Type-directed Trace Analysis of Security Protocols in Process Calculus

Guoqiang Li Bochao Liu Li Xin Mizuhito Ogawa

Japan Advanced Institute of Science and Technology
Asahidai, Nomi, Ishikawa, 923-1292 Japan
{guoqiang, bochao, li-xin, mizuhito}@jaist.ac.jp

Trace analysis, one of the formal methods to verify security protocols based on the process calculus, represents the environment as traces and a deductive system. However, the state space of the approach is infinite because the environment knowledge is too large to predict. In this paper, a type-directed trace analysis method is proposed to cut down the state space to be finite by type matching.

1 Introduction

Trace analysis is one of the formal methods to verify security protocols based on the process calculus, in which behaviors of the principals in the protocol are regarded as processes, and the environment is represented by the sequences of actions(trace) that the protocol may execute and a deductive system on the messages the sequences includes. Then it explicitly generates the model in order to check whether any insecure state is reachable.[1, 2, 3, 4].

The main problem of trace analysis is that the execution of a protocol typically generates infinite traces, because the environment is too large to predict. A principal of the protocol waiting for an input at a given moment may expect any of the infinite messages the environment can produce. We can cut down the state space to a convenient finite size by imposing upper-bounds upon the critical parameters. In [2], a symbolic method approach is given to cut down state space of the model.

In this paper, we explore an alternative approach to reduce the state space by type matching. The general idea is that when a principal waits for an input, the type of message it expects can be decided by actions the principal performs after the input. On the other hand, when a principal sends messages to the environment, the types of the messages can be predicted. And still we also can infer the type of the message which the environment deduces by deduction rules. So we only need to generate messages from the environment whose type matches the input type. In this approach, we can show that the model state space is cut down to be finite.

2 Model

The syntax of the model we use is taken essentially from the Spi calculus[5]. However, unlike Spi calculus regarding the channel as the name, we use the label to replace the channel which differ in the name.

2.1 Syntax

We firstly assume three countable disjoint sets: $\mathcal{L}$ of labels, $\mathcal{N}$ of names and $\mathcal{V}$ of variables. We let $a, b, c..$ range over labels, $m, n, k...$ range over names and $x, y, z...$ range over variables. The message M in the set $\mathcal{M}$ is defined as follows:

$$M, N, L ::= n \mid x \mid (M, N) \mid \{M\}_L$$

Let $\mathcal{P}$ be countable set of processes which was ranged by $P, Q, R....$ The grammar of processes is defined as follows:

$P, Q, R ::=$

$\mathbf{0}$	Nil
$\overline{a}M.P$	output
$a(x).P$	input
$[M = N]P$	match
$(\nu n)P$	new
$let\ (x, y) = M\ in\ P$	pair splitting
$case\ M\ of\ \{x\}_L\ in\ P$	decryption
$P \| Q$	composition

Intuitively,the Nil process $\mathbf{0}$ does nothing. $\overline{a}M.P$ outputs the message M to the environment and then evolves as P. $a(x).P$ awaits an input and evolves as P under the substitution. If $M = N$, $[M = N]P$ acts as P. Here $(\nu n).P$ means P generates the name a, which can not be seen until P send it. And if the M is a pair (N, L), $let\ (x, y) = M\ in\ P$ evolves as $P\{N/x, L/y\}$. For $case\ M\ of\ \{x\}_L\ in\ P$ performs as $P\{N/x\}$ when M is a encrypted message $\{N\}_L$. A difference from spi calculus is that the input and output labels must not be regarded as channels, but rather "tags" attached to the process actions, since we assume that there is just an open public environment which every principals send their messages to and receive their messages from.

In the processes, x in $a(x).P$ is bound, x, y in $let\ (x, y) = M\ in\ P$, and x in $case\ M\ of\ \{x\}_L\ in\ P$ are also bound. We use $f_v(P)$ and $b_v(P)$ to represent free variable and bound variable respectively. A process is closed if $f_v(P) = \emptyset$. Furthermore, we use $N(P)$ to represent the name in the process P.

