Верификация криптографических протоколов с использованием Тамарин (Tamarin)

Винарский Евгений
по всем вопросам писать на vinevg2015@gmail.com

Институт Системного программирования

6 ноября 2025 г.

Содержание

Криптографичекские сообщения в системе Tamarin

Система Tamarin доступна на tamarin-prover.github.io/

- *Public constants*: идентификаторы и метки агентов (Agent)
- *Fresh constants*: закрытые ключи, nonce

Встроенные функции:

- fst(pair(x,y)) = x
- snd(pair(x,y)) = y
- $< x_1, x_2, \ldots, x_{n-1}, x_n >$
- функции и арность: $f_1/a_1, \ldots, f_n/a_n$

Типы переменных:

- $\sim x$ означает, что x – fresh (секретная информация)
- $\$x$ означает, что x – pub (открытая информация)
- $\#i$ означает, что x – temporal (вспомогательная переменная)

Встроенные теории и примитивы в системе Tamarin

- Теория **hashing**:
  - определяет функцию $h/1$
- Теория **asymmetric-encryption**:
  - определяет функции: $aenc/2$, $adec/2$, $pk/1$
  - определяет аксиомы: $adec(aenc(m, pk(sk)), sk) = m$
- Теория **symmetric-encryption**:
  - определяет функции: $senc/2$, $sdec/2$
  - определяет аксиомы: $sdec(senc(m, k), k) = m$
- Теория **diffie-hellman**:
  - определяет функции: $inv/1$
  - определяет аксиомы:

```
(x^y)^z   = x^(y*z)
x^1       = x
x*y       = y*x
(x*y)*z   = x*(y*z)
x*1       = x
x*inv(x)  = 1
```

Состояния и переходы в системе Tamarin

Текущая конфигурация в системе Tamarin определяется совокупностью состояний агентов и переходами между этими состояниями
Состояние агента определяется выполненными шагами протокола:

- Ключевая пара сгенерирована
- Сообщение отправлено
- ...

Переход – изменение состояния системы в результате выполнения некоторого (возможно пустого) действия

- Соединение установлено
- Общий сессионный ключ выработан
- ...

Состояние, в котором никакой агент ещё не осуществил ни одного шага, объявляется начальным состоянием

Правила в системе Tamarin

- **Left part**: правило может быть применено к состоянию, в котором верны соответствующие утверждения
- **Action part**: переход помечается соответствующими действиями $Act1(\ n)$ и $Act2(x)$
- **Right part**: после применения правила будет переход в состояние, в котором верен набор соответствующих утверждений

Пусть верны утверждения $Pre(x)$ и $Fr(\sim n)$
Тогда, применяя правило *fictitious*, переходим в состояние с утверждением $Out(< x,\ n >)$

```
rule fictitious:
    [ Pre(x), Fr(~n) ]
  --[ Act1(~n), Act2(x) ]-->
    [ Out(<x,~n>) ]
```

Утверждение (Facts) в системе Tamarin

Встроенные утверждения в системе Tamarin:

- **In**: Получение сообщения из общего канала (может быть применён только в левой части)
- **Out**: Отправка сообщения в общий канал (может быть применён только в правой части)
- **Fr**: Генерация случайного секрета: закрытый ключ, nonce (может быть применён только в левой части)

persistent facts (!) – никогда не удаляются из состояний

Каждый участник протокола (в том числе и противник) может отправлять и получать все сообщения, находящиеся в общем канале

Tamarin Prover: символьная верификация криптопротокола

- Σ – сигнатура (переменные, функциональные символы, предикаты, утверждения)
- E – уравнения, связывающие функциональные символы
- P – система переписывания правил (протокол)
- Dependency Graph (dg) – граф зависимости, который строится по Σ, E и P
- $paths(dg)$ – множество путей, порождаемых в графе зависимости dg
- Формула φ

Проверяем выполнимость формулы φ на всех путях множества $paths(dg)$ ($paths(dg) \models \varphi$)

