Heuristic Methods for Security Protocols[*]

Qurat ul Ain Nizamani
Department of Computer Science
University of Leicester, UK
`qn4@mcs.le.ac.uk`

Emilio Tuosto
Department of Computer Science
University of Leicester, UK
`et52@mcs.le.ac.uk`

Model checking is an automatic verification technique to verify hardware and software systems. However it suffers from state-space explosion problem. In this paper we address this problem in the context of cryptographic protocols by proposing a *security property-dependent* heuristic. The heuristic weights the state space by exploiting the security formulae; the weights may then be used to explore the state space when searching for attacks.

1 Introduction

Security protocols present many interesting challenges from both pragmatic and theoretical points of view as they are ubiquitous and pose many theoretical challenges despite their apparent simplicity. One of the most interesting aspects of security protocols is the complexity of the verification algorithms to check their correctness; in fact, under many models of the intruder, correctness is undecidable and/or computationally hard [15, 12, 11, 26].

Many authors have formalised security protocols in terms of process calculi suitable to define many verification frameworks (besides model checking, path analysis, static analysis, etc.) [21, 1, 7, 8]. Model checking (MC) techniques have been exploited in the design and implementation of automated tools [13, 23] and *symbolic* techniques have been proposed to tackle the state explosion problem [3, 9, 10, 18].

This paper promotes the use of directed MC in security protocols. We define a heuristic based on the logic formulae formalising the security properties of interest and we show how such heuristic may drive the search of an attack path. Specifically, we represent the behaviour of a security protocol in the context of the (symbolic) MC framework based on the cIP and $\mathscr{PL}$, respectively a cryptographic process calculus and a logic for specifying security properties introduced in [18]. An original aspect of this framework is that it allows to explicitly represent instances[1] of participants and predicate over them.

Intuitively, the heuristic ranks the nodes and the edges of the state space by inspecting (the syntactical structure of the) formula expressing the security property of interest. More precisely, the state space consists of the transition system representing possible runs of a protocol; our heuristic weights states and transitions considering the instances of principals that joined the context and how they are quantified in the security formula. Weights are designed so that most promising paths are tried before other less promising directions. Rather interestingly, the heuristic can rule out a portion of the state space by exploring only a part of it. In fact, we also show that the heuristic may possibly cut some directions as they cannot lead to attacks; in fact, the heuristic is proved to be correct, namely no attacks can be found in the portion of the state space cut by our heuristic.

---

[*] The authors thank the anonymous reviewers for their valuable suggestions and Alberto Lluch-Lafuente for the useful discussions.

[1] This key feature of the framework is supported by the use of *open variables*, a linguistic mechanism to tune and combine instances that join the run of a protocol.

M. Boreale, S. Kremer (Eds.):
Security Issues in Concurrency (SecCo'09)
EPTCS 7, 2009, pp. 61–75, doi:10.4204/EPTCS.7.5

At the best of our knowledge, heuristic methods have not yet been explored to analyse security protocols (at least in the terms proposed in this paper) which may be surprising. In general, many of the features of heuristics fit rather well with the verification of security protocols. More precisely, (i) optimality of the solution ("the attack") is not required when validating protocols (violations of properties of interest are typically considered equally harmful), (ii) the graph-like structures (e.g., labeled transition systems) representing the behaviour of security protocols usually have 'symmetric regions' which may be ignored once one of them is checked, (iii) heuristic search may be easily combined with several verification frameworks and particularly with MC. The lack of such research thread is possibly due to the fact that it is in general hard to define heuristics for security protocols. In fact, typically heuristics are tailored on (properties of) a *goal state* and measure the "distance" from a state to a goal state. In the case of the verification of security protocols, this would boil down to measure the distance from an attack where a security property is violated. Therefore, designing heuristics suitable to improve MC of security protocols is hard as attacks cannot be characterised beforehand.

Here, we argue that heuristic search may be uniformly used in the verification of security protocols and provide some interesting cases of how our approach improves efficiency. In fact, we will illustrate how the use of the heuristic can greatly improve the efficiency of the search by cutting the directions that cannot contain attacks.

Our approach seems to be rather promising, albeit this research is in an initial stage, it can be extended in many directions. Finally, we argue that our proposal can be applied to other verification frameworks like [8] or inductive proof methods like [24, 17] (see § 5).

Structure of the paper. § 2 summarizes the concepts necessary to understand our work; § 3 yields the definition of our heuristic which is then evaluated and proved correct in § 4; § 5 concludes the paper and discusses related work.

2 Background

This section fixes our notation (§ 2.1) and a few basic concepts on informed search largely borrowed from [25] (§ 2.2)

2.1 Expressing security protocols and properties in cIP and $\mathscr{PL}$

We adopt the formal framework introduced in [18] consisting of the cIP (after *cryptographic Interaction Pattern*) process calculus and the $\mathscr{PL}$ logic (after *protocol logic*) to respectively represent security protocols and properties. Here, we only review the main ingredients of cIP and $\mathscr{PL}$ by means of the Needham-Shroeder (NS) public key protocol and refer the reader to [18] for a precise presentation.

The NS protocol consists of the following steps

$$\begin{array}{lll} 1. & A \to B: & \{na, A\}_{B^+} \\ 2. & B \to A: & \{na, nb\}_{A^+} \\ 3. & A \to B: & \{nb\}_{B^+} \end{array}$$

where, in step 1 the initiator A sends to B a nonce na and her identity encrypted with B's public key B^+; in step 2, B responds to the nonce challenge by sending to A a fresh nonce nb and na encrypted with A^+, the public key of A; A concludes the protocol by sending back to B the nonce nb encrypted with B's public key.