In our model, we use a pair $\langle s, P\rangle$ named *configuration* to model a state of the system, where s is a *trace* which records the history of messages the P sends and recieves and P is a closed process. The trace is defined follows: Let an *action* α is a term in the form of $\overline{a}M$(output action) or $a(M)$ (input action), Act is an action set and a trace s is an closed action string $s \in Act^*$. We also use $N(s)$ to represent the names occurs in s.

We characterize the messages that the environment can produce at a given moment, starting from the current knowledge, a finite set $S \subseteq \mathcal{M}$ via a deductive system We also presuppose a countable sets $\mathcal{E}$, for those environmental names such as public keys. Let $\rhd$ be the least binary relation generated by the deductive systems as follows:

$$\frac{}{S \rhd n}\ n \in \mathcal{E} \quad ENV$$

$$\frac{}{S \rhd M}\ M \in S \quad AX$$

$$\frac{S \rhd M \quad S \rhd N}{S \rhd (M, N)} \quad PAIR$$

$$\frac{S \rhd (M, N)}{S \rhd M} \quad PROJ1$$

$$\frac{S \rhd (M, N)}{S \rhd N} \quad PROJ2$$

$$\frac{S \rhd \{M\}_k \quad S \rhd k}{S \rhd M} \quad DEC$$

$$\frac{S \rhd M \quad S \rhd k}{S \rhd \{M\}_k} \quad ENC$$

Note that the set of messages that S can produce is infinite whatever S is, due to rules *ENV*, *PAIR* and *ENC*. Given a trace s, we write $s \rhd M$ if the set of messages in s, represented by $msg(s)$, holds $msg(s) \rhd M$.

2.2 Type

The sets of types (ranged over by τ) are given by the following syntax:

τ ::=

α	type variables
b	base type
$unit$	nil type
$\tau * \tau$	pair type
$\ominus\tau$	encryption type
$\tau + \tau$	disjoint type
$\tau \to \tau$	bound type

where α ranges over a given countably infinite set of type variables. b is a base type for those atomic plain messages such as nonce, key, etc. $unit$ is a nil type for Nil process. Pair type is given to tuples in the protocol. Encryption type, $\ominus\tau$, is given to an encryption message, and τ is the type for the message it encrypts. Disjoint type means an expression is composed of expressions of type τ_1, τ_2 respectively, given to composition process, $P \| Q$. Bound type means a variable is bound to the following process. Next section, we will give the precise typing rule to the model.

What we introduce is to translate untyped expressions into explicit typed expressions by our type system, then each time when input is needed, we perform type matching. Firstly, we will give typing rules for expression: an expression $e \in \mathcal{P} \cup \mathcal{M}$. Let

Γ be a type environment which maps from $\mathcal{V} \cup \mathcal{N}$ to types $\mathcal{T}$.

$$\frac{}{\Gamma \vdash x : \tau}\ (x,\tau) \in \Gamma \quad \text{Msg\_Variable}$$

$$\frac{}{\Gamma \vdash n : b}\ b = TypeOf(n) \quad \text{Msg\_Name}$$

$$\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash (M,N) : \tau_1 * \tau_2}\text{Msg\_Pair}$$

$$\frac{\Gamma \vdash M : \tau \quad \Gamma \vdash k : b}{\Gamma \vdash \{M\}_L : \ominus\tau}\text{Msg\_Enc}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : unit}\text{Nil}$$

$$\frac{\Gamma + \{x : \tau_1\} \vdash P : \tau_2}{\Gamma \vdash a(x).P : \tau_1 \rightarrow \tau_2}\text{Input}$$

$$\frac{\Gamma \vdash n : b \quad \Gamma + \{n : b\} \vdash P : \tau}{\Gamma \vdash (\nu n)P : \tau}\text{New}$$

$$\frac{\Gamma \vdash P : \tau_1 \quad \Gamma \vdash M : \tau_2}{\Gamma \vdash \overline{a}M : \tau_2.P : \tau_1}\text{Output}$$

$$\frac{\Gamma + \{x : \tau_1, y : \tau_2\} \vdash P : \tau_3 \quad \Gamma \vdash M : \tau_1 * \tau_2}{\Gamma \vdash let\ (x,y) = M : \tau_1 * \tau_2\ in\ P : \tau_3}\text{Pair}$$

