DOI: 10.15514/ISPRAS-2021-33(5)-7

Формальный язык первичных спецификаций криптографических протоколов

С.Е. Прокопьев, ORCID: 0000-0002-4410-2565 <sepr@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
АО «НПК «Криптонит»,
115114, Москва, Шлюзовая набережная, 4 (БЦ «Россо Рива»)

Аннотация. Применение формальных методов в разработке реализаций сетевых протоколов способствует повышению качества этих реализаций. Наибольший эффект даст изложение на формальном языке первичных спецификаций протоколов, публикуемых в документах RFC. В статье предлагается формальный язык описания криптографических протоколов, нацеленный на более полное, по сравнению с существующими языками первичных спецификаций, выполнение следующих требований: лаконичность, декларативность, выразительность, однозначность, исполнимость, возможность автоматического извлечения из спецификаций наборов тестов высокого качества. Основа языка – вычислитель, специально разработанный для предметной области криптографических протоколов, – т. н. C2-машина. В статье при изложении принципов построения и функционирования C2-машины используется метод последовательного уточнения модели: сначала вводится недетерминированный автомат, на котором в наиболее общем виде задается операционный контракт протокола, затем вводится машина C0, на которой демонстрируется предлагаемый подход к специфицированию недетерминизма протоколов, и далее – машина C1, на которой в формализованном виде задается семантика базовых инструкций C2-машины. Статья иллюстрирована примерами для протокола TLS.

Ключевые слова: криптографические протоколы; формальные спецификации; C2-машина

Для цитирования: Прокопьев С.Е. Формальный язык первичных спецификаций криптографических протоколов. Труды ИСП РАН, том 33, вып. 5, 2021 г., стр. 117-136. DOI: 10.15514/ISPRAS–2021–33(5)–7

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

A formal language for primary specifications of the cryptographic protocols

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
JSRPC «NPK Kryptonite»,
4, Shluzovaya emb., Moscow, 115114, Russia

Abstract. Use of the formal methods in the development of the protocol implementations improves the quality of these implementations. The greatest benefit would result from the formalizing of the primary specifications usually contained in the RFC documents. This paper proposes a formal language for primary specifications of the cryptographic protocols, which aims at fulfilling (in a higher degree than in the existing approaches) the conditions required from primary specifications: they have to be concise, declarative, expressive, unambiguous

and executable; in addition, the tools supporting the language have to provide the possibility of automatic deriving of the high quality test suites from the specifications. The proposed language is based on a machine (dubbed the C2-machine) specifically designed for the domain of the cryptographic protocols. Protocol specification is defined as a program of the C2-machine. This program consists of two parts: the definition of the protocol packets and the definition of the non-deterministic behavior of the protocol parties. One execution of the program simulates one run of the protocol. All the traces, which can be potentially produced by such execution, by definition, comprise all conformant traces of the protocol; in other words, the program of the C2-machine defines the operational contract of the protocol. In the paper, to make the design and operational principles of the C2-machine easier to understand, two abstractions of the C2-machine are presented: C0-machine and C1-machine. C0-machine is used to explain the approach taken in expressing non-determinism of the protocols. The abstraction level of the C1-machine (which is a refinement of C0-machine) is enough to define the semantics of the basic C2-machine instructions. To enhance the readability of the programs and to reach the level of declarativeness and conciseness of the formalized notations used in some of conventional RFCs (e.g. TLS RFCs), C2-machine implements some syntactic tricks on top of the basic instructions. To use C2-specifications in the black-box testing, the special form of the C2-machine (C2-machine with excluded role) is presented. Throughout the paper the proposed concepts are illustrated by examples from the TLS protocol.

Keywords: cryptographic protocols; formal specifications; C2-machine

For citation: Prokopev S.E. A formal language for primary specifications of the cryptographic protocols. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 117-136 (in Russian). DOI: 10.15514/ISPRAS-2021-33(5)-7

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

1. Введение

Перед тем, как пользователи получат в свое распоряжение реализацию криптографического протокола, специалисты проводят ряд обязательных исследований, включая анализ криптографических свойств протокола в разных моделях (в стандартной модели, модели Долева-Яо и др.) и анализ корректности программной реализации протокола с использованием статических и динамических методов.

Формальные методы позволяют повысить качество исследований, для некоторых из них позволяют задать метрики оценки достигнутого уровня качества, а также упрощают задачу верификации полученных результатов. При этом использование формальных методов не всегда означает удорожание исследований, так как удачная формализация может обеспечить возможность их автоматизации и, как следствие, снизить трудоемкость их проведения.

На текущий момент формальные методы используются в разработке программных реализаций [2, 3], в тестировании [4, 1], в анализе криптографических свойств протоколов [5, 6].

Во всех перечисленных случаях по первичным описаниям (спецификациям) протоколов, изложенным на естественном языке (например, в документах RFC), вручную разрабатываются вторичные формальные спецификации (к которым мы относим и исходные тексты реализаций).

Можно попытаться уменьшить объем ручного труда, обеспечив автоматическую трансляцию вторичных спецификаций из одной нотации в другую. Например, спецификации в нотации ESPI (пригодной для анализа инструментом ProVerif) можно пытаться конвертировать в исходные тексты на языке Java (см. [7]), или наоборот, исходные тексты на языке C пытаться конвертировать в спецификации для ProVerif (см. [8]).

Хотя вторичные спецификации, полученные автоматически из других вторичных спецификаций, могут решать свою задачу менее эффективно, чем написанные вручную, на практике фактор минимальной трудоемкости их получения (усиливаемый постоянными корректировками первичных спецификаций в ходе жизненного цикла разработки протоколов), этот недостаток может перевесить. Как отмечено в [9], *«...a developing*

environment where several tools can be used for different purposes on the base of the same specification model can be much more convenient than having different tools, even if more efficient, working on input models with their own different languages».

Помимо автоматической трансляции спецификаций из одной нотации в другую, есть и путь встраивания методов анализа, работающих в своих моделях со своей специфической нотацией, в другие модели. Например, в работе [10] метод верификации протоколов в модели Долева-Яо на основе дизъюнктов Хорна встроен во фреймворк Java Cryptography Architecture.

Важнейшим вопросом является то, насколько точно вторичная спецификация отражает требования, предъявленные к протоколу в первичной спецификации, то есть насколько она полна и корректна. Возникает желание уменьшить «семантический разрыв» между первичной неформальной спецификацией и вторичной формальной спецификацией путем повышения уровня формализации первичной спецификации. В первичных спецификациях многих протоколов такие попытки делаются.

