

Верификация криптографических протоколов с использованием Тамарин (Tamarin)

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

Институт Системного программирования

28 ноября 2024 г.

- 1 Синтаксис системы Tamarin
- 2 Tamarin: Пример описания протокола Диффи-Хеллмана
- 3 Tamarin: Пример описания протокола BADN
- 4 Задание для самостоятельной работы

Криптографические сообщения в системе Tamarin

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

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

Типы переменных:

Встроенные функции:

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

- $\sim x$ 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 $\#$
- \implies : 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

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

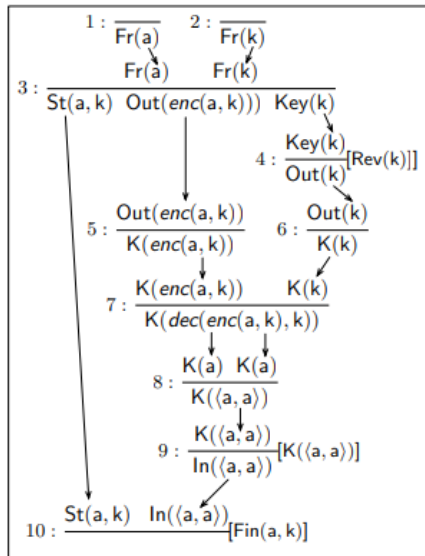


Может ли Tamarin выполнить протокол?

Tamarin Prover: граф зависимостей

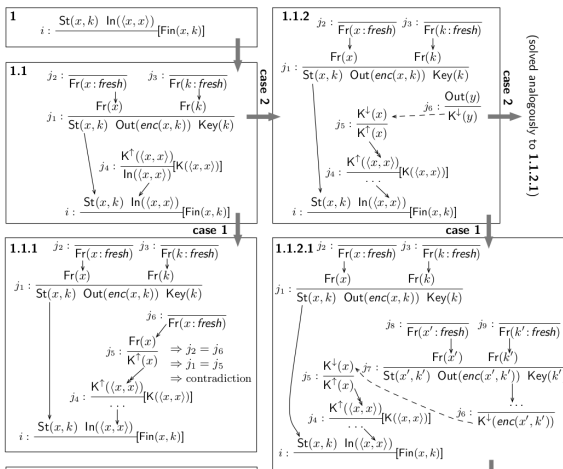
$$P = \{ [Fr(x), Fr(k)] \vdash [St(x, k), Out(enc(x, k)), Key(k)] \\ , [St(x, k), In(\langle x, x \rangle)] \vdash [Fin(x, k)] \vdash [] \\ , [Key(k)] \vdash [Rev(k)] \vdash [Out(k)] \} .$$


Может ли Tamarin выполнить протокол?



Tamarin Prover: пример построения системы ограничений

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



Тamarin 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 ($pk_C, priv_C$)		Server ($pk_S, priv_S$)
- генерация g^x	$\langle pk_C, g^x \rangle$ ----->	
	$\langle pk_S, g^y \rangle$ <-----	- генерация g^y
- $client_key = g^{xy}$	$\langle NB \rangle pk(B)$ ----->	- <u>$server_key = g^{xy}$</u>
$key = g^{(x*y)}$ -- 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_secretcy:
  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	$\langle g^x \rangle$ ----->	
	$\langle g^y, pkS, sig_S(g^x, g^y) \rangle$ <-----	- генерация g^y
- $client_key = g^{xy}$	$\langle pkC, sig_C(g^y, g^x) \rangle$ ----->	- $server_key = g^{xy}$
$key = g^{x*y}$ -- общий секрет		

Тamarin Prover: пример описания протокола BADH (нестойкость)

Client (pkC, privC)		Server (pkS, privS)
- генерация g^x	$\langle g^x \rangle$ ----->	
	$\langle g^y, pkS, \text{sig}_S(g^x, g^y) \rangle$ <-----	- генерация g^y
- $client_key = g^y \wedge x$	$\langle pkC, \text{sig}_C(g^y, g^x) \rangle$ ----->	- $server_key = g^x \wedge y$
$key = g^{(x*y)}$ -- общий секрет		

Противник отправляет серверу сообщение $\langle pk_E, \text{sig}_E(g^y, g^x) \rangle$ от лица клиента



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 (pk_C , $priv_C$)		Server (pk_S , $priv_S$)
- генерация g^x	$\langle pk_C, g^x \rangle$ ----->	
	$\langle pk_S, g^y, sig_S(g^x, g^y, pk_C) \rangle$ <-----	- генерация g^y
- $client_key = g^{xy}$	$\langle sig_C(g^y, g^x, pk_S) \rangle$ ----->	- <u>$server_key = g^{xy}$</u>
$key = g^{(x*y)}$ -- общий секрет		

Задание для самостоятельной работы (2)

Дедлайн – 23:59 11

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 – 3 балла

- Выполненное домашнее задание присылать в телеграмм @evgenii1996 или на почту vinevg2015@gmail.com, крайний срок отправки домашнего задания – 11 декабря 23:59
- В случае, если задание на доработку было отправлено больше одного раза, то максимальная оценка снижается до 1 балла