$$\frac{\Gamma + \{x : \tau_1\} \vdash P : \tau_2 \quad \Gamma \vdash M : \ominus\tau_1}{\Gamma \vdash case\ M : \ominus\tau_1\ of\ \{x\}_L\ in\ P : \tau_2}\text{Dec}$$

$$\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_1 \quad \Gamma \vdash P : \tau_2}{\Gamma \vdash [M : \tau_1 \ = \ N : \tau_1]P : \tau_2}\text{Match}$$

$$\frac{\Gamma \vdash P : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash P \| Q : \tau_1 + \tau_2}\text{Composition}$$

Here for constants, we assume a function $TypeOf : \mathcal{N} \rightarrow \mathcal{T}$, that assigns type to name. In this type system, we only have one type for name, b.

2.3 Semantics

The transition relation of configurations is defined by the rules as follows, with type matching. Note that in rule COM, no reaction relation is provided between the composition processes, all messages go through the environment.

$$\begin{array}{rl}
(INPUT) & \langle s, a(x).P : \tau_1 \rightarrow \tau_2 \rangle \longrightarrow \\
& \quad \langle s.a(M : \tau_1), P\{M/x\} \rangle \\
& \quad s \rhd M, \Gamma \vdash M : \tau_1 \\
(OUTPUT) & \langle s, \overline{a}M : \tau.P \rangle \longrightarrow \langle s.\overline{a}M : \tau, P \rangle \\
(DEC) & \langle s, case\ \{M\}_L\ of\ \{x\}_L\ in\ P \rangle \longrightarrow \\
& \quad \langle s, P\{M/x\} \rangle \\
(PAIR) & \langle s, let\ (x,y) = (M,N)\ in\ P \rangle \longrightarrow \\
& \quad \langle s, P\{M/x, N/y\} \rangle \\
(NEW) & \langle s, (\nu n)P \rangle \longrightarrow \langle s, P\{m/n\} \rangle \\
& \quad m \notin N(s) \cup N(P) \\
(MATCH) & \langle s, [M = M]P \rangle \longrightarrow \langle s, P \rangle \\
(COM) & \dfrac{\langle s, P \rangle \longrightarrow \langle s', P' \rangle}{\langle s, P \| Q \rangle \longrightarrow \langle s', P' \| Q \rangle} \\
& \text{plus symmetric version of } (COM)
\end{array}$$

Here, We can see that in $INPUT$, only messages that match the type of variable can be instantiated through the trace. And when process sends the message to the trace, in $OUTPUT$, the type of the message has been inferred already.

3 Analysis

Our model can verify many properties which are similar to [2]. Here we take the authentication as a case study and an protocol introduced in [5] as an example to explain the analysis of model.

3.1 Authentication

Authentication is a security property that has been widely studied in the formal research of security protocols. Here, we exploit an already existing and widely used way of specifying authentication properties, called correspondence assertion, which was first introduced by Woo and Lam in [6]. The main idea is to annotate the code of the protocol with labelled events that show the progress of a principal during the communication. These labelled events are divided into two main categories, begin-event and end-event. The protocol guarantees authentication if in every run, and in the presence of every possible intruders, every assertion of an end-event corresponds to a distinct, earlier as-

sertion of a begin-event with the same label. To use the correspondence assertion in our model, we firstly define a satisfaction relation.

Given a configuration $\langle s, P\rangle$ and a trace s', we say that $\langle s, P\rangle$ *generates* the s', if $\langle s, P\rangle \longrightarrow^* \langle s', P'\rangle$ for some P'. Given a string of actions s, and two actions α and β, we say that α *occurs prior to* β in s if whenever $s = s'.\beta.s''$ then $\alpha \in s'$. Let ρ range over ground substitutions, that is, finite maps from a set $dom(\rho) \subseteq \mathcal{V}$ to closed messages; $t\rho$ denotes the result of replacing each $x \in f_v(t) \cap dom(\rho)$ by $\rho(x)$. So we have the following definition:

Definition 1 *Let α and β be actions, with $f_v(\alpha) \subseteq f_v(\beta)$, and let s be a trace. We write $s \models \alpha \hookleftarrow \beta$, if for each ground substitution ρ it holds that $\alpha\rho$ occurs prior $\beta\rho$ in s. We say that a configuration satisfies $\alpha \hookleftarrow \beta$, write as $\langle s, P\rangle \models \alpha \hookleftarrow \beta$, if all traces generated by the configuration satisfy $\alpha \hookleftarrow \beta$.*

With the definition, we can define and verify authentication property of security protocols. Let's take an example to explain.