Например, формализованная запись форматов сообщений в RFC 8446 (TLS версии 1.3) делает спецификацию протокола в этой части более строгой (уменьшает неоднозначность). Однако этот формализм не исполним, то есть он не поддержан инструментом трансляции в некоторый «полезный» язык, например, в промышленный язык программирования, на котором реализуется протокол, или в нотацию, принятую в некоторой системе тестирования. Пример исполнимого формализма – первичная спецификация протокола Signal, содержащая вставки кода на языке Python. Эти вставки можно без изменений перенести в исходные тексты реализации, при этом они достаточно хорошо читаемы ввиду лаконичности синтаксиса этого языка. Другой пример: в работе [11] для первичных спецификаций предложена формальная нотация для описания управляющей логики протокола, поддержанная инструментом трансляции в код на C++ и Java.

Сформулируем следующие требования к «идеальному» языку первичных спецификаций криптографических протоколов:

1) хорошая читаемость (лаконичность и декларативность);
2) высокая выразительность: часть спецификации, выраженная на нем, должна охватывать на порядок больше требований к протоколу, чем часть, выраженная на естественном языке;
3) однозначность;
4) возможность генерации из спецификации произвольного числа контрольных примеров с параметрами, интересующими исследователя;
5) возможность автоматической трансляции спецификаций в реализации, пригодные для встречной работы с промышленными реализациями протоколов;
6) возможность автоматического извлечения из спецификаций наборов тестов (как тестов на конформность, так и фаззинг-тестов), существенно превышающих по качеству наборы, которые можно сформировать вручную. При этом должна выдаваться адекватная оценка качества проведенного тестирования;
7) возможность автоматической трансляции спецификаций в языки специализированных инструментов анализа криптографических свойств протоколов (ProVerif, Tamarin, CryptoVerif и др.).

В настоящей работе представлена формальная нотация, нацеленная на более полное, чем в существующих подходах, выполнение предъявленных выше требований. Нотация основана на вычислителе (так называемой *C2-машине),* специально разработанном для предметной области промышленных криптографических протоколов, таких как TLS, IKEv2 и др.

C2-машина исполняет программы, содержащие *гардированные команды* (guarded command programs, [12]). В таких программах существуют точки выполнения, в которых дальнейшее

вычисление задается наборами пар *guard/update*, где *guard* – предикат (гард), задающий условия, при которых действие *update* может быть выполнено. В случае, когда одновременно истинны несколько гардов, недетерминированно выбирается один из них. Детали исполнения программ C2-машины, касающиеся «гардирования», можно найти в разд. 2 и 3 настоящей статьи.

Тексты программ для C2-машины и предлагается использовать в качестве первичных спецификаций протоколов.

Каждый запуск спецификации на C2-машине дает трассу сообщений протокола. Так же, как в подходе NCT [13], протокол определяется множеством своих конформных трасс, которое, в свою очередь, задается как множество всех трасс сообщений, которые потенциально может произвести машина, выполняя спецификацию. В целом идея использования специализированных машин при моделировании криптографическими протоколов, не нова (примеры: машины CrAM [14] и CVM [8]).

В настоящей статье изложение принципов построения C2-машины проводится методом последовательного уточнения модели (model refinement). Сначала в разд. 2 представлен простой недетерминированный автомат *A* (первое приближение C2-машины), в контексте которого рассматривается понятие *операционного контракта* протокола. В разд. 3 рассмотрен предлагаемый подход к захвату недетерминизма протоколов с помощью *последовательностей* и *гардирования* значений полей сообщений протоколов и представлено второе приближение C2-машины – машина *C0*. В разд. 4 рассмотрены *инструкции*, *процедуры*, *выражения* и *программы* C2-машины и представлено ее третье приближение – машина *C1*. Разд. 5 посвящен способам представления процедур C2-машины в лаконичном и декларативном виде, разд. 6 – тестированию реализаций протоколов в режиме «черного ящика» с помощью так называемой *C2-машины с исключенной ролью*.

2. Операционный контракт протокола

Рассмотрим недетерминированный автомат *A*, определение которого адаптировано к предметной области протоколов передачи сообщений:

A = (State, Role, Packet, ProtocolSpec, calcpacket, s0), где

State – множество состояний автомата, *Role* – множество ролей протокола, *Packet* – множество пакетов протокола, *ProtocolSpec* – множество спецификаций протоколов, *s0 :: State* – начальное состояние автомата,

calcpacket :: (ProtocolSpec, State, Role, Role) →: (State, Packet) ∪{stop, fail} – недетерминированная функция вычисления пакета протокола и перехода в следующее состояние; *stop, fail* – особые значения, доопределяющие функцию *calcpacket*.

Примечание: двойное двоеточие :: обозначает принадлежность элемента к множеству; запись (X,...Y) – декартово произведение множеств X,...Y; стрелочка с двоеточием "→:" – недетерминированную функцию.

Равенство *calcpacket (protSpec, si, src, dst) = (sj, pkt)* интерпретируется следующим образом: автомат, работая по спецификации *protSpec*, находясь в состоянии *si* и получая в качестве входного символа пару *(src, dst)*, переходит в состояние *sj* и выдает выходной символ *pkt*.

Алгоритм работы автомата *A* (для некоторого значения *protSpec*):

0. в момент запуска автомат находится в состоянии *s0*;
1. из множества *(Role,Role)* недетерминированно выбирается *(src,dst)*;
2. если *calcpacket (protSpec, si, src, dst) :: {stop, fail}*, то конец работы;
3. если *calcpacket (protSpec, si, src, dst) = (sj, pkt)*, то возврат на шаг 1.

Трассой запуска автомата A называется список троек *(src,dst,pkt)*, снимаемых внешним наблюдателем на шаге 3. В общем случае автомат может не останавливаться и производить бесконечные трассы.

Ввиду недетерминизма шага 1 (*недетерминизма чередования*, interleaving choice) и недетерминизма функции *calcpacket* (*недетерминизма спецификации*, specification choice), разные запуски автомата, как правило, будут производить разные трассы. Таким образом, автомат задает язык, словами которого являются все трассы, которые он потенциально может произвести (трассы, завершившиеся *fail*, в язык не включаются).

Если язык, производимый автоматом *A*, совпадает со множеством конформных трасс моделируемого протокола, то говорят (см., например, [15]), что этот автомат задает *операционный контракт* протокола.

Автомат *A* является первым приближением С2-машины.

3. Специфицирование недетерминизма протоколов

3.1 Последовательности сессий, пакетов и фрагментов пакетов

Ключевым понятием предлагаемого подхода к моделированию протоколов являются *последовательности*. Предполагается, что в любом протоколе всегда присутствуют две последовательности: *_sessions* – последовательность сессий протокола (это последовательность всех пакетов из трассы автомата *A* из раздела 1) и *_packets* – последовательность пакетов, принадлежащих одной сессии протокола.

Обычно последовательности образуют иерархию. Например, элемент последовательности *_sessions* содержит элементы последовательности *_packets*. В каждом пакете протокола может быть своя иерархия последовательностей фрагментов.

