbols and we introduce reduction constant *MAX*. So introduction function symbols into datalog program cannot cause that such extended datalog program don't have the fixpoints.
End of the proof.

3.3.3 DLA datalog safety

Definition 14 (DLA datalog safety): Let H_C be a logic program that interprets evaluation of a communication protocol *C*. Let *p,q* the secret messages that the participants *a,b* of the protocol want to exchange. Let *kab* be a new session key between ***a*** and *b*. Let ***i*** be the adversary. Let the intepretation ***I*** be the DLA-computed answer for relation *Messages* and program H_C. The protocol *C* represented by logic program H_C is **DLA-datalog computionaly safe**, if in the implementation ***I*** of the relation *Messages* by the program H_C holds:

$$Messages^I \supseteq \{(a,q), (b,p)\} \text{ and } Messages^I \cap \{(i,kab),(i,p),(i,q)\} = \emptyset.$$

3.3.4 DLA abduction safety

Definition 15 (DLA abduction safety): Let H_C be a logic program that interprets evaluation of a communication protocol *C*. Let $<H_C,A,IC>$ be abductive theory. Let ***p,q*** the secret messages that the participants ***a,b*** of the protocol want to exchange. Let *kab* be a new session key between ***a*** and *b*. Let ***i*** be the adversary. Then the protocol *C* represented by logic program H_C is **DLA-abduction computionaly safe**, if there is only such ADepth-computed answer Δ for abductive theory $<H_C,A,IC>$ and observations *Messages(i,kab), Messages(i,p), Messages(i,q)* that in Δ is only instances of the predicate *Keys* in form *Keys(i,X),* where *X* is a key.

4 Various aspects of analysis of security protocols

Datalog is a *declarative* (programming) language. At the beginning we want to use for evaluation any datalog engine and modify it for our purposes, while classical datalog don't support the function symbols and concatenation and projection of list of elements. But we found only engines without source codes so for generation of closure of relations we used the Amzi Logic Interpreter with the same rules that we plan to use in datalog and which were presented in previous chapter.

4.1 Needham-Schroeder protocol

In Fig. 4.1 is shown evaluation of Needham-Schroeder protocol.

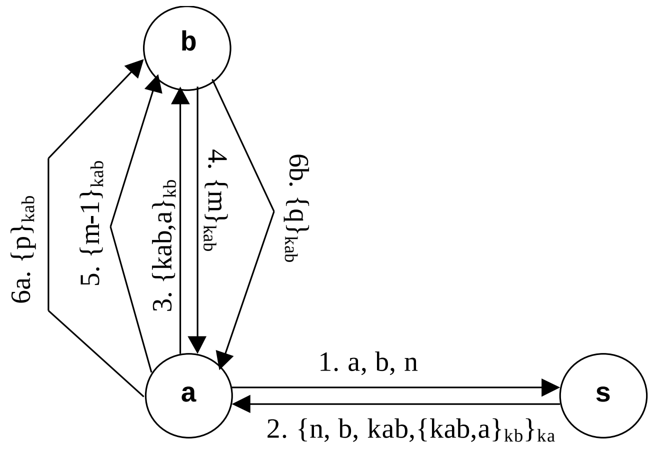


Fig. 4.1. Needham-Schroeder protocol

The description of Needham-Schroeder protocol in rules you can see in Figures 2.3 and 2.5 in Chapter 2.

Our analysis shows that directly from protocol the adversary cannot learn content of any encrypted message and cannot obtain the session key. Conclusion is that we didn't find any oracle flaw in protocol as in analyses by another methods. So we try to state conditions for possible attacks by abduction. Abduction for question *(i,kab)* is part of abduction for question *(i,p)* or *(i,q)* so in Fig. 4.2 is shown evaluation of abduction only for question *(i,p)*. Abduction for *(i,q)* is very similar to abduction for *(i,p)*.

[I.1]Messages (i , p) :- Messages (i , e (p, L) & Keys (i , L)
L from domain *Skey* => L must befrom {kab, ka, kb, ki} :
L = kab
1. Messages (i , p) :- Messages (i , e (p, kab) & Keys (i , kab)
1.1 Messages (i , e (p, kab)
(i,e(p,kab)) is in relation *Messages*
1.2 Keys(i, kab)
[R.2]1.2.1 Keys(i, kab):- Messages(i, kab)&Key(kab).
kab is in relation *Key*
[I.5] 1.2.2 Messages (i, kab) :- Messages (i , [kab, a]).
[I.1] 1.2.3 Messages (i, [kab, a]) :- Messages (i , e([kab, a], kb)) & Keys(i, kb).
1.2.3.1 Messages (i , e([kab, a], kb)).
(i, e([kab, a],kb)) is in Messages
1.2.3.2 Keys(i, kb).
And this fact is possible to reach (for adversary) only by cryptoanalysis.

Result: ***Messages(i, p) <= Keys (i, kb).***

Fig. 4.2. Abduction for question *(i,p)* in Needham-Schroeder protocol

The results of the abduction show that the adversary can do any attack only with the knowledge of the keys that participants share with the key server. It means that safety of the protocol depends on the strength of the keys. From our point the protocol is safe, we didn’t find any attack on its.

4.2 “Optimized” Otway-Rees protocol

For demonstration possibilities of our method we also analyze optimized Otway-Rees protocol that was optimized by Dexter in [7]. The optimized protocol is shown in Fig. 4.3.

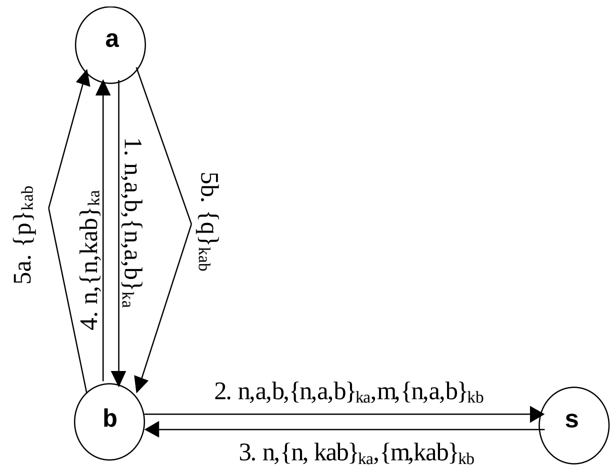


Fig. 4.3. “Optimized” Otway-Rees protocol

4.2.1 Results of analysis

In this protocol there exists the attack and our method found this attack. The attacker is possible to produce all messages as the participant ***a*** and at the end the attacker send $\{r\}_{kab}$. When we focus on the protocol messages that the attacker needs to generate, it is sufficient to produce messages $\{m,k\}_{kb}$ and $\{r\}_{kab}$ and the participant ***b*** will believe that he is talking with the participant ***a***. So how does the attacker produce these messages?

- $\{r\}_{kab}$ – the attacker needs to know ***r*** and ***kab***, but ***r*** is arbitrary word. So the attacker needs only to obtain ***kab***. The attacker can learn the key ***kab*** from the third message of the protocol. In this message the attacker learn $\{U,k\}_{ka}$ and he needs to know the key ***ka***. The attacker can learn whole third message if he is possible to produce second message - $[X, A, b, \{X, A, b\}_K, m, \{X, A, b\}_{kb}]$.

First of all the attacker must have all initial knowledge and this includes also knowledge from legitimate protocol evaluation of the attacker as the legitimate participant ***c*** with the participant ***b***.

1. $c \rightarrow b$: $n,c,b,\{n,c,b\}_{kc}$
2. $b \rightarrow s$: $n,c,b,\{n,c,b\}_{kc},m_1,\{n,c,b\}_{kb}$
3. $s \rightarrow b$: $n,\{n,kcb_1\}_{kc},\{m_1,kcb_1\}_{kb}$
4. $b \rightarrow c$: $n,\{n,kcb_1\}_{kc}$

Fig. 4.4. Legitimate protocol evaluation of the attacker

From this legitimate evaluation the attacker learns $\{n,c,b\}_{kc}$ and $\{n,c,b\}_{kb}$ and now he can start new evaluation with the participant ***b***, where the attacker will masquerade as the participant ***a***:

1. $i \rightarrow b$: $n,a,b,\{n,c,b\}_{kc}$
2. $b \rightarrow s$: $n,a,b,\{n,c,b\}_{kc},m_2,\{n,a,b\}_{kb}$ – but this message is intercepted by the attacker and the attacker modifies it

   $i \rightarrow s$: $n,\mathbf{c},b,\{n,c,b\}_{kc},m_2,\mathbf{\{n,c,b\}_{kb}}$
3. $s \rightarrow b$: $n,\{n,kcb_2\}_{kc},\{m,kcb_2\}_{kb}$
4. $b \rightarrow a$: $n,\{n,kcb_2\}_{kc}$ – this message is again intercepted by the attacker and is not delivered to the participant *a*

Fig. 4.5. The attack on the “optimized” Otway-Rees protocol

And when the participant ***b*** receives third message from the key server ***s*** he will believe that he talks with the participant ***a***. And the communication key kcb_2 is shared also with the participant ***a***.

