АВТОМАТИЧЕСКИЙ АНАЛИЗ КРИПТОГРАФИЧЕСКИХ ПРОТОКОЛОВ: ПОСТРОЕНИЕ ЗАПРОСОВ В PROVERIF

Ф. Б. Дасько

Белорусский государственный университет,
Минск, Беларусь, dasko@bsu.by

В статье предложен подход к формированию запросов в системе автоматизированного анализа криптографических протоколов ProVerif. Представлены запросы для проверки шести ключевых свойств безопасности: конфиденциальности, аутентификации, защиты от атак повтора, подтверждения ключа, обновления ключа и защиты от «чтения назад». Результаты могут быть использованы для повышения качества анализа за счет формального подхода к построению запросов, исключающего возможные ошибки и неточности.

Ключевые слова: криптографические протоколы; автоматизированный анализ протоколов; ProVerif; запросы безопасности; свойства безопасности.

AUTOMATED ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS: CONSTRUCTION OF QUERIES IN PROVERIF

F. B. Dasko

Belarusian State University,
Minsk, Belarus, dasko@bsu.by

We present an approach to construct queries in ProVerif, an automated cryptographic protocol analyzer. We develop query templates for verifying six fundamental security properties: confidentiality, authentication, resistance to replay attacks, key confirmation, fresh key derivation and perfect forward secrecy. Our methodology enhances the reliability of verification results through a rigorous formal approach to query formulation that eliminates potential ambiguities and errors.

Keywords: cryptographic protocols; automated protocol verification; ProVerif; security queries; security properties.

1. Введение

Формальный анализ криптографических протоколов является эффективным инструментом, позволяющим ускорить и упростить процесс проверки свойств безопасности протоколов. При этом наличие или отсутствие ожидаемых свойств доказывается строго математически.

Одним из популярных средств автоматизированного анализа является ProVerif [1]. В основе работы данного инструмента лежит понятие модели

протокола (model) – формального описания протокола на специализированном языке. Модель определяет каналы связи, криптографические примитивы, участников протокола и их действия. Ключевые шаги протокола маркируются с помощью событий (events) – специальных меток, обозначающих значимые состояния или действия в ходе выполнения протокола.

Свойства безопасности, которые необходимо проверить, формулируются в виде запросов (queries) – утверждений формальной логики над над одним или несколькими событиями. ProVerif анализирует модель и по каждому запросу выносит один из трех вердиктов: `true` (доказано, что утверждение истинно для всех возможных исполнений протокола), `false` (найден контрпример), или `can not be proved` (инструмент не смог ни доказать, ни опровергнуть свойство).

Для корректного анализа свойств протокола критически важна точная формулировка запросов. Ошибка в построении запроса приведет к тому, что он не будет соответствовать проверяемому свойству безопасности, что обесценивает результаты анализа.

Цель настоящей работы – предложить систематический подход к формулированию запросов в ProVerif для проверки основных свойств безопасности: конфиденциальности, аутентификации, защиты от атак повтора (replay protection), подтверждения ключа (key confirmation), обновления ключа (fresh key derivation) и защиты от «чтения назад» (perfect forward secrecy).

В [3] мы иллюстрируем предлагаемый подход на примере протокола Station-to-Station (STS) [2]. Этот протокол выбран благодаря своей минималистичности, хорошей изученности. На примере STS легко проследить выполнение вышеперечисленных свойств.

В разделе 2 представлены используемые соглашения. Разделы 3–9 посвящены анализу свойств безопасности. В данных разделах приводятся определения исследуемых свойств, затем они формулируется в виде строгих логических высказываний, которые, в свою очередь, трансформируется в конкретные запросы ProVerif. В разделе 10 обсуждаются полученные результаты.

Мы предполагаем, что читатель знаком с синтаксисом ProVerif.

2. Соглашения

При описании событий модели, запросов и результатов анализа мы используем синтаксис ProVerif, а также определенные именования элементов протокола для повышения читаемости запросов.

В модели используются следующие объекты:

- `pk` – открытые долговременные ключи;

- `sk` –личные долговременные ключи;
- `eph_pk` – открытые сеансовые ключи;
- `eph_sk` – личные сеансовые ключи;
- `K` – секретные ключи;
- `id` – идентификаторы сторон;
- `cert` – сертификаты;
- `ctext` – шифртексты.

При построении модели протокола STS мы учитывали, что протокол проводится в параллельном режиме, т.е. стороны действуют симметрично и одновременно. Симметричность действий сторон позволяет определить в модели для обеих сторон одну и ту же последовательность действий, формализованную как один процесс, запускаемый с различными входными параметрами.

Поскольку один процесс используется для обеих сторон протокола, то указать, что данные принадлежат определенной стороне протокола (данные стороны A или данные стороны B) не удастся. Поэтому используется указание на то, что данные созданы текущей стороной или получены извне (собственные данные и принятые данные).

Для указания принадлежности данных используются постфиксы:

- `self` – данные, определяемые текущей стороной процесса;
- `peer` – данные, определяемые второй стороной процесса;
- `t` – данные, определяемые третьей доверенной стороной.

В протоколе STS стороны используют свой идентификатор и идентификатор второй стороны, владеют личным и открытым ключом. Открытый ключ заверен третьей доверенной стороной в виде сертификата. Стороны протокола владеют ключом, позволяющим проверять сертификаты третьей доверенной стороны. Все эти данные передаются в процесс стороны как входные параметры:

```
processParty(id_self: identifier, id_peer: identifier,
sk_self: privKey, cert_self: bitstring, pk_t: pubKey).
```

3. Конфиденциальность

Свойство конфиденциальности гарантирует, что определённые данные (передаваемые как часть содержимого защищенного сообщения или сформированные на основе данных, которыми обмениваются стороны), не будут доступны нелегитимным сторонам и останутся неизвестными для злоумышленника.

В системе ProVerif для проверки доступности данных противнику применяется логический предикат `attacker(data)`. Данный предикат

принимает истинное значение в случае, если атакующий способен получить значение `data`.

Данные `data` могут быть представлены через свободное имя или через переменную. Использование свободных имен более гибкое, но требует привязки к событию протокола.

Свойство может быть формализовано так: «Если участник протокола обладает данными `data`, то они не известны атакующему».

Мы выделяем два типа событий, рекомендуемых для использования в запросах конфиденциальности: «участник протокола получил значение `data`» (обозначим `PartyKnowsData(data)`) и «участник завершил протокол со значением `data`» (обозначим `PartyFinishes(data)`). Первый тип события фиксирует знание на наиболее раннем этапе, в то время как второй – на наиболее позднем.

В зависимости от выбора события запрос конфиденциальности в ProVerif примет следующий вид:

```
query data: datatype; event(PartyKnowsData(data)) &&
attacker(data) ==> false.
```

или

```
query data: datatype; event(PartyFinishes(data)) &&
attacker(data) ==> false.
```

Запрос записан в виде импликации с посылкой `false` для того, чтобы избежать применения ключевого слова `not`, использование которого в запросах затруднено.

Необходимо отметить, что указанные запросы не эквивалентны, и запрос, использующий событие завