DOI: 10.15514/ISPRAS-2023-35(6)-10

Извлечение опорных тестовых наборов из спецификаций криптопротоколов на предметно-ориентированном языке

С.Е. Прокопьев, ORCID: 0000-0002-4410-2565 <*sepr@ispras.ru*>

Институт системного программирования РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.
АО «НПК «Криптонит»,
Россия, 115114, Москва, Шлюзовая набережная, 4

Аннотация. Работа посвящена описанию инструмента тестирования безопасности реализаций криптографических протоколов, работающего на основе спецификаций, написанных на декларативном интероперабельном встроенном предметно-ориентированном языке (emdedded [in Haskell] DSL). Рассмотрена задача формирования качественных опорных тестовых наборов для использования в тестировании безопасности реализаций протоколов. Предложен метод решения этой задачи в контексте представленного инструмента.

Ключевые слова: криптографические протоколы; формальные спецификации криптографических протоколов; тестирование криптографических протоколов; тестирование на основе формальных спецификаций; EDSL; виртуальная машина SBSM/C2.

Для цитирования: Прокопьев С.Е. Извлечение опорных тестовых наборов из спецификаций криптопротоколов на предметно-ориентированном языке. Труды ИСП РАН, том 35, вып. 6, 2023 г., стр. 167–178. DOI: 10.15514/ISPRAS-2023-35(6)-10.

Благодарности: Исследование поддержано грантом Минобрнауки России № 075-15-2020-788.

Extracting the Reference Test Suites from Cryptographic Protocol Specifications Written on a Domain-Specific Language

S.E. Prokopev, ORCID: 0000-0002-4410-2565 <sepr@ispras.ru>

Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
JSC "NPK Kryptonite",
4, Shluzovaya emb., Moscow, 115114, Russia

Abstract. The paper describes a tool for testing the security of cryptographic protocol implementations working on the basis of specifications written in a declarative interoperable domain-specific language implemented as EDSL (Embedded [in Haskell] DSL). The problem of forming high-quality reference test suites for testing the security of cryptoprotocol implementations is considered. A method of addressing this problem within the tool being developed is discussed.

Keywords: cryptographic protocol specifications, cryptographic protocol testing, specification-based testing, DSL embedded in Haskell.

For citation: Prokopev S.E. Extracting reference test suites from cryptographic protocol specifications written on a domain-specific language. *Trudy ISP RAN/Proc. ISP RAS*, vol. 35, issue 6, 2023. pp. 167-178 (in Russian). DOI: 10.15514/ISPRAS-2023-35(6)-10.

Acknowledgements. The work is supported by a grant from the Ministry of Education and Science of Russia.

1. Введение

Защита взаимодействия компонентов распределенных систем по каналам связи с использованием криптографических протоколов подразумевает проведение комплекса работ, связанных с разработкой спецификаций практических протоколов, оформляемых в виде документов типа RFC, а также анализом конформности, безопасности и совместимости программных реализаций протоколов. В рамках данных работ можно выделить ряд проблемных моментов:

- низкая степень формализации спецификаций протоколов (формально описываются только форматы сообщений и некоторые вычисления, при этом используемый формальный язык не является машинно-интерпретируемым);
- отсутствие признанных эффективных подходов к тестированию реализаций криптопротоколов, как в части проверки конформности, так и в части проверки безопасности (из-за сложных зависимостей сообщений протокола от всей предыстории выполнения протокола, реализации криптопротоколов являются очень неудобными объектами для стандартных промышленных фаззеров, основанных как на грамматиках (например, Peach), так и на использовании обратной связи по покрытию ветвей вычислений (например, фаззеры семейства AFL));
- отсутствие признанных эффективных методик проверки совместимости реализаций криптопротоколов от разных разработчиков.

Под эффективностью тестирования понимается выполнение требований, перечисленных в табл. 1 (для справки в таблице также даны оценки по трехбалльной шкале для ряда инструментов, построенных по сильно отличающимся друг от друга технологиям: tlsfuzzer (tf) [1], tlspuffin (tp) [2], UniTESK (un) [3] и NCT (nc) [4]).

Табл.1. Требования к инструментам тестирования реализаций протоколов
Table 1. Protocol testing tool requirements

	Требование	tf	tp	un	nc
1	Способность автоматически генерировать неожиданные неконформные сообщения	–	+	–	–
2	Способность автоматически генерировать неожиданные конформные сообщения	–	+	–	+
3	Способность генерировать неожиданные сообщения на уровне логики протокола	–	=	–	+
4	Способность различать сгенерированные конформные и неконформные сообщения	–	–	+	+
5	Наличие адекватного критерия качества тестирования	–	–	+	–
6	Прослеживаемость результатов тестирования к проверкам свойств безопасности из явно заданного перечня	+	–	+	+
7	Простота адаптации инструмента к новым протоколам	–	–	–	=
8	Скорость работы инструмента	+	+	+	–

Генерация неожиданных сообщений необходима для тестирования безопасности реализаций, при этом важно уметь генерировать сообщения, неожиданные не только на уровне мутаций на битовом уровне, но и на уровне логики протокола.