Леммы в системе Tamarin

Леммы в системе Tamarin записываются на языке логики предикатов первого порядка

- **All**: квантор всеобщности, вспомогательные переменные помечаются префиксом #
- **Ex**: квантор существования, вспомогательные переменные помечаются префиксом #
- ==>: импликация
- &: конъюнкция, |: дизъюнкция, *not*: отрицание
- $f@arg$: действие на функцию в момент *arg* (запись $K(k)@j$ используется для факта раскрытия противником ключа k в момент времени j)
- $a < b$ ($a = b$), если значение переменной a меньше значения (равно значению) переменной b
- $\#a < \#b$ ($\#a = \#b$): если значение вспомогательной переменной a меньше значения (равно значению) вспомогательной переменной b

Tamarin Prover: граф зависимостей

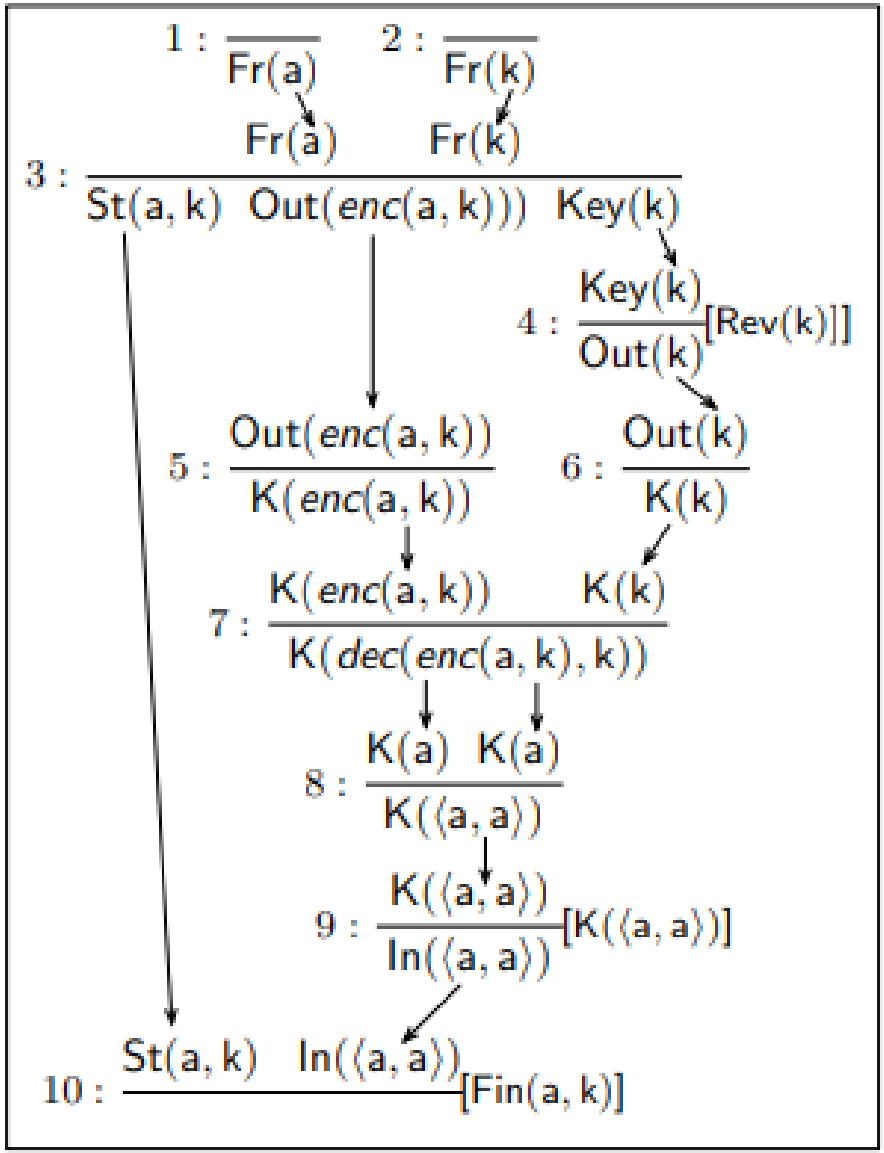
$$
\begin{aligned}
P = \{\ & [\mathsf{Fr}(x), \mathsf{Fr}(k)] -[]\rightarrow [\mathsf{St}(x,k), \mathsf{Out}(\mathrm{enc}(x,k)), \mathsf{Key}(k)] \\
, \ & [\mathsf{St}(x,k), \mathsf{In}(\langle x, x\rangle)] -[\ \mathsf{Fin}(x,k)\ ]\rightarrow [] \\
, \ & [\mathsf{Key}(k)] -[\ \mathsf{Rev}(k)\ ]\rightarrow [\mathsf{Out}(k)]\ \}\ .
\end{aligned}
$$