In cIP principals consist of their identity, the list of *open variables* and the actions they have to perform in the protocol. A cIP principal can either send or receive messages from a public channel using the *out* and *in* actions respectively. The NS protocol can be formalized in cIP as follows:

$$
\begin{array}{llll}
A:(r)[& out(\{na,A\}_{r^+}). & \qquad B:()[& in(\{?x,?y\}_{B^-}). \\
 & in(\{na,?z\}_{A^-}). & & out(\{x,nb\}_{y^+}). \\
 & out(\{z\}_{r^+}) \ ] & & in(\{nb\}_{B^-}) \ ]
\end{array}
\tag{1}
$$

The principal A (resp. B) in (1) represents the initiator (resp. the responder) of the NS protocol. The open variable r is meant to be bound to the identity of the responder. The principal A first executes the output action and then waits for a message which should match the pattern specified in the *in* action. More precisely, A will receive any pair encrypted with her public key whose first component is the nonce na; upon a successful match, the second component of the pair will be assigned to the variable z. For instance, the $\{na,M\}_{A^+}$ matches $\{na,?z\}_{A^-}$ for any M and would assign M to z.

We adopt the definition of $\mathscr{PL}$ formulae given in [18]:

$$\phi,\psi \ ::= \ x_i = m \ \mid \ \kappa \rhd m \ \mid \ Qi:A.\psi \ \mid \ \neg\psi \ \mid \ \psi\wedge\phi \ \mid \ \psi\vee\phi$$

where Q ranges over the set of quantifiers $\{\forall,\exists\}$ and x_i are indexed variables (a formula without quantifiers is called *quantifier-free*).

The atomic formulae $x_i = m$ and $\kappa \rhd m$ hold respectively when the variable x_i is assigned the message m and when κ (representing the intruder's knowledge) can derive m. Notice that quantification is over indexes i because $\mathscr{PL}$ predicates over the *instances* of the principals concurrently executed. A principal *instance* is a cIP principal indexed with a natural number; for example, the instance of the NS initiator obtained by indexing the principal A in (1) with 2 is

$$A_2:(r_2)[out(\{na_2,A_2\}_{r_2^+}).in(\{na_2,?z_2\}_{A_2^-}).out(\{z_2\}_{r_2^+})] \tag{2}$$

we let $[X]$ be the set of all instances of a principal X; e.g., the instance[2] in (2) is in $[A]$ (§ 3 illustrates how the transition system of cIP instances of a protocol is obtained).

As an example of $\mathscr{PL}$ formula consider the formula ψ_{NS} predicating on (instances of) the NS protocol:

$$\forall i:A.\ \exists j:B\ (x_j = na_i \ \wedge \ z_i = nb_j).$$

The formula ψ_{NS} states that for all instances of A there should be an instance of B that has received the nonce na_i sent by A_i and the nonce nb_j is received by the instance A_i.

2.2 Basics of heuristics

As mentioned in § 1, the approaches such as symbolic MC can be used to tackle the problem of state space explosion. However, even with the use of such approaches the search space can grow enormously. It is therefore desirable to look into methods through which search space can be generated/explored more efficiently using *informed search algorithms* that are characterized by the use of a *heuristic function* (also called evaluation function). A heuristic function assigns a weight to nodes by estimating their "distance" from a *goal node*.

---

[2] Hereafter, we denote an instance simply by the indexed name of the principal; for example, the instance above will be referred to as A_2.

We recall here the basic concepts on heuristic algorithms by means of a simple example and refer the reader to [25] for a deeper presentation.

The *n*-puzzle (also known as the sliding-block or tile-puzzle) is a well-known puzzle in which the goal is to move square tiles by sliding them horizontally or vertically in one empty tile. For $n = 8$ the goal configuration is depicted in Figure 1; a possible initial configuration is in Figure 2. The problem of finding the shortest path leading to the goal configuration is NP-hard.

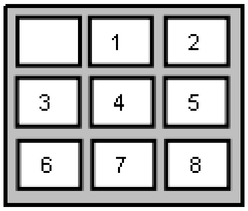


Figure 1: The 8-puzzle goal configuration

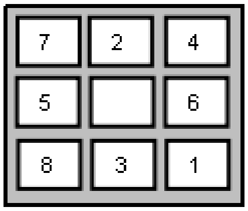


Figure 2: A possible start configuration

A very simple heuristic (cf. [25]) for 8-puzzle can be given by

$$h_1 = \text{number of misplaced tiles.}$$

For each configuration, h_1 counts the number of misplaced tiles with respect to the goal configuration. For instance, h_1 weights 8 the configuration in Figure 2 since all the tiles are misplaced.

Another heuristic (cf. [25]) for 8-puzzle is the one that exploits the so called *Manhattan distance*

$$h_2 = \text{sum of the Manhattan distances of non-empty tiles from their target positions.}$$

So the configuration in Figure 2 is weighted 18 by h_2.

An important property of heuristics is *admissibility*; an admissible heuristic is one that never over estimates the cost to reach the goal node. Both h_1 and h_2 are admissible; in fact, h_1 is clearly admissible as each misplaced tile will require at least one step to be on its right place, and h_2 is also admissible as at each step the tiles will be at most one step closer to goal. A non-admissible heuristic is $h_3 = h_1 * 4$; in fact, if only one tile is misplaced with respect to a goal configuration, h_3 will return 4 which is an overestimation of the distance to goal.

3 A Heuristic for Security Protocols

This section introduces our original contribution (§ 3.2), namely the heuristic for effectively searching the state space generated for MC cryptographic protocols. As mentioned earlier, there is not much work done in this regard. Specifically (to the best of our knowledge) no work exists that can prune a state space in verification of cryptographic protocols.

The heuristic function is defined on the security formula expressed in $\mathscr{PL}$.

The heuristic is efficient as it will not only guide the searching algorithm towards promising regions of the graph but can also prune those parts of the state space where attack cannot happen under a given security formula.

The heuristic is defined in terms of two mutually recursive functions $\mathscr{H}_s$ and $\mathscr{H}_t$ which assign weights to states and transitions respectively. The state space is obtained according to the semantics of cIP defined in [18]. For lack of space, an informal presentation of the semantics is given here.

3.1 The state space

A state consists of a tuple $\langle \mathscr{C}, \chi, \kappa \rangle$ where

- $\mathscr{C}$ is a context containing principal instances which joined the session,
- χ is a mapping of variables to messages, and
- κ is a set of messages representing the intruder knowledge.