The protocol specific rules for evaluation:

[P.1] *Messages(X, [o, a, b, e([o,a,b], ka)]):- X=a v X=i.*

Messages(X,[G, a, b, e([G, B, b], L)]):- Messages(i, [o,a,b,e([o,a,b],ka)])&Nonces(i,G)&´
&Agents(i,B)&Keys(i,L)&Messages(i,[G,a,b,e([G, B, b],L)]).

[P.2] *Messages(i,[G,A,b,Y),F,e([G,A,b],kb)]):-Messages(b,[G, A, b, Y])&Nonces(i,G)&Agent(i,A)&*
&Com_Nonces(b,A,F).

Messages(X,[G,B,b,Y,F,e([I,C,b],kb)]):-Messages(i,[G,A,b,Y,F,e([G,A,b], kb)])&
&Messages(i,e([I,C,b],kb))& Nonce(G)&Nonce(F)&Agent(a)&
&Nonces(i,H)&Nonces(i,J)&Nonces(i,I)&Agents(i,B)&Agents(i,C).

[P.3] *Messages(i,[G,e([G,L],K),e([F,L],kb)]):-Com_Keys(s,A,K)&Com_keys(b,A,L)&*
& Messages(s, [G, A, b, e([G, A, b],K), F, e([G, A, b], kb)])&
&Messages(s, [G, e([G, L],K), e([F, L],kb)]).

Messages(X,[H,e([H,P],K1),e([I,P],kb)]):- Messages(i,[G,e([G, L],K), e([F, L],kb)])&
&Messages(i, [H, e([H, P],K1), e([I, P],kb)])&Key(P)&
&Key(K1)&Key(L)&Key(K)&Nonce(G)&Nonce(F)&
&Nonce(H)&Nonce(I) .

[P.4] *Messages(i,[G,M]):-Messages(b,[G,A,b,Y])&Messages(b,[G,M,e([F,L],kb)])&Key(L)&*
&Com_Nonces(b, A, F)& &Nonce(G)&Agent(A).

Messages(X, [F, M]):-Messages(i, [G,M])&Messages(i,[F,M])&Nonce(G)&Nonces(i,F).

[P.5] *Messages(i,e(p,L)):-Com_Keys(s,A,K)&Com_keys(b,A,L)&*
&Messages(s, [G, A, b, e([G, A, b],K), F, e([G, A, b], kb)])&
&Messages(s, [G, e([G, L],K), e([F, L],kb)]).

Messages(X,e(Q,L)):- Messages(i,e(p,,K))&Messages(i,e(Q,L))&Key(L).

[P.6] *Messages(i,e(q,L)):-Messages(b,[G,A,b,Y])&Messages(b,[G,M,e([F,L],kb)])&&Key(L)&*
&Com_Nonces(b, A, F)& &Nonce(G)&Agent(A).

Messages(X, e(P,L)):-Messages(i, e(q,K))&Messages(i,e(P,L))&Key(L).

4.2.2 How to make a fault protocol secure

The problem in the protocol is that server cannot identify the freshness of part of message encrypted by the participant *b*. This is caused by the nonce of the participant *b* that is not included into $\{n,a,b\}_{kb}$ in the second message and so for the attacker it is easy to change that word in second messages for old one. If the word will look like $\{m,a,b\}_{kb}$ then it is impossible for the attacker change the second message (only if the attacker will have the key *b* from cryptoanalysis) and masquerade as the participant ***a***. After this change the second message looks like (whole protocol is shown on Fig. 4.6): $[n, a, b, \{n, a, b\}_{ka}, m, \{ m, a, b\}_{kb}]$.

```
1. a → b : n,a,b,{n,a,b}ka
2. b → s : n,a,b,{n,a,b}ka,m,{m,a,b}kb
3. s → b : n,{n,kab}ka,{m,kab}kb
4. b → a : n,{n,kab}ka
```

Fig. 4.6. Modification of nonce in “optimized” Otway-Rees protocol

In this case we can try to find the conditions of some attack by abduction. But the only conditions that abduction found was that the attacker need to know the keys shared with the key server. So the safety of the protocol depends on the strength of the keys.

```
[I.1]Messages ( i , p ) :- Messages ( i , e ( p, K ) & Keys ( i , K )
     K from domain Skey  =>  K must befrom {kab, ka, kb, ki} :
     K = kab
     1. Messages ( i , p ) :- Messages ( i , e ( p, kab ) & Keys ( i , kab )
              1.1 Messages ( i , e ( p, kab )
              (i,e(p,kab)) is in relation Messages
              1.2 Keys( i, kab )
              [R.2]1.2.1 Keys( i, kab ):- Messages( i, kab )&Key(kab).
                   kab is in relation Key
                   In this point there are two possibilities:
              [I.5] 1.2.2 a) Messages ( i, kab ) :- Messages ( i , [ n, kab ] ).
                    1.2.2 b) Messages ( i, kab ) :- Messages ( i , [ m, kab ] ).
          a)  [I.1] 1.2.3 Messages ( i, [ n, kab] ) :- Messages ( i , e( [ n, kab ], L )) & Keys( i, L ).
                                      L=ka
                                      1.2.3.1 Messages ( i , e( [ n, kab ], ka )).
                                      (i, e( [ n, kab],ka)) is in relation Messages
                                      1.2.3.2 Keys( i, ka ).
                                      And this fact is possible to reach (for adversary) only by
                                      cryptoanalysis.
Result:   Messages( i, p ) <= Keys ( i, ka ).
          b)  [I.1] 1.2.3 Messages ( i, [ m, kab] ) :- Messages ( i , e( [ m, kab ], L )) & Keys( i, L ).
                                      L=kb
                                      1.2.3.1 Messages ( i , e( [ n, kab ], kb )).
                                      (i, e( [ n, kab],kb)) is in relation Messages
                                      1.2.3.2 Keys( i, kb ).
                                      And this fact is possible to reach (for adversary) only by
                                      cryptoanalysis.
Result:   Messages( i, p ) <= Keys ( i, kb ).
```

Fig. 4.7. Abduction for question *(i,p)* in modified "optimized" Otway-Rees protocol

Another way how to prevent such attack is included into words in the third message the identifiers of the participants. Then the third message looks like(whole protocol is shown on Fig. 4.8):

$$[n, \{b, n, kab\}_{ka}, \{a, n, kab\}_{kb}]$$

In this case the participant *b* after receiving of the third message will know with which agent he will talk with the new session key. So that he really doesn't talk with the participant *a* (by the shown attack).

```
1. a → b : n,a,b,{n,a,b}ka
2. b → s : n,a,b,{n,a,b}ka,m,{n,a,b}kb
3. s → b : n,{b,n,kab}ka,{a,n,kab}kb
4. b → a : n,{b,n,kab}ka
```

Fig. 4.8. Modification of identities in Otway-Rees protocol

After modification it is impossible to make shown attack so now we can evaluate abduction and try to find the conditions for some attack by abduction.

```
    [I.1]Messages ( i , p ) :- Messages ( i , e ( p, K ) & Keys ( i , K )
         K from domain Skey  =>  K must befrom {kab, ka, kb, ki} :
         K = kab
         1. Messages ( i , p ) :- Messages ( i , e ( p, kab ) & Keys ( i , kab )
                    1.1 Messages ( i , e ( p, kab )
                    (i,e(p,kab)) is in relation Messages
                    1.2 Keys( i, kab )
                    [R.2]1.2.1 Keys( i, kab ):- Messages( i, kab )&Key(kab).
                          kab is in relation Key
                          In this point there are two possibilities:
                    [I.5]  1.2.2 a) Messages ( i, kab ) :- Messages ( i , [ n, kab ] ).
                           1.2.2 b) Messages ( i, kab ) :- Messages ( i , [ m, kab ] ).
          a)        [I.5]  1.2.3  Messages ( i, [n, kab] ) :- Messages ( i , [ b, n, kab ] ).
                    [I.1]  1.2.4 Messages ( i, [ b, n, kab] ) :- Messages ( i , e( [ b, n, kab ], L )) & Keys( i, L ).
                                                             L=ka
                                                             1.2.4.1 Messages ( i , e( [ b, n, kab ], ka )).
                                                             (i, e( [ b, n, kab],ka)) is in relation Messages
                                                             1.2.4.2 Keys( i, ka ).
                                                             And this fact is possible to reach (for adversary) only by
                                                             cryptoanalysis.
Result:   Messages( i, p ) <= Keys ( i, ka ).
          b)        [I.5]  1.2.3  Messages ( i, [m, kab] ) :- Messages ( i , [ a, m, kab ] ).
                    [I.1]  1.2.4 Messages ( i, [ a, m, kab] ) :- Messages ( i , e( [ a, m, kab ], L )) & Keys( i, L ).
                                                             L=kb
                                                             1.2.4.1 Messages ( i , e( [ a, m, kab ], kb )).
                                                             (i, e( [ n, kab],kb)) is in relation Messages
                                                             1.2.4.2 Keys( i, kb ).
                                                             And this fact is possible to reach (for adversary) only by
                                                             cryptoanalysis.
Result:   Messages( i, p ) <= Keys ( i, kb ).
```

Fig. 4.9. Abduction for question *(i,p)* in modified "optimized" Otway-Rees

So after modification it was not possible to find direct attack on the protocol and from the abduction we can show that any attack is possible only with knowledge of participant's keys that they share with the key server. And the attacker can obtain them only by crypto-analysis.

