Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (2) 323-328 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Directed Model Checking for Security Protocols

Q.U.A. NIZAMANI, H.A. NIZAMANI, F.H. CHANDIO , M.S. CHANDIO , N.H. ARIJO*

Institute of Maths and Comp. Science, University of Sindh, Jamshoro, Pakistan
Email: hanizamani@gmail.com : chandiofida@gmail.com : mschandio@gmail.com : niaz_arijo@hotmail.com
Corresponding author: Q.U. A. NIZAMANI, Email: q_agro@hotmail.com Ph. No. +92-229213177

Received 28th May 2012 and Revised 10th June 2012

Abstract: We present in this paper the experimental results for the heuristic suggested in (Nizamani, *et al.*, 2009) for security protocol verification. The results show that approach can considerably reduce the size of the state space while finding attacks. Further, we have shown that our approach can verify large protocols which could not be verified earlier due to state space explosion problem.

Keywords: Model Checking, Heuristics, Security Protocols.

1. INTRODUCTION

In this paper we give account of a few experimental results on the use of directed methods in model checking of security protocols. The main aim of this paper is to highlight the merits of a theoretical framework we presented in (Nizamani and Tuosto, 2009) where we introduced a simple heuristic and proved its correctness. Technically, our heuristic method is applied to a symbolic model checker hinging a process calculus and on an ad-hoc logic introduced in (Ferrari, *et al.*, 2008). The calculus (called cIP after cryptographic interaction pattern) is used to model protocols and its operational semantics yields the state space upon which security properties expressed in the logic (called PL after protocol logic) are checked. Although, the heuristic is defined for c*IP* and *PL*, it is rather general and could be adapted to other verification frameworks; in fact, it is based on how variables ranging over (instances of) principals are quantified. For instance, it would be straightforward to adapt the heuristic to a framework based on correspondence assertions (as e.g. (Boreale *et al.*, 2005). Directed methods have been applied to classical model checking but as observed e.g., in (Eldekemp *et al.*, 2001, Eldekemp *et al.*, 2004), it could be argued that they are not as effective as one would have expected. One of the issues left open in (Nizamani and Tuosto, 2009) was actually the lack of evidence of the effectiveness of our heuristic methods. Here, we tackle this problem and give some experimental results that confirm what we conjectured in (Nizamani, and Tuosto, 2009), namely that directed model checking of security protocols could yield more effective searches of attacks. In fact, we have implemented the heuristic introduced in (Nizamani and Tuosto, 2009) by integrating it into an existing symbolic model-checker for security protocols based on c*IP* and *PL*. More precisely, we modified the symbolic model checker ASPASyA (Baldi *et al.*, 2005, Ferrari *et al.*, 2008) so that the state space is cut and searched according to our heuristic. The heuristic analyses the PL formula expressing the security property of interest to weight the state space according to the likeliness that a state is on an attack trace. Interestingly, some states can be cut[1] and the search in (the smaller state space) is *then* driven according to the weights of states computed by the heuristic.

The initial experimental results presented here are rather encouraging and show that, unlike in the general case, directed methods provide substantial improvements when applied to the model checking of security protocols. For instance, in one case, the state space is reduced by a factor of ten. Remarkably, the gain in efficiency that we obtain is quite considerable despite the simplicity of our heuristic. This hints that further improvements could be obtained by exploiting other kinds of heuristics. The correctness of our heuristic (that is, cut states cannot be on attack traces) is shown in (Nizamani and Tuosto, 2009).

2. BACKGROUND

We describe the ASPASyA symbolic model checker and our heuristic.To illustrate our results we use a symbolic model checker for security protocol

*Institute of Information and Communication Technology, University of Sindh, Jamshoro, Pakistan

verification called ASPASyA (after Automated Tool for Security Protocol Analysis via a Symbolic Approach). The framework consists of a process calculus called *cIP* (after cryptographic interaction pattern) and of a logic called *PL* (after protocol logic) to respectively represent security protocols and their properties.

We elucidate the main aspects (and the notation) of *cIP* and *PL* by means of the well-known Needham-Shroeder (NS) public key protocol:

i) $A \rightarrow B : \{A, na\}_B^+$
ii) $B \rightarrow A : \{na, nb\}_A^+$
iii) $A \rightarrow B : \{nb\}_B^+$

where, in step i) the initiator *A* sends the responder *B* her identity and a nonce na encrypted with *B*'s public key *B*+ ; in step ii), B responds to the nonce challenge with a fresh nonce *nb* encrypted with *A*+ , the public key of A; the protocol ends when A sends back B the nonce *nb* encrypted with *B*'s public key.