3.2 States

We consider the protocol introduced in [5] as an example. Firstly, we give the informal description of the protocol. There are three interaction flows in the protocol, which we explain flow-by-flow as follows:

$$\begin{aligned} A \longrightarrow S: &\quad A, \{B, K_{AB}\}_{K_{AS}} \\ S \longrightarrow B: &\quad \{A, K_{AB}\}_{K_{SB}} \\ A \longrightarrow B: &\quad A, \{M\}_{K_{AB}} \end{aligned}$$

The meaning of the protocol is quite simple: principal A wants to send a message M to principal B. It first passes a shared key to B through server S, then uses the key to encrypt M and sends it to B.

We define sender A and receiver B as follows.

$$\begin{aligned} A \triangleq &(\nu K_{AB})(\nu M)\overline{a1}(A, \{B, K_{AB}\}_{K_{AS}}) \\ &.\overline{a2}(A, \{M\}_{K_{AB}}) \qquad (1) \\ B \triangleq &b1(x).case\ x\ of \{x'\}_{K_{SB}}\ in \\ &let\ (y, z) = x'\ in\ b2(w).let\ (w', w'') = w \\ &in\ [y = w']\ case\ w''\ of\ \{u\}_z\ in\ F(u) \qquad (2) \\ S \triangleq &c1(x).let\ (y, z) = x\ in\ case\ z\ of\ \{u\}_{K_{AS}} \\ &in\ let\ (u', u'') = u\ in\ \overline{c2}\{y, u''\}_{K_{SB}} \qquad (3) \\ SYS \triangleq &(\nu K_{AS})(\nu K_{SB})A \| S \| B \qquad (4) \end{aligned}$$

Here $F(u)$ means the action B will perform after the communication. And we assume $F(u) = \overline{accept}\, w$($w$ is the variable bound in B) for defining authentication of the protocol as follows:

Definition 2 *Given the formal definition of the protocol, we say that sender is correctly authenticated to receiver, if $\langle \epsilon, SYS\rangle \models \overline{a2}\, t \hookleftarrow \overline{accept}\, t$.*

To begin with the analysis of this protocol, let's assume that there is a intruder named I want to destroy the protocol. The name of the intruder is known before the protocol running. So we let $I \in \mathcal{E}$. Unfortunately, we can find a trace that does not satisfied the definition:

$$\begin{aligned} &\overline{a1}(A, \{B, K_{AB}\}_{K_{AS}}).c1(I, \{B, K_{AB}\}_{K_{AS}}). \\ &\overline{c2}\,\{I, K_{AB}\}_{K_{SB}}.b1(\{I, K_{AB}\}_{K_{SB}}). \\ &\overline{a2}(A, \{M\}_{K_{AB}}).b1(I, \{M\}_{K_{AB}}). \\ &\overline{accept}(I, \{M\}_{K_{AB}}) \end{aligned}$$

3.3 Finite Search

Trace analysis in the model without the type will generate infinite traces. The reason lies in the modelling of the environment(In this paper, it is the traces that model the environment), whose behavior is largely unpredictable. We can see that given a trace s, it can generate infinite messages. However, with the type system we give in this paper, the analysis becomes finite even without the upper-bounds. To illuminate how it works, let's begin with the $(INPUT)$ transition rule. If there is no type variable appearing in the type of x, $\ominus(b * b) * b$ for example, which means, messages which x can be instantiated into is finite because the atomic messages of the environment is finite. However, if there exist

any type variables in the type of x, type variable can be unified to any type through type matching. So messages which x can be instantiated is infinite. How to reduce such kind of instantiation to finite states is the main task of our research.