5 Internet Key Exchange Protocol (IKE)

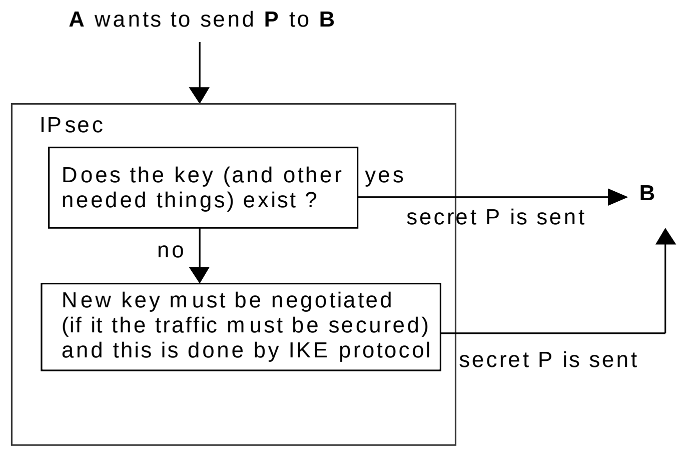
In the middle of nineties under the leading **IETF** (**I**nternet **E**ngineering **T**ask **F**orce) where published first documents about the protocol **IPsec** (**I**nternet **P**rotocol **sec**urity services). This protocol is well-known because of it's interoperation directly with network layer in networks based on TCP/IP protocol stack. IPsec is designed to provide interoperable, high quality, cryptographically based security for Internet Protocol (IP) version 4 and version 6. The set of security services offered includes access control, connectionless integrity, data origin authentication, protection against replays (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. These services are provided at the IP layer, offering protection for IP and/or upper layer protocols. IPsec uses two protocols to provide traffic security - Authentication Header (AH) and Encapsulating Security Payload (ESP).

- The IP Authentication Header (AH) [14] provides connectionless integrity, data origin authentication, and an optional anti-replay service.
- The Encapsulating Security Payload (ESP) protocol [15] may provide confidentiality (encryption), and limited traffic flow confidentiality. It also may provide connectionless integrity, data origin authentication, and an anti-replay service. (One or the other set of these security services must be applied whenever ESP is invoked.)

These protocols may be applied alone or in combination with each other to provide a desired set of security services in IPv4 and IPv6. Each protocol supports two modes of use: transport mode and tunnel mode. In transport mode the protocols provide protection primarily for upper layer protocols; in tunnel mode, the protocols are applied to tunneled IP packets.

Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols. So if we want to show that the protocol IPsec is really secure first of all we must to show that the negotiation of used cryptographic keys (and other needed things) are secure. In Fig. 5.1 is shown how IPsec cooperate with IKE [5].

The Internet Key Exchange protocol is intended to provide the security support for client protocols of the Internet Protocol. As such it does much more than simply distributes keys. It also is intended to use to establish *Security Associations* [16] that specify such things as the protocol format used, the cryptographic and hashing algorithms used, and other necessary features for secure communication. Since it is intended to be flexible, it supports a number of different types of key exchange options, including digital signatures, public key encryption and conventional encryption using shared keys. IKE has evolved from a number of different protocols, including ISAKMP [20], Oakley [29], the Station-to-Station protocol [8] and SKEME [18].



A wants to send P to B
IPsec
Does the key (and other needed things) exist ?
yes
B
secret P is sent
no
New key must be negotiated (if it the traffic must be secured) and this is done by IKE protocol
secret P is sent

Fig. 5.1. Simplification of how IPsec and IKE cooperate

A typical key establishment protocol proceeds in one phase, in which two participants use master keys to establish shared keying material. IKE, however, proceeds in two phases. In the first phase, two participants use master key to agree, not only keying material, but on the various mechanisms (e.g. cryptographic algorithms, hash functions, etc.), that they will use in the second phase. The keying material and set of mechanisms thus agreed upon is also called a security association (it is different from that, which is negotiated). In the second phase, the keys and mechanisms produced in the first phase are used to agree upon new keys and mechanisms (these will be in new negotiated security association for IPsec), that will be used to protect and authenticate further communications. The security association in the first phase is bidirectional. So the initiator in the first phase can be either initiator or responder in the second phase.

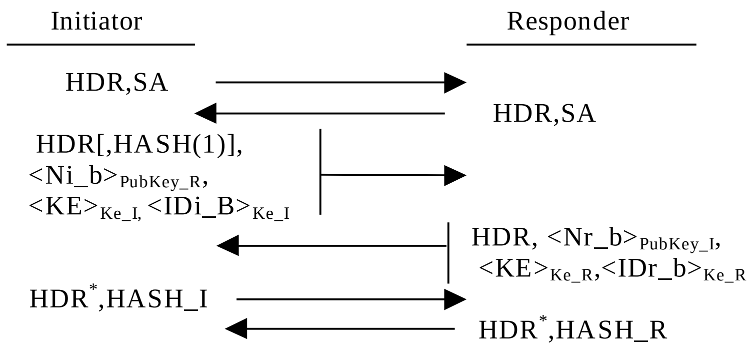
In the Phase One may be used two modes – *main* and *aggressive*. These modes offer different types of services. In main mode, some certain types of identifying information will not be exchanged until some initial authentication has occurred. In the aggressive mode, this level of protection is not provided, but the exchange is accomplished in fewer messages. Main and aggressive mode can be implemented in different ways: using shared keys, signatures or public keys in two different ways for authentication. In all of these the Diffie-Hellman protocol is used to generate the keying material.

Quick mode is used as part of the Phase Two negotiation process. In quick mode the key material generated in Phase One is used to encrypt and authenticate messages used in Phase Two.

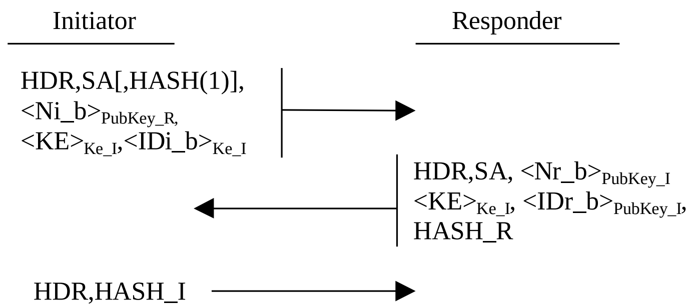
As we can see IKE is really a combination of a number of different subprotocols. Communication via IKE protocol looks like follows:

IKE Phase I – authentication with revised public key encryption

A) Main mode:

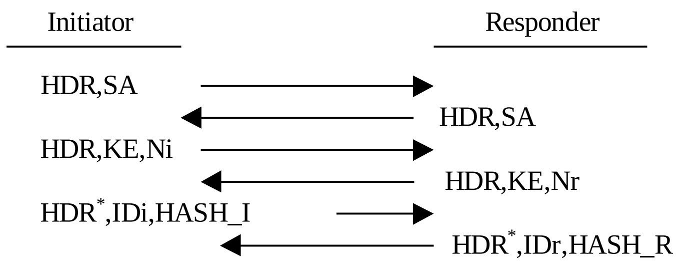

Initiator
Responder
HDR,SA
HDR,SA
HDR[,HASH(1)], <Ni_b>$_{PubKey\_R}$, <KE>$_{Ke\_I}$, <IDi_B>$_{Ke\_I}$
HDR, <Nr_b>$_{PubKey\_I}$, <KE>$_{Ke\_R}$,<IDr_b>$_{Ke\_R}$
HDR*,HASH_I
HDR*,HASH_R

B) Aggressive mode

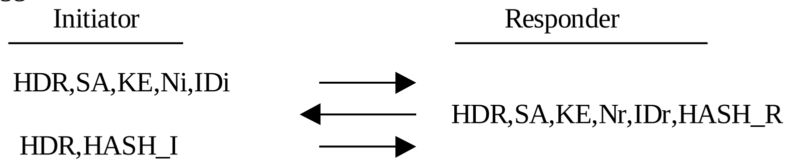

Initiator
Responder
HDR,SA[,HASH(1)], <Ni_b>$_{PubKey\_R}$, <KE>$_{Ke\_I}$,<IDi_b>$_{Ke\_I}$
HDR,SA, <Nr_b>$_{PubKey\_I}$ <KE>$_{Ke\_I}$, <IDr_b>$_{PubKey\_I}$, HASH_R
HDR,HASH_I