In *cIP*, principals have an identity, a list of open variables, and the communication actions corresponding to their role in the protocol. Open variables are meant for modeling sharing of information (keys and identities) among principals. For instance, the participants of the NS in *cIP* are:

A :(x) [out({A, na}x+). i n({na, ?y}A−). out({y}x+)] (1)
B : ( )[in({?z, ?u}B−). out({u, nb}z+). in({nb}B−)]

In **(1)**, the open variable x of principal A is meant to be instantiated with the identity of an instance [1] of B. The principal A first executes the output action and then waits for a message which should match the pattern specified in the in action; A will accept[2] any pair encrypted with her public key whose first component is the nonce *na* and assign the second component to *y*. The responder *B* has the complementary behaviour.

The semantics of *cIP* models the Dolev-Yao (Dolev et al., 1983) intruder and is specified as a labeled transition system where states are triples $\langle C, \chi, \kappa \rangle$ made of a finite set *C* of principal instances (called context), a mapping χ of variables to messages, and a set of messages κ representing the intruder's knowledge. Transitions take the format:

$$t : \langle C, \chi, \kappa \rangle \xrightarrow{\alpha} \langle C', \chi', \kappa' \rangle$$

where the label α can be any cIP action (e.g., out(M), or in(M)) or a join transition. A join transition can be defined as Transitions take the form

$$t_{join} : \langle C, \chi, \kappa \rangle \xrightarrow{join\ A_i} \langle C', \chi', \kappa' \cup \{A_i, A_{i^+}\} \rangle$$

We adopt the definition of PL formulae given in [Ferrari, G, et al., 2008]:

$\varphi, \psi ::= x_i = m \mid \triangleright m \mid \forall i : A.\psi \mid \exists i : A.\psi \mid \neg\psi \mid \psi \wedge \varphi \mid \psi \vee \varphi$

where x_i are indexed variables and m stands for any message. The atomic formulae $x_i = m$ and $\triangleright m$ hold respectively when the variable x_i is assigned the message m and when the intruder "knows" *m*. Notice that quantification is over indexes i because *PL* predicates over the instances of the principals.

As an example of *PL* security formula consider the formula ψ_{NS} predicating on (instances of) the NS protocol:

$\psi_{NS} = \forall i : B. (\triangleright nb_i \Longrightarrow zi = I) \vee (\exists j : A. z_i = A_j \Longrightarrow x_j = B_i)$

Informally, ψ_{NS} states that for all instances i of *B*, either the intruder is able to get nb_i when he is the intended responder of the session or there exists an instance *j* of *A* such that B_i assumes to be connected to A_j then A_j also assumes to be connected to B_i.

Another interesting feature of ASPASyA is the use of join formulae, which permits to specify constraints on the way principals can be connected to each other. For instance, consider following join formula for NS protocol: $(\exists j : A. \text{true}) \wedge (\exists i : B. \text{true})$ which specifies that we are interested only in those runs of verification in which there is at least an instance of A and an instance of *B*.

3. HEURISTIC H_1

The mutually recursive functions H_s and H_t below are borrowed from (Nizamani, and Tuosto.,

1 A principal instance is a cIP principal indexed with a natural number; for example, two instances corresponding to principals in **(1)** are

A2 : (x2)[out({A2 , na2 }x+). in({na2 , ?y2 }A−). out({y2 }x+)]

B1 : ( )[in({?z1 , ?u1 }B−). out({u1 , nb1 }z+). in({nb1 }B−)

2 Communication in cIP is regulated by a pattern matching mechanism; a message {na, m}A+ matches the pattern {na, ?y}A− for any m.

2009) and define our first heuristic. Let s be a state and φ a PNF formula, H_s is given by

$$\mathcal{H}_s(s,\phi)=\begin{cases}\max\limits_{t\in s\Upsilon}\mathcal{H}_t(t,\phi), & s\Upsilon\neq\emptyset\\ -\infty, & \phi\equiv\forall i:A.\,\phi'\wedge s\Upsilon=\emptyset\wedge s\cap[A]=\emptyset\\ 0, & \text{otherwise}\end{cases}$$

where $s\gamma$ is the set of join transitions departing from s, [A] is the set of all instances of A and assuming $s = \langle\ C, \chi, \kappa\ \rangle$, $s \cap [A]$ stands for $\kappa \cap [A]$.