Пример: иерархия (неполная) последовательностей протокола TLS:

_sessions → _packets
_packets → _records
_records → _handshMess
_handshMess → _clientSuites
_handshMess → _clientHelloExts
_clientHelloExts → _clntSuppGroups
_clientHelloExts → _clntPointFormats
_clientHelloExts → _clntSigAlgs
_clientHelloExts → _clntSuppVersions
_clientHelloExts → _clntKeyShares
_handshMess → _serverHelloExts
_handshMess → _certRequestExts
_certRequestExts → _certRequestSigAlgs
_handshMess → _certificates
_certificates → _certificateExts

Каждому фрагменту пакета, включая пакет целиком, можно сопоставить список пар («идентификатор последовательности», «номер элемента последовательности»), отражающий место этого фрагмента в иерархии последовательностей протокола. Будем называть этот список *sequence-индексом* (или просто, *индексом*) фрагмента пакета.

Пример: в протоколе TLS фрагмент пакета, содержащий четвертый по порядку элемент расширения *ServerHelloExtensions* сообщения *ServerHello* может иметь следующий sequence-индекс:

[(_sessions,1), (_packets,2), (_records,1), (_handshMess,1), (_serverHelloExts,4)].

Разные части одной последовательности могут быть «раскиданы» по разным пакетам и по разным местам пакетов. Например, в протоколе TLS версии 1.3 в первых двух пакетах первой сессии могут быть фрагменты со следующими индексами:

пакет 1:

[(_sessions, 1), (_packets, 1), (_records, 1), (_handshMess, 1)] – фрагмент пакета, в котором размещено сообщение *ClientHello* подпротокола TLSHandshakeMessages подпротокола TLSRecords;

пакет 2:

[(_sessions, 1), (_packets, 2), (_records, 1), (_handshMess, 1)] – фрагмент пакета, в котором размещено сообщение *ServerHello* подпротокола TLSHandshakeMessages подпротокола TLSRecords;

[(_sessions, 1), (_packets, 2), (_records, 2)] – фрагмент пакета, в котором размещено сообщение *ChangeCipherSpec* подпротокола TLSRecords;

[(_sessions, 1), (_packets, 2), (_records, 3), (_handshMess, 1)] – фрагмент пакета, в котором размещено сообщение *EncryptedExtensions* подпротокола TLSHandshakeMessages подпротокола TLSRecords;

[(_sessions, 1), (_packets, 2), (_records, 3), (_handshMess, 2)] – -//- CertificateRequest -//-;
[(_sessions, 1), (_packets, 2), (_records, 3), (_handshMess,3)] – -//-Certificate -//-;
[(_sessions, 1), (_packets, 2), (_records, 3), (_handshMess, 4)] – -//- CertificateVerify -//-;
[(_sessions, 1), (_packets, 2), (_records, 3), (_handshMess, 5)] – -//- Finished -//-.

В данном примере последовательность *_handshMess* разбита на три части: первая часть (состоящая из одного элемента) размещена в первом пакете, вторая (состоящая из одного элемента) и третья (состоящая из пяти элементов) – во втором.

3.2 Второе приближение С2-машины – машина *C0*

Машина *C0* – это абстракция С2-машины, достаточная для формализованного изложения предлагаемого способа специфицирования недетерминизма протоколов.

Ниже используется псевдокод, содержащий следующие интуитивно понятные конструкции:

if ... then (do) ... (else (do) ...) *return ...* *... && ...*
case ... do ... case ... do ... *let ... = (do) ...*

Используются следующие операции со списками и другие соглашения:

[] пустой список,
a:as добавление в начало списка *as* элемента *a,*
as ++ bs конкатенация списков *as* и *bs,*
null as если список *as* пустой, то истина, иначе ложь,
length as длина списка *as,*
take n as список из первых *n* элементов списка *as,*
drop n as список *as* без первых *n* элементов,
reverse as перестановка элементов списка *as* в обратном порядке,
init as список *as* без последнего элемента,
last as последний элемент списка *as,*
fst ab левый элемент пары *ab,*
snd ab правый элемент пары *ab,*
empty пустая частичная функция,
Maybe A множество *{Just x | x :: A}* ∪*{Nothing},*
A \ B разность множеств *A* и *B,*
a :← A элемент *a* недетерминированно выбран из множества *A,*
a ← f *a* принимает значение, возвращаемое функцией *f,*
v := x обновление переменной *v* состояния машины значением *x,*
[X] множество списков конечной длины, составленных из элементов множества *X*, включая пустой список,
Map X Y множество всех частичных функций из *X* в *Y,*
Dom f область определения частичной функции *f,*
insert a b f добавление в частичную функцию *f* пары *(a, b)* (с замещением пары вида *(a, *)*, если таковая в *f* была),

f a b c f :: (A, B, C) → Y; параметры функций (в данном случае параметрами функции *f* являются *a :: A, b :: B и c :: C*) обычно будут записываться без круглых скобок и запятых – через пробелы,

N1 натуральные числа, начиная с единицы,

Bool *{True, False}*.

C0 = (State, Role, SeqID, FieldName, FieldValue, protSpec).

Состояние машины *C0* – это три переменных, значениями которых являются частичные функции:

State = {packetRoles :: Map (N1, N1) (Role, Role),
fieldValues :: Map (FieldName, SeqIndex) FieldValue,
sequenceInfo :: Map SeqID (Map SeqIndex N1)},

где *SeqID* – множество идентификаторов последовательностей, *FieldName* и *FieldValue* – множества имен и значений полей пакетов протокола, *SeqIndex = [(SeqID, N1)]* – множество всех sequence-индексов.

Для *seqid :: Dom sequenceInfo* частичная функция *sequenceInfo seqid* состоит из множества пар *(index, num)*, где *index :: SeqIndex* – это индекс фрагмента пакета, внутри которого создана часть последовательности с идентификатором *seqid*, а *num :: N1* – число элементов в этой части.

Спецификация протокола разбита на два компонента:

protSpec = {packetDef :: PacketDef, controlDef :: ControlDef}, где

PacketDef = (State, SeqID, SeqIndex) → Fragment – функция вычисления фрагментов пакетов протокола,

Fragment = Field FieldName | Sequence SeqID | B [Fragment] – (заданное в виде алгебраического типа данных) множество фрагментов пакетов (то есть фрагмент – это либо именованное поле, либо последовательность фрагментов, либо конкатенация фрагментов),

ControlDef = SeqID → ([(GuardFun, [Assignment])], ReadyFun) – функция, для каждого индекса задающая, во-первых, варианты возможных значений именованных полей фрагмента пакета с этим индексом вместе с *гардами* этих вариантов (то есть условиями, при которых они допустимы):

Assignment = (FieldName, [FieldValue]),

GuardFun = (State, SeqIndex) → Bool;

во-вторых, условие, при котором возможен выход из последовательности:

ReadyFun = (State, SeqIndex) → Bool.