IKE Phase I – authentication with pre-shared key:

A) Main mode:

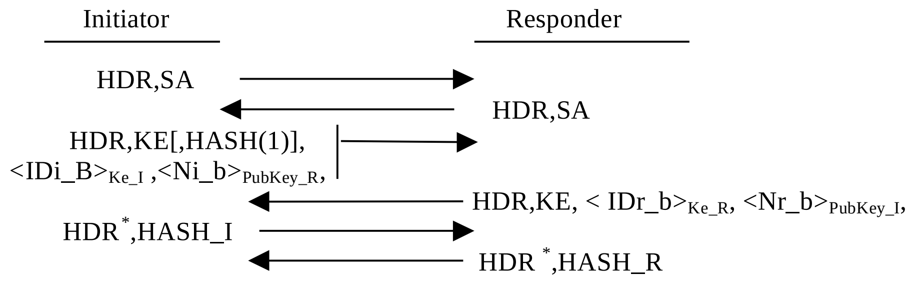


B) Aggressive mode:

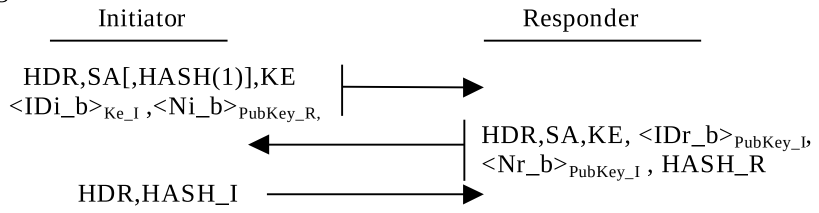


IKE Phase I – authentication with public key encryption

A) Main mode:

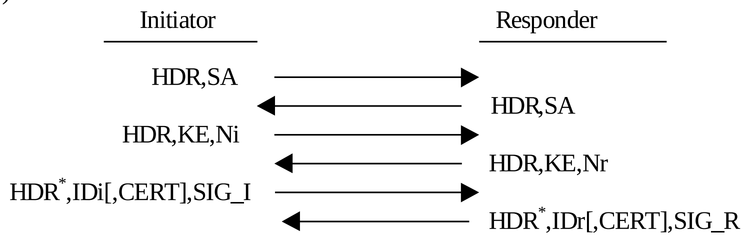

Initiator
Responder
HDR,SA
HDR,SA
HDR,KE[,HASH(1)], <IDi_B>$_{Ke\_I}$,<Ni_b>$_{PubKey\_R}$,
HDR,KE, < IDr_b>$_{Ke\_R}$, <Nr_b>$_{PubKey\_I}$,
HDR*,HASH_I
HDR *,HASH_R

B) Aggressive mode

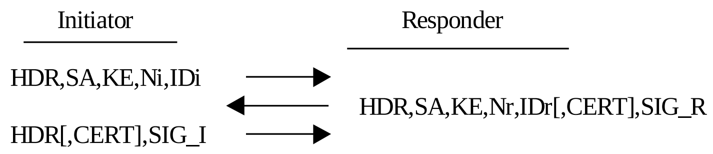


IKE Phase I – authentication with digital signature:

A) Main mode:



B) Aggressive mode:



The IKE protocol has been analyzed by Martius [19] and later by Meadows [24].

In our analysis we focused on Phase One. It is because of Phase Two does not authenticate some one, it only serves for key refreshment. Therefore, we don’t need to analyze this phase, because its security depends only on the security of the Phase One. Another simplification was done that we analyze now only sorter aggressive mode that doesn’t offer such security as the main mode.

As it was said the IKE support more ways of authentication, so we part our analysis on three parts. In part one we analyze the IKE using pre-shared key, then in part two we analyze the IKE with digital signature and at the end the IKE with private key encryption.

5.1 Extension of DLA model for IKE protocol

By analyzing of IKE protocol we will need to define new types, function symbols, relations and rules for these relation schemes.

- *Cookie(* Cookie*: Dcookies)* – says that “Cookie” is a cookie.
- *SecAssociation(* SA*: DsecAss)* – says that “SA” is a security association.
- *DHgroup(*Group*: Dgroups)* - says that “Group” is a Diffie-Hellman group.

- *Number(*Number*: Dnumbers)* – says that "Number" is a random number for D-H evaluation.
- *Cookies(*Agent*: Dagents;* Cookie*: Dcookies)* – says that participant "Agent" knows cookie "Cookie".
- *SecAssociations(*Agent*: Dagents;* SA*: DsecAss)* – says that participant "Agent" knows security association "SA".
- *Numbers(*Agent*: Dagents;* Number*: Dnumbers)* – says that participant "Agent" knows random number "Number".
- *Com_Cookies(*Agent1,Agent2*: Dagents;* Cookie*: Dcookies)* – says that cookie "Cookie" is used by participant "Agent1" in communication with participant "Agent2".
- *Com_SecAssociation(*Agent1,Agent2*: Dagents;* SA*: DsecAss)* – says that security association "SA" is negotiated in the session between "Agent1" and "Agent2".
- *Com_Numbers(*Agent1,Agent2*: Dagents;* Number*: Dnumbers)* – says that random number "Number" is used by communication between participants "Agent1" and "Agent2".
- *Secret(*Agent1,Agent2*: Dagents;* Secret*: Dsecrets)* – says that participants "Agent1" and "Agent2" have common pre-shared secret "Secret".

The relations *Cookies, SecAssociations* and *Numbers* are binary relations that describe some type of knowledge of participants and they can change during evaluation of protocol by using following rules:

[R.5] *Cookies(X, C):-Messages(X, C)&Cookie(C).*
[R.6] *SecAssociations(X, S):-Messages(X, K)&SecAssociation(S).*
[R.7] *Numbers(X, N):-Messages(X, N)&Number(N).*
[R.8] *Messages(X, N):-Number(N)&DHkey(N,G,A,B)&Messages(X,A)& &Messages(X,B)&Number(A)& &Number(B).* – to obtain the D-H private number

Fig. 5.2. Additional relation rules for IKE

Beside new relations and rules we also will need new function symbols to express some functionality, especially the solution of Diffie-Hellman generation of public numbers and the solution of pseudo random function that is used by generation of some values.

New function symbols and their rules:
exp(G,A) – express public Diffie-Hellman value:

$$X = \text{generator}(G)^{A} \bmod \text{prime}(G)$$

Where "prime(G)" is a prime belonging to group G and "generator(G)" is generator belonging to group G.

prf(T,L) – express the result of evaluation of pseudo random function on message *T* with the key *L*.

[I.8] *Messages(X,exp(G,A)):-Numbers(X, A)&DHgroup(G).* – D-H exponentiation
[I.9] *Messages(X, prf(T, L)):-Messages(X, T)&Messages(X, L).* – prf

Fig. 5.3. Additional protocol-independent cryptographic rules for IKE

5.2 DLA Analysis IKE with pre-shared key

The description of messages used by evaluation of IKE protocol with pre-shared key (PSK) authentication is shown on Fig. 5.4.

1. $a \rightarrow b$: ca,sa,g^a,n,a
2. $b \rightarrow a$: ca,cb,sa,g^b,m,b,hash_b
3. $a \rightarrow b$: hash_a

4a. $a \rightarrow b$: $\{p\}_{kab}$
4b. $b \rightarrow a$: $\{q\}_{kab}$

Fig. 5.4. The IKE protocol with pre-shared key

Description of used constants:

ca,cb – cookies of the participants *a, b* respectively
sa – negotiated security association
g^a, g^b – the public Diffie-Hellman values
m, n – nonces of the participants
a, b – identifiers of the participants
hash_a, hash_b – authentication value (result of auth. function)
p, q – arbitrary word

By specification protocol we made one simplification. Typically the participant *a*, the initiator, sends to the participant *b* offers for potential security associations. The participant *b* accept one of offered security associations and sends it in the second message to the participant *a*. We made simplification that the participant *a* offers only one security associations *sa* so the participant *b* must accept it and sends the same security association in the second message. We can do this because if the attacker can modify in the messages the offer ,if there are more security associations, it is possible to do also if there are only one.

The protocol specification rules for evaluation the IKE protocol with pre-shared key authentication:

[P.1] *Messages(X, [cab,sab,exp(g1,x1),n,a]):- X=a v X=i.*
Messages(X,[C1,S,exp(G,A),N,Y]):-Messages(i,[C1,S,exp(G,A),N,Y])&Nonce(N)&SecA(S)&Cookie(C1)&

&Agent(Y)&DHgroup(G)&Number(A).