Способность отличать конформные сообщения от неконформных необходима для получения сильного тестового оракула: в ответ на отправку конформного сообщения инструмент ожидает нормальные ответы, а на отправку неконформного – сообщение об ошибке или разрыв соединения.

Критерий качества тестирования должен оценивать способность инструмента проникать в разнообразные (включая глубокие) состояния протокола, а также степень разнообразия генерируемых сообщений с точки зрения используемых сочетаний алгоритмов криптографических примитивов.

Прослеживаемость к заданному перечню свойств безопасности означает, что по результатам тестирования должно быть ясно, какие именно типы ошибок реализации искались.

Простота адаптации к новым протоколам означает, что с каждым новым протоколом трудоемкость проведения тестирования с использованием инструмента резко уменьшается.

Скорость работы инструмента измеряется как число выполненных тестов в секунду на одно ядро процессора (при наличии адекватного критерия качества тестирования данный критерий рассматривается как второстепенный).

Выполнение требований 4 и 5 из табл. 1 возможно только в рамках подхода к тестированию на базе выразительных формальных моделей, например, таких как расширенные автоматы (рис. 1).

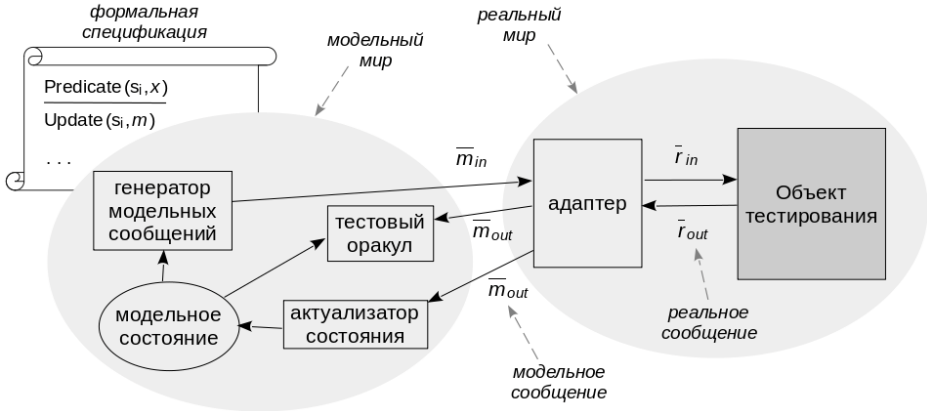

Рис. 1. Тестирование протоколов на основе расширенного автомата
Fig. 1. Protocol implementation testing based on an extended state machine

Под расширенными автоматами понимается семейство концептуально родственных друг другу автоматов с бесконечным числом состояний [5]. Упрощенно, модель протокола на базе расширенного автомата состоит из множества пар $\mathrm{Predicate}(s_i,x)$ / $\mathrm{Update}(s_i,m)$, где s_i – это состояние автомата, $m = (\mathrm{src}, \mathrm{dst}, a_1, \ldots, a_n)$ – это вектор полей модельного сообщения, включая адресата и отправителя сообщения.

Множество конформных сообщений протокола в состоянии s_i задается как множество всех векторов $(\mathrm{src}, \mathrm{dst}, a_1, \ldots, a_n)$ обращающих $\mathrm{Predicate}(s_i,x)$ в истину. В рамках методологии UniTESK извлечение этих решений проводится с помощью сценариев, которые необходимо дополнительно разрабатывать и прикладывать к спецификации протокола. В методе NCT решения извлекает SMT-решатель, что является медленной процедурой.

Основной проблемой здесь является сложность реализации в схеме на рис. 1 актуализатора состояния и адаптера: фактически для каждого протокола необходимо разработать

отдельную реализацию, совместимую с тестируемой реализацией на бинарном уровне, что противоречит требованию 7 из табл. 1.

Ранее в работах [6-7] автором был предложен высокоуровневый язык формальных спецификаций криптопротоколов – лаконичный, декларативный и выразительный. Язык основан на расширенном автомате SBSM, специально разработанном для предметной области криптографических протоколов, и реализован как встроенный предметно-ориентированный язык – то есть как EDSL (Embedded DSL), где в качестве хост-языка использован язык Haskell. Предложенный EDSL является интероперабельным, то есть совместимым с промышленными реализациями протоколов на уровне бинарных сообщений. Семантика языка задана на основе виртуальной машины SBSM/C2. Машина SBSM/C2 реализована на Haskell и обеспечивает возможность машинной интерпретации спецификаций на EDSL.

Так как язык спецификаций интероперабелен, то адаптер в схеме на рис. 1 не нужен. Кроме того, в реализацию виртуальной машины SBSM/C2 встроены функции автоматической актуализации модельного состояния.

В отличие от промышленных языков разработки, предложенный формальный язык спецификаций нацелен на максимально полное описание протокола: на описание всех разрешенных в RFC вариантов формирования сообщений протокола и их последовательностей. При этом ввиду того, что множество всех возможных траекторий выполнения протокола, задаваемых спецификацией, для промышленного криптопротокола (например, такого как TLS) огромно (и даже бесконечно), возникает задача автоматического извлечения из формальной спецификации небольшого по мощности, но качественного подмножества тестируемых траекторий.