A transition from one state to another can be the result of *out* and *in* actions performed by principal instances or of *join* operations non-deterministically performed by the intruder; join transitions may instantiate open variables by assigning them with the identity of some principal (provided it is in κ). Initially, $\mathscr{C}$ is empty and therefore the only possible transitions are join ones. When $\mathscr{C}$ contains an instance ready to send a message, an *out* transition can be fired so that the sent message is added to κ. If $\mathscr{C}$ contains a principal ready to receive a message, the intruder tries to derive from the messages in κ a message that matches the pattern specified in the input action (see § 2); if such a message is found χ is updated to record the assignments to the variables occurring in the input action.

For instance, a few possible transitions for the NS protocol are

$$s_0 \xrightarrow{join} s_1 \xrightarrow{join} s_2 \xrightarrow{out} s_3 \xrightarrow{in} s_4$$

where $s_0 = \langle\ \emptyset, \emptyset, \kappa_0\ \rangle$ with $\kappa_0 = \{I, I^+, I^-\}$, namely initially no principal instance joined the context, there is no assignment to variables, and the intruder only knows its identity and public/private keys.

The join transition from s_0 to s_1 adds a principal instance B_2 to the context yielding

$$\begin{aligned} s_1 = \langle\ & \{()[in(\{?x_2, ?y_2\}_{B_2^-}).out(\{x_2, nb_2\}_{y_2^+}).in(\{nb_2\}_{B_2^-})]\}, \\ & \emptyset, \\ & \kappa_1 = \kappa_0 \cup \{B_2, {B_2}^+\}\ \rangle \end{aligned}$$

that is, the intruder now knows B_2's identity and (by default) its public key. Similarly, the transition from s_1 to s_2 adds the principal instance A_1 to context and therefore

$$\begin{aligned} s_2 = \langle\ & \{()[in(\{?x_2, ?y_2\}_{B_2^-}).out(\{x_2, nb_2\}_{y_2^+}).in(\{nb_2\}_{B_2^-})], ()[out(\{na_1, A_1\}_{B_2^+}).in(\{na_1, ?z_1\}_{A_1^-}).out(\{z_1\}_{B_2^+})]\}, \\ & \{r_1 \mapsto B_2\}, \\ & \kappa_2 = \kappa_1 \cup \{A_1, {A_1}^+\}\ \rangle \end{aligned}$$

Notice that the open variable r_1 is now mapped to B_2.

The transitions from s_2 to s_3 is due to an *out* action

$$\begin{aligned} s_3 = \langle\ & \{()[in(\{?x_2, ?y_2\}_{B_2^-}).out(\{x_2, nb_2\}_{y_2^+}).in(\{nb_2\}_{B_2^-})], ()[in(\{na_1, ?z_1\}_{A_1^-}).out(\{z_1\}_{B_2^+})]\}, \\ & \{r_1 \mapsto B_2\}, \\ & \kappa_3 = \kappa_2 \cup \{na_1, A_1\}_{B_2^+}\ \rangle \end{aligned}$$

the prefix of A_1 is consumed and the message is added to the intruder's knowledge.

Finally, the transition from s_3 to s_4 is due to an *in* transition for the input prefix of B_2. The message $\{na_1, A_1\}_{B_2^+}$ added to the intruder's knowledge in the previous transition matches the pattern $\{?x_2, ?y_2\}_{B_2^-}$

specified by B_2, therefore the x_2 and y_2 are assigned to na_1 and A_1 respectively. Hence,

$$\begin{aligned} s_4 = \langle\ & \{out(\{na_1, nb_2\}_{A_1^+}).in(\{nb_2\}_{B_2^-})], ()[in(\{na_1, ?z_1\}_{A_1^-}).out(\{z_1\}_{B_2^+})]\}, \\ & \{r_1 \mapsto B_2, x_2 \mapsto na_1, y_2 \mapsto A_1\}, \\ & \kappa_3\ \rangle \end{aligned}$$

In our framework, join transitions can be safely anticipated before any other transition (Observation 10.1.3 in [28], page 174).

3.2 The heuristic

For simplicity and without loss of generality, we define the heuristic on *Prenex Normal Form* (PNF) formulae defined below.

Definition 1. *[Prenex Normal Form] A $\mathscr{PL}$ formula is in* prenex normal form *if it is of the form*

$$Q_1 i_1 : A_1. \cdots .Q_n i_n : A_n.\phi$$

where ϕ is a quantifier-free formula and, for $1 \leq j \leq n$, $Q_j \in \{\forall, \exists\}$, each i_j is an index variable, and A_j is a principal name.

Basically, a PNF formula is a formula where all the quantifiers are ”at top level”. Notice that, in Definition 1, it can be $n = 0$ which amounts to say that a quantifier free formula is already in PNF.

Theorem 3.1. *Any $\mathscr{PL}$ formula can be transformed into a logically equivalent PNF formula.*

Proof. Let the function $pnf : \mathscr{PL} \rightarrow \mathscr{PL}$ be defined as follows:

$$pnf(\psi) = \begin{cases} \psi & \psi \text{ is a quantifier-free formula.} \\ Qi : A.pnf(\psi') & \psi \equiv Qi : A.\psi' \\ \overline{Q}i : A.pnf(\neg\psi'') & \psi \equiv \neg\psi' \text{ and } pnf(\psi') \equiv Qi : A.\psi'' \\ Qi' : A.pnf(\psi_1'[i'/i] \wedge \psi_2) & i' \text{ fresh, } \psi \equiv \psi_1 \wedge \psi_2 \text{ and } pnf(\psi_1) \equiv Qi : A.\psi_1' \\ Qi' : A.pnf(\psi_1'[i'/i] \vee \psi_2) & i' \text{ fresh, } \psi \equiv \psi_1 \vee \psi_2 \text{ and } pnf(\psi_1) \equiv Qi : A.\psi_1' \end{cases}$$

The proof of theorem 3.1 follows from the properties of *pnf* given by Lemmas 3.2 and 3.3 below. □

Lemma 3.2. *For any $\mathscr{PL}$formula ψ, $pnf(\psi)$ is in PNF.*

Proof. We proceed by induction on the structure of ψ.

If ψ is a quantifier free formula then it is in PNF and, by definition of *pnf*, $pnf(\psi) = \psi$.

The inductive case is proved by case analysis.