[P.2] *Messages(i,[C1,C2,S,exp(G,B),M,b,prf(SKEYID,L)]):-Messages(b [C1,S,A,N,Y])& DHgroup(G)&*
&Nonce(N)&Agent(Y)&Number(A)&SecA(S)&Cookie(C1)&Com_Cookies(b,Y,C2)&
&Com_Nonces(b,Y,M)&Secrets(b,T)&Com_Numbers(b,Y,B)&
&L=[exp(G,B),A,C2,C1,S,b]&Secret(b,Y,T)&SKEYID=prf(T,[N,M]).
Messages(X,[C1,C2,S,exp(G,B),M,b,prf(SKEYID,K)]):-Messages(i, [C1,C2,S,exp(G,B),M,b,prf(SKEYID,K)])&
&Cookie(C1)&Cookie(C2)&SecA(S)&Nonce(M)&DHGroup(G)&Number(A)&
&K=[exp(G,B),A,C2,C1,S,b]&Messages(i,[C1,S,A,N,Y])&SKEYID=prf(T,[N,M])
&Secret(T).

[P.3] *Messages(i,prf(SKEYID,L)]):-Messages(Y,[C1,C2,S,B,M,b,HASH_B])&Com_Cookies(Y,b,C1)&*
&Cookie(C2)&Nonce(M)&DHgroup(G)&Com_SecAss(Y,b,S)&Com_Numbers(Y,b,A)&
&HASH_B=prf(SKEYID,K)&Secret(Y,b,T)&SKEYID=prf(T,[N,M])&
&K=[B,exp(G,A),C2,C1,S,b]&L=[exp(G,A),B,C2,C1,S,Y].
Messages(X,prf(SKEYID,L)):-Messages(i,prf(SKEYID,L))&Messages(i,[C1,C2,S,B,M,b,prf(SKEYID,K)])&
&Cookie(C1)&Cookie(C2)&SecA(S)&Number(B)&Nonce(M)&SKEYID=prf(T,[N,M])&
&Nonces(N)&Secret(T)&K=[B,A,C2,C1,S,b]&L=[A,B,C2,C1,S,Y]&
&Messages(i,[C1,S,A,N,Y])&Number(A)&Agent(Y).

[P.4] *Messages(i,e(q,H)):-Messages(b,[C1,S,A,N,Y])&Messages(b,HASH_A)&DHgroup(G) &Agent(Y)&*
&Com_keys(b,Y,H)&Messages(b,SKEYID)&Number(A)&SecA(S)&Cookie(C1)&
&Com_Cookies(b,Y,C2)&DH &Com_Nonces(b,Y,M)&HASH_A=prf(SKEYID,K) &
&Nonce(N) &Com_Numbers(b,Y,B)&K=[exp(G,B),A,C2,C1,S,b]&Secret(b,Y,T)&
&SKEYID=prf(T,[N,M])&L=[exp(G,A),B,C2,C1,S,Y].
Messages(X,e(R,L)):- Messages(i,e(q,K))&Messages(i,e(R,L)).

[P.5] *Messages(i,e(p,H)):-Messages(Y,[C1,C2,S,B,M,b,HASH_B])&Com_Cookies(Y,b,C1)&Cookie(C2)&*
&Nonce(M)&DHgroup(G)&Com_SecAss(Y,b,S)&Com_Numbers(Y,b,A)&
&HASH_B=prf(SKEYID,K)&Com_keys(Y,b,H)&Secret(Y,b,T)&
&SKEYID=prf(T,[N,M])&K=[B,exp(G,A),C2,C1,S,b]&
&L=[exp(G,A),B,C2,C1,S,Y]&Messages(Y,SKEYID).
Messages(X,e(R,L)):- Messages(i,e(p,K))&Messages(i,e(R,L)).

Fig. 5.5. Protocol specification rules for IKE with PSK

By the evaluation of protocol we didn't find any direct attack on the protocol so we try to state the conditions with abduction. The abduction saws that only condition that can lead to the attack is to know the pre-shared key. But negotiation of the pre-shared key is out the scope of the IKE protocol. So the safety of the IKE with pre-shared key is based on safety of negotiation of this pre-shared key. The results show that we didn't find any attack on this part of protocol as in other analyses.

```
[I.1] Messages ( i , p ) :- Messages ( i , e( p, L ) & Keys ( i , L )
    L is from Dkeys = {kab,kac,kbc,pka,pkb,pkc,vka,vkb,vkc} :
    L = kab
    1. Messages ( i , p ) :- Messages ( i , e ( p, kab ) & Keys ( i , kab )
    True       1.1 Messages ( i , e ( p, kab )
               1.2 Keys( i, kab )
               [R.1]   Keys( i, kab ):- Messages( i, kab ).
                       kab is derived from SKEYID_D=prf( prf(asb, [n,m]),[xy11,ca,cb,0])
                       so we need to look for: Messages(i,[asb,ca,cb,n,m,xy11]).
                       if the intruder knows [ca.cb.n.m.xy11] then he can derive (learn) the key.
                       Messages ( i, kab ) :- Messages ( i , [asb, ca, cb, n, m, xy11] ).
```

```
        [I.6]   Messages ( i, [ asb, ca, cb, n, m, xy11] ) :- Messages ( i , asb)&
                                                &Messages(i, [ca,cb,n,m,xy11]).
        true    1.2.1 Messages ( i , asb )
Result:         This is pre-shared secret that knows only A and B.
                So this fact is impossible to obtain for the intruder.
                1.2.2 Messages(i, [ca,cb,n,m,xy11])
        [I.6]   Messages ( i, [ ca, cb, n, m, xy11] ) :- Messages ( i , ca)&
                                                & Messages(i, [cb,n,m,xy11]).
        true    1.2.2.1 Messages ( i , ca )
                1.2.2.2 Messages(i, [cb,n,m,xy11])
                [I.6]   Messages(i, [cb,n,m,xy11]):-Messages(i,cb)&
                                                &Messages(i,[n,m,xy11]).
                true    1.2.2.2.1 Messages(i,cb)
                        1.2.2.2.2 Messages(i,[n,m,xy11])
                        [I.6]   Messages(i, [n,m,xy11]):-Messages(i,n)&
                                                &Messages(i,[m,xy11]).
                        true    1.2.2.2.2.1 Messages ( i , n )
                                1.2.2.2.2.2 Messages(i, [m,xy11])
                                [I.6]   Messages(i, [m,xy11]):-Messages(i,m)&
                                                &Messages(i,xy11).
                                true    1.2.2.2.2.2.1 Messages ( i , m )
                                        1.2.2.2.2.2.2 Messages(i, xy11)
                                    a)  [R.8]   Messages(i, xy11):-Number(xy11)&
                                                    &DHkey(xy11,g1,x1,exp(g1,y1)&
                                                    &Messages(i,x1)&Messages(i,exp(g1,y1))&
                                                    &Number(x1)&Number(exp(g1,y1)).
                                        true    Number(xy11)
                                        true    Number(x1)
                                        true    Number(exp(g1,y1))
                                        true    DHKey(xy11,g1,x1,exp(g1,y1))
                                        true    Messages(i, exp(g1,y1))
                                                Messages(i,x1)
Result:                                         And this fact is impossible to obtain for the
                                                intruder.
                                    b)  [R.8]   Messages(i, xy11):-Number(xy11)&
                                                    &DHkey(xy11,g1,exp(g1,x1),y1)&
                                                    &Messages(i,exp(g1,x1))&Messages(i,y1)&
                                                    &Number(exp(g1,x1))&Number(y1).
                                        true    Number(xy11)
                                        true    Number(exp(g1,x1))
                                        true    Number(y1)
                                        true    DHKey(xy11,g1,x1,exp(g1,y1))
                                        true    Messages(i, exp(g1,x1))
                                                Messages(i,y1)
Result:                                         And this fact is impossible to obtain for the
                                                intruder.
```

Fig. 5.6. Abduction for question *(i,p)* in IKE with PSK

5.3 DLA Analysis IKE with digital signature

As by the IKE protocol with pre-shared key in IKE with digital signature (SIG) we also use the new defined relation and rules [R.5]-[R-7], [I.8] and [I.9] (see chapter 5.1). The difference is in the protocol specific rules because of different authentication method and different evaluation of key material SKEYID.

```
1. a → b : ca,sa,g^a,n,a
2. b → a : ca,cb,sa,g^b,m,b,sig_b
3. a → b : sig_a
4a. a → b : {p}kab
4b. b → a : {q}kab
```

Fig. 5.7. The IKE protocol with digital signature

Description of used constants are the same as by IKE with pre-shared key with exception *sig_a*, *sig_b* – authentication value (result of authentication function). The signed data, *sig_a* or *sig_b*, is the result of the negotiated digital signature algorithm applied to *HASH_A* or *HASH_B* respectively.