Может ли Tamarin выполнить протокол?

Tamarin Prover: граф зависимостей

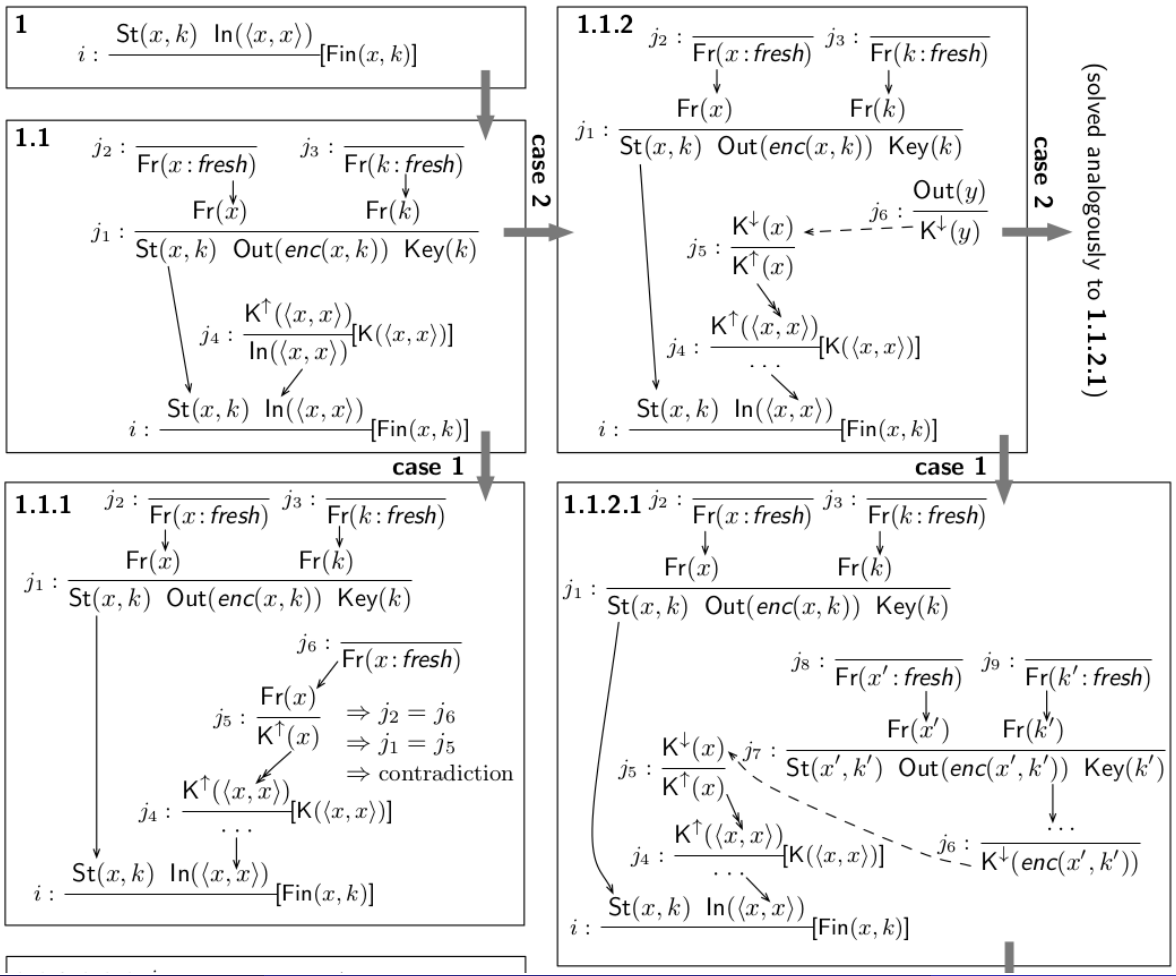
$$P = \{\ [\mathsf{Fr}(x), \mathsf{Fr}(k)] -[\,]\rightarrow [\mathsf{St}(x,k), \mathsf{Out}(\mathrm{enc}(x,k)), \mathsf{Key}(k)]$$
$$,\ [\mathsf{St}(x,k), \mathsf{In}(\langle x, x\rangle)] -[\ \mathsf{Fin}(x,k)\ ]\rightarrow [\,]$$
$$,\ [\mathsf{Key}(k)] -[\ \mathsf{Rev}(k)\ ]\rightarrow [\mathsf{Out}(k)]\ \}\ .$$