- Assume ψ is $Qi : A.\psi'$, then by definition of *pnf*, $pnf(\psi) = Qi : A.pnf(\psi')$. By inductive hypothesis $pnf(\psi')$ is in PNF and therefore $pnf(\psi)$ is in PNF.
- If $\psi = \psi_1 \wedge \psi_2$ then, assuming $pnf(\psi_1) = Qi : A.\psi_1'$, by definition of *pnf*

  $$pnf(\psi) = Qi' : A.pnf(\psi_1'[i'/i] \wedge \psi_2)$$

  For fresh index i' not occuring in ψ_2. By inductive hypothesis $pnf(\psi_1'[i'/i] \wedge \psi_2)$ is in PNF, therefore $pnf(\psi)$ is in PNF.

- The case $\psi = \psi_1 \vee \psi_2$ is analogous.
- If $\psi = \neg\psi'$ then, assuming $pnf(\psi') = Qi : A.\psi''$, by definition of pnf, $pnf(\psi) = \overline{Qi} : A.pnf(\neg\psi'')$. By inductive hypothesis $pnf(\neg\psi'')$ is in PNF, therefore $pnf(\psi)$ is in PNF.

□

Lemma 3.3. *$pnf(\psi) \Leftrightarrow \psi$.*

Proof. We proceed by induction on the structure of ψ.

If ψ is a quantifier free formula then $pnf(\psi) = \psi$ and therefore $pnf(\psi) \Leftrightarrow \psi$.

Again the proof for the inductive case is given by case analysis.

- Assume ψ is $Qi : A.\psi'$, then by definition of pnf, $pnf(\psi) = Qi : A.pnf(\psi')$. By inductive hypothesis $pnf(\psi') \Leftrightarrow \psi'$ hence $pnf(\psi) \equiv Qi : A.\psi'$ and therefore $pnf(\psi) \Leftrightarrow \psi$.
- If $\psi = \psi_1 \wedge \psi_2$ then, assuming $pnf(\psi_1) = Qi : A.\psi_1'$, by definition of pnf

  $$pnf(\psi) = Qi' : A.pnf(\psi_1'[i'/i] \wedge \psi_2)$$

  For i' fresh (namely, i' does not occur in ψ_2). By inductive hypothesis $pnf(\psi_1) \Leftrightarrow \psi_1$ hence

  $$\psi \Leftrightarrow pnf(\psi_1) \wedge \psi_2 = (Qi : A.\psi_1') \wedge \psi_2$$

  It is trivial to prove that for any $\mathscr{PL}$ formula $(Qi : A.\psi) \wedge \phi \Leftrightarrow Qi : A.(\psi \wedge \phi)$ and therefore $pnf(\psi) \Leftrightarrow \psi$.
- The proof for $\psi = \psi_1 \vee \psi_2$ is similar.
- If $\psi = \neg\psi'$ then, assuming $pnf(\psi') = Qi : A.\psi''$, by definition of pnf, $pnf(\psi) = \overline{Qi} : A.pnf(\neg\psi'')$. By inductive hypothesis $pnf(\psi') \Leftrightarrow \psi'$ and therefore $\psi \Leftrightarrow \neg pnf(\psi')$, hence $\psi \Leftrightarrow \neg(Qi : A.\psi'') \Leftrightarrow \overline{Qi} : A.\neg\psi''$ and therefore $pnf(\psi) \Leftrightarrow \psi$.

□

The heuristic function $\mathscr{H}_s$ is given in Definition 2 and depends on the function $\mathscr{H}_t$ given in Definition 3 below.

Definition 2 (Weighting states)**.** *Given a state s and a formula ϕ, the* state weighting function *is given by*

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

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

The function $\mathscr{H}_s$ takes a state, say $s = \langle \mathscr{C}, \chi, \kappa \rangle$, and a formula ϕ as input and returns the maximum among the weights computed by $\mathscr{H}_t$ on the join transitions departing from s for ϕ. The weight $-\infty$ is returned if

- ϕ is a universal quantification on a principal instance A ($\forall i : A.\ \phi'$),
- s does not have outgoing join transitions ($s\Upsilon = \emptyset$), and

- there is no instance of A in the context ($s \cap [A] = \emptyset$).

The heuristic $\mathscr{H}_s$ has been designed considering that a formula universally quantified on instances of A is falsified in those states where there is at least one instance of A. Therefore a context that does not have an instance of the quantified principal, has no chance of falsifying the formula. In fact, the condition $s\Upsilon = \emptyset$ ensures that no principal instance can later join the context. As a result, there is no possibility of falsifying the property in all paths emerging from this state which can therefore be pruned. This justifies the second case of $\mathscr{H}_s$ where the value $-\infty$ is assigned to such states.

The heuristic that assigns weights to transitions is given in Definition 3.

Definition 3 (Weighting transitions)**.** *Given a state s and a transition t from s to* $s' = \langle \mathscr{C}', \chi', \kappa' \rangle$ *in* $s\Upsilon$*, the* weighting transitions *function* $\mathscr{H}_t$ *is*

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

The function $\mathscr{H}_t$ takes as input a transition t and invokes $\mathscr{H}_s$ to compute the weight of t depending on the structure of the formula ϕ. As specified in Definition 3, the value of the weight of the arrival state is incremented if either of the two following mutually exclusive conditions hold:

- ϕ universally quantifies on a principal instance A ($\forall i : A.\ \phi'$) for which some instances have already joined the context ($\kappa' \cap [A] \neq \emptyset$);
- ϕ existentially quantifies on a principal instance A ($\exists i : A.\ \phi'$) which is not present in the context ($\kappa' \cap [A] = \emptyset$).

Instead, the heuristic $\mathscr{H}_t$ does not increment the weight of the arrival state if either of the following mutually exclusive[3] conditions hold:

- ϕ existentially quantifies on instances of A ($\exists i : A.\ \phi'$) and present in the context ($\kappa' \cap [A] \neq \emptyset$);
- ϕ universally quantifies on instances of A ($\forall i : A.\ \phi'$) and the context does not contain such instances ($\kappa' \cap [A] = \emptyset$).

Again the intuition behind $\mathscr{H}_t$ is based on quantifiers. The formula ϕ that universally (resp. existentially) quantifies on instances of A can be falsified only if such instances will (resp. not) be added to the context. Therefore all transitions that (resp. do not) add an instance of A get a higher value. It is important to mention that in the first and third cases of Definition 3, the recursive call to $\mathscr{H}_s$ takes in input ϕ', the subformula of ϕ in the scope of the quantifier. This is due to the fact that once an instance of the quantified principal has been added we are not interested in more instances and therefore consume the quantifier. The heuristic $\mathscr{H}_t$ returns 0 when ϕ is a quantifier free formula. In fact, due to the absence of quantifiers we cannot assess how promising is t to find an attack for ϕ. We are investigating if in this case a better heuristic is possible.

Finally, we remark that $\mathscr{H}_s$ and $\mathscr{H}_t$ terminate on a finite state space because the sub-graph consisting of the join transitions forms a tree by construction[4]. Therefore, the recursive invocations from $\mathscr{H}_t$ to $\mathscr{H}_s$ will eventually be resolved by the last two cases of $\mathscr{H}_s$ in Definition 2.

---

[3] Note that all the conditions of the definition of $\mathscr{H}_t$ are mutually exclusive.

[4] For page limits we do not prove it formally, but it can easily be checked by the informal description of join transitions given in this section.

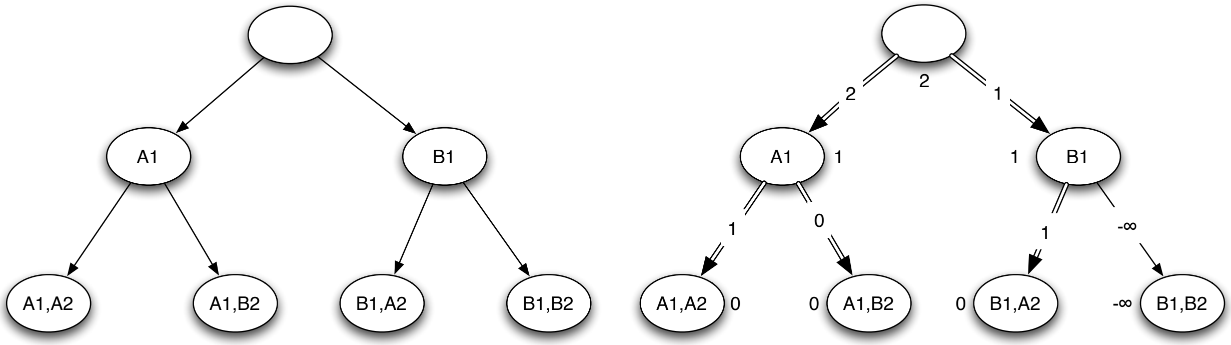
4 Evaluation of the Heuristic

In this section we describe with the help of examples how $\mathscr{H}_s$ and $\mathscr{H}_t$ can find attacks without exploring the complete state space. In the first example the heuristic is applied on the NS protocol and in the second example it is applied on the KSL protocol.

We also prove the correctness of the heuristic.

4.1 Applying the heuristic to the Needham-Schroeder protocol

Let us consider the property ψ_{NS} given in § 2.1 as $\forall i : A.\ \exists j : B\ (x_j = na_i \ \wedge\ z_i = nb_j)$. Figure 3(a) illustrates a portion of the state space of the NS protocol after the first two join transitions when ψ_{NS} is considered. Notice that ψ_{NS} can be falsified in a path where there is a context containing at least one instance of A and no instances of B.



(a) Join transitions of the NS protocol (b) Weighted states in the NS protocol

Figure 3: Join transitions and weighted states in NS protocol

The heuristic will assign weights to states and transitions as in Figure 3(b). The highlighted paths (those with 'fat' arrows) are the one to be explored; the context $\{A_1, A_2\}$ contains the attack reported below:

$$\begin{array}{lll}
1.\ A_1 \to I: & \{na_1, A_1\}_{I^+} & \\
2.\ A_2 \to I: & \{na_2, A_2\}_{I^+} & \\
3.\ I \to A_1: & \{na_1, na_2\}_{A_1^+} & \kappa \rhd na_1, na_2 \\
4.\ I \to A_2: & \{na_2, na_1\}_{A_2^+} & \kappa \rhd na_1, na_2 \\
5.\ A_1 \to I: & \{na_2\}_{I^+} & \\
6.\ A_2 \to I: & \{na_1\}_{I^+} &
\end{array}$$

The intruder acts as responder for both A_1 and A_2. As a result of step 1 and 2, κ contains na_1 and na_2; enabling the intruder to send messages to A_1 and A_2 at step 3 and 4 respectively. This results into assignments like $z_{A_1} = na_2$ and $z_{A_2} = na_1$, which is the falsification of stated property which requires a nonce generated by an instance of B to be assigned to the variables.

The other two highlighted paths contain a similar attack, we report the one with context $\{A_1,B_2\}$.

$$\begin{array}{lll} A_1 \rightarrow I: & \{na_1,A_1\}_{I^+} & \\ I \rightarrow A_1: & \{na_1,I\}_{A_1{}^+}, & \kappa \triangleright na_1,I \\ A_1 \rightarrow I: & \{I\}_{I^+} & \\ I \rightarrow B_2: & \{na_1,I\}_{B_2{}^+}, & \kappa \triangleright na_1,I \\ B_2 \rightarrow I: & \{na_1,nb_2\}_{I^+} & \\ I \rightarrow B_2: & \{nb_2\}_{B_2{}^+}, & \kappa \triangleright nb_2 \end{array}$$

Again at step 2 and 4, A_1 and B_2 are receiving the identity of intruder instead of nonce by B, resulting into an attack.

It is evident from the Figure 3(b) that heuristic assigns appropriate weights to the paths that contain an attack. It is worthy mentioning that the context $\{B_1,B_2\}$ has been labeled $-\infty$, therefore the search will never explore this state. This suggests that approximately 1/4th of the state space can be pruned by applying heuristic. This is a rough estimate taking into consideration the symmetry in the state space (the context $\{A_1,A_2\}$ is similar to $\{B_1,B_2\}$ and $\{A_1,B_2\}$ is similar to $\{B_1,A_2\}$).

4.2 Applying the heuristic to the KSL protocol

We consider the analysis of (the second phase of) KSL [20], done in [18]. The protocol provides repeated authentication and has two phases; in the first phase (i) a trusted server S generates a session key kab to be shared between A and B, and (ii) B generates the *ticket* $\{Tb,A,kab\}_{kbb}$ for A (where Tb is a timestamp and kbb is known only to B).

In the second phase, A uses the ticket (until it is valid) to repeatedly authenticate herself to B without the help of S. The second phase can be specified as follows:

$$\begin{array}{lll} 1. & A \rightarrow B: & na, \{Tb,A,kab\}_{kbb} \\ 2. & B \rightarrow A: & nb, \{na\}_{kab} \\ 3. & A \rightarrow B: & \{nb\}_{kab} \end{array}$$

A sends a fresh nonce na and the ticket to B that accepts the nonce challenge and sends nb together with the cryptogram $\{na\}_{kab}$ to A. In the last message, A confirms to B that she got kab.

In *cIP*, A and B can be represented as follows:

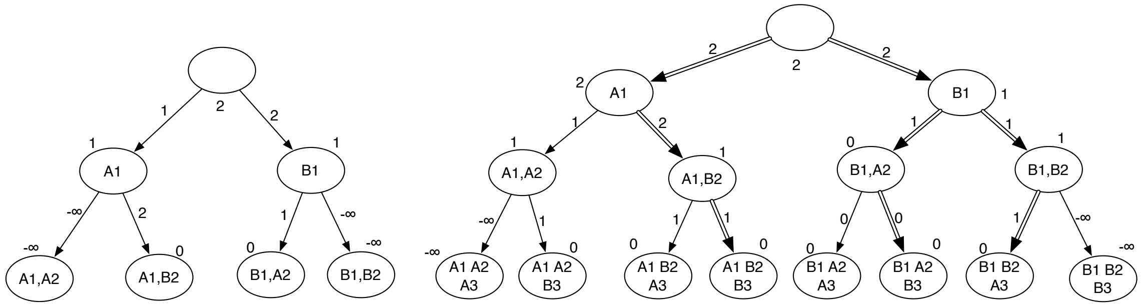
$$\begin{array}{llll} A:(b,sk,tk)[& out(na,\{b,A,sk\}_{tk}). & \qquad B:(a,sk,tk)[& in(?x,\{B,a,sk\}_{tk}). \\ & in(?y,\{na\}_{sk}). & & out(nb,\{x\}_{sk}). \\ & out(\{y\}_{sk})] & & in(\{nb\}_{sk})] \end{array}$$

(where for simplicity the timestamp generated by B is substituted by his identity). Authentication is based on the mutually exchanged nonces, and formalized as follows:

$$\psi_{KSL} = \forall l:B.\ \forall j:A.(b_j = B_l \wedge a_l = A_j \rightarrow x_l = na_j \wedge y_j = nb_l)$$

which reads any pair of properly connected "partners" B_l and A_j ($b_j = B_l \wedge a_l = A_j$) eventually exchange the nonces na_j and nb_l.

Figures 4(a) and 4(b) depict the weighted join transitions for 2 and 3 principal instances respectively. The verification with 2 principal instances reports no attack and the conclusion can be derived by just exploring half of the state space (the context $\{A_1,A_2\}$ and $\{B_1,B_2\}$ are labeled $-\infty$; see Figure 4(a)).



(a) 2 principals (b) 3 principals

Figure 4: Join transitions of KSL

In case of 3 principal instances the attacks are found in highlighted paths (those with 'fat' arrow in Figure 4(b)). The heuristic assigns appropriate weights to such paths and 2 states are labeled $-\infty$, suggesting a rough cut down of 1/4th of the state space.

The examples show that heuristic is able to guide the searching algorithm towards promising paths containing attacks. Moreover a considerable part of the state space is pruned, reducing the number of states to be explored by searching algorithm.

4.3 Properties of $\mathscr{H}_s$ and $\mathscr{H}_t$

First we would like to briefly comment on the admissibility of our proposed heuristic. Admissibility of heuristics is important in certain problems where it is possible to reach many goal states along different paths each path having a different cost. Hence, it may be not only important to find a goal state, but also find the goal state on the path with the best (or an acceptable) cost (as discussed in § 2.2 for the n-puzzle). In such cases, it is important for heuristic function to return an estimation of the cost to reach a goal from the state.

We contend that for security protocols the situation is different. In fact, the goal state in this case is an "attack", namely a state that violates the security property. Typically, it is very hard to compare the importance of different attacks as the violation of a property may be due to many causes as for the NS example in § 4.1). Therefore, optimality of the attack is of less concern when validating protocols; what matters in the first instance is to find an attack, if any. However, we envisage the problem of finding optimal solutions as important but we do not consider it in this paper.

It is also important to remark that the weights assigned by $\mathscr{H}_s$ and $\mathscr{H}_t$ to states or transitions do not correspond to evaluate the proximity to a target state. Rather they estimate the likeliness for the state to lead to an attack. This leads to a different scenario where the heuristic function does not have to return the cost to reach at goal node. Rather our heuristic returns a value that corresponds to the chance that nodes and transitions are on a path leading to an attack. We therefore contend that admissibility is not an issue in our case.

The following theorem proves the correctness of our heuristic; namely, it shows that pruned parts of the state space do not contain any attack.

Theorem 4.1. *If* $\mathscr{H}_s(s,\phi) = -\infty$ *then for any state s' =* $\langle \mathscr{C}', \chi', \kappa' \rangle$ *reachable from s=* $\langle \mathscr{C}, \chi, \kappa \rangle$, $\kappa' \models_{\chi'} \phi$

Proof. (Sketch) Suppose that there is a s' reachable from s such that $\kappa' \not\models_{\chi'} \phi$. Then $\kappa' \models_{\chi'} \neg\phi$ and by Definition of $\models$ (the relation $\models$ is reported in Appendix A), there is $A_n \in \kappa'$(because $\phi \equiv \forall i : A.\ \phi'$).

However by hypothesis $s \cap [A] = \emptyset$ and $s\Upsilon = \emptyset$ hence $s' \cap [A] = \emptyset$ and therefore s' does not satisfy $\neg\phi$. $\square$

5 Concluding Remarks

We have designed a heuristic that can be applied to improve the MC of cryptographic protocols. The proposed heuristic can drive the searching algorithm towards states containing attacks with respect to a security formula. Our heuristic may possibly prune parts of the state space that do not contain attack. We have shown that the heuristic is correct, namely we showed that pruned parts of the state space do not contain attacks.

The formal context to define the heuristic is the one proposed in [18] which features the cIP calculus and an ad-hoc logical formalism, called $\mathscr{PL}$, to respectively express protocols and security properties. An original aspect of $\mathscr{PL}$ is that it can quantify over principal instances. Formulas of $\mathscr{PL}$ are checked against the (symbolic) semantics of cIP by a tool called $\mathscr{A}$SPASyA (Automatic Security Protocol Analysis via a SYmbolic model checking Approach) [4].

5.1 Related work

At the best of our knowledge, the use of heuristics to analyse cryptographic protocols has not been much studied.

The concept of heuristics in cryptographic protocol verification has been utilized in [14]. The idea is to construct a pattern[5], $pt = (E, \rightarrow)$, where E is a set of events and $\rightarrow$ is a relation on the events. Afterwards, it is checked if a pt can give realizable patterns which are actual traces of the protocol and represent an attack. For each event execution, there are certain terms that need to be in intruder's knowledge or that are added to intruder's knowledge represented by $in(e)$ and $out(e)$ respectively. A process called pattern refinement is applied to get realizable patterns for those events whose $in(e)$ requirements are not satisfied. An open goal represents such requirements and is selected from set of potential open goals on the basis of the heuristics. In [14], 5 heuristics have been reported (e.g., an open goal is selected randomly, open goals that require a decryption key have higher priority). However, the whole state space must be searched if there are no attacks. We argue that our approach can give better results as it can prune certain parts of state space even when there are no attacks (as seen in § 4.2).

In [5], heuristics have been used to minimize the branching factor for infinite state MC of security protocols. Mainly, the heuristics in [5] reorder the nodes, for instance actions involving intruder are rated higher than actions initiated by honest participants. However these heuristics are very basic and as noted in [6], the tool does not scale to most of the protocols.

Though heuristic methods have not received much attention for MC security protocols, they have been studied for MC in general.

In [19], a heuristic has been defined in terms of model and formula to be verified, that can also prune the state space. Our heuristic seems to fall under the general conditions considered in [19] and we plan a deeper comparison.

In [2], the heuristic namely 'NEXT' compresses a sequence of transitions into a single meta transition. This eliminates transient states and therefore searching algorithm does less work to find the goal node. Similarly, in [16] heuristics for safety and liveness for communication protocols are given. At the best of our knowledge the heuristics in [16] (and references therein) allow to cut the state space only in few trivial cases.

---

[5] E.g. a pattern can be a representative of all traces violating secrecy

5.2 Future work

This paper proposes the first step of a research program that may develop in several directions.

First, other heuristics can be designed and studied; in fact, we are planning to define two heuristics. The former exploits the intruder's knowledge κ and the cIP protocol specification while the other heuristic exploits *joining formulae*, another feature (also supported by $\mathscr{A}$SPASyA) of cIP. Joining formulae are $\mathscr{PL}$ formulae which enable the analyst to express conditions on how principals should be joined (by predicating over open variables)[6].

The first heuristic will rank states considering the actions that principal instances are ready to execute with respect to the formula to falsify. For instance, if the goal is to prove that a variable should not be assigned a given value, the heuristic may rank higher those states that assigns such variable.

The second heuristic may instead be used to avoid the anticipation of all the joining formulae at the beginning (which may be computationally expensive) and use them to decide which instance to introduce in a given state.

It will also be rather interesting to study the combined effect of those heuristics (e.g., to consider their sum, or the max, etc.) or also use multiple heuristics during the search depending on the structure of the state. For instance, in one state one heuristic might be more suitable than others. Further we intend to implement these heuristics into existing tool in order to determine the efficiency achieved in terms of space and time.

We also plan to consider heuristics in other verification contexts. For instance, using *strand spaces* [17], the approaches in [27, 22] express properties in terms of connections between strands. A strand can be parameterised with variables and a trace is generated by finding a substitution for which an interaction graph exists. These approaches provide devices very similar to the join mechanism of cIP and possibly be suitable for heuristics similar to ours to help in finding the solution of the constraints. Also, in [9] a symbolic semantics based on unification has been adopted to verify security protocols with correspondence assertions and the use of trace analysis. We think that also in this case heuristics may drive the search for an attack in a more efficient way.

References

[1] ABADI, M., AND GORDON, A. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation 148*, 1 (January 1999), 1–70.

[2] ALUR, R., AND WANG, B.-Y. "Next" Heuristic for On-the-Fly Model Checking. In *CONCUR '99: Proceedings of the 10th International Conference on Concurrency Theory* (London, UK, 1999), Springer-Verlag, pp. 98–113.

[3] AMADIO, R., LUGIEZ, D., AND VANACKÈRE, V. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science 290*, 1 (2003), 695–740.

[4] BALDI, G., BRACCIALI, A., FERRARI, G., AND TUOSTO, E. A Coordination-based Methodology for Security Protocol Verification. In *International Workshop on Security Issues with Petri Nets and other Computational Models* (Bologna (Italy), June 26 2004, Feb. 2005), N. Busi, R. Gorrieri, and F. Martinelli, Eds., vol. 121 of *Electronic Notes in Theoretical Computer Science*, Elsevier, pp. 23–46.

[5] BASIN, D. A. Lazy infinite-state analysis of security protocols. In *Proceedings of the International Exhibition and Congress on Secure Networking - CQRE (Secure) '99* (London, UK, 1999), Springer-Verlag, pp. 30–42.

---

[6] For example, runs of the NS protocol with at least an initiator and a responder can be specified by the formula $(\exists i : A.true) \wedge (\exists j : B.true)$ which rules out states that contain only instances of A or of B.