$HASH\_A=prf(SKEYID,[g^a, g^b,ca,cb,sa,a])$
$HASH\_B=prf(SKEYID, [g^b, g^a,cb,ca,sa,b])$
$SKEYID=prf([n,m],g^{ab})$ where g^{ab} is D-H secret value
$sig\_a=sig(HASH\_A,pka)$ and $sig\_b=sig(HASH\_B,pkb)$

Where *pka* and *pkb* are private digital signature keys. The verification of signature is done with *vka* and *vkb* public digital signature keys.

$b: HASH\_A=ver(sig\_a,vka)$ and $a: HASH\_B=ver(sig\_b,vkb)$

So we will need two new function symbols and so additional protocol-independent cryptographic rulesrules:

[I.10] *Messages(X,sig(M,K)):-Messages(X, M)&Keys(X,K)&Digital(X,K,L).* - signature
[I.11] *Messages(X,M):-Messages(X,sig(M,K))&Keys(X,L)&Digital(X,K,L).* – verification

Fig. 5.8. Additional rule for IKE with digital signature

We will need one new relation to express relation between Diffie-Hellman public and secret values:

- *DHkey(*Svalue*: Dnumbers;* Group*: Dgroups;* Num1,Num2*: Dnumbers)* – says that public values “Num1” and “Num2” generated by group “Group” have the secret value “Svalue”.
- *Digital(*Agent*: Dagents*; private, public*: Dkeys)* – says that the participant “Agent” for digital signature uses private key “private” and public key “public”.

The protocol specific rules for evaluation the IKE protocol with digital signature authentication:

[P.1] *Messages(X, [cab,sab,exp(g1,x1),n,a]):- X=a v X=i.*
Messages(X,[C1,S,exp(G,A),N,Y]):-Messages(i,[C1,S,exp(G,A),N,Y])&
&.Nonce(N)&SecA(S)&Cookie(C1)&Agent(Y)&DHgroup(G)&
&Number(A).

[P.2] *Messages(i,[C1,C2,S,exp(G,B),M,b,sig(HASH_B,PK)]):-*
Messages(b [C1,S,A,N,Y])& DHgroup(G)&Nonce(N)&Agent(Y)&
&Number(A)&SecA(S)&Cookie(C1)&Com_Cookies(b,Y,C2)&
&Com_Nonces(b,Y,M)&Messages(b,HASH_B)&
&Digital(b,PK,VK)&Keys(b,PK)&HASH_B=prf(SKEYID,L)&
&Com_Numbers(b,Y,B)&L=[exp(G,B),A,C2,C1,S,b]&
&DHkey(T,G,A,exp(G,B))&SKEYID=prf([N,M],T).
Messages(X,[C1,C2,S,exp(G,B),M,b, sig(HASH_B,PK)]):-
Messages(i, [C1,C2,S,exp(G,B),M,b,sig(HASH_B,PK)])&Cookie(C1)&
&Cookie(C2)&SecA(S)&Nonce(M)&DHGroup(G)&Number(A)&
&K=[exp(G,B),A,C2,C1,S,b]&Messages(i,[C1,S,A,N,Y]&
&Digital(b,PK,VK)&Keys(b,PK)&HASH_B=prf(SKEYID,L)&
&SKEYID=prf([N,M],T)&Number(T).

[P.3] *Messages(i,sig(HASH_A,I)]):-Messages(Y,[C1,C2,S,B,M,b,sig(HASH_B,J)])&*
&Digital(Y,I,IV)&Digital(b,J,JV)&Keys(Y,JV)&Keys(Y,I)&
&Messages(Y,HASH_B)&Message(Y,HASH_A)&
&Com_Cookies(Y,b,C1)&Cookie(C2)&Nonce(M)&
&DHgroup(G)&Com_SecAss(Y,b,S)&Com_Numbers(Y,b,A)&
&HASH_A=prf(SKEYID,K)&SKEYID=prf([N,M],T)&
&K=[exp(G,A),B,C2,C1,S,Y] &HASH_B=prf(SKEYID,L)&
&L=[B,exp(G,A),C2,C1,S,b]&DHkey(T,G,exp(G,A),B).
Messages(X,sig(HASH_A,I)):-Messages(i,sig(HASH_A,I))&
&Messages(i,[C1,C2,S,B,M,b,sig(HASH_B,J)])&Cookie(C1)&
&Digital(Y,I,IV)&Digital(b,J,JV)&Keys(Y,JV)&Keys(J,I)&
&Cookie(C2)&SecA(S)&Number(B)&Nonce(M)&
&HASH_B=prf(SKEYID,K)&HASH_A=prf(SKEYID,L)&
&SKEYID=prf([N,M],T)&Nonces(N)&K=[B,A,C2,C1,S,b]&
&L=[A,B,C2,C1,S,Y]&Messages(i,[C1,S,A,N,Y])&
&Number(A)&Agent(Y)&Number(T).

[P.4] *Messages(i,e(q,H)):-&Messages(b [C1,S,A,N,Y])& DHgroup(G)&Nonce(N)&*
& Cookie(C1)&Com_Cookies(b,Y,C2)&Digital(b,PK,VK)&
&Com_Nonces(b,Y,M)&Messages(i,HASH_B)& Number(A)&
&SecA(S)&HASH_B=prf(SKEYID,L)&Com_Numbers(b,Y,B)&

& Keys(b,PK) &L=[exp(G,B),A,C2,C1,S,b]&DHkey(T,G,A,exp(G,B))&
&SKEYID=prf([N,M],T)&Agent(Y)&Com_keys(b,Y,H).
Messages(X,e(R,L)):- Messages(i,e(q,K))&Messages(i,e(R,L)).

[P.5] *Messages(i,e(p,H)):- Messages(Y,[C1,C2,S,B,M,b,sig(HASH_B,J)])&*
&Digital(Y,I,IV)&Digital(b,J,JV)&Keys(Y,JV)&Keys(J,I)&
&Messages(Y,HASH_B)&Message(Y,HASH_A)& Nonce(M)&
&Com_Cookies(Y,b,C1)&Cookie(C2)& DHkey(T,G,exp(G,A),B)&
&DHgroup(G)&Com_SecAss(Y,b,S)&Com_Numbers(Y,b,A)&
&HASH_A=prf(SKEYID,K)&SKEYID=prf([N,M],T)&
&K=[exp(G,A),B,C2,C1,S,Y] &HASH_B=prf(SKEYID,L)&
&L=[B,exp(G,A),C2,C1,S,b]&Com_keys(Y,b,H).
Messages(X,e(R,L)):- Messages(i,e(p,K))&Messages(i,e(R,L)).

Fig. 5.9. The protocol specification rules for IKE with SIG

In this case our analysis it is possible for the attacker to do the denial of service attack on the protocol. In the attack the participant *a*, the initiator of the protocol evaluation, will believe that he shares the session key with the participant *b*, but the participant will share no session key with the participant *a*. The attack is shown on Fig. 5.10.

1. a → b : ca,sa,g^a,n,a - is intercepted
   i → b : ca,sa,g^a,n,**c**
2. b → c : ca,cb,sa,g^b,m,b,sig_b – is intercepted
   i → a : ca,cb,sa,g^b,m,b,sig_b
3. a → b : sig_a - *b* reject this message
4. 4a. a → b : $\{p\}_{kab}$ - *b* reject this message

Fig. 5.10. The attack on the IKE with digital signature

The main problem, that causes the attack, is that by authentication of the participant *b* it is not clear for who he makes the authentication. By pre-shared key it is simply to achieve, while the participant *b* shares some data with the participant *a* and some other data with *c*. So if the participant *b* makes authentication he uses these shared data. It means that the participant *a* will know whether the authentication of the participant *b* is assigned to him or to other participant. To prevent this attack it must be changed the protocol that way to the participant know to achieve that the authentication of the session is dedicated to him not to some other participant. We didn't find other attack.

5.4 DLA Analysis IKE with public key encryption

As by the IKE protocol with pre-shared key and digital signature in IKE with public key encryption (PKE) we also use the new defined relation and rules [R.5]-[R-7], [I.8] and

[I.9] (see chapter 5.1). The difference is in the protocol specific rules because of different authentication method and different evaluation of key material SKEYID.

