Modelling Security Protocol Synthesis using AI Planning

Samuel Gabrielsson
University of Toronto, Canada
samuel@mie.utoronto.ca

June 1, 2007

Abstract

In this paper, we try to see if its possible to map the work done in [1] into AI Planning. By trying to translate the architecture of the Automatic Synthesis Protocol Builder (ASPB) and making it run with the Fast-Forward planner, maybe we could get a better time performance than the original ASPB and the APG from [2] We do not know this yet until we successfully manage to describe the domain of a security protocol synthesis using PDDL. More work on describing the domain needs to be done in the future. Hopefully this paper will lead the way towards a solution.

1 Introduction

Automated planning is a branch in Artificial Intelligence (AI) where planning is the reasoning side of acting. Automated planning technology is used in areas as simple as playing a game of bridge and as complicated as controlling space vehicles and robots. But the field of planning is still in an early stage. Ghallab has made a contribution for the development of planning in [3]. But not much work has been done to date in AI planning with regards to security protocols which is the area we wish to study. One interesting article to represent the verification of security protocols as a classical AI Planning problem can be found in [4].

The Planning Domain Definition Language (PDDL) [5] was developed to express the physics of the domain. We use PDDL to define the domain of our security protocol. The user defines what kind of predicates there exists in the current state, the possible actions, the preconditions and effects of the actions applied on the different states depending on the goals that the planner should try to accomplish (see Appendix A). In this paper we use the planner called Fast-Forward (FF) [6] which is a domain independent planning system developed by Jörg Hoffmann. It uses forward search in the state space, guided by a heuristic function that is extracted from the domain description.

A protocol is a set of rules or conventions defining an exchange of messages among a set of two or more partners. These partners are users, agents, processes or machines called principals. A basic example of a protocol is when you have two principals A and B that share a channel C_{ab} where only A and B can send data or listen on this channel. A channel has a set of principals that can receive messages and send messages via the channel. The protocol is basically that A uses C_{ab} for sending a single message X to B. We can write this in the following notation:

$$\text{Message 1} \qquad A \rightarrow B: \ X \qquad \text{on } C_{ab}$$

In a security protocol, also called a cryptographic protocol, the whole part of some or all of the messages is encrypted. The cryptographic version of the previous example would then be written as:

$$\text{Message 1} \qquad A \rightarrow B: \ \{X\}_{K_{ab}} \qquad \text{on } C_{ab}$$

where the two principals share the key K_{ab} on the public channel C_{ab} that they can use for communication, but which is in no way secure. The security protocol is simply that A sends a message X under K_{ab} to B, on C_{ab}.

Security protocols provide secure communications on the Internet for data or message transfers through channels between different principals. It is common to find channels on which only a given set of principals are allowed to send data or listen. Examples of protocols are application protocols like HTTP, FTP, RTP, RTSP, TELNET and Internet protocols like IP, TCP and UDP.

In section 2 we describe the abstract logic that can be used to reason about authentication protocols. In section 3 we also give a brief introduction to AI Planning and in section 4 we analyze the problem. In section 5 we present the problems and questions encountered during our work.

2 The BSW Beliefs Logic

We will use a logical notation in generating and describing protocols. It enables us to reason about authentication protocols and it models the behavior of the principals and the channels in the system. The Buttyán-Staamann-Wilhelm (BSW) Logic in [7] is a beliefs logic very similar to Burrows-Abadi-Needham (BAN) Logic [8].The symbols P, Q range over principals, C ranges over channels and X range over statements (messages, formulas and nonces) where ϕ represents a formula.

- $P \triangleleft X$: Principal P sees message X. Someone has sent a message containing X via a channel that P can read.
- $P \triangleleft C(X)$: Principal P sees $C(X)$. Someone has sent X via a channel C. If P can not read C then P can not discover the contents of X.
- $P| \sim X$: P once said X. The principal P at some time in the past sent a message including the statement X. We do not know the exact moment the message was sent but it is known that P believed X then.
- $P \parallel \sim X$: P says X. P sent X in the current run of the protocol.
- $\sharp(X)$: Message X is fresh. X has never been said before the current run of the protocol. This is usually true for messages containing nonces. Nonces usually include a time-stamp or a number that is used only once.
- $P| \equiv \phi$: P believes that ϕ is true. It does not mean that ϕ is really true, but P believes it.

Other logical formulas are also used like implication ($\rightarrow$), conjunction ($\wedge$) and disjunction ($\vee$) and some basic notation from set theory.

A demonstration of the language defined above can be seen in [7]. With the logic Buttyán, Staamann and Wilhelm derived the inference rules and the synthetic rules needed to guide the (manual) systematic calculation of a protocol from its goals. The general form of the synthesis rules look like:

$$\begin{array}{ll} G & \\ \hookrightarrow & G_1 \\ \hookrightarrow & G_2 \\ \hookrightarrow & \cdots \end{array}$$

In order to reach the goal G all the given goals $G_1, G_2, \cdots$ have to be reached. A goal G can have the form G''/G''', which means that either or G'' G''' have to be reached.

