CARiSMA - New Sequence-Diagram Crypto FOL-Analyzer

Daniel Korner, *SHK, Tu Dortmund - Informatik - Lehrstuhl XIV*

Index Terms—CARiSMA, Sequence Diagram, Crypto Analyzer, journal, Sequence-Diagram Crypto FOL-Analyzer

1 Introduction

Security is today more important then even. Finding security issues early in development not only increases security but also decreases the costs to fix these issues. One often found security issue is insecure communication and a common solution is using cryptographic encryption on the messages to secure the communication.

But whether or not a communication is secure, isn't only decided by the used encryption, but also by whether or not an attacker needs to break the encryption.

No matter how good an encryption may be, if an attack is able to gain knowledge of the used keys or the encrypted data, the used encryption is as good as no encryption.

One way an attacker could gain this knowledge is by monitoring the communication. Afterward the attacker may not only read and alter the content of messages but also inject new messages, in case the attacker could gain enough information out of the communication, .

To prevent this, one not only needs to use cryptographic encryption on a communication but also needs to make sure an attacker can't gain knowledge of the keys or other data that may leads to enough knowledge to read/alter a messages content or even inject new ones.

2 Analyzing UML Sequence Diagrams

A common model for messages passing is the UML sequence diagram. This diagram will be use to determine whether or not an attack is able to gain enough knowledge.

To be able to impersonate as another Object , an man-in-the-middle attacker must be able to reproduce to some degree the communication. Which means the attacker has to answer the way the other Objects expect the Object , which the attacker impersonate, to answer.

The degree to which a man-in-the-middle attacker needs to reproduce the communication depends on whether or not the whole communication is symmetric or not. In case of a symmetric communication, the attacker may just pass the first few request to the original Object and only alters later messages.

Which means, the attacker doesn't need to be able to reproduce the whole communication but a fraction of it. As the attacker knows the order in which the messages are send and received, he knows exactly at which point he has to alter a message and sees directly if he succeeded or not.

On the other hand, if the communication is asymmetric, the attacker can't predict what might should happens next or could happens next. The attacker doesn't know the order or relation between the messages. Which means, he can't predict whether or not a message

should come and how the next message should look like.

2.1 Messages

2.1.1 Syntax

A message is composed of a name and arguments. A valid message follows this **syntax**

$$
\begin{aligned}
\text{Msg} &\rightarrow \text{Msgname Args}\\
\text{Msgname} &\rightarrow \text{string}\\
\text{Args} &\rightarrow ()\\
\text{Args} &\rightarrow (\text{Vals})\\
\text{Args} &\rightarrow \epsilon\\
\text{Vals} &\rightarrow \text{Val, Vals}\\
\text{Vals} &\rightarrow \text{Val}\\
\text{Val} &\rightarrow \text{string}\\
\text{Val} &\rightarrow \text{Fun}\\
\text{Fun} &\rightarrow \text{Unfun}\\
\text{Fun} &\rightarrow \text{Bifun}\\
\text{Unfun} &\rightarrow \text{Unfunname(Val)}\\
\text{Unfunname} &\rightarrow inv\\
\text{Bifun} &\rightarrow \text{Bifunname(Val, Val)}\\
\text{Bifunname} &\rightarrow symenc\\
\text{Bifunname} &\rightarrow symdec\\
\text{Bifunname} &\rightarrow conc\\
\text{Bifunname} &\rightarrow enc\\
\text{Bifunname} &\rightarrow dec\\
\text{Bifunname} &\rightarrow sign\\
\text{Bifunname} &\rightarrow ext
\end{aligned}
$$

2.1.2 Axioms

The following axioms apply to the unary and binary (message-)functions

Let a, b, c, k be values

$$dec(enc(a,k),inv(k)) = a \tag{1}$$
$$symdec(symenc(a,k),k) = a \tag{2}$$
$$ext(sign(a,inv(k)),k) = a \tag{3}$$
$$inv(inv(k)) = k \tag{4}$$
$$conc(a,conc(b,c)) = conc(conc(a,b),c) \tag{5}$$

2.2 Guards

A guard is modeled as a precondition of a message. Each message has a precondition, which is on default simply $[true]$. Let $Constrains$ be the set of all valid constrains, the function $guard : Messages \rightarrow Constrains$ maps each message to the guard it has in the UML diagram.

2.2.1 Syntax

Let c be a valid constrain, than c has to confirm to the following **syntax**.

$$
\begin{aligned}
\text{Guard} &\rightarrow \text{[Conditions]}\\
\text{Guard} &\rightarrow []\\
\text{Conditions} &\rightarrow \text{(Condition Junction Conditions)}\\
\text{Conditions} &\rightarrow \text{Condition}\\
\text{Junction} &\rightarrow \text{AND}\\
\text{Junction} &\rightarrow \text{OR}\\
\text{Junction} &\rightarrow \text{IMPLIES}\\
\text{Junction} &\rightarrow \text{XOR}\\
\text{Condition} &\rightarrow \text{Val = Val}\\
\text{Condition} &\rightarrow \text{Val != Val}\\
\text{Condition} &\rightarrow true\\
\text{Condition} &\rightarrow false\\
\text{Val} &\rightarrow \text{string}\\
\text{Val} &\rightarrow \text{Fun}\\
\text{Fun} &\rightarrow \text{Unfun}\\
\text{Fun} &\rightarrow \text{Bifun}\\
\text{Unfun} &\rightarrow \text{Unfunname(Val)}\\
\text{Unfunname} &\rightarrow inv\\
\text{Bifun} &\rightarrow \text{Bifunname(Val, Val)}\\
\text{Bifunname} &\rightarrow symenc\\
\text{Bifunname} &\rightarrow symdec\\
\text{Bifunname} &\rightarrow conc\\
\text{Bifunname} &\rightarrow enc\\
\text{Bifunname} &\rightarrow dec\\
\text{Bifunname} &\rightarrow sign\\
\text{Bifunname} &\rightarrow ext
\end{aligned}
$$

2.2.2 Axioms

Let v_0, v_1 be valid Val strings

$$v_0 \text{ = } v_1 \Leftrightarrow v_0 = v_1 \tag{6}$$
$$v_0 \text{ != } v_1 \Leftrightarrow v_0 \neq v_1 \tag{7}$$

Example:

$$inv(inv(k)) = k \overset{(4)}{\Leftrightarrow} k = k \overset{(6)}{\Leftrightarrow} k = k$$

2.3 Attackers Knowledge

To model what the attacker knows, the set $knows$ is used. $knows$ can also be used as predicate. $knows(value)$ means, the attacker knows $value$ respectively $value \in knows$. The following **axioms** applies to $knows$

Let a, b, k be values

$$knows(a) \wedge knows(b) \Rightarrow knows(conc(a,b)) \quad (8)$$
$$knows(a) \wedge knows(b) \Rightarrow knows(enc(a,b)) \quad (9)$$
$$knows(a) \wedge knows(b) \Rightarrow knows(symenc(a,b)) \quad (10)$$
$$knows(a) \wedge knows(b) \Rightarrow knows(dec(a,b)) \quad (11)$$
$$knows(a) \wedge knows(b) \Rightarrow knows(symdec(a,b)) \quad (12)$$
$$knows(a) \wedge knows(b) \Rightarrow knows(ext(a,b)) \quad (13)$$
$$knows(a) \wedge knows(b) \Rightarrow knows(sign(a,b)) \quad (14)$$
$$knows(conc(a,b)) \Rightarrow knows(a) \wedge knows(b) \quad (15)$$
$$knows(symenc(a,k)) \wedge knows(k) \Rightarrow knows(a) \quad (16)$$
$$knows(symdec(a,k)) \wedge knows(k) \Rightarrow knows(a) \quad (17)$$
$$knows(enc(a,k)) \wedge knows(inv(k)) \Rightarrow knows(a) \quad (18)$$
$$knows(sign(a,inv(k))) \wedge knows(inv(b)) \Rightarrow knows(a) \quad (19)$$

As a man-in-the-middle attacker is assumed, the attacker also reads each message m and following that, the attacker knows about m and all arguments in m i.e. $knows(m)$ and $\forall i \in \mathbb{N}_0 . i \leq n . knows(name(m)_i)$ with $args(m) = \{arg_0, ..., arg_n\}$ as set of arguments of m and $name(m)$ as the messages name.

As the man-in-the-middle attacker doesn't know about the internal names of the values, the attacker only know the arguments position and value. That is why a message like $msg(a,b,c)$ is modeled as $knows(msg_0), knows(msg_1)$ and $knows(msg_2)$.

2.3.1 Attackers initial knowledge

As an attacker may already has knowledge, the set $knows$ may initial not be empty.

To model the initial Knowledge, the following **syntax** is used:

$$\begin{aligned}
\text{InitialKnowledge} &\rightarrow \text{Vals} \\
\text{Vals} &\rightarrow \text{Val}, \text{Vals} \\
\text{Vals} &\rightarrow \text{Val} \\
\text{Val} &\rightarrow \text{string} \\
\text{Val} &\rightarrow \text{Fun} \\
\text{Fun} &\rightarrow \text{Unfun} \\
\text{Fun} &\rightarrow \text{Bifun} \\
\text{Unfun} &\rightarrow \text{Unfunname}(\text{Val}) \\
\text{Unfunname} &\rightarrow inv \\
\text{Bifun} &\rightarrow \text{Bifunname}(\text{Val}, \text{Val}) \\
\text{Bifunname} &\rightarrow symenc \\
\text{Bifunname} &\rightarrow symdec \\
\text{Bifunname} &\rightarrow conc \\
\text{Bifunname} &\rightarrow enc \\
\text{Bifunname} &\rightarrow dec \\
\text{Bifunname} &\rightarrow sign \\
\text{Bifunname} &\rightarrow ext
\end{aligned}$$

2.3.2 Able to gain knowledge

As the man-in-the-middle attacker gains knowledge, it is also of interest whether or not a man-in-the-middle attacker is able to gain a specific knowledge or not. To model the knowledge which is to check wether or not the man-in-the-middle attacker is able to gain it, the following **syntax** is used:

SeekKnowledge → Val, SeekKnowledge
SeekKnowledge → Condition, SeekKnowledge
SeekKnowledge → Val
SeekKnowledge → Condition
Condition → Val = Val
Condition → Val != Val
Condition → $true$
Condition → $false$
Val → string
Val → Fun
Fun → Unfun
Fun → Bifun
Unfun → Unfunname(Val)
Unfunname → inv
Bifun → Bifunname(Val, Val)
Bifunname → $symenc$
Bifunname → $symdec$
Bifunname → $conc$
Bifunname → enc
Bifunname → dec
Bifunname → $sign$
Bifunname → ext

2.4 Phase 1 : Attacker impersonate Objects

In the first phase of the analysis, it is checked whether or not an man-in-the-middle attacker could be able to impersonate another Object after sniffing at least once the whole communication. To do this a first order logic formula is constructed.

In case the whole communication was symmetric, all the man-in-the-middle attacker has to do is follow a defined message order. Knowing what another Object expect so see in a message, the man-in-the-middle attacker sends a message and receives an answer. With the answer the attack gains knowledge and based on the answer the attacker may again sends a new message and so forth. Which means, success with a messages implies new knowledge to be used for the next message. Which is why this behavior is modeled using

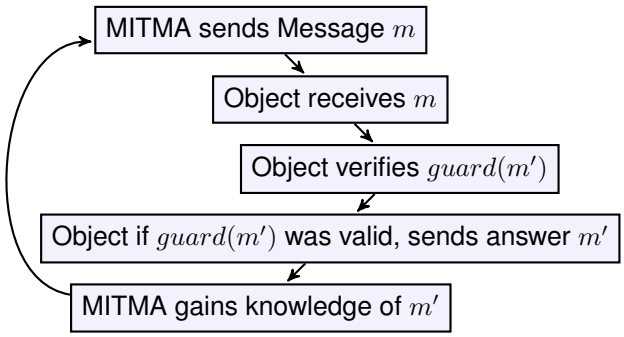

Fig. 1. Schematic of how an man-in-the-middle attacker gains knowledge

implications.