1.a $\rightarrow$ b : ca,sa,g^a,$\{a\}_{vkb}$,$\{n\}_{vkb}$
2.b $\rightarrow$ a : ca,cb,sa,g^b,$\{b\}_{vka}$,$\{m\}_{vka}$,hash_b
3.a $\rightarrow$ b : hash_a
4a. a $\rightarrow$ b : $\{p\}_{kab}$
4b. b $\rightarrow$ a : $\{q\}_{kab}$

Fig. 5.11. The IKE protocol with public key encryption

HASH_A=prf(SKEYID,[g^a, g^b,ca,cb,sa,a])
HASH_B=prf(SKEYID, [g^b, g^a,cb,ca,sa,b])
SKEYID=prf(hash([n,m]),[ca,cb])

The rules [I.1] and [I.2] express only the symmetric encryption with the same key used to encrypt and decrypt any message. In PKE authentication we need the asymmetric encryption so we must define additional rules:

[I.10] *Messages(X,pke(M,L)):-Messages(X, M)&Keys(X,L)&PKEKeys(X,K,L).* - encryption
[I.11] *Messages(X,M):-Messages(X,pke(M,L))&Keys(X,K)&PKEKeys(X,K,L).*– decryption

Fig. 5.12. Additional rule for IKE with public key encryption

The relation scheme *PKEKeys(*Agent*: Dagents*; private, public*: Dkeys)* means that to send a message to the participant "Agent" we need the public key "public" and the participant "Agent" uses to read such message the private key "private".

The protocol specific rules for evaluation the IKE protocol with digital signature authentication:

[P.1] *Messages(X, [cab,sab,exp(g1,x1),pke(a,vka) ,pke(n,vka)]):- X=a v X=i.*
Messages(X,[C1,S,exp(G,A),pke(Y,K) ,pke(N,K)]):- Number(A)&Key(K)&
&Messages(i,[C1,S,exp(G,A),pke(Y,K),pke(N,K)])&
&.Nonce(N)&SecA(S)&Cookie(C1)&Agent(Y)&DHgroup(G).

[P.2] *Messages(i,[C1,C2,S,exp(G,B),pke(b,O),pke(M,O),HASH_B]):- DHgroup(G)&*
&Messages(b [C1,S,A,pke(N,K),pke(Y,K)])&Nonce(N)&Agent(Y)&
&PKEKeys(b,P,K)&Keys(b,K)&Messages(b,N)&Number(A)&
&SecA(S)& Cookie(C1)&Com_Cookies(b,Y,C2)&Com_Nonces(b,Y,M)&
&Messages(b,HASH_B)&HASH_B=prf(SKEYID,L)&PKEKeys(Y,J,O)&
&Com_Numbers(b,Y,B)&L=[exp(G,B),A,C2,C1,S,b]&
&SKEYID=prf([N,M],[C1,C2]).
Messages(X,[C1,C2,S,exp(G,B),pke(b,J),pke(M,J), HASH_B]):- Cookie(C1)&

&Messages(i, [C1,C2,S,exp(G,B),pke(b,J),pke(M,J),HASH_B])&
&Cookie(C2)&SecA(S)&Nonce(M)&DHGroup(G)&Number(A)&
&K=[exp(G,B),A,C2,C1,S,b]&Messages(i,[C1,S,A,pke(Y,I),pke(N,I)]&
&PKEKeys(b,PK,I)&PKEKeys(Y,PV,J)&Keys(b,PK)&
&HASH_B=prf(SKEYID,L)&SKEYID=prf([N,M],[C1,C2]).

[P.3] *Messages(i,HASH_A):-Messages(Y,[C1,C2,S,B,pke(b,J),pke(M,J),HASH_B])&*
&PKEKeys(Y,I,J)&Keys(Y,I)&L=[B,exp(G,A),C2,C1,S,b]&
&Messages(Y,HASH_B)&Message(Y,HASH_A)&
&Com_Cookies(Y,b,C1)&Cookie(C2)&Nonce(M)&
&DHgroup(G)&Com_SecAss(Y,b,S)&Com_Numbers(Y,b,A)&
&HASH_A=prf(SKEYID,K)&SKEYID=prf([N,M],[C1,C2])&
&K=[exp(G,A),B,C2,C1,S,Y] &HASH_B=prf(SKEYID,L).
Messages(X,HASH_A):-Messages(i,HASH_A)&Number(A)&Agent(Y)&
&Messages(i,[C1,C2,S,B,pke(b,J),pke(M,J),HASH_B])&Cookie(C1)&
&PKEKeys(Y,I,J)&Cookie(C2)&SecA(S)&Number(B)&Nonce(M)&
&HASH_B=prf(SKEYID,K)&HASH_A=prf(SKEYID,L)&Keys(i,J)
&SKEYID=prf([N,M],[C1,C2])&Nonces(N)&K=[B,A,C2,C1,S,b]&
&L=[A,B,C2,C1,S,Y]&Messages(i,[C1,S,A,pke(Y,I),pke(N,I)]).
[P.4] *Messages(i,e(q,H)):- DHgroup(G) &Com_keys(b,Y,H)&*
&Messages(b [C1,S,pke(N,K),pke(Y,K)])&Nonce(N)&Agent(Y)&
&PKEKeys(b,P,K)&Keys(b,K)&Messages(b,N)&Number(A)&
&SecA(S)& Cookie(C1)&Com_Cookies(b,Y,C2)&Com_Nonces(b,Y,M)&
&Messages(b,HASH_B)&HASH_B=prf(SKEYID,L)&PKEKeys(Y,J,O)&
&Com_Numbers(b,Y,B)&L=[exp(G,B),A,C2,C1,S,b] &
&DHkey(T,G,A,exp(G,B))&SKEYID=prf([N,M],[C1,C2]).
Messages(X,e(R,L)):- Messages(i,e(q,K))&Messages(i,e(R,L)).

[P.5] *Messages(i,e(p,H)):- Messages(Y,[C1,C2,S,B,pke(b,J),pke(M,J),HASH_B])&*
&PKEKeys(Y,I,J)&Keys(Y,I)&L=[B,exp(G,A),C2,C1,S,b]&
&Messages(Y,HASH_B)&Message(Y,HASH_A)& Nonce(M)&
&Com_Cookies(Y,b,C1)&Cookie(C2) &Com_keys(Y,b,H)&
&DHgroup(G)&Com_SecAss(Y,b,S)&Com_Numbers(Y,b,A)&
&HASH_A=prf(SKEYID,K)&SKEYID=prf([N,M],[C1,C2])&
&K=[exp(G,A),B,C2,C1,S,Y] &HASH_B=prf(SKEYID,L).
Messages(X,e(R,L)):- Messages(i,e(p,K))&Messages(i,e(R,L)).

Fig. 5.13. The protocol specification rules for IKE with PKE

By the evaluation of first step our method we didn't find any attack of this part of IKE protocol. So in the second step by abduction we try to state possible attacks. Only conditions that we reached by abduction where that we need to know the private Diffie-Hellman numbers to attack the evaluation of the protocol. And it is impossible for the attacker to obtain such information. So the protocol is safe.

[I.1] Messages (i , p) :- Messages (i , e(p, L) & Keys (i , L)
L is from Dkeys = {kab,kac,kbc,pka,pkb,pkc,vka,vkb,vkc} :
L=kab
1. Messages (i , p) :- Messages (i , e (p, kab) & Keys (i , kab)
True 1.1 Messages (i , e (p, kab)

```
1.2 Keys( i, kab )
[R.1]   Keys( i, kab ):- Messages( i, kab ).
        kab is derived from SKEYID_D=prf(prf([n,m],[ca,cb]),[xy11,ca,cb,0])
        so we need to look for: Messages(i,[ca,cb,n,m,xy11]).
        if the intruder knows [ca.cb.n.m.xy11] then he can derive (learn) the key.
        Messages ( i, kab ) :- Messages ( i , [ ca, cb, n, m, xy11] ).
[I.6]   Messages(i,[ca,cb,n,m,xy11]):-Messages(i,ca)&
                                    &Messages(i,[cb,n,m,xy11]).
true    1.2.1 Messages ( i , ca )
        1.2.2 Messages(i, [cb,n,m,xy11])
        [I.6]   Messages(i, [cb,n,m,xy11]):-Messages(i,cb)&
                                    &Messages(i,[n,m,xy11]).
        true    1.2.2.1 Messages(i,cb)
                1.2.2.2 Messages(i,[n,m,xy11])
                [I.6]   Messages(i, [n,m,xy11]):-Messages(i,n)&
                                    &Messages(i,[m,xy11]).
                true    1.2.2.2.1 Messages ( i , n )
                        1.2.2.2.2 Messages(i, [m,xy11])
                        [I.6]   Messages(i, [m,xy11]):-Messages(i,m)&
                                            &Messages(i,xy11).
                        true    1.2.2.2.2.1 Messages ( i , m )
                                1.2.2.2.2 Messages(i, xy11)
                          a) [R.8]   Messages(i, xy11):-Number(xy11)&
                                        &DHkey(xy11,g1,x1,exp(g1,y1)&
                                        &Messages(i,x1)&Messages(i,exp(g1,y1))&
                                        &Number(x1)&Number(exp(g1,y1)).
                             true    Number(xy11)
                             true    Number(x1)
                             true    Number(exp(g1,y1))
                             true    DHKey(xy11,g1,x1,exp(g1,y1))
                             true    Messages(i, exp(g1,y1))
                                     Messages(i,x1)
Result:                              And this fact is impossible to obtain for the
                                     intruder.
                          b) [R.8]   Messages(i, xy11):-Number(xy11)&
                                       &DHkey(xy11,g1,exp(g1,x1),y1)&
                                      &Messages(i,exp(g1,x1))&Messages(i,y1)&
                                      &Number(exp(g1,x1))&Number(y1).
                             true    Number(xy11)
                             true    Number(exp(g1,x1))
                             true    Number(y1)
                             true    DHKey(xy11,g1,x1,exp(g1,y1))
                             true    Messages(i, exp(g1,x1))
                                     Messages(i,y1)
Result:                              And this fact is impossible to obtain for the
                                     intruder.
```

Fig. 5.14. Abduction for question *(i,p)* in IKE with PKE

6 Conclusions and future work

We understood the problem of analysis of communication protocols as two-dimensional problem. One dimension is the method used by analysis of protocols and second dimension is the complexity of analyzed protocols. The analysis of protocols is focused especially on the key exchange protocols, because if we know to make an attack on such protocol we can disabled or learn all following traffic.