Let t be a transition from s to $s' = \langle\ C', \chi', \kappa'\ \rangle$ in $s\gamma$, H_t is

$$\mathcal{H}_t(t,\phi)=\begin{cases}1+\mathcal{H}_s(s',\phi') & \phi\equiv\forall i:A.\,\phi'\wedge\kappa'\cap[A]\neq\emptyset,\\ 1+\mathcal{H}_s(s',\phi), & \phi\equiv\exists i:A.\,\phi'\wedge\kappa'\cap[A]=\emptyset,\\ \mathcal{H}_s(s',\phi'), & \phi\equiv\exists i:A.\,\phi'\wedge\kappa'\cap[A]\neq\emptyset,\\ \mathcal{H}_s(s',\phi), & \phi\equiv\forall i:A.\,\phi'\wedge\kappa'\cap[A]=\emptyset,\\ 0 & otherwise.\end{cases}$$

We explain the heuristic discussing how it works on the NS protocol given in **(1)** and refer the reader to (Baldi et al., 2005) for a deeper and detailed presentation. **(Fig. 1)** shows join transitions for NS with two instances (the top tree is isomorphic to the bottom one and it is given to assign names to states and transitions for illustration purposes); we consider states only by contexts as the other components of states (namely χ and κ) are immaterial for our heuristic.

Initially Hs is invoked with s_0 and ψ_{PNF} that satisfies the first condition in the definition of H_s; hence H_t is invoked to weight s_0 with the maximum of the weights of t_1 and t_2. In order to weight t_1 (resp. t_2), H_t is invoked on t_1 (resp. t_2) alongwith ψ_{PNF} and finds that case 4 (resp. 1) of the definition of Ht applies. Justification follows from the fact that first quantifier in ψ_{NS} is ∀ quantified on instances of B. Thus case 4 (resp. case 1) is applied in case of of t_1 (resp. t_2) as an instance of B is added (not added) in the upcoming state. Hence

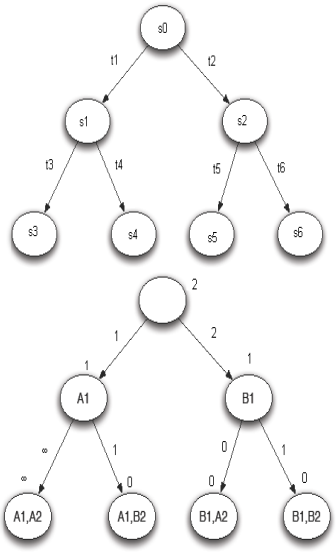

Fig. 1. Weighted join transitions for NS with two participants

Ht ($t1$, ψ_{PNF}) yields the weight of s_1 for ψ_{PNF} while H_t (t_2, ψ_{PNF}) increments and assigns t2 the weight of s2 for ψ_0 where

$$\Psi' = \exists 1 : A.(\triangleright\ nbk \Longrightarrow z_i = I) \vee (z_k = A_l \Longrightarrow x_l = B_k)$$

namely ψ' is obtained by consuming the universal quantifier from ψ_{PNF}. The intuition behind this is that s_2 contains an instance of B which eliminates the need to look for an instance of B in upcoming states.

We illustrate only the calculation of the weight of s_1 (the weight of s_2 is computed similarly). The invocation Hs (s_1, ψ_{PNF}) (done by Ht (t_1, ψ_{PNF})) enables case 1 of the definition of Hs ; therefore, the weights of t_3 and t_4 which respectively qualify for case 4 and case 1 of Ht are computed by H_t which returns the weight of s_3 for t_3 and the incremented weight of s_4 for t_4. Finally, Hs for s_3 will return $-\infty$ as it is weighted on ψ_{PNF} and s_3 does not contain an instance of B (case 2 of Hs). Instead the weight of s_4 is 0 as it is weighted on ψ' (case 3 of Hs). The resulting weighted tree is given in the bottom tree of **(Fig. 1)**.

4 EXPERIMENTAL RESULTS AND THEIR ANALYSIS

We have implemented the heuristic given in section 3 and tested our code on various protocols under different conditions and the results are reported

in the (**Tables. 1, 2 and 3 shown below).** The first column of the Table1 and Table 2 list the protocols that have been taken as test cases. Column no 2 in both tables report no. of instances taken into consideration in each run of the verification. The last two columns in both tables show the results . The column with complete state space heading means that number refers to the states explored for complete state space. Whereas column with the heading first attack means that the number corresponds to the number of states explored in discovering first attack.