2. Постановка задачи извлечения тестовых наборов из формальной спецификации на EDSL

Постановку задачи будем рассматривать на примере протокола TLS.

В записи протокола TLS (рис. 2) выделим три типа полей: информационные поля (белые прямоугольники), неинформационные поля (черные прямоугольники; это либо поля-константы, либо поля, которые автоматически вычисляются из других полей) и поля-последовательности (серые прямоугольники с пунктирными разделителями).

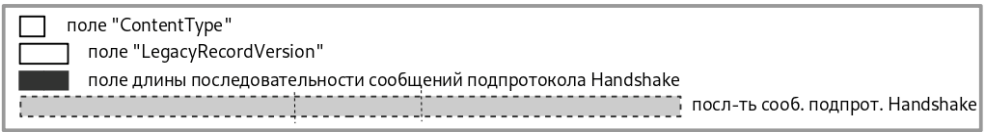

Рис. 2. Структура сообщений (записей) протокола TLS
Fig. 2. The structure of the records of the TLS protocol

Если взять сообщение ClientHello последовательности Handshake (рис. 3), то оно также содержит поля-последовательности: последовательность идентификаторов шифрнаборов и последовательность расширений сообщения ClientHello. В свою очередь, расширения сообщения ClientHello также могут включать в свой состав поля-последовательности, например, расширение Groups содержит последовательность двухбайтовых идентификаторов групп точек эллиптической кривой (рис. 4).

Фактически, структурное разнообразие конформных сообщений протокола TLS представляет собой разнообразие способов формирования последовательностей, входящих в состав сообщений: какие элементы и в каком порядке будут включены в каждую последовательность. И это в целом характерно для современных промышленных криптопротоколов. Например, в протоколе IPSEC IKEv2 можно обнаружить следующие

последовательности: payloads, SA proposals, transforms, configurations attributes, traffic selectors и др.

В EDSL при кодировании структур сообщений каждая такая последовательность описывается с помощью оператора Sequence, и этой последовательности назначается имя (рис. 5).

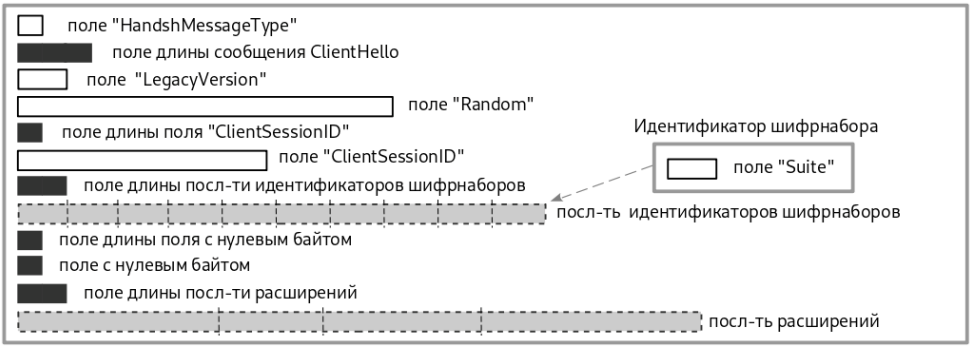

Рис. 3. Структура сообщения ClientHello подпротокола TLS Handshake
Fig. 3. The structure of the TLS ClientHello message

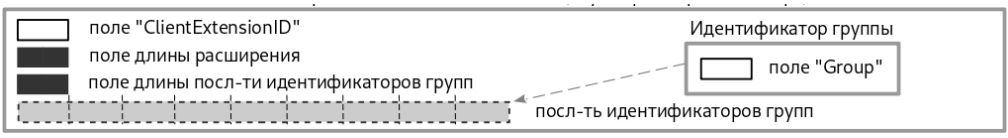

Рис. 4. Структура расширения Groups сообщения ClientHello
Fig. 4. The structure of the Groups extension of the TLS ClientHello message