Может ли Tamarin выполнить протокол?



Tamarin Prover: пример построения системы ограничений

$$P = \{\ [\mathsf{Fr}(x), \mathsf{Fr}(k)] \dashv [] \rightarrow [\mathsf{St}(x,k), \mathsf{Out}(\mathsf{enc}(x,k)), \mathsf{Key}(k)]$$
$$,\ [\mathsf{St}(x,k), \mathsf{In}(\langle x,x\rangle)] \dashv [\ \mathsf{Fin}(x,k)\ ] \rightarrow []$$
$$,\ [\mathsf{Key}(k)] \dashv [\ \mathsf{Rev}(k)\ ] \rightarrow [\mathsf{Out}(k)]\ \}\ .$$



Tamarin Prover: пример описания протокола Диффи-Хеллмана

- (pr_C, pk_C) – долговременные (закрытый, открытый) ключи клиента
- (x, g^x) – сессионные (закрытый, открытый) ключи клиента
- (pr_S, pk_S) – долговременные (закрытый, открытый) ключи сервера
- (y, g^y) – сессионные (закрытый, открытый) ключи сервера

Диффи-Хэллман

Client (pkC, privC)		**Server (pkS, privS)**
- генерация g^*x*	<*pkC*, g^*x*> ---------->	
	<*pkS*, g^*y*> <----------	- генерация g^*y*
- *client_key* = g^*y*^*x*	<*N*B>pk(B) ---------->	- *server_key* = g^*x*^*y*
key = g^(*x***y*) -- общий секрет		

Tamarin Prover: пример описания протокола (2)

```
rule Clnt_Step_1:
let
    pk_C = pk(~pr_C)
    g_x = 'g'^(~x)
in
    [
        Fr(~pr_C),
        Fr(~x)
    ]
    --[
        Send_1($Client, $Server, <pk_C, g_x>)
    ]->
    [
        Out(<pk_C, g_x>),
        !Clnt_Step_1(~pr_C, pk_C, ~x, g_x)
    ]
```

```
rule Clnt_Step_2:
let
    client_key = g_y^(~x)
in
    [
        !Clnt_Step_1(~pr_C, pk_C, ~x, g_x),
        In(<pk_S, g_y>)
    ]
    --[
        Client_Key(client_key),
        Receive_2($Client, $Server, <pk_S, g_y>),
        Client_Finished($Client, client_key)
    ]->
    [
        !Clnt_Step_2(client_key, pk_S, g_y)
    ]
```

```
rule Serv_Step_1:
let
    pk_S = pk(~pr_S)
    g_y = 'g'^(~y)
    server_key = g_x^(~y)
in
    [
        Fr(~pr_S),
        Fr(~y),
        In(<pk_C, g_x>)
    ]
    --[
        Server_Key(server_key),
        Receive_1($Server, $Client, <pk_C, g_x>),
        Send_2($Server, $Client, g_y),
        Server_Finished($Server, server_key)
    ]->
    [
        !Serv_Step_1(~pr_S, pk_S, ~y, g_y),
        Out(<pk_S, g_y>)
    ]
```

Tamarin Prover: пример описания лемм протокола (1)

- $Client\_Finished(Client, key)$ – клиент выработал сессионный ключ key (установил соединение)
- $Server\_Finished(Server, key)$ – сервер выработал сессионный ключ key (установил соединение)

Лемма: Существует сервер, клиент и сессионный ключ такие, что клиент смог установить соединение с сервером

Tamarin Prover: пример описания лемм протокола (1)

- *Client_Finished*(*Client*, *key*) – клиент выработал сессионный ключ *key* (установил соединение)
- *Server_Finished*(*Server*, *key*) – сервер выработал сессионный ключ *key* (установил соединение)

Лемма: Существует сервер, клиент и сессионный ключ такие, что клиент смог установить соединение с сервером

```
lemma executable_Finished:
    exists-trace
    "Ex Client Server key #i #j.
        Client_Finished(Client, key) @i &
        Server_Finished(Server, key) @j
    "
```

Tamarin Prover: пример описания лемм протокола (2)

- $Client\_Finished(Client, key)$ – клиент выработал сессионный ключ key (установил соединение)
- $Server\_Finished(Server, key)$ – сервер выработал сессионный ключ key (установил соединение)
- $K(key)$ – противник узнал ключ key

Лемма: Для любого сервера, клиента и сессионного ключа, если клиент смог установить соединение с сервером, то противник не может узнать сессионный ключ

Tamarin Prover: пример описания лемм протокола (2)

- $Client\_Finished(Client, key)$ – клиент выработал сессионный ключ key (установил соединение)
- $Server\_Finished(Server, key)$ – сервер выработал сессионный ключ key (установил соединение)
- $K(key)$ – противник узнал ключ key

Лемма: Для любого сервера, клиента и сессионного ключа, если клиент смог установить соединение с сервером, то противник не может узнать сессионный ключ

```
lemma Key_secrecy:
    all-traces
    "All Client Server key
         #t1 #t2.
    (
        Client_Finished(Client, key) @t1 &
        Server_Finished(Server, key) @t2
    )
    ==>
    (
        not
        (
            Ex #k.
            (
                K(key) @ #k
            )
        )
    )
```

Tamarin Prover: пример описания лемм протокола (3)

- $Client\_Finished(Client, key)$ – клиент выработал сессионный ключ key (установил соединение)
- $Server\_Finished(Server, key)$ – сервер выработал сессионный ключ key (установил соединение)
- $Receive\_1(Server, Client, mess)$ – сервер получил от клиента сообщение $mess$
- $Send\_1(Client, Server, mess)$ – клиент отправил серверу сообщение $mess$

Лемма: Для любого сервера, клиента и сообщения, если клиент смог установить соединение с сервером и сервер получил сообщение, то его отправил именно клиент

Tamarin Prover: пример описания лемм протокола (3)

- $Client\_Finished(Client, key)$ – клиент выработал сессионный ключ key (установил соединение)
- $Server\_Finished(Server, key)$ – сервер выработал сессионный ключ key (установил соединение)
- $Receive\_1(Server, Client, mess)$ – сервер получил от клиента сообщение $mess$
- $Send\_1(Client, Server, mess)$ – клиент отправил серверу сообщение $mess$

Лемма: Для любого сервера, клиента и сообщения, если клиент смог установить соединение с сервером и сервер получил сообщение, то его отправил именно клиент

```
lemma auth_1:
    "All Client Server key mess
        #t1 #t2 #t.
    (
        Client_Finished(Client, key) @t1 &
        Server_Finished(Server, key) @t2 &
        Receive_1(Server, Client, mess) @t
    )
    ==>
        (Ex #j. Send_1(Client, Server, mess) @j & j<t)
    "
```

Tamarin Prover: пример описания протокола BADH

- (pr_C, pk_C) – долговременные (закрытый, открытый) ключи клиента
- (x, g^x) – сессионные (закрытый, открытый) ключи клиента
- $sig_C(mess)$ – ЭЦП на закрытом ключе клиента
- (pr_S, pk_S) – долговременные (закрытый, открытый) ключи сервера
- (y, g^y) – сессионные (закрытый, открытый) ключи сервера
- $sig_S(mess)$ – ЭЦП на закрытом ключе сервера

.