Let $Sends(A) = \{m_{s_0}, ..., m_{s_j}\}$ be the set of messages Object A sends with

$$m_{s_k} < m_{s_j} =_{def} k < j$$

and let F_m be the first order logic formula constructed for the sent message $m \in Sends(A)$. The first order logic formula in case of whole symmetric communication for Object A would look like:

$$F_A = (...((F_{m_{s_0}} \implies F_{m_{s_1}}) \implies F_{m_{s_2}})... \implies F_{m_{s_j}})$$

But in case at least one message wasn't symmetric, every message whether they were symmetric or not are seen as asymmetric, as there is no determinable order of all messages. In that case, the formula has to work with any possible order, which leads to the conjunction of each message formula like this:

$$F_A = F_{m_1} \wedge F_{m_2} \wedge F_{m_3} ... \wedge F_{m_n}$$

Iff F_A is a $contradiction$ then the attack is probably not able to impersonate as Object A, which makes communication with A secure.

2.4.1 Message formula

The first order logic formula for each sent message is constructed like this.

Let $Messages(A) =_{def} \{m_0, ..., m_n\}$ be the set of all messages Object A receives or sends with

$$\begin{aligned} Receives(A) =_{def} &\{m \mid m \in Messages(A) \wedge \\ &target(m) \neq A \wedge source(m) \neq A\} \\ = &\{m_{r_0}, ..., m_{r_k}\} \\ Sends(A) =_{def} &\{m \mid m \in Messages(A) \wedge \\ &source(m) \neq A \wedge target(m) \neq A\} \\ = &\{m_{s_0}, ..., m_{s_j}\} \\ m_k < m_j =_{def} &\ k < j \end{aligned}$$

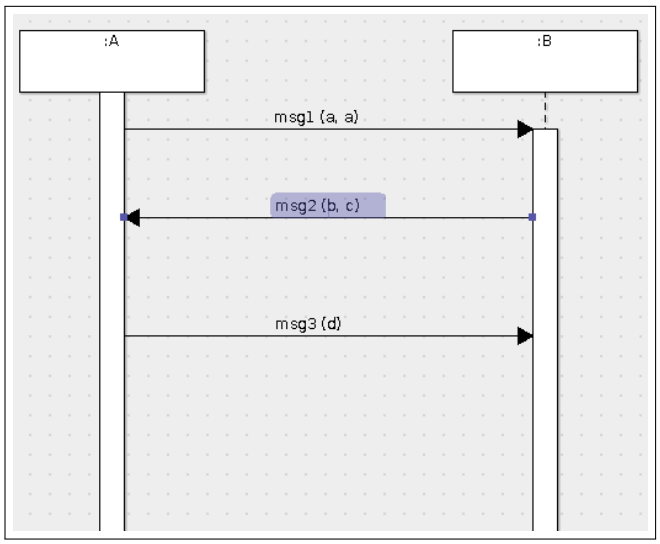

Fig. 2. Example UML Sequence Diagram

Then the constructed formula for each message $m \in Messages(A)$ is

$$\begin{aligned} F_{m_{s_n}} = ((&\forall m_{r_q} \in Receives(A).m_{s_{max(n-1,0)}} < m_{r_q}. \\ &\forall i \in \mathbb{N}_0.i \leq |args(m_{r_q})| : knows(name(m_{r_q})_i)) \\ &\wedge (guard(m))) \Rightarrow \forall a \in args(m_{s_n}).knows(a) \end{aligned}$$

2.5 Phase 2 : Check attackers knowledge

In the second phase of the analysis, it is check whether or not a man-in-the-middle attacker knows specific values after successfully sniffing the whole communication.

Let $seek = \{val_0, val_1, ...val_n\}$ be a set of values, which are to be check whether a man-in-the-middle attacker is able to gain knowledge about them. If $val_k \in knows \Leftrightarrow val_k$ is compromised .

2.6 Example Analysis

Given the sequence diagram of figure 2, initial knowledge $knows = \emptyset$ and the following sets

$$\begin{aligned} Messages =&\{Msg1(a,a), \\ &Msg2(b,c), \\ &Msg3(d)\} \\ Messages(A) =&\{Msg1(a,a), \\ &Msg2(b,c), \\ &Msg3(d)\} \\ Messages(B) =&\{Msg1(a,a), \\ &Msg2(b,c), \\ &Msg3(d)\} \\ Sends(A) =&\{Msg1(a,a), \\ &Msg3(d)\} \\ Receives(A) =&\{Msg2(b,c)\} \\ Sends(B) =&\{Msg2(b,c)\} \\ Receives(B) =&\{Msg1(a,a), \\ &Msg3(d)\} \end{aligned}$$