Недетерминизм спецификации (specification choice) «рассредоточен» по трем местам: 1) выбор «продолжать или не продолжать последовательность» – при условии, что выход возможен, 2) выбор гарда *guard :: GuardFun* из числа тех, что обратились в истину, 3) выбор значений именованных полей из числа возможных вариантов, задаваемых списком *assignments :: [Assignment]*, стоящим в паре с этим *guard*. Ниже в псевдокоде Листинга 1 эти места выделены полужирным шрифтом.

```
runC0 = run_session 1
run_session s = do
    do_sequence _packets 1 [(_sessions,s)]
    b :← Bool
    if b then run_session (s+1) else stop

do_sequence seqid i index = do
  b ← proceed_or_drop seqid i index
  if not b then return else do
      if seqid == _packets then do
          (src,dst) :← (Role,Role)
          packetRoles := insert (s,i) (src,dst) packetRoles
      new_seqelement seqid i index
      do_sequence seqid (i+1) index

proceed_or_drop seqid i index = do
    let checkready = snd (controlDef seqid)
    let b = checkready (index ++ [(seqid,i)])
    if not b then return True else do
        c :← Bool
        if c then return True
             else return False

new_seqelement seqid i index = do
    let index' = index ++ [(seqid,i)]
    let guardedassignments = fst (controlDef seqid)
    truelists ← select_truepairs guardedassignments []
    assignment :← truelists
    assign_fieldvalues assignment
    let frag = packetDef seqid index'
    do_fragment frag
    sequenceInfo := insert (seqid,index) i sequenceInfo

select_truepairs xs acc =
  if xs == [] then return acc else do
    let (guard,assignments):xs' = xs
    if guard index'
        then select_truepairs xs' (acc ++ [assignments])
        else select_truepairs xs' acc

assign_fieldvalues xs =
  if xs == [] then return else do
      let (fieldname,variants):xs' = xs
      vnt :← variants
      fieldValues := insert (fieldname,index') vnt fieldValues
      assign_fieldvalues xs'

do_fragment frag = do
  case frag == Field fieldname do
      if (fieldname,index') :: Dom fieldValues
          then return else fail
  case frag == Sequence seqid' do
      do_sequence seqid' 1 index'
  case frag == B frags do
      do_concat frags

do_concat xs =
  if xs == [] then return else do
      let frag':xs' = xs
      do_fragment frag'
      do_concat_of xs'
```

Листинг 1. Алгоритм работы машины C0
Listing 1. Algorithm of the machine C0

4. Дальнейшие детали построения C2-машины

4.1 Инструкции и процедуры

Процедура C2-машины – это последовательность *инструкций*, исполняемых на стеке: аргументы инструкций берутся из стека и результаты выполнения инструкций кладутся в стек. Элементами стека являются байт-строки переменной конечной длины.

При исполнении инструкций используется и обновляется внутреннее состояние С2-машины. Кроме того, используется информация, снимаемая со встроенных ДСЧ и часов реального времени.

Результирующим набором байт-строковых значений инструкции (процедуры) называется содержимое стека после завершения исполнения инструкции (процедуры). Если этот набор состоит из одного элемента, то используется термин *байт-строковое значение* инструкции (процедуры).

Примеры инструкций С2-машины (с неформальным и упрощенным пояснением семантики):

C [0x15, 0x21] –положить в стек байт-строку *[0x15, 0x21]*;

V_ "ChosenSuite" [(_sessions, 1)] – если машине известен фрагмент данных с именем *"ChosenSuite"*, вычисленный в первой сессии протокола, то положить в стек этот фрагмент данных;

Random (Plain 4) Clnt (V_ "Key" [(_sessions, 3)]) – если сейчас идет третья сессия протокола и отправитель текущего формируемого пакета – роль *Clnt*, и если фрагмент данных с именем *"Key"* в третьей сессии еще не был вычислен, то снять с ДСЧ четыре случайных байта и положить их в стек, при этом считать эти байты значением фрагмента данных *"Key"* из третьей сессии;

*Length_* – взять из стека крайний элемент и положить обратно длину этого элемента в байтах (в формате big endian без лидирующих нулевых байтов);

I2Bytes_ 32 – дополнить крайний элемент стека слева лидирующими нулевыми байтами до длины 32;

*Hash_* – взять из стека два крайних элемента (идентификатор алгоритма хэширования и текст), вычислить и положить в стек хэш;

Encrypt_ 4 – взять из стека четыре крайних элемента (идентификатор алгоритма шифрования, открытый текст, ключ и инициализирующий вектор), вычислить на этих параметрах шифртекст и положить его в стек;

B_ 3 – взять из стека три крайних элемента и положить обратно их конкатенацию;

*Not_* – взять из стека крайний элемент, если он равен *[0x01]*, то положить в стек *[0x00]*, иначе положить *[0x01]*;

And_ 2 – взять из стека два крайних элемента, если они оба равны [0x01], то положить в стек *[0x01]*, иначе положить *[0x00]*;

Src Serv – если отправитель текущего пакета – роль *Serv*, то положить в стек *[0x01]*, иначе положить *[0x00]*;

Equals_ 2 – взять из стека два крайних значения и сравнить их байт-строковые значения; если равны, то положить в стек *[0x01]*, иначе - положить *[0x00]*;

P instrs – выполнить процедуру instrs.

Машине не всегда хватает информации, хранящейся во внутреннем состоянии, чтобы вычислить байт-строковое значение инструкции. Такое событие является нормальным, и существует специальная инструкция, заданная на этом событии:

Known instr – если байт-строковое значение инструкции *instr* может быть вычислено, то положить в стек *[0x01]*, иначе положить *[0x00]*.

Примечание: запись [a,b...], где a, b и т.д. – элементы некоторого множества, обозначает список из этих элементов.

Язык инструкций С2-машины не поддерживает циклы или рекурсии. Есть поддержка ветвлений посредством инструкции *Select instr cases*.

Пример:

Select (V_ "ChosenSuite" [(_sessions,1)]) [([0x13,0x01], ins1), ([0x13,0x02], ins2), ...] – если байт-строковое значение инструкции *V_ "ChosenSuite" [(_sessions,1)]* равно *[0x13,0x01]*, то выполняется инструкция *ins1*, если *[0x13,0x02]*, то *ins2* и т.д.

Последовательности фрагментов внутри пакетов создаются с помощью инструкции *Sequence seqid seqelem*, где *seqid* – идентификатор последовательности, а *seqelem* – процедура вычисления очередного элемента последовательности.

Ввиду наличия таких «макроинструкций», как *P instrs*, *Select instr cases* и др., термины «инструкция» и «процедура» С2-машины являются взаимозаменяемыми.

4.2 Выражения С2-машины

Контекст выполнения процедуры – это sequence-индекс фрагмента пакета, вычисляемого этой процедурой. В зависимости от контекста, одна и та же процедура С2-машины может иметь разные байт-строковые значения.

Выражения – это формулы, конструируемые С2-машиной в ходе выполнения процедур, с помощью которых она запоминает результаты вычислений с учетом контекста выполнения.