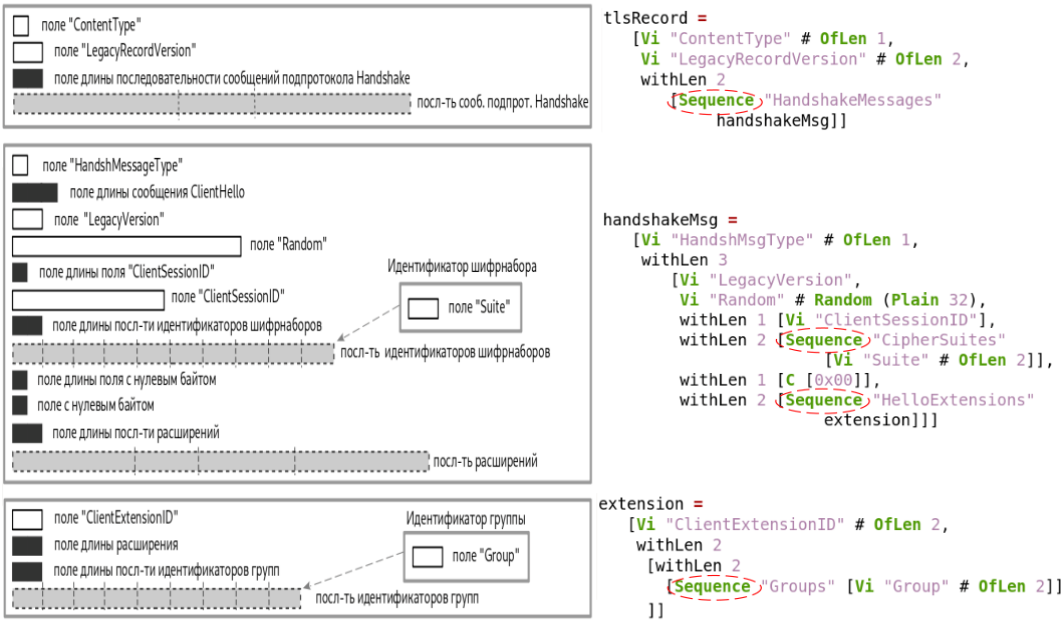

Рис. 5. Кодирование полей и последовательностей сообщений в EDSL
Fig. 5. Description of message fields and sequences in EDSL

С каждой последовательностью в спецификации ассоциировано дерево вариантов формирования элемента этой последовательности (рис. 6). Каждое ребро этого дерева

помечено парой Guard/Assign, где Guard – это гард (предикат), заданный на текущем состоянии автомата, Assign – список значений, назначаемых информационным полям формируемого элемента поля-последовательности. Проход по некоторому пути до конца дерева соответствует формированию одного элемента последовательности (при проходе будут присвоены значения всем информационным полям этого элемента). Путь в дереве может оканчиваться на Next (означает, что после формирования текущего элемента последовательности следует приступить к формированию следующего элемента), на Last (текущий формируемый элемент является последним) или на End (окончание формирования последовательности без формирования текущего элемента).

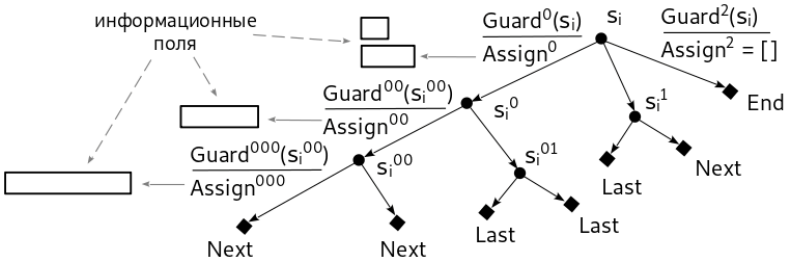

Рис. 6. Дерево вариантов формирования элемента последовательности
Fig. 6. Tree of variants of the sequence elements

При формировании конформных сообщений протокола тестировщику необходимо выбирать те пути, на которых все гарды обращаются в истину, а при формировании неконформных – все остальные. Таким образом, тестировщик всегда знает, какое сообщение он сформировал – конформное или неконформное.

Язык спецификаций предоставляет возможность гибкого управления выбором очередных элементов последовательности. Например, гард

```
ExistsIn ClientExtensions (Vi "ClientExtensionID" == [0x00,0x0a]) IsCurSess &&
Not (ExistsIn ClientExtensions (Vi "ClientExtensionID" == [0x00,0x33]) IsCurSess)
```

задает следующее условие: «в текущей сессии протокола сформированная ранее часть последовательности ClientExtensions содержит элемент, в котором информационному "ClientExtensionID" было присвоено значение [0x00,0x0a], и не содержит элементов, в которых информационному полю "ClientExtensionID" присвоено значение [0x00,0x33]».

Все возможные сочетания вариантов формирования последовательностей, содержащихся в сообщении TLS ClientHello, приводят к комбинаторному взрыву числа способов построения сообщения ClientHello (рис. 7).

На рис. 7 в виде $tpath_{seqName}^{x}$ обозначается путь с номером x в дереве вариантов формирования элемента последовательности с именем seqName (очевидно, что все такие *tpath-пути* в этом дереве можно перенумеровать). И это только одно сообщение протокола. Если составить из деревьев отдельных сообщений общее дерево траекторий протокола (рис. 8), то даже если ограничить число сессий протокола (например, тремя) и число сообщений в каждой сессии (например, двадцатью), то все равно обойти за разумное время все пути в этом дереве траекторий невозможно.

На рис. 8 тройка (clnt,srvr,msg) означает отправку участником clnt участнику srvr сообщения msg, сформированного в результате прохождения по некоторой последовательности tpath-путей.

Данное дерево траекторий задано неявно – это результат гипотетической работы автомата SBSM, моделирующего протокол, в недетерминированном режиме (то есть, когда в точках выбора вариантов продолжения траекторий автомат идет параллельно по всем путям). Вид этого дерева заранее не известен, вершины и помеченные ребра возникают динамически в процессе работы автомата.