3 Planning

A plan is a set of actions. Actions are chosen and organized for changing the state of a system. The changes of states are called transitions. The solution of our problem, if there exists a solution, is called a plan. A plan is a sequence of actions. When the actions are executed in a world satisfying the initial state description, it achieves the goal.

Our planning problem in the context of security protocols where principals exchange messages can be formulated as follows:

1. the identification and creation of objects (messages, channels and principals)
2. the initial state is described and usually no messages are exchanged at the time the protocol starts.
3. the goal state that the principal will try to achieve.
4. a description of the possible actions that the principal can perform and what preconditions and effects are related to the actions.

4 Analysis

FF should do the same job as Algorithm 1 in [1] making the function syn() unnecessary when we map our model to planning. The planner should also automatically take care of the synthesis rules derived in [7] that are based on the inference rules of the logic. One action in our PDDL file should be send(). Knowing how to implement send() is the first approach (see Appendix).

We use the BSW-logic to analyze the Woo-Lam protocol. It uses three principals and three channels. We assume that in the initial state of the protocol, all the principals should be able to read from the public channel. So there is a channel C_p between principals A, B and S for which

$$r(C_p) = w(C_p) = \{A, B, S\}$$

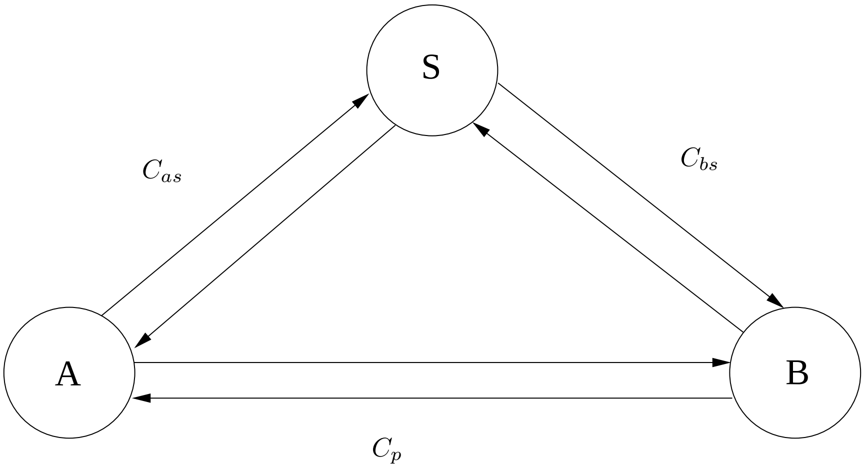

Figure 1: The Woo-Lam protocol with principal A, B and the server S.

meaning that only A, B and S can send and receive messages via C_p. We also have

$$r(C_{as}) = w(C_{as}) = \{A, S\}$$

and

$$r(C_{bs}) = w(C_{bs}) = \{B, S\}$$

meaning that only A and S can send and receive messages via C_{as} and only B and S can send and receive messages via C_{bs}. With this information our planner has some of its initial values. So the requirement specification in [1] section $A1$ needs to be mapped into our fact file. We start by creating the two predicates

$$(\text{readable } ?c \ - \ \mathit{channel} \ ?p \ - \ \mathit{principal})$$

meaning that channel C is readable by principal P or principal P is a reader of the channel C, denoted by $P \in r(C)$. Similarly, principal P is a writer of the channel C, denoted by $P \in w(C)$ written in PDDL as

$$(\text{writable } ?c \ - \ \mathit{channel} \ ?p \ - \ \mathit{principal})$$

The assumptions in section $A1$ are our initial values in the fact file. The initial values for readable and writable for the objects are straightforward, but when implementing the formula ϕ, it gets a lot more complicated. For example, the initial formula

$$A| \equiv \sharp(Na)$$

meaning principal A believes that the nonce Na is fresh where $\phi = \sharp(Na)$. How do we implement this in PDDL?

Let us show another example that has to do with a formula. Principal A believes that it can write to the channel C_{as}. It also believes that S can write to the same channel C_{as}. The implementation for A believes X is very easy

$$(\text{believes } A \ X)$$

but because X can be a formula too, the implementation for A believes A is writable in PDDL should look something like

$$\text{(believes } A \text{ writable } Cas\ A)$$

how to implement that in PDDL is the major obstacle for the moment. The planner will display an error "unknown constant WRITABLE in literal BELIEVES. check input files".

We express two goals of the protocol with the following formula:

$$G_1 \triangleq A| \equiv (B \parallel\!\sim (A, Na))$$

meaning principal A believes that principal B says the nonce Na to principal A. The goal of this protocol is to convince A that it talks to B

$$G_2 \triangleq B| \equiv (A \parallel\!\sim (B, Nb))$$

meaning principal B believes that principal A says the nonce Nb to principal B. The goal of this protocol is to convince B that it talks with A.