Далее *Word8* – множество целых чисел в диапазоне *0..255*, *String* – множество символьных строк.

Выражения и вычисленные байт-строковые значения, сопоставленные им, хранятся в следующих переменных состояния С2-машины:

expressions :: Map ExprDesc (CoreInstr, [ExprDesc]), где *CoreInstr* – множество базовых (не зависящих от контекста) инструкций С2-машины, *ExprDesc* – множество дескрипторов выражений;

values :: Map ExprDesc [Word8].

Таким образом, выражение – это пара *(coreinstr, ds)*, где *coreinstr* – базовая инструкция, а *ds* – список дескрипторов выражений параметров.

Примеры выражений:

(V_ "ChosenSuite" [(_sessions, 1), (_packets, 2)], [])

(C [0x15, 0x21], [])

(Encrypt_ 4, [34, 53, 57, 60])

Пример инструкции машины, зависящей от контекста: *Vi str*, где *str :: String*. Встретив инструкцию *Vi str* в процедуре, выполняемой в контексте *ix,* машина заменяет *Vi str* на *V_ str ix*.

В С2-машину в неизменном виде переносится хранилище *sequenceInfo*, заданное нами выше для машины *C0*.

Механизм построения выражений и сохранения их вместе с байт-строковыми значениями обеспечивает С2-машине следующие свойства:

1) машина никогда не дублирует вычисления: сначала она строит выражение, соответствующее процедуре, потом проверяет, известно ли уже байт-строковое значение этого выражения; если да, то выполнение процедуры на этом заканчивается;
2) одинаковые процедуры в одинаковом контексте выполнения дают один и тот же результат, несмотря на случайность генерируемых ключей;
3) появляется возможность иметь следующие важные инструкции (с учетом информации о последовательностях, накопленной в *sequenceInfo*):

   LastIn seqid instr filtercondition – найти последний по порядку индекс, оканчивающийся (в левой части пары) на *seqid*, такой что в этом контексте инструкция *filtercondition*, выступающая в роли предиката, истинна, и вычислить в этом контексте инструкцию *instr (примечание: если байт-строковое значение инструкции равно [0x01], то считается, что эта инструкция обращается в истину, иначе – в ложь)*;

AllIn seqid instr filtercondition – записать по порядку в список все индексы, оканчивающиеся на *seqid*, такие что в их контексте инструкция-предикат *filtercondition* истинна, и вычислить в контексте каждого из них инструкцию *instr*; результирующие байт-строковые значения положить в стек согласно порядку отобранных sequence-индексов.

В С2-машине в отдельной переменной состояния *curstep :: (N1, N1)* хранится пара *(s, p)*, где *s* – номер текущей сессии, *p* – номер текущего пакета. От значения *curstep* зависит результат выполнения инструкций *InCurrSess* и *InCurrPack* (ниже считаем, что начальный отрезок текущего контекста имеет вид *[(_sessions, s'), (_packets, p'), …]*):

InCurrSess если *s' == s*, то результат *[0x01]*, иначе *[0x00*]

InCurrPack если *p' == p*, то результат *[0x01]*, иначе *[0x00*]

4.3 Программа С2-машины (спецификация протокола)

Двухкомпонентная спецификация, определенная для машины *C0*, в С2-машине детализируется следующим образом:

packetDef :: Instr;

controlDef :: Map SeqID [((Instr, [(Instr, Instr)]), Instr)].

Здесь *Instr* – это множество всех инструкций С2-машины.

В структуре (*(guard, assignments), ready) = controlDef seqid* компоненты *guard* и *ready* – это инструкции-предикаты. Элемент *assignments* – это список, состоящий из пар вида *(instr, values)*, где *instr* – инструкция с (пока что) неизвестным байт-строковым значением (обычно это инструкция *Vi str*), *values* – инструкция, результирующий набор байт-строковых значений которой составляет множество возможных значений для *instr*.

Пример: элемент списка *assignments*, определенный как

```
(P [LastIn _handshMess (P [Vi "HandshType", C [0x01], Equals_ 2]) (Src Clnt),
    Known (LastIn _handshMess (P [Vi "HandshType", C [0x02], Equals_ 2]) (Src Serv)),
    Not_,
    And_ 2],
 [(Vi "HandshType",  C [0x02]),
  (Vi "ChosenSuite",  P [C [0x13, 0x01], C [0x13, 0x02]])]),
```

означает следующее: если в последнем (на текущий момент) элементе последовательности *_handshMess* из пакета, отправленного ролью *Clnt*, фрагмент данных *"HandshType"* имел значение *[0x01]*, и ранее в *_handshMess* не существовало элемента из пакета, отправленного ролью *Serv*, в котором фрагмент данных *"HandshType"* имел значение *[0x02]*, то в текущем формируемом элементе последовательности *_handshMess* присвоить фрагменту данных *"HandshType"* значение *[0x02]*, а фрагменту данных *"ChosenSuite"* присвоить значение, недетерминированным образом выбранное из двух вариантов – *[0x13, 0x01]* и *[0x13, 0x02]*.

Как было показано на уровне абстракции машины *C0*, С2-машина совершает недетерминированный выбор (в части specification choice) тогда и только тогда, когда наступает момент формирования очередного элемента последовательности. Резюмируем поведение машины в этот момент для инструкции *Sequence seqid seqelem*:

1) перед формированием очередного элемента последовательности последовательно перебираются все пары *(guard, assignments)*, относящиеся к *seqid*, и в этих парах вычисляются гарды; далее из числа тех гардов, что обратились в истину, недетерминированно выбирается один;
2) обрабатывается список *assignments*, стоящий в паре с выбранным гардом (здесь также совершается недетерминированный выбор значений); если спецификация корректна, то после обработки этого списка процедура *seqelem* становится детерминированно

вычислимой: всем нужным инструкциям, не имевшим байт-строковых значений, эти значения назначены;

3) выполняется процедура *seqelem*, вычисляющая очередной элемент последовательности *seqid*.

4.4 Третье приближение С2-машины – машина *C1*

Особенности функционирования С2-машины, рассмотренные в предыдущих параграфах раздела, опишем в формализованном виде в виде машины *C1*.