Table 1. Results for H_1 heuristic for True Join

Protocols	instances	Complete State Space		First Attack	
		A	H 1	A	H 1
NS	2	158	129	67	25
NS	3	5183	4942	1288	197
KSL	3	10240	3332	NA	NA
KSL	4	OF	2717158	NA	NA
WMF	3	1055	563	451	386
Carlsen	3	9792	867	NA	NA
Denning	3	76273	31379	39438	22
BY	2	1292	1134	335	873
BY	3	236497	226562	28995	132115
Bilke	3	6316	5700	NA	NA

Table 2. Results for H_1 heuristic for a specified join

Protocols	instances	Complete State Space		First Attack	
		A	H 1	A	H 1
NS	2	50	49	39	38
NS	3	1589	1588	1048	599
KSL	3	550	275	NA	NA
WMF	3	522	347	497	222
Carlsen	3	534	259	NA	NA

Table 3. Results for H_1 for Carlsen Protocol

	Tot. starting nodes				Total Nodes			
No of instances	1	2	3	4	1	2	3	4
A	3	6	10	15	18	249	9802	OF
H1	1	3	6	10	7	4	873	42377

An entry marked NA in the 'First Attack' column means that the numbers are same as given in complete state space exploration as the protocol is attack free. The columns with the heading A and H1 gives results respectively for ASPASyA and H-ASPASyA (i.e., ASPASyA with heuristic H1). The column entitled starting nodes in Table 3 refers to a stage in verification process where the required no. of instances have joined the context. In **(Fig 2),** this corresponds to level 1 of the tree where total number of nodes is 3. However, one node is shaded depicting the pruned part of the state space in H-ASPASyA. Note that in **(Table 3)**, the starting nodes in case of NS with 2 instances is 3 in ASPASyA and 2 in H-ASPASyA.

Table 1 gives the number of states explored for various protocols with different number of instances and with respect to 'true' join which means there is no constraint on the principal instances to join the protocol. Table 2 shows the number of states explored for various protocols, all tested with respect to their specified join and 3 instances in each case. Table 3 has been designed so that the behavior of a single protocol can be analyzed; in this case we have run the Carlsen protocol with various number of instances, each run has used the same *PL* formula and true join.

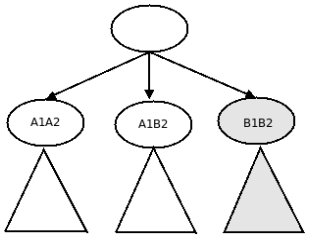

Fig. 2 Partial state space for NS

We report following observations after analyzing the results:

The number of attacks found in ASPASyA and H-ASPASyA is same and nature of the attacks is also similar.

Increasing the no of instances increase the size of the state space that can be controlled by applying heuristic. It follows from **(Table 1)** that in case of NS executed with 3 principal instances there is a reduction of 4.63 % of the total state space explore in ASPASyA. We get even better results in widefrog protocol where the reduction in case of 4 instances is almost 46% of the original state space.

The join formula as mentioned earlier is a mechanism to restrict the state space. In the absence of any join formula the possibilities for attacks increase and result into bigger state spaces. Our heuristic plays an important role in the absence of join formulae and gives impressive results. It can be observed from **(Table 1)** that there is a huge reduction in state space in case of carlsen protocol. In case of carlsen

protocol without a join formula we are able to cut-down almost 90% of original state space. Note that in few cases such as NS checked with respect to their join formulae and 2 instances, ASPASyA and SPASyA give same results (in case of complete state space exploration). This reflects the fact that a carefully designed join formula can also successfully avoid the exploration of less likely states for attacks. However, we comment that join formulae are to be specified by the user and a "weaker" join formula will not give impressive results as our heuristic produces. This is supported by the verification of Widefrog protocol with 3 instances where almost 33% state space is reduced by H-ASPASy Although a join formula has also been specified **(Table 2)**. Further with a built-in heuristic there is no additional overhead over the user to design a PL join formula.