Again, the problem with implementing the formula in PDDL, but this time it is a bit more complicated because of the compound message (A, Na) and (B, Na)

5 Problems and Questions?

1. X can be both a message and a formula. How should that be implemented in PDDL?
2. Should Na,Nb, phi and x all be of type message and should the types nonce and formula be removed so that only the type message is left?
3. Could we solve the formula problem if we create an object of type formula and work from that, thus separating the formula from the type message? That would give us the predicate

   ```
   (believes ?p - principal ?phi - formula)
   ```

4. Are there more possible actions than just send?
5. Is the predicate *has* necessary when we have the predicate *received*? Does it have the same properties? If so, can we then delete *has* and only use received?

6 Conclusions

This paper shows that PDDL is not the action language to use when trying to model an abstract dynamic world like a security protocol based on beliefs logic. No known planners are yet released that can solve this problem. There is an action language called GOLOG [9] developed here at the University of Toronto that looks very promising. GOLOG has been formalized using the situation calculus which is basically a second order language designed for dynamically changing worlds. At this moment work is done by Jens Classen, Patrick Eyerich and Gerhard Lakemeyer [10]to integrate GOLOG and planning in the sense that planning problems, formulated as part of a GOLOG program, are solved by a modern planner during the execution of the program.

7 Acknowledgments

Many thanks to Professor J. Christopher Beck at the university of Toronto for suggesting the idea for this project. The weekly meetings discussing the work has been very helpful. Of course, discussion and support has also been provided by the very helpful people at the Toronto Intelligent Decision Engineering Laboratory.

References

[1] Hongbin Zhoou and Simon N. Foley. Fast automatic synthesis of security protocols using backward search. In *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 1–10. ACM Press, 2003.

[2] Adrian Perrig and Dawn Song. Looking for diamonds in the desert — extending automatic protocol generation to three-party authentication and key agreement protocols. In *In Proceedins of 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[3] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning Theory and Practice*. Morgan Kaufman, 2004.

[4] Fabio Massacci. Breaking security protocols as an AI planning problem. In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, pages 286–298. Springer-Verlag, 1997.

[5] D. McDermott. Pddl – the planning domain definition language, 1998.

[6] J. Hoffmann. FF: The fast-forward planning system, 2001.

[7] Buttyan, Staamann, and Wilhelm. A simple logic for authentication protocol design. In *PCSFW: Proceedings of The 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[8] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. In *ACM Transactions on Computer Systems*, volume 8, pages 18–36. IEEE Computer Society Press, 1990.

[9] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[10] Jens Claen, Patrick Eyerich, Gerhard Lakemeyer, and Bernhard Nebel. Towards an integration of golog and planning. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*. AAAI Press, 2007.

Appendix

Code

```
;; Specifications in PDDL2.1 of the project operator file
;;
;; By Samuel Gabrielsson
;;

(define (domain operators)
 (:requirements :strips :typing )
 (:types
  channel       ; Three channels declared Cas, Cbs and Cp
  principal     ; Three principals (agents) A, B and S
  message       ; Message X.
;  nonce         ; Nonce - values used only once (usually a random number), Na and Nb
;  formula       ; Formula phi
 )

 (:predicates
   (readable ?c - channel ?p - principal)       ; Channel C is readable by principal P.
   (writable ?c - channel ?p - principal)       ; Channel C is writable by principal P.

   (has ?p - principal ?x - message)            ; The principal P has message X.
   (received ?p - principal ?x - message ?c - channel)
                                                ; Principal P has received message X
                                                ; via the channel C.

   (sees ?p - principal ?x - message)           ; Principal P sees message X.
   (once_said ?p - principal ?p - message)      ; Principal P once said message X.
   (says ?p - principal ?x - message)           ; Principal P says message X
   (fresh ?x - message)                         ; Message X is fresh meaning it really
                                                ; is a valid message.
   (believes ?p - principal ?x - message)       ; Principal P belies that X is true.
                                                ; Should be a formula?
 )

;; sends a message X between two adjecent principals via some channel C.
 (:action send
   :parameters (?x - message ?from ?to - principal ?c - channel)
   :precondition (and (readable ?c ?from) (writable ?c ?from)
      (has ?from ?x))
   :effect (and (readable ?c ?to) (writable ?c ?to)
(has ?to ?x) (not (has ?from ?x)))
 )
)

;; Project fact file
;; The Woo-Lam protocol
;; By Samuel Gabrielsson
```

```
;;

(define (problem fact1)
  (:domain operators)

  (:objects
   A B S - principal
   Cas Cbs Cp - channel
   X Na Nb phi - message
;    Na Nb - nonce
;    phi - formula
  )

;; The initial state.
  ; A can not read Cbs and B can not read Cas
  (:init
;    (believes A writable Cas A)
   (readable Cas A) (readable Cas S)
   (readable Cbs B) (readable Cbs S)
   (readable Cp A) (writable Cp A)
   (readable Cp B) (writable Cp B)
   (readable Cp S) (writable Cp S)
   (has A X)
  )

;; The task!
  (:goal
;    (has B X)
;    (and (has B X) (has S X))
   (or (has B X) (has S X))
  )
)
```