C1 = (State, Instr, N1, Word8, String, Bool, SeqID, argsnum, calcfun, protSpec)

```
State = {curstep        :: (N1,N1),
         packetRoles    :: Map (N1,N1) (Role, Role),
         expressions    :: Map Desc (CoreInstr, [ExprDesc]),
         values         :: Map Desc BS,
         sequenceInfo   :: Map SeqID (Map SeqIndex N1)}
```

SeqIndex = [(SeqID, N1)]

```
Instr ::= CoreInstr | Vi String | V1 String | V2 String | Known Instr | P Instrs |
          Src Role | Dst Role | InCurrSess | InCurrPack |
          Sequence SeqID Instr | Select Instr [([Word8], Instr)] | B [Instr]
CoreInstr ::= C [Word8] | V_ String SeqIndex | Encrypt_ N1 | Hash_ | Length_ |
              B_ N1 | Known_ SeqIndex | Equals_ N1 | Not_ | And_ N1 | ...
```

SeqID ::= _sessions | _packets | _records | ...

argsnum :: CoreInstr → N1

calcfun :: (CoreInstr, [[Word8]]) → [Word8]

protSpec :: (Instr, Map SeqID [(Instr, [(Instr, Instr)])])

Определения вспомогательных функций процедуры функционирования машины *C1*:

dochoices :: (SeqID, SeqIndex) → Bool

symexecute :: ([Instr], SeqIndex, [ExprDesc]) → [ExprDesc]

calcvalue :: ExprDesc → [Word8]

Реализация вспомогательных функций процедуры функционирования машины *C1* и Алгоритм работы машины *C1* показаны в Листингах 2 и 3 соответственно.

```
dochoices seqid ix = do
  let filtguards ac xs =
    if null xs then return ac
      else do
        let (guard,assignments):xs' = xs
        [d] ← symexecute [B [guard]] ix []
        bs ← calcvalue d
        if bs == [0x01]
          then filtguards (ac ++ [assignments]) xs'
          else filtguards ac xs'
  let doassignments xs =
    if null xs then do
          let info = if seqid :: Dom sequenceInfo
            then sequenceInfo(seqid) else empty
          let info' = insert (init ix) (snd (last ix)) info
          sequenceInfo := insert seqid info' sequenceInfo
          return True
      else do
        let (ins,values):xs' = xs
        [d] ← symexecute [B [ins]] ix []
```

```
        if d :: Dom values then fail
          else do
            ds ← symexecute [values] ix []
            d' :← ds
            mbs ← calcvalue d'
            if mbs == Just bs
              then do
                  values := insert d bs values
                  doassignments xs'
              else fail
  let controlDef = snd protSpec
  updlist ← filtguards [] (controlDef(seqid))
  if null updlist then return False
    else do assignments' :← updlist
                doassignments assignments'
findcase bs cases =
  if null cases then fail else do
    let (cs,ins1):cases' = cases
    if cs == bs then symexecute [B [ins1]] ix []
                else findcase cases'
doexpress coreins ix stk = do
  let push d stk = return (d:stk)
  let pops n stk =
      if length stk < n then fail
                        else return (take n stk, drop n stk)
  (ds, stk') ← pops (argsnum coreins) stk
  let expr = (coreins, reverse ds)
  if d :: Dom expressions && expressions(d) == expr
    then push d stk'
    else do let e :: DescExpr \ Dom expressions
                expressions := insert e expr expressions
                push e stk'
symexecute instrs ix stk = do
  if null instrs then return stk else do
    let chkbool expr = if expr then C [0x01] else C [0x00]
    let [(_sessions,s),(_packets,p)] = take 2 ix
    let ins:insss = instrs
    case ins :: CoreInstr do
        stk' ← doexpress ins ix stk
        symexecute insss ix stk'
    case ins == P inss' do (symexecute (inss' ++ inss) ix stk)
    case ins == Src r do chkbool (fst (packetRoles(s,p)) == r)
    case ins == Dst r do chkbool (snd (packetRoles(s,p)) == r)
    case ins == InCurrSess do chkbool (fst curstep == s)
    case ins == InCurrPack do chkbool (snd curstep == p)
    case ins == B inss' do
        ds ← symexecute inss' ix []
        d ← symexecute [B_ (length ds)] ix []
        symexecute inss ix (d:ds ++ stk)
    case ins == Vi str do (symexecute [V_ str ix] ix stk)
    case ins == V1 str do (symexecute [V_ str (take 1 ix)] ix stk)
    case ins == V2 str do (symexecute [V_ str (take 2 ix)] ix stk)
    case ins == Known ins' do
        symexecute [B [ins'], Known_ ix] ix stk
    case ins == Select ins' cases do
        [d] ← symexecute [B [ins']] ix []
        mbs ← calcvalue d
```



```
        case mbs == Just bs do [e] ← findcase bs cases
                                          symexecute inss ix (e:stk)
        case mbs == Nothing do fail
    case ins == Sequence seqid ins1 do
      let addelem i stk' = do
        let ix' = ix ++ [(seqid,i)]
        b ← dochoices seqid ix'
        if b then do [d] ← symexecute [B [ins1]] ix'
                     addelem (i+1) (d:stk')
             else symexecute inss ix stk
      addelem 1 []
calcvalue d =
  let mapMcalcdesc zs = do
      let calc ac xs =
        if null xs then return (Just ac)
          else do let x:xs' == xs
                  if calcdesc x == Just y
                      then calc (ac ++ [y]) xs'
                      else return Nothing
      calc [] zs
  if d :: Dom values then return (Just (values(d)))
    else do let (coreins,ds) = calcexpress (expressions(d))
                if mapMcalcdesc ds == Just bss
                    then do let bs = calcfun coreins bss
                                values := insert d bs values
                                return (Just bs)
                    else return Nothing
```

Листинг 2. Реализация вспомогательных функций процедуры функционирования машины C1
Listing 2. Implementation of the auxiliary functions of the procedure for the functioning of the C1 machine

```
      runC1 = do
                let packetDef = fst protSpec
              let run (s, p) = do
                       let chksess = do b' :← Bool
                                        if b' then run (s, p+1) else run (s+1, p)
                    curstep := (s, p)
                    (src, dst) :← (Role, Role)
                    packetRoles := insert (s, p) (src, dst) packetRoles
                    let ix = [(_sessions, s), (_packets, p)]
                    b ← dochoices _packets ix
                    if b then do [d] ← symexecute [B [packetDef]] ix
                                 mbs ← calcvalue d
                                 if mbs == Just bs then chksess else fail
                         else run (s+1, p)
              packetRoles := empty
              run (1,1)
```

Листинг 3. Алгоритм работы машины C1
Listing 3. Algorithm of the C1 machine

Последовательность троек *(src, dst, bs)*, снимаемых внешним наблюдателем на каждой итерации функции *run*, называется *трассой запуска машины C1*.

5. Декларативное представление процедур C2-машины

Так как C2-машина является стековой машиной, процедуры вычисления формул представляют собой запись этих формул в обратной «польской» нотации. Например, формулу *EncryptAES128 (plaintext ||HashSHA256 (plaintext), key, iv)*, где *plaintext, key, iv –*

некоторые байт-строки, можно вычислить следующей процедурой С2-машины: *[C iv, C key, C plaintext, C sha256oid, Hash_, C plaintext, B_ 2, C aes128oid, Encrypt_ 4]*, где *sha256oid*, *aes128oid* – это байт-строки-идентификаторы алгоритмов.