[6] BASIN, D. A., MÖDERSHEIM, S., AND VIGANÒ, L. An on-the-fly model-checker for security protocol analysis. In *Proceedings of Esorics'03*, LNCS 2808. Springer-Verlag, Heidelberg, 2003, pp. 253–270.

[7] BODEI, C., DEGANO, P., NIELSON, F., AND NIELSON, H. R. Static analysis for secrecy and non-interference in networks of processes. *Lecture Notes in Computer Science 2127* (2001), 27–34.

[8] BOREALE, M. Symbolic trace analysis of cryptographic protocols. In *Colloquium on Automata, Languages and Programming* (July 2001), F. Orejas, P. Spirakis, and J. van Leeuwen, Eds., vol. 2076 of *Lecture Notes in Computer Science*, Springer-Verlag.

[9] BOREALE, M., AND BUSCEMI, M. A method for symbolic analysis of security protocols. *Theoretical Computer Science 338, Issues 1-3* (2005), 393–425.

[10] BORGSTRÖM, J., BRIAIS, S., AND NESTMANN, U. Symbolic Bisimulation in the Spi Calculus. In *International Conference in Concurrency Theory* (2004), P. Gardner and N. Yoshida, Eds., vol. 3170 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 161–176.

[11] CHEVALIER, Y., KÜSTERS, R., RUSINOWITCH, M., AND TURUANI, M. Deciding the security of protocols with diffie-hellman exponentiation and products in exponents. In *Foundations of Software Technology and Theoretical Computer Science* (2003), P. Pandya and J. Radhakrishnan, Eds., vol. 2914 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 124–135.

[12] CHEVALIER, Y., KÜSTERS, R., RUSINOWITCH, M., AND TURUANI, M. An NP decision procedure for protocol insecurity with XOR. In *Annual Symposium on Logic in Computer Science* (2003), IEEE Computer Society, pp. 261–270.

[13] CLARKE, E., JHA, S., AND MARRERO, W. Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols. In *IFIP Working Conference on Programming Concepts and Methods (PROCOMET)* (1998).

[14] CREMERS, C. J. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security* (New York, NY, USA, 2008), ACM, pp. 119–128.

[15] DURGIN, N., LINCOLN, P., MITCHELL, J., AND SCEDROV, A. Undecidability of bounded security protocols, Aug. 10 1999.

[16] EDELKAMP, S., LEUE, S., AND LLUCH-LAFUENTE, A. Directed explicit-state model checking in the validation of communication protocols. *Int. J. Softw. Tools Technol. Transf. 5*, 2 (2004), 247–267.

[17] FABREGA, J., HERZOG, J., AND GUTTMAN, J. Strand spaces: Proving security protocols correct. *Journal of Computer Security 7*, 2,3 (January 1999), 181–230.

[18] FERRARI, G., BRACCIALI, A., AND TUOSTO, E. A symbolic framework for multi-faceted security protocol analysis. *Internation Journal of Information Security 7*, 1 (2008), 55–84.

[19] GRADARA, S., SANTONE, A., AND VILLANI, M. L. Formal verification of concurrent systems via directed model checking. *Electron. Notes Theor. Comput. Sci. 185* (2007), 93–105.

[20] KEHNE, A., SCHÖNWÄLDER, J., AND LANGENDÖRFER, H. Multiple authentications with a nonce-based protocol using generalized timestamps. In *Proc. ICCC '92* (Genua, 1992).

[21] LOWE, G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems* (1996), vol. 1055, Springer-Verlag, pp. 147–166.

[22] MILLEN, J. K., AND SHMATIKOV, V. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security* (2001), pp. 166–175.

[23] MITCHELL, J., MITCHELL, M., AND STERN, U. Automated analysis of cryptographic protocols using murϕ. In *Computer Security Foundation Workshop* (1997), IEEE Computer Society, pp. 141–151.

[24] PAULSON, L. Proving properties of security protocols by induction. In *Computer Security Foundation Workshop* (1997), IEEE Computer Society.

[25] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[26] SHMATIKOV, V. Decidable analysis of cryptographic protocols with products and modular exponentiation.

In *European Symposium on Programming* (April 2004), D. Schmidt, Ed., vol. 2986 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 355–369.

[27] SONG, D., BEREZIN, S., AND PERRIG, A. Athena, a novel approach to efficient automatic security protocol analysis. *Computer Security 9(1,2)* (2001), 47–74.

[28] TUOSTO, E. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Dipartimento di Informatica, Università di Pisa, May 2003. TD-8/03.

A Model for $\mathscr{PL}$ formulae

We borrow from [18] the definition of models for $\mathscr{PL}$.

Definition 4 (Model for $\mathscr{PL}$ formulae)**.** *Let* χ *be a mapping from indexed variables to indexed messages,* κ *a knowledge and* ϕ *a closed formula of* $\mathscr{PL}$*. Then* $\langle \kappa, \chi \rangle$ is a model for ϕ *if* $\kappa \models_\chi \phi$ *can be proved by the following rules (where n stands for an instance index):*

$$\frac{xa_n\chi = m\chi}{\kappa \models_\chi xa_n = m}\,(=) \qquad \frac{\kappa \rhd m\chi}{\kappa \models_\chi \kappa \rhd m}\,(\rhd)$$

$$\frac{\textit{exists } n \textit{ s.t. } A_n \in \kappa \quad \kappa \models_\chi \phi[n/i]}{\kappa \models_\chi \exists i : A.\ \phi}\,(\exists)$$

$$\frac{\textit{forall } n \textit{ s.t. } A_n \in \kappa \quad \kappa \models_\chi \phi[n/i]}{\kappa \models_\chi \forall i : A.\ \phi}\,(\forall)$$

$$\frac{\kappa \models_\chi \phi \quad \kappa \models_\chi \psi}{\kappa \models_\chi \phi \wedge \psi}\,(\wedge) \qquad \frac{\kappa \models_\chi \phi}{\kappa \models_\chi \phi \vee \psi}\,(\vee 1)$$

$$\frac{\kappa \models_\chi \psi}{\kappa \models_\chi \phi \vee \psi}\,(\vee 2) \qquad \frac{\kappa \not\models_\chi \phi}{\kappa \models_\chi \neg\phi}\,(\neg)$$