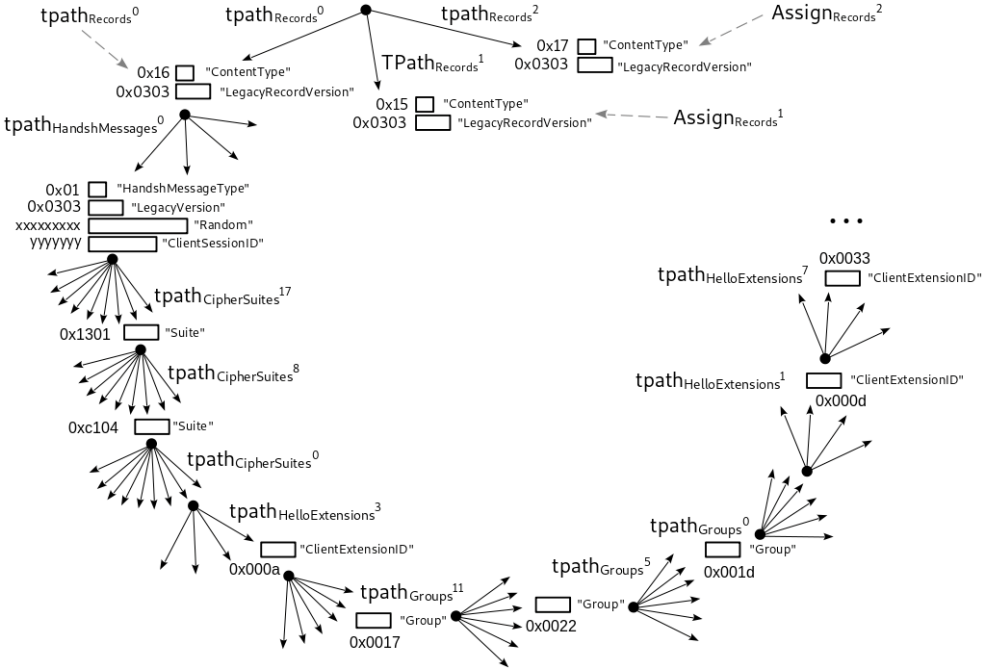

Рис. 7. Дерево вариантов формирования сообщения ClientHello
Fig. 7. Tree of variants of the ClientHello message

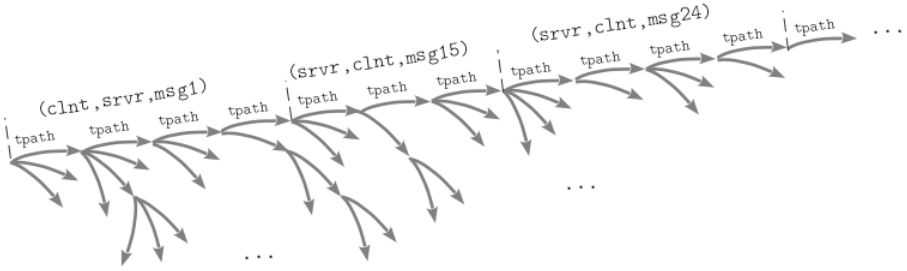

Рис. 8. Дерево траекторий протокола
Fig. 8. The tree of the protocol trajectories

Тестирование можно проводить методом случайного блуждания, то есть выбирать tpath-пути в развилках случайным образом. Однако ясно, что не все подмножества траекторий равноценны с точки зрения полезности при тестировании. Необходимо уметь автоматически выделять качественные подмножества траекторий, которые использовать и для проверки конформности, и для проверки безопасности реализаций.

В качестве критерия качества подмножества траекторий предлагается использовать критерий полноты покрытия типа MC/DC в следующей редакции: каждый гард рассматривается как формула, состоящая из атомарных предикатов (например, вида P1 && (P2 || P3)), и подмножество траекторий считается качественным, если каждый атомарный предикат каждого гарда спецификации побывал истинным, ложным и существенным (то есть значение гарда существенно зависит от этого атомарного предиката).

3. Метод факторизации множества траекторий тестирования

Процедура тестирования разбивается на две фазы. Цель первой фазы – провести тестирование конформности и одновременно факторизовать множество траекторий

протокола (остающееся огромным даже если ограничить длину траекторий) на классы эквивалентных траекторий и явно выписать эти классы в виде компактного автомата с конечным числом состояний. Вторая фаза – это тестирование безопасности реализации (фаззинг).

В рамках предлагаемого метода для борьбы с комбинаторным взрывом числа траекторий применяется последовательность приемов абстрагирования дерева траекторий протокола.

Шаг 1. В язык спецификаций добавлена возможность специальным образом помечать произвольные развилки дерева траекторий, давая тестировщику подсказки следующих двух типов:

тип 1: «эта развилка tpath-вариантов относится к последовательности, в которой нет зависимостей элементов ни от состояния, ни друг от друга; число вариантов формирования этой последовательности огромно, но все они распадаются на небольшое число классов эквивалентности»,

тип 2: «эта развилка tpath-вариантов относится к последовательности, в которой есть зависимости элементов друг от друга, но эти зависимости находятся исключительно на уровне гардов данной последовательности; число вариантов формирования этой последовательности большое, но не огромное, и среди них много эквивалентных».

Когда при обходе дерева траекторий тестировщик обнаруживает помеченную развилку, он продолжает обход лишь по одному ребру (первому или случайному), а остальные ребра откладывает в запас (рис. 9).