$$Msg1(a,a) < Msg2(b,c) < Msg3(d)$$

$$\begin{aligned} m_0 =&Msg1(a,a) \\ m_1 =&Msg2(b,c) \\ m_2 =&Msg3(d) \\ guard(m_0) =&[] \\ guard(m_1) =&[\ Msg1_1 = Msg1_2\ ] \\ guard(m_1) =&[\ (Msg2_1 = \text{b AND } Msg2_1 = \text{c})\ ] \\ F_A =&F_{m_0} \implies F_{m_2} \\ F_B =&F_{m_1} \\ F_{m_0} =&(\top \wedge guard(m_0)) \Rightarrow knows(a) \wedge knows(a) \\ F_{m_1} =&((knows(Msg1_1) \wedge knows(Msg1_2)) \wedge guard(m_1)) \\ &\Rightarrow knows(b) \wedge knows(c) \\ F_{m_2} =&((knows(Msg2_1) \wedge knows(Msg2_2)) \wedge guard(m_2)) \\ &\Rightarrow knows(d) \end{aligned}$$

$$\begin{aligned} KnowsFraction =&\{knows(Msg1_1), knows(Msg1_2), \\ &knows(Msg2_1), knows(Msg2_2), \\ &knows(Msg3_1)\} \\ KnowsFraction \subset knows& \end{aligned}$$

A valid assignment would be e.g.

$$\begin{aligned} Msg1_1 &= a \\ Msg1_2 &= a \\ Msg2_1 &= b \\ Msg2_2 &= c \\ Msg3_1 &= d \end{aligned}$$

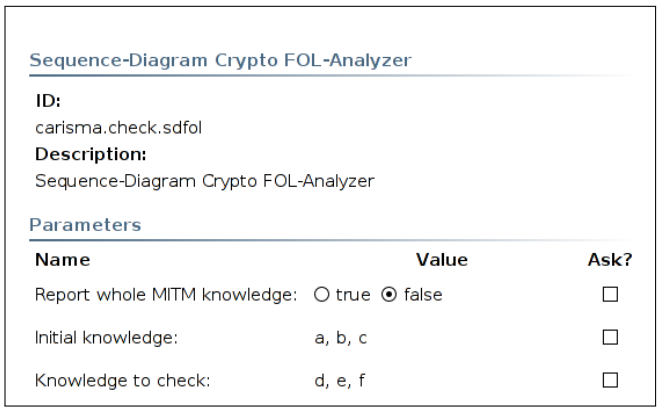

Fig. 3. Sequence-Diagram Crypto FOL-Analyzer Configuration

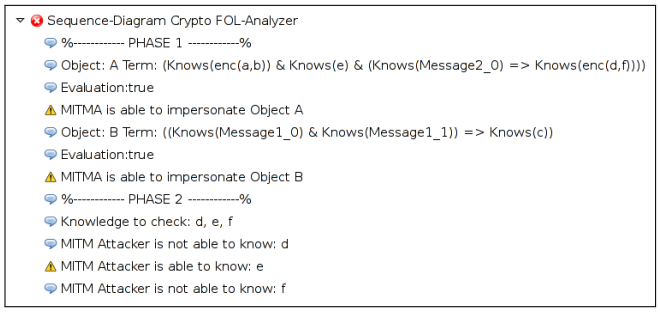

Fig. 4. Example Analysis Report

With this assignment F_A, F_B are both true, which means communication isn't secure with Objects A, B as a man-in-the-middle attacker is able to impersonate A and B.

3 Analyzing using the CARiSMA Plugin

The usage of the plugin is quite easy. Firstly a valid UML Sequence Diagram in the .uml2 format is needed. **Papyrus** or **AlgoUML** may be use to create such an UML Sequence Diagram. Also the guards have to be valid under the syntax of 2.2.1.

Then load the .uml2 file in CARiSMA, select the Sequence-Diagram Crypto FOL-Analyzer , add initial knowledge (valid under the syntax of 2.3.1) and knowledge which is to check whether or not the attacker is able to gain it (valid under the syntax of 2.3.2).

4 References

This Plugin is based on the work *Werkzeugunterstützte Sicherheits-Analyse von kryptographischen Protokollen mit automatischen Theorem-Beweisern* (Tool supported security analysis of cryptographic protocols of automatic theorem solver) by **Andreas Gilg**.