Если добавить в язык инструкций С2-машины «синтаксический сахар», в частности, записывая формулы в привычном инфиксном виде, то процедуры С2-машины приобретают удобный для восприятия декларативный вид. Например, описанную выше процедуру можно переписать следующим образом:

Encrypt (C aes128oid) [text, C key, C iv] where

text = B [C plaintext, Hash (C sha256oid) (C plaintext)]

Существуют плохо читаемые процедуры, часто встречающиеся в разных протоколах. Их можно вынести в библиотеки стандартных процедур. Например, процедура *withLen n instr* приписывает слева к результирующей байт-строке инструкции *instr* длину этой байт-строки, уложенную в *n* байтов в порядке big endian:

withLen n instr = B [B [instr], Length_, I2Bytes_ n, instr]

Весь «синтаксический сахар» языка С2-машины, ввиду ограничений на объем текста, в настоящей статье не приводится. Конечный результат применения приемов по улучшению читаемости показан в Листинге 4.

```
packetDef = Sequence _records tlsRecord
  where
  tlsRecord =
   B [Vi "ContentType" # OfLen 1,
        Vi "Version"         # OfLen 2,
        withLen 2
         (Select (Vi "ContentType")
           [Case [0x14] changeCipherSpecMsg,
            Case [0x15] alertMsg,
            Case [0x16] handshakeMsgs,
            Case [0x17] encryptedMsg])]
        where
        changeCipherSpecMsg = C [0x01]
        handshakeMsgs =
          Sequence _handshMess handshMsg
            where
            handshMsg = B
             B [Vi "HandshType" # OfLen 1,
                  withLen 3 (Vi "HandshakeMsg" ##
                    Select (Vi "HandshType")
                       [Case [0x01] clientHello,
                        Case [0x02] serverHello,
                        Case [0x05] endOfEarlyData,
                        Case [0x08] encrypExtensions,
                        Case [0x0d] certificateRequest,
                        Case [0x0b] certificate,
                        Case [0x0f] certificateVerify,
                        Case [0x14] finished,
                        Case [0x04] newSessionTicket,
                        Case [0x18] keyUpdate])]

clientHello =
 B [C [0x03,0x03],
      random Clnt,
      withLen 1 clientSessionID,
      withLen 2 clientSuites,
      withLen 1 (C [0x00]),
      withLen 2
```



```
        (Sequence _clientHelloExts extension)]
         where
         extension =
          B [Vi "ClientExtensionID" # OfLen 2,
               withLen 2 (Vi "ClientHelloExtension" ##
                  Select (Vi "ClientExtensionID")
                     [Case [0x00,0x00] serverName,
                      Case [0x00,0x0a] groups,
                      Case [0x00,0x0b] pointFormats,
                      Case [0x00,0x23] sessionTicket,
                      Case [0x00,0x16] encryptThenMAC,
                      Case [0x00,0x17] extendedMaster,
                      Case [0x00,0x0d] supportedSigAlgs,
                      Case [0x00,0x2b] supportedVersions,
                      Case [0x00,0x31] postHandshAuth,
                      Case [0x00,0x33] keyShare,
                      Case [0x00,0x2a] earlyData,
                      Case [0x00,0x2d] pskKeyExchMode,
                      Case [0x00,0x29] preSharedKey]]
                where
                serverName = withLen 2
                    (B [Vi "NameType" # OfLen 1,
                          …
```

Листинг 4. Начальная часть спецификации пакета протокола TLS версии 1.3
Listing 4. An initial part of the specification for a TLS protocol package version 1.3

Такая нотация, являясь по-настоящему формальной (исполнимой на некоторой машине) по восприятию не сильно отличается от нотации формализованных вставок, используемых в документах RFC 5246 (TLS версии 1.2) и RFC 8446 (TLS версии 1.3), – тот же декларативный и лаконичный стиль описания форматов сообщений и определений функций.

Пример дальнейшей формализации RFC 8446: тексту на естественном языке *"Many of the cryptographic computations in TLS make use of a transcript hash. This value is computed by hashing the concatenation of each included handshake message, including the handshake message header carrying the handshake message type and length fields, but not including record layer headers. I.e., Transcript-Hash(M1, M2, ... Mn) = Hash(M1 || M2 || ... || Mn)."* будет соответствовать следующая формальная запись на языке С2-машины:

```
Hash hashOID (AllIn   _handshMess
                      (B [Vi "HandshType",  withLen 3 (Vi "HandshakeMsg")])
                      filtercondition)
```

где *filtercondition* – это инструкция-предикат, которая, в свою очередь, формально описывает условия отбора сообщений *M1, M2, ..., Mn*, неформально заданные в RFC 8446.

6. Тестирование «черного ящика» и криптографическая машина с исключенной ролью

Спецификация протокола задана на одной машине, совместно используемой всеми ролями протокола. Запуская спецификацию на С2-машине, разработчики промышленных реализаций протокола могут генерировать эталонные трассы с интересующими их параметрами и использовать их в качестве контрольных примеров.

Если же рассматривать задачу тестирования «черного ящика», то тестируемая реализация протокола – это отдельный объект. Тестирующей системе не известны секретные ключи объекта тестирования и результаты разрешения недетерминизма протокола на стороне объекта тестирования. Как следствие, предварительная генерация из спецификации некоторого множества трасс в целях проверки конформности «черного ящика» не имеет смысла. Тесты должны генерироваться динамически с учетом реакции «черного ящика».

C2-машина с исключенной ролью role (далее – C2ER-машина) – это вариант C2-машины, который работает в паре с «черным ящиком», реализующим протокол на стороне роли *role*.

C2ER-машина должна уметь решать две задачи: 1) проверять конформность пакетов, поступающих от «черного ящика» (message verification); 2) по пакетам, поступающим от «черного ящика», актуализировать свое состояние так, чтобы сохранялась конформность последующего поведения C2ER-машины (state updating).

Процедура функционирования *runC1ER*, описывающая C2ER-машину во втором приближении (Листинг 5), отличается от процедуры *runC1* наличием двух дополнительных функций – *receivefrom* (получение пакета извне) и *verifyupdate* (верификация пакета и актуализация состояния C2ER-машины; в случае успеха функция возвращает *True*, иначе *False*).

```
receivefrom :: Role → Maybe [Word8]
verifyupdate :: (ExprDesc, [Word8]) → Bool
runC1ER role = do
         let packetDef = fst protSpec
    let run (s, p) = do
             let chksess = do b' ←: Bool
                              if b' then run (s, p+1) else run (s+1, p)
          curstep := (s, p)
          (src, dst) :← (Role, Role)
          packetRoles := insert (s, p) (src, dst) packetRoles
               let ix = [(_sessions, s), (_packets, p)]
          if src == role
              then do mbs ← receivefrom role
              if mbs == Just bs
                      then do [d] ← symexecute [B [packetDef]] ix
                                    b ← verifyupdate d bs
                                    if b then chksess else fail
                               else stop
              else do b ← dochoices _packets ix
                   if b then do [d] ← symexecute [B [packetDef]] ix
                                      mbs ← calcvalue d
                             if mbs == Just bs then chksess else fail
                               else run (s+1, p)
    packetRoles := empty
    run (1,1)
```

Листинг 5. Процедура функционирования runC1ER
Listing 5. RunC1ER Functional Procedure

Реализация функции *verifyupdate* – непростая задача. Например, в работе [16] продемонстрирована нетривиальность задачи применения диссекторов общего назначения (на примере фреймворка Python Scapy) для разбора пакетов криптографических протоколов (на примере QUIC).