If type of x includes any type variables, there lie two possibilities: Firstly, the principal of x does not care value of x. Thus, value of x will not affect security properties. For example, $P \triangleq p1(x).\overline{p2}\,x.0$. Under such condition, we do not need to instantiate every possible messages to x, rather only one closed message are enough for trace analysis. However, such kind of condition does not likely happen when we model a practical protocol.

Secondly, when we infer the type of principal B in the above example. the type of x in B is $\ominus(\alpha * b)$ if the type of w' is α. so after type matching, when α is unified to any other type, x is also instantiated to a encryption message whose plain message is a pair. Its first component is any message, and second component is a name. So it seems to be infinite since first component of plain message can be any messages. However, we can see that, our type system can guarantee once the first component is instantiated to a pair, α must be inferred to $\alpha_1 * \alpha_2$ where α_1 and α_2 are all type variable. Under such condition, we only need to substitute type variable to base type after type inference, and instantiate such kind of message variable to name, which is finite in the model.

4 Typing Inference

The type rules given in section 2.2 does not provide an easy method for finding, given Γ and expression e, that $\Gamma \vdash e : \tau$. We now present an algorithm for this purpose, then we will verify its correctness. In fact, our algorithm goes a step further. Given Γ and expression e, if a substitution θ and a type τ are found, which are most general in a sense to be made precise below, such that

$$\Gamma\theta \vdash e : \tau$$

Firstly, we provide the unification algorithm for our type system, which can be used not only in the type inference but also message type matching in the trace transition (That is, in *INPUT* of translation relation). Here we presume a function to check whether a free variable occurs in a type, $FTV(\tau, \alpha) = True$, means α does not occur in τ. Now we introduce the algorithm $Unify(\tau, \tau') = (\vartheta, \sigma)$, given two types τ and τ', which either returns a substitution ϑ and the result σ satisfying $\tau\vartheta = \tau'\vartheta = \sigma$, or fails.

$Unify(\tau, \tau') = (\vartheta, \sigma)$

1. $Unify(\alpha, \beta) = (\{\beta/\alpha\}, \beta)$.
2. $Unify(\alpha, \tau') = (\{\tau'/\alpha\}, \tau')$, if $FTV(\tau', \alpha)$.
3. $Unify(b, b) = (Id, b)$.
4. let $Unify(\tau_1, \tau_1') = (\vartheta_1, \sigma_1)$, $Unify(\tau_2\vartheta_1, \tau_2'\vartheta_1) = (\vartheta_2, \sigma_2)$ in $Unify(\tau_1 * \tau_2, \tau_1' * \tau_2') = (\vartheta_1\vartheta_2, \sigma_1 * \sigma_2)$.
5. let $Unify(\tau_1, \tau_1') = (\vartheta, \sigma_1)$, $\sigma = \ominus\sigma_1$ in $Unify(\ominus\tau_1, \ominus\tau_1') = (\vartheta, \sigma)$.
6. let $Unify(\tau_1, \tau_1') = (\vartheta_1, \sigma_1)$, $Unify(\tau_2\vartheta_1, \tau_2'\vartheta_1) = (\vartheta_2, \sigma_2)$ in $Unify(\tau_1 + \tau_2, \tau_1' + \tau_2') = (\vartheta_1\vartheta_2, \sigma_1 * \sigma_2)$.
7. let $Unify(\tau_1, \tau_1') = (\vartheta_1, \sigma_1)$, $Unify(\tau_2\vartheta_1, \tau_2'\vartheta_1) = (\vartheta_2, \sigma_2)$ in $Unify(\tau_1 \to \tau_2, \tau_1' \to \tau_2') = (\vartheta_1\vartheta_2, \sigma_1 * \sigma_2)$.
8. raise error

Now we give the type inference algorithm $Infer(\Gamma, e) = (\theta, \tau)$:

$Infer(\Gamma, e) = (\theta, \tau)$

1. $Infer(\Gamma, x) = (Id, \tau)$ that $(x, \tau) \in \Gamma$.
2. $Infer(\Gamma, n) = (Id, b)$ that $(n, b) \in \Gamma$.
3. Let $Infer(\Gamma, M) = (\theta_1, \tau_1)$, $Infer(\Gamma\theta_1, N) = (\theta_2, \tau_2)$ in $Infer(\Gamma, \langle M, N\rangle) = (\theta_1\theta_2, (\tau_1\theta_2) * \tau_2)$.
4. Let $Infer(\Gamma, M) = (\theta_1, \tau_1)$, $Infer(\Gamma\theta_1, L) = (\theta_2, \tau_2)$, $Unify(\tau_2, b) = (\vartheta, b)$ in $Infer(\Gamma, \{M\}_L) = (\theta_1\theta_2\vartheta, \ominus(\tau_1\theta_2))$.
5. $Infer(\Gamma, \mathbf{0}) = (Id, unit)$.

6. Let $Infer(\Gamma \cup \{(x,\alpha)\}, P) = (\theta,\tau)$, in $Infer(\Gamma, a(x).P) = (\theta, \alpha\theta \rightarrow \tau)$, in which $\forall \tau\ of\ (x',\tau) \in \Gamma,\ FTV(\tau,\alpha) = True$.

7. Let $Infer(\Gamma, M) = (\theta_1,\tau_1)$, $Infer(\Gamma\theta_1, P) = (\theta_2,\tau_2)$ in
$Infer(\Gamma, \overline{a}M.P) = (\theta_1\theta_2, \tau_2)$.

8. Let $Infer(\Gamma, n) = (\theta_1, b)$, $Infer(\Gamma\theta_1 \cup \{(n,b)\}, P) = (\theta_2,\tau)$ in
$Infer(\Gamma, (\nu n)P) = (\theta_1\theta_2, \tau)$.

9. Let $Infer(\Gamma, M) = (\theta_1,\tau_1), Unify(\alpha * \beta, \tau_1) = (\vartheta, \varrho)$,
$Infer(\Gamma\theta_1\vartheta \cup \{(x,\alpha),(y,\beta)\}\vartheta, P) = (\theta_2,\tau_2)$ in
$Infer(\Gamma, let\ (x,y) = M\ in\ P) = (\theta_1\vartheta\theta_2, \tau_2)$
in which $\forall \tau\ of\ (x',\tau) \in \Gamma,\ FTV(\tau,\alpha) = FTV(\tau,\beta) = True$.

10. Let $Infer(\Gamma, M) = (\theta_1,\tau_1)$, $Unify(\ominus\alpha, \tau_1) = (\vartheta,\varrho)$,
$Infer(\Gamma\theta_1\vartheta \cup \{(x,\alpha)\}\vartheta, P) = (\theta_2,\tau_2)$ in
$Infer(\Gamma, case\ M\ of\ \{x\}_L\ in\ P) = (\theta_1\vartheta\theta_2, \tau_2)$
in which $\forall \tau\ of\ (x',\tau) \in \Gamma,\ FTV(\tau,\alpha) = TRUE$.

11. Let $Infer(\Gamma, M) = (\theta_1,\tau_1)$, $Infer(\Gamma\theta_1, N) = (\theta_2,\tau_2)$, $Unify(\tau_1\theta_2, \tau_2) = (\vartheta,\varrho)$
$Infer(\Gamma\theta_1\theta_2\varrho, P) = (\theta_3,\tau_3)$ in $Infer(\Gamma, [M = N]P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$.

12. Let $Infer(\Gamma, P) = (\theta_1,\tau_1)$, $Infer(\Gamma\theta_1, Q) = (\theta_2,\tau_2)$ in
$Infer(\Gamma, P\|Q) = (\theta_1\theta_2, \tau_1\theta_2 + \tau_2)$.

We can see that the substitutions of messages types inferred by the algorithm are all identification functions, because there is not unification during the type inference.

We will verify our algorithm is sound, which means every types inferred by algorithm can be derived by the typing rules. Before this proposition, we introduce another one that means the typing judgement is stable under substitution: if we can prove $\Gamma \vdash e : \tau$, then the judgements obtained by substituting any types for any variables in Γ and τ are still provable.