One of the advantage of this heuristic is verification of large protocols with H-ASPASyA which previously could not be done with ASPASyA due to state space explosion problem. In **(Table 3)**, it can be seen that carlsen verification with 4 instances in case of ASPASyA gives an error and terminates as it could not handle the large state space (OF stands for Stack-Overflow). However, H-ASPASyA successfully verifies the same protocol after exploring 42377 states and gives results.

5. CONCLUDING REMARKS

We have given account of some experimental results of a heuristic for model checking security protocols. The heuristic exploits a security formula to prune and re-order states accordingly. The experimental results show a gain in efficiency achieved both in terms of space. Furthermore, the heuristic is effective in those cases also when there is no attack. It is also important to mention that attacks reported by ASPASyA and H-ASPASyA are the same (as shown in (Nizamani and Tuosto, 2009), the heuristic is correct).

The results are much noticeable in case of large state spaces (protocols run with large number of principal instances). In particular, we are able to verify larger protocols with H-ASPASyA which ASPASyA could not verify due to larger state space.

Heuristic methods have been widely applied to general model checking. At the best of our knowledge, their feasibility for security protocol verification in particular has not been studied much. In (Basin, 1999), some heuristics for directed model checking of security protocols have been reported. However, the heuristics are very basic and as observed in (Basin *et al.,* 2003), they do not give much efficiency. The most significant work for general directed model checking is reported in (Eldekemp, *et al.,* 2004, Eldekemp *et al.,* 2001). In (Eldekemp, *et al.,* 2004), the heuristics are defined for safety and liveliness while in (Eldekemp, *et al.,* 2001) they are tailored for assertion violation and deadlock detection. However, at the best of our knowledge, the heuristics in (Eldekemp, *et al.,* 2001, Eldekemp *et al.,* 2003) allows pruning of state space for few trivial cases.

Some promising heuristics for model checking have been reported in (Kupferschmid *et al,* 2007) which are based on calculating the distance to an error state by first doing some abstractions. The heuristics provide substantial reduction in memory as compared to the ones suggested in (Eldekemp *et al.,* 2001, Eldekemp *et al.,* 2004). However, they do not provide any cut in the state space resulting in complete state space exploration when there is no attack. We consider the heuristics suggested in (Gradara *et al.,* 2007) to be most relevant to our approach where the heuristics utilize the model and formula to be verified for rating the states. We intend to do a deeper comparison of both works.

REFERENCES:

Baldi, G., A. Bracciali, G. Ferrari, and E. Tuosto, (2005) A Coordination-based Methodology for Security Protocol Verification. In Electronic Notes in Theoretical Computer Science, Elsevier, 23-46.

Basin, D., (1999) 'Lazy infinite-state analysis of security protocols'. In Proceedings of the International Exhibition and Congress on Secure Networking - CQRE (Secure) '99, Springer-Verlag, London, UK 30-42.

Basin, D., S. Mödersheim, and L. Vigano, (2003) An on-the-fly model-checker For security protocol analysis. In Proceedings of Esorics'03, Springer-Verlag, Heidelberg, LNCS (2808) 253-270.

Boreale, M., and M. Buscemi, (2005) A Method for Symbolic Analysis of Security Protocols. Theoretical Computer Science 338 **(1-3):** 393-425.

Dolev, D., and A.Yao, (1998) On the security of public key protocols. IEEE Transactions on Information Theory 29 **(2):**198-208.

Edelkamp, S., A. L Lafuente, and S. Leue, (2001) Protocol verification with heuristic search. In In AAA Symposium on Model-based Validation of Intelligence , 75-83.

Edelkamp, S., S. Leue, and A. L. Lafuente, (2004) Directed explicit-state model checking in the validation of communication protocols. Int. J. Softw. Tools Technol. Transf. 5 **(2):** 247-267.

Ferrari, G., A. Bracciali, and E. Tuosto, (2008) A symbolic framework for multifaceted security protocol analysis. Internation Journal of Information Security 7**(1):** 55-84.

Gradara, S., A. Santone, and M. L.Villani, (2007) Formal verification of concurrent systems via directed model checking. Electron. Notes Theor. Comput. Sci. **(185):** 93-105.

Kupferschmid, S., K. Draäger, J. Hoffmann, B. Finkbeiner, H. Dierks, A. Podelski, and G. Behrmann, (2007) UPPAAL/DMC- abstraction-based heuristics for directed model checking. In TACAS , 679-682.

Nizamani, Q., and E. Tuosto, (2009) Heuristic Methods for Security Protocols. EPTCS **(7):** 61-75.