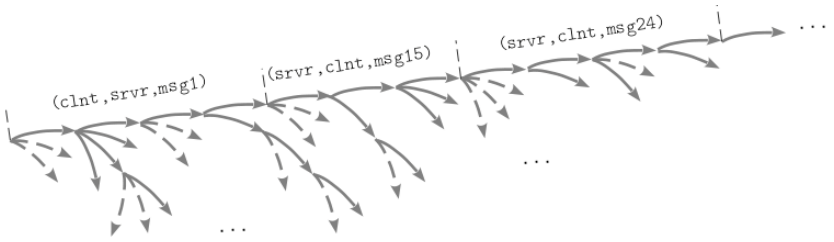

Рис. 9. Дерево траекторий протокола с отложенными ребрами
Fig. 9. The tree of the protocol with postponed edges

В случае развилки типа 2 тестировщик откладывает явно вычисленные tpath-варианты, а в случае развилки типа 1 – символическое описание отложенного множества tpath-вариантов.

Шаг 2. Вводится функция абстрагирования сообщения msg:

abstract(msg) = (true_atomic(msg), false_atomic(msg)),

где true_atomic – это мультимножество атомарных предикатов, входящих в состав гардов, встретившихся при формировании сообщения и обратившихся в истину, а false_atomic – это мультимножество остальных встретившихся гардов.

Для каждой вершины дерева, ребра сообщений msg, отображающиеся в одинаковые абстрактные сообщения, заменяются на одно ребро (рис. 10).

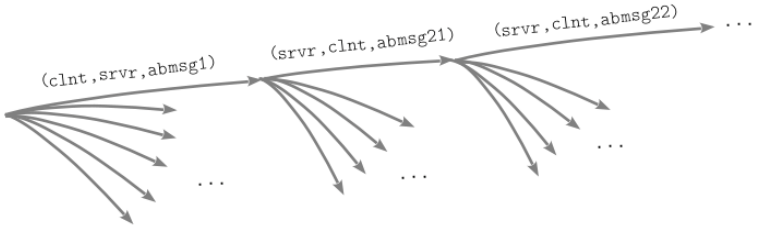

Рис. 10. Дерево траекторий протокола с абстрактными сообщениями
Fig. 10. The tree of the protocol with abstract messages as edges

Шаг 3. Наложим следующее условие: поведение объекта тестирования детерминировано, то есть на одни и те же воздействия он отвечает одинаково. Используя данное ограничение, будем строить дерево траекторий для конкретной конфигурации объекта тестирования: из всех ребер сообщений, которые потенциально мог бы отправить объект тестирования в некоторой вершине дерева, оставляем только то, которое было выбрано им фактически (рис. 11; здесь тестировщик работает в роли клиента, а объект тестирования – в роли сервера).

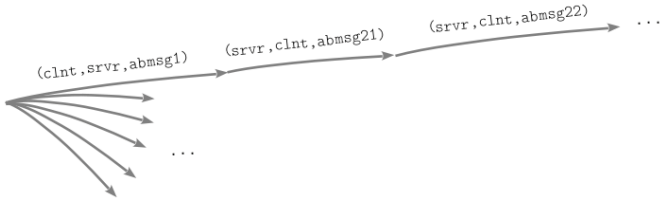

Рис. 11. Дерево траекторий протокола с детерминированным участником
Fig. 11. Customized tree of the protocol

Шаг 4. Ребра сообщений, отправляемых тестировщиком, и последующие ребра ответных флайтов (то есть ребра серии ответных сообщений) объекта тестирования заменяются на одно ребро (рис. 12), при этом флайты ответов могут быть пустыми.

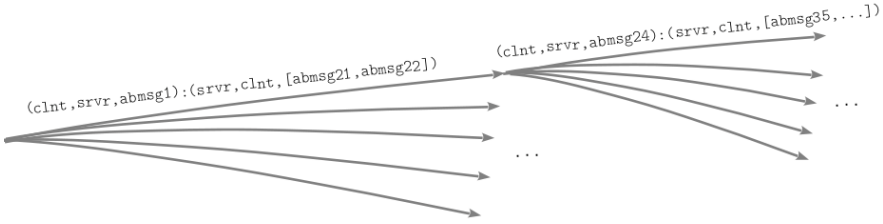

Рис. 12. Дерево траекторий протокола с флайтами сообщений
Fig. 12. The tree of the protocol with message flights as edges

Шаг 5. Согласно стандартному подходу, два состояния некоторого автомата, генерирующего сообщения, считаются эквивалентными, если из них «растут» одинаковые деревья сообщений. С учетом того, что длина каждой траектории в дереве бесконечна, данное сравнение возможно провести лишь на некоторых конечных поддеревьях этих деревьев.

В нашем методе точностью сравнения управляет специальный (изначально пустой) накопитель Precision, содержащий списки идентификаторов абстрактных сообщений abmsg, отправляемых тестировщиком. Эти списки интерпретируются как префиксы путей в дереве, изображенном на рисунке 1.12.

Абстрактным состоянием для состояния S_i тестировщика называется следующее конечное поддерево бесконечного дерева, растущего из вершины S_i: в первом слое оставлены все ребра, далее, для каждого элемента каждого префикса пути, входящего в Precision, дерево продлевается еще на один слой (рис. 13).