Proposition 1 (Stable under substitution) *Let e be an expression, τ be a type, Γ be a typing environment and θ be a substitution. If $\Gamma \vdash e : \tau$, then $\Gamma\theta \vdash e : \tau\theta$.*

Proof 1 *by structural induction on expression e. We show the base cases and two inductive steps here; the remaining cases are quite similar.*

1. *Case $e = x$: Because $\Gamma \vdash x : \tau$, we have $(x,\tau) \in \Gamma$. Let $\Gamma_1 = \Gamma \backslash \{(x,\tau)\}$, so $\Gamma\theta = \Gamma_1\theta \cup \{(x,\tau)\}\theta = \Gamma_1\theta \cup \{(x,\tau\theta)\}$, and $\Gamma\theta \vdash x : \tau\theta$ by typing rule $Msg\_Variable$.*

2. *Case $e = n$: Because $\Gamma \vdash n : \tau$, we have $\tau = TypeOf(n)$, and $\tau\theta = \tau$ for every substitution θ. So by typing rule Msg_Name, $\Gamma\theta \vdash n : TypeOf(n)$.*

3. *Case $e = \langle M, N\rangle$: By typing rules, we can get $\Gamma \vdash M : \tau_1$ and $\Gamma \vdash M : \tau_2$ from $\Gamma \vdash \langle M,N\rangle : \tau_1 * \tau_2$. Given a substitution θ, we have $\Gamma\theta \vdash M : \tau_1\theta$ and $\Gamma\theta \vdash M : \tau_2\theta$ by induction hypothesis. So we have $\Gamma\theta \vdash \langle M,N\rangle : (\tau_1 * \tau_2)\theta$.*

4. *Case $e = a(x).P$: Applying the induction hypothesis, we can get $\Gamma\theta \cup \{x,\tau_1\}\theta = \Gamma\theta \cup \{x,\tau_1\theta\} \vdash P : \tau_2\theta$. So we have $\Gamma\theta \vdash a(x)P : (\tau_1 \rightarrow \tau_2)\theta$.*

Proposition 2 (Soundness of type inference) *Let e be an expression, Γ be a typing environment and θ be a substitution. If $Infer(\Gamma, e) = (\theta,\tau)$ is defined, then we can derive $\Gamma\theta \vdash e : \tau$.*

Proof 2 *the proof is an inductive argument over the structure of e, and makes heavy use of the fact that the typing judgement is stable under substitution (proposition 1). We show one base step(we omit the other one.) and three inductive steps; the remaining cases are simple and similar. We use the same notations as in the algorithm.*

1. *Case $e = x$: We have $Infer(\Gamma, x) = (Id,\tau)$, then because $\Gamma Id = \Gamma$ and $(x,\tau) \in \Gamma$, applying the typing rule $Msg\_Variable$, we have $\Gamma \vdash x : \tau$.*

2. *Case $e = \langle M,N\rangle$: Applying the induction hypothesis to the recursive call to $Infer$, we get two derivations of*

$$Infer(\Gamma, M) = (\theta_1,\tau_1) \quad \mapsto \quad \Gamma\theta_1 \vdash M : \tau_1 \quad (5)$$
$$Infer(\Gamma\theta_1, N) = (\theta_2,\tau_2) \quad \mapsto \quad \Gamma\theta_1\theta_2 \vdash N : \tau_2 \quad (6)$$

Using proposition 1 to (5), we have

$$\Gamma\theta_1\theta_2 \vdash M : \tau_1\theta_2 \quad (7)$$

Using typing rule Msg_Pair to (6) and (7), we have

$$\Gamma\theta_1\theta_2 \vdash \langle M, N\rangle : \tau_1\theta_2 * \tau_2$$

That is the expected result of $Infer(\Gamma, \langle M, N\rangle) = (\theta_1\theta_2, (\tau_1\theta_2) * \tau_2)$.

3. *Case* $e = a(x).P$*: Applying the induction hypothesis, we get a derivation of*

$$Infer(\Gamma \cup \{(x,\alpha)\}, P) = (\theta, \tau) \mapsto \Gamma\theta \cup \{(x,\alpha)\}\theta \vdash P : \tau \quad (8)$$

Using typing rule Input to (8), we have

$$\Gamma\theta \vdash a(x).P : \alpha\theta \to \tau$$

which is the result of $Infer(\Gamma, a(x).P) = (\theta, \alpha\theta \to \tau)$.