C2ER-машину будем называть *корректной проекцией* C2-машины, если выполняется следующее условие: для любой спецификации протокола множество трасс, производимых C2ER-машиной, является подмножеством множества трасс, производимых C2-машиной.

7. Заключение

Для проверки работоспособности предложенного подхода автором разработана реализация C2-машины с исключенной ролью на языке Haskell и спецификации протокола TLS (версий 1.2 и 1.3). Спецификация существенной части протокола TLS 1.3 (в частности, поддерживается двусторонняя аутентификация и восстановление сессий с использованием тикетов) на языке C2-машины занимает около 800 строк (часть *packetDef* – 500 строк и часть *controlDef* – 300 строк). Еще 200 строк (обработка сертификатов и др. общеупотребимые

процедуры) вынесены из *packetDef* в библиотеку, общую для всех протоколов. В роли «черного ящика» выступал программный пакет openssl.

Применительно к задаче тестирования «черного ящика» реализация машины умеет делать полный обход всех возможных траекторий протокола, зависящих от тестирующей стороны, последовательно перебирая различные варианты разрешения ситуаций недетерминированного выбора (в предположении, что объект тестирования детерминирован и для него возможен *надежный рестарт*, reliable reset).

На данном этапе реализации подхода ограничение перебора предполагается осуществлять путем ручного сокращения недетерминизма спецификации. Реализация эффективных стратегий обхода на полноценной спецификации – тема дальнейших исследований. Цель – на базе формализма C2-машины задать метрики качества тестирования криптопротоколов, схожие с метриками, принятыми в стандартах тестирования программ в авионике. Возможные подходы к решению этой задачи рассмотрены, например, в работе [17] (в контексте *машины с абстрактными состояниями* (ASM) [18]). В следующей работе также планируется представить результаты моделирования на языке C2-машины репрезентативного, с точки зрения обоснования выразительности предложенного языка, набора промышленных криптографических протоколов (IPSec (IKEv2), QUIC и др.).

Дополнительно авторская реализация C2-машины с исключенной ролью включает реализацию алгоритма Д. Англуин (Dana Angluin) [19], с помощью которого полученное дерево обхода сворачивается в более компактное (удобное для визуального анализа) представление в виде конечного автомата, а также транслятор трасс запуска C2-машины в спецификации в нотации анализатора протоколов ProVerif.

Литература

[1] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман. Тестирование реализаций клиента протокола TLS. Труды ИСП РАН, том 27, вып.2, 2015 г., стр. 145-160 / A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. TLS clients testing. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 2, 2015, pp. 145-160 (in Russian), DOI: 10.15514/ISPRAS-2015-27(2)-9.

[2] K. Bhargavan, A. Delignat-Lavaud et al. Implementing and Proving the TLS 1.3 Record Layer. In Proc. of the 38th IEEE Symposium on Security and Privacy, 2017, pp.463-482.

[3] J. Goubault-Larrecq and F. Parrenne. Cryptographic protocol analysis on real C code. Lecture Notes in Computer Science, vol. 3385, 2005, pp. 363-79.

[4] J. Bozic, L. Marsso et al. A formal TLS handshake model in LNT. In Proc. of the Third Workshop on Models for Formal Analysis of Real Systems and Sixth International Workshop on Verification and Program Transformation, 2018, pp. 1-40.

[5] K. Bhargavan, B. Blanchet, N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In Proc. of the IEEE Symposium on Security and Privacy; 2017, pp.483-502.

[6] J. Zhang, L. Yang et al. Formal analysis of 5G EAP-TLS authentication protocol using ProVerif. IEEE Access, vol. 8, 2020, pp. 23674-23688.

[7] M. Backes, A. Busenius, C. Hriţcu. On the Development and Formalization of an Extensible Code Generator for Real Life Security Protocols. Lecture Notes in Computer Science, vol. 7226, 2-12, pp. 371-387.

[8] M. Aizatulin, A. Gordon, J. Jurjens. Extracting and verifying cryptographic models from C protocol code by symbolic execution. In Proc. of the 18th ACM Conference on Computer and Communications Security (CCS'11), 2011, pp. 331-340.

[9] P. Arcaini, A. Gargantini, E. Riccobene. AsmetaSMV: a way to link high-level ASM models to low-level NuSMV specifications. Lecture Notes in Computer Science, vol. 5977, 2010, pp. 61-74.

[10] J. Jürjens. Automated Security Verification for Crypto Protocol Implementations: Verifying the Jessie Project. Electronic Notes in Theoretical Computer Science, vol. 250, issue 1, 2009, pp. 123-136.

[11] I. Abdullah, D. Menascé. Protocol Specification and Automatic Implementation Using XML and CBSE. In Proc. of the International Conference on Communications, Internet and Information Technology. (CIIT2003), 2003, pp. 1-7.

[12] E.W. Dijkstra. Guarded commands, non-determinacy and a calculus for the derivation of programs. Communications of the ACM, vol. 18, issue 8, 1975, pp. 453-457.
[13] K.L. McMillan and L.D. Zuck. Formal specification and testing of QUIC. In Proc. of ACM Special Interest Group on Data Communication (SIGCOMM'19), 2019, 14 p.
[14] D. Rozenzweig, D. Runje. The Cryptographic Abstract Machine. Lecture Notes in Computer Science, vol. 3052, 2004, pp. 202-217.
[15] M. Veanes, C. Campbell et al. Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. Lecture Notes in Computer Science, 2008, vol. 4949, pp. 39-76.
[16] E. Gagliardi, O. Levillain. Analysis of QUIC session establishment and its implementations. In Proc. of the IFIP International Conference on Information Security Theory and Practice, 2019, pp.169-184.
[17] A. Gargantini, E. Riccobene. ASM-based Testing: Coverage Criteria and Automatic Test Sequence Generation. Journal of Universal Computer Science, vol. 7, no. 11, 2001, pp. 1050-1067.
[18] Y. Gurevich. Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic, vol. 1, issue 1, 2000, pp. 77-111.
[19] D. Angluin. Learning Regular Sets from Queries and Counterexamples. Information and Computation, vol. 75, issue 2, 1987, pp. 87-106.

Информация об авторах / Information about authors

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

Sergey Evgenievich PROKOPEV – PhD, 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 JSRPC «NPK Kryptonite».