При тестировании тестировщик ходит по траекториям протокола, вычисляя и запоминая абстрактные сообщения и соединяя их ребрами. Если абстрактное состояние повторилось, то тестировщик далее по этой траектории не идет.

Когда все траектории завершатся попаданием в известное абстрактное состояние, будет получена текущая гипотеза конечного автомата тестирования (далее, автомат-гипотеза).

Шаг 6. Для текущего автомата-гипотезы осуществляется поиск контрпримеров двух типов:

1) опробование ребер, отложенных в запас: в помеченных развилках, в которых на шаге 1 было оставлено только одно tpath-ребро, это оставленное ребро заменяется на одно из tpath-ребер, отложенных в запас, после чего проверяется, приводит ли данная

замена к изменению автомата-гипотезы; если не приводит, то данное tpath-ребро записывается в класс эквивалентности первого tpath-ребра; если приводит, то это означает, что найден контрпример;

2) проход по различным путям автомата-гипотезы с целью поиска вычисленного абстрактного состояния, не совпадающего с абстрактным состоянием, которое должно быть в этой точке этого пути согласно автомату-гипотезе.

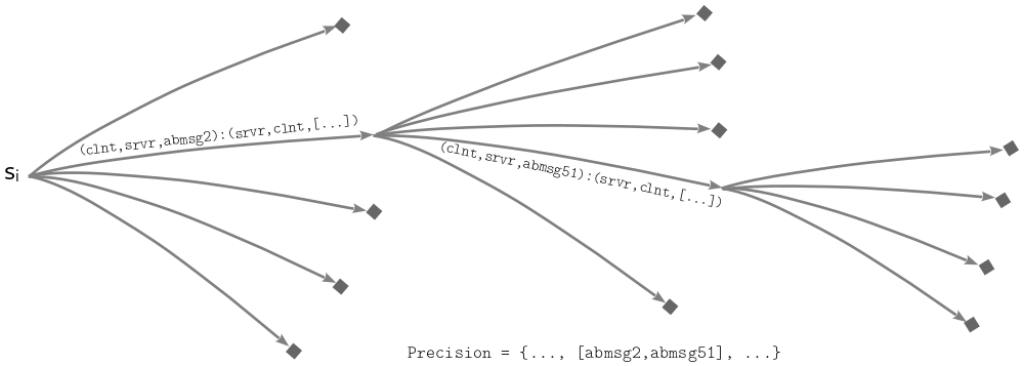

Рис. 13. Абстрактное состояние протокола
Fig. 13. Protocol abstract state

После нахождения контрпримера первого типа в помеченную развилку, в которой было найдено tpath-ребро-контрпример, добавляется это ребро (то есть в развилке теперь не одно, а два ребра), и шаги 1-5 проделываются заново. На последующих итерациях, при извлечении нового ребра из запаса последовательно проверяется, какому ребру из развилки он эквивалентен. Если никакому, то найден контрпример (новый класс эквивалентности) и т. д. После нахождения контрпримера второго типа в накопитель Precision добавляется путь в состояние тестировщика, на котором было выявлено несоответствие абстрактных сообщений, и шаги 1-5 проделываются заново для уточненных (refined) абстрактных состояний.

Отсутствие контрпримеров на некоторой итерации выполнения шагов 1-6 в течение продолжительного времени означает успешное завершение построения автомата тестового набора.

4. Заключение

Сходимость (постоянное увеличение уровня детализации автомата) и завершимость описанной итеративной процедуры тестирования на практических протоколах были проверены экспериментально на примере протоколов TLS и IPSec IKEv2. На рис. 14 представлены примеры автоматов-гипотез на четырех итерациях алгоритма при тестировании одной из конфигураций сервера OpenSSL.

Так как абстрагирование сообщений протокола и состояний автомата осуществляется на базе значений атомарных предикатов гардов, то представленный в настоящей работе метод построения опорного тестового набора нацелен на достижение критерия MC/DC в приведенной выше редакции. Однако автоматическое выполнение данного критерия не гарантируется. Непокрытые атомарные предикаты выводятся аналитику, который должен построить недостающие траектории с помощью специального инструмента ручного построения траекторий, после чего эти траектории будут автоматически встроены в опорный тестовый набор.

Построение опорного тестового набора означает завершение первой фазы тестирования реализации криптопротокола – фазы проверки конформности.

Далее этот тестовый набор будет использоваться во второй фазе тестирования – фазе проверки безопасности (фаззинге) – в качестве несущих траекторий мутационных воздействий на объект тестирования. Знание фаззером логического смысла содержимого полей сообщений протоколов (шифртекст, точка эллиптической кривой и т.д) обеспечит возможность применения предметно-ориентированных мутаций (например, отправку открытого ключа, не лежащего на согласованной эллиптической кривой) и исключит бессмысленные мутации (например, перебор искажений шифртекста или значений хэш-функции). Как следствие, может быть выполнено требование 6 из табл. 1 (прослеживаемость результатов тестирования к проверкам свойств безопасности из явно заданного перечня угроз).