We can distinguish two types of key exchange protocols:

1. with key server that distributes the new keys
2. without key server and without key distribution (the participants exchange only key material used for key derivation}

The complexity of analysis of these groups is much different. The analysis of first group of protocols is simple (in this group are simple protocols as Needham-Schroeder). The complexity of second group is not simple. In such protocols we must include into analysis also the key derivation. In this group belong protocols like IKE protocol. In Fig. 6.1 it is shown comparison of methods and analyzed protocols.

Explanations: Y/N – Y = analyzed / N = no attack found
? – no information about analsis
Y/Y – Y = analyzed / Y = an attack found
N/- – N = not analyzed / -

			Methods of analysis			
Protocols			BAN logic	NRL Analyzer	Dexter prover	DLA method
Simple	Needham-Schroeder		Y/N	Y/N	Y/N	Y/N
	Otway-Rees		?	?	Y/Y	Y/Y
Complex	IKE (aggressive mode)	with SIG	Y/N	Y/Y	N/-	Y/Y
		with PSK	Y/N	Y/N	N/-	Y/N
		with PKE	Y/N	Y/N	N/-	Y/N

Fig. 6.1. Methods and analyzed protocols

Some type of analyzes based on logic approaches have some weak places. As we mentioned using of some protocol independent rules can lead to infinity cycles There are logic's analyses that ignore. In our method we try to minimize such problem with infinite search space and we limited by analysis the number of possible evaluation of all functions (decryption, encryption, concatenation and others). We also try to use new methods by analysis and to show that such methods and limitations are correct and sound.

Another weak place analyses is description of the possibilities of the attacker. It is because the description of an attacker strongly depends on known flaws of protocols.

The most of methods are independent on used cryptographic algorithms in protocol and this can lead that these methods declare that an protocol is correct and secure but by using with some cryptographic algorithm it is possible to do an attack on this protocol. The influence of such relationship (between the protocol and used cryptographic algorithm) was shown by Rajsky in [32]. He shows that in Needham-Schroeder protocol using with cipher of Feistel type in ECB mode is possible to find direct attack on session key.

In the future work we want to improve our program for translation of the protocol into inference rules. In this field it will be useful to translate a protocol such way that no additional correct will be needed. But it is hard to say whether or not it will be possible to do this. The problem is if we want to use this method by analysis of the key distribution protocols without key server. In such protocols we need to make a lot of additional extensions, that are specific for chosen protocol. But our DLA method was successfully used by this type of protocols too. We also want to use our method by analysis of others protocols.

Bibliography

1. Bellare M., Rogaway P.: Entity authentication and key distribution. Advances in Cryptology – Crypto'93 Proceedings, (1993).
2. Bieber P.: A logic of Communication in a Hostile Environment. Proceedings the Computer Security Foundations Workshop III. IEEE Computer Society Press, (1990) 14-22
3. Burrows, M., Abadi, M., Needham, R.M.: A logic of authentication. ACM Transactions on Computer Systems, (1990)
4. Carlsen U.: Cryptographic protocol flaws: Know your enemy. Computer security Foundations Workshop VII, IEEE Computer Society Press (1994)
5. Carrel D., Harkins D.: Internet Key Exchange Protocol, RFC 2409 (1998).
6. Compton K.J., Dexter S.: Proofs Techniques for Cryptographic Protocols. Workshop at ECAI'98, (1998) 25-39
7. Dexter S.: An Adversary-Centric Logic of Security and Authenticity. PhD thesis, University of Michigan (1998).
8. Diffie W., van Oorschot P.C., Wiener M.J.: Authentication and authenticated key exchanges. Design, Codes and Cryptography (1992).
9. Ferguson N., Schneier B.: A Cryptographic evaluation of IPSec. http://www.counterpane.com (1999)
10. Jenčušová E., Jirásek J.: Formal methods of analysis of security protocols. TATRACRYPT 2001 (2001). Submitted to Tatra Mountains Math. Publ.
11. Kakas A.C., Michael A.: An Abductive-Based Scheduler for Air-Crew Assignment.
12. Kakas A.C., Michael A.: Abductive Logic Programming. Journal of Logic and Computation 2, (1993).
13. Kemmerer R., Meadows C., Millen J.: Three systems for cryptographic protocol analysis. Journal of Cryptology, (1994) 79-130
14. Kent S., Atkinson R.: IP Authentication Header. RFC 2402 (1998).
15. Kent S., Atkinson R.: IP Encapsulating Security Payload. RFC 2406 (1998).
16. Kent S., Atkinson R.: Security Architecture for the Internet Protocol. RFC 2401 (1998).

17. Kindred D., Wing J.M.: Fast automatic checking of security protocols. Proccedings of 2nd Usenix Workshop on Electronic Commerce, (1996) 41-52.
18. Krawczyk H.: SKEME – A versatile secure key exchange mechanism for Internet. ISOC Secure Networks and Distributed Systems Symposium. San Diego (1996).
19. Martius K.: IKE Protocol Analysis. http://www.imib.med.tu-dresden.de/imib/personal/Kai.html (1998)
20. Maughan D. Schertler M., Schneider M., Turner J.: Internet Security Association and Key Management Protocol (ISAKMP) RFC 2408 (1998).
21. Meadows C.: Applying formal methods to the analysis of key management protocols. Journal of Computer Security, (1992) 5-35
22. Meadows C.: Formal Verification of Cryptographic Protocol: A Survey. Asiacrypt'94 (1994)
23. Meadows C.: The NRL Protocol Analyzer: an overview. The Journal of Logic Programming, (1996) 113-131
24. Meadows C.: Analysis of the IKE Protocol Using the NRL Protocol Analyzer. IEEE'99 (1999)
25. Merritt M.J.: Cryptographic Protocols. PhD thesis. Georgia Institute of Technology, (1983)
26. Moser L.: A logic of Knowledge and Belief for Reasoning About Computer Security. Proceedings the Computer Security Foundations Workshop II. IEEE Computer Society Press, (1989) 57-63
27. Needham, R.M., Schroeder M.D.: Using encryption for authentication in large networks of computers. Communications of ACM, Vol. 21,(1978) 993-999
28. Nilsson U., Małuszyński: Logic, Programming And Prolog. Published by John Wiley & Sons Ltd. (1995). http://www.ida.liu.se/~ulfni/lpp
29. Orman H.: The Oakley Key Determination Protocol, RFC 2412 (1998).
30. Paulson L.C.: ML for working programmer. Cambridge University Press (1996)
31. Perlman R., Kaufman Ch.: Analysis of the IPSec Key Exchange Standard. (2001)
32. Rajsky P.: Automatic proving of the correctness of cryptographic protocols. FEI STU Bratislava (2001).
33. Rangan P.V.: An Axiomatic Basis of Trust in Distributed Systems. Proceedings of the 1988 Symposium on Security and Privacy. IEEE Computer Society Press, (1988) 204-211
34. Syverson P.: A Taxonomy of Replay Attacks. Proceedings of the Computer security Foundations Workshop VII, IEEE Computer Society Press, (1994)
35. Syverson P.: Formal Semantic for Logic of Cryptographic Protocols. Proceedings the Computer Security Foundations Workshop III. IEEE Computer Society Press, (1990) 32-41
36. Ullman J.D.: Databases and Knowledge-base systems. Volume I, Computer Science Press (1988).
37. Vojtáš P.: Tunable fuzzy logic programming for abduction under uncertainty. In Proceeding Workshop “Many valued logic for Computer Science Application” at ECAI'98, University of Brighton. (1998) 7 pages.
38. Woo T.Y.C., Lam S.S.: A semantic model for authentication protocols. Proceedings of the Symposium on Research in Security and Privacy, (1993) 178-194
39. Yahalom R., Klein B., Beth T.: Trust relationship in secure systems: A distributed authentication perspective. Proceedings of the 1993 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, (1993) 150-164