Client (pkC, privC)		**Server (pkS, privS)**
- генерация g^x	<g^x> ---------->	
	<g^y, pkS, sig_S(g^x, g^y)> <----------	- генерация g^y
- client_key = g^y^x	<pkC, sig_C(g^y, g^x)> ---------->	- server_key = g^x^y
key = g^(x*y) -- общий секрет		

Tamarin Prover: пример описания протокола BADH (нестойкость)

.

Client (pkC, privC)		**Server (pkS, privS)**
- генерация g^x	<g^x> ---------->	
	<g^y, pkS, sig_S(g^x, g^y)> <----------	- генерация g^y
- client_key = g^y^x	<pkC, sig_C(g^y, g^x)> ---------->	- server_key = g^x^y
key = g^(x*y) -- общий секрет		

Противник отправляет серверу сообщение $\overset{<pk_E, sig_E(g^y, g^x)>}{\rightarrow}$ от лица клиента

$\Downarrow$

Протокол не обеспечивает аутентификацию клиента перед сервером

Задание для самостоятельной работы

Необходимо построить модель на языке Tamarin протокола **ISO**

- (pr_C, pk_C) – долговременные (закрытый, открытый) ключи клиента
- (x, g^x) – сессионные (закрытый, открытый) ключи клиента
- $sig_C(mess)$ – ЭЦП на закрытом ключе клиента
- (pr_S, pk_S) – долговременные (закрытый, открытый) ключи сервера
- (y, g^y) – сессионные (закрытый, открытый) ключи сервера
- $sig_S(mess)$ – ЭЦП на закрытом ключе сервера

Client (pkC, privC)		**Server (pkS, privS)**
- генерация g^*x*	<*pkC*, g^*x*> ---------->	
	<*pkS*, g^*y*, sig_S(g^*x*, g^*y*, *pkC*)> <----------	- генерация g^*y*
- *client_key* = g^*y*^*x*	<sig_C(g^*y*, g^*x*, *pkS*)> ---------->	- *server_key* = g^*x*^*y*
key = g^(*x***y*) -- общий секрет		

Задание для самостоятельной работы (2)

Для построенной модели **протокола ISO** проверить следующие свойства

- + Существует сервер, клиент и сессионный ключ такие, что клиент смог установить соединение с сервером
- + Существует сервер, клиент и сообщение такое, что клиент отправил первое сообщение, и сервер его получил
- + Существует сервер, клиент и сообщение такое, что сервер отправил второе сообщение, и клиент его получил
- + Существует сервер, клиент и сообщение такое, что клиент отправил третье сообщение, и сервер его получил

Задание для самостоятельной работы (3)

Для построенной модели **протокола ISO** проверить следующие свойства

- Для любого сервера, клиента и сессионного ключа, если клиент смог установить соединение с сервером, то противник не может узнать сессионный ключ
- Для любого сервера, клиента и сообщения, если клиент смог установить соединение с сервером и сервер получил первое сообщение, то его отправил именно клиент
- Для любого сервера, клиента и сообщения, если клиент смог установить соединение с сервером и клиент получил второе сообщение, то его отправил именно сервер
- Для любого сервера, клиента и сообщения, если клиент смог установить соединение с сервером и сервер получил третье сообщение, то его отправил именно клиент

Правила сдачи самостоятельной работы

Максимальная оценка за домашнюю работу – 2 балла

- Выполненное домашнее задание присылать на почту vinevg2015@gmail.com, крайний срок отправки домашнего задания – 20 ноября 23:59