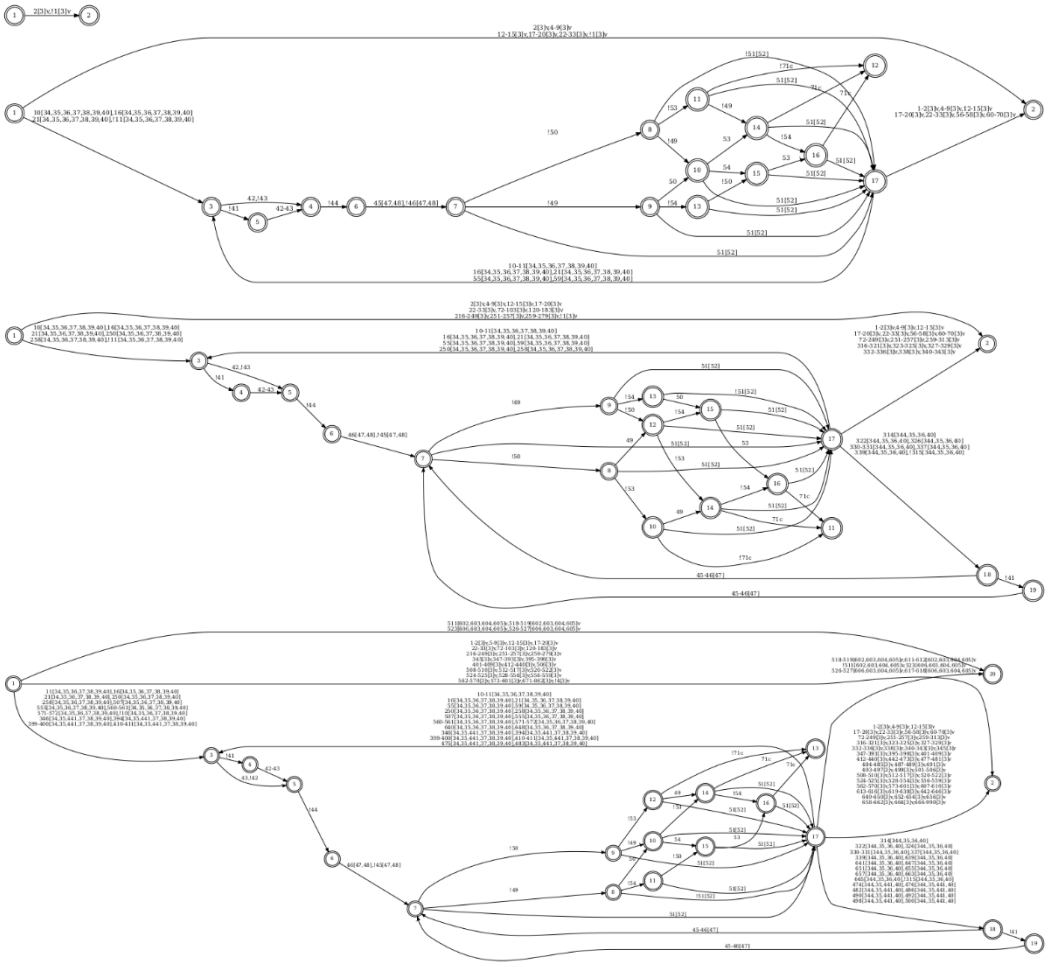

Рис. 14. Автоматы-гипотезы на разных итерациях метода
Fig. 14. Hypotheses of the automaton on the iterations of the method

Список литературы / References

[1]. H. Kario. Tlsfuzzer. Available at: https://github.com/tomato42/tlsfuzzer, accessed 10.11.2023.
[2]. M. Ammann, L. Hirschi, S. Kremer. DY Fuzzing: Formal Dolev-Yao Models Meet Cryptographic Protocol Fuzz Testing. https://eprint.iacr.org/2023/057.
[3]. Пакулин Н.В., Шнитман В.З., Никешин А.В. Разработка тестового набора для верификации реализаций протокола безопасности TLS // Труды Института системного программирования РАН, том 23, 2012 г., стр. 387-404.

[4]. Kenneth L. McMillan and Lenore D. Zuck. 2019. Formal specification and testing of QUIC. In Proceedings of ACM Special Interest Group on Data Communication (SIGCOMM'19). ACM, New York, NY, USA, 14 pages.
[5]. ABZ – International Conference on Rigorous State Based Methods, Available at: https://abz-conf.org, accessed 10.11.2023.
[6]. Прокопьев С.Е. Формальный язык первичных спецификаций криптографических протоколов. Труды ИСП РАН. 2021;33(5):117-136. https://doi.org/10.15514/ISPRAS-2021-33(5)-7.
[7]. Prokopev, S. Cryptographic protocol conformance testing based on domain-specific state machine. J Comput Virol Hack Tech (2023). https://doi.org/10.1007/s11416-023-00474-1.

Информация об авторах / Information about authors

Сергей Евгеньевич ПРОКОПЬЕВ – старший научный сотрудник отдела технологий программирования ИСП РАН, руководитель направления сетевых протоколов безопасности лаборатории сетевой и информационной безопасности АО НПК «Криптонит».

Sergey Evgenevich PROKOPEV – research associate in the Department of Programming Technologies of the ISP RAS, senior researcher in the area of security protocols in the Laboratory of the Network Security of JSC NPK "Kryptonite".