4. *Case* $e = case\ M\ of\ \{x\}_L\ in\ P$*: Applying the induction hypothesis, we get two derivations and an unification of*

$$Infer(\Gamma, M) = (\theta_1, \tau_1) \mapsto \Gamma\theta_1 \vdash M : \tau_1 \quad (9)$$

$$Unify(\ominus\alpha, \tau_1) = (\vartheta, \varrho) \mapsto \ominus(\alpha\vartheta) = \tau_1\vartheta = \varrho \quad (10)$$

$$Infer(\Gamma\theta_1\vartheta \cup \{(x,\alpha)\}\vartheta, P) = (\theta_2, \tau_2) \mapsto \Gamma\theta_1\vartheta\theta_2 \cup \{(x,\alpha)\}\vartheta\theta_2 \vdash \tau_2 \quad (11)$$

Using proposition 1 to (9), we have

$$\Gamma\theta_1\vartheta\theta_2 \vdash M : \tau_1\vartheta\theta_2 \quad (12)$$

Note that α *does not occur in* θ_1*, we have* $\{(x,\alpha)\}\vartheta\theta_2 = \{(x,\alpha)\}\theta_1\vartheta\theta_2$*. So after unification type of* x *is* $\alpha\vartheta\theta_2$*, and that of* $\{x\}_L$ *is* $\ominus(\alpha\vartheta\theta_2)$*. And because of (10), we have*

$$\ominus(\alpha\vartheta\theta_2) = \tau_1\vartheta\theta_2$$

So, using typing rule Dec, we have

$$\Gamma\theta_1\vartheta\theta_2 \vdash case\ M\ of\ \{x\}_L\ in\ P : \tau_2$$

which is the result of $Infer(\Gamma, case\ M\ of\ \{x\}_L\ in\ P) = (\theta_1\vartheta\theta_2, \tau_2)$.

5 Future Prospects

We have presented a type-directed trace analysis method for analyzing security protocols, which is well suited for an efficient mechanization. Compared with symbolic optimization in [2], which adopts unification mechanism in messages, performing unification during the "run-time"(that is, unify the messages during the states translation), our type system brings out a static check for trace analyzing, more like the conception of "compile-time" in programming language (that is, perform type inference before the states translation).

This method can be applied into public-key security protocols, with the redesign of type system, which will be our next work. Furthermore, since our type system is just an assistant work of trace analysis, and recently, type-based analysis of security protocols becomes such a research area that has gained increasing interest over the last few years[7, 8, 9]. So we also try to enhance our type system for verifying properties of security protocols directly.

参考文献

[1] Amadio, R.M., Prasad, S.: The game of the name in cryptographic tables. In: PPDP'99. Volume 1702 of LNCS., Spinger (1999) 15–26

[2] Boreale, M.: Symbolic trace analysis of cryptographic protocols. In: ICALP'01. Volume 2076 of LNCS., Spinger (2001) 667–681

[3] Lowe, G.: Breaking and fixing the needham-schroeder public-key using fdr. In: TACAS'96. Volume 1055 of LNCS., Spinger (1996) 147–166

[4] Boreale, M., Buscemi, M.: Experimenting with sta, a tool for automatic analysis of security protocols. In: Proceedings of the 2002 ACM symposium on Applied computing table of contents. (2002) 281–285

[5] Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols:the spi calculus. In: Proceedings of the Fourth ACM Conference on Computer and Communications Security. (1997)

[6] Woo, T.Y., Lam, S.S.: A semantic model for authentication protocols. In: RSP: IEEE Computer Society Symposium on Research in Security and Privacy. (1993) 178–194

[7] Abadi, M.: Secrecy by typing in security protocols. Journal of the ACM **46** (1999) 749–786

[8] Abadi, M., Blanchet, B.: Secrecy types for asymmetric communication. In: FoSSaCS'01. Volume 2030 of LNCS., Spinger (2001)

[9] Gordon, A., Jeffrey, A.: Authenticity by typing for security protocols. In: 14th IEEE Computer Security Foundations Workshop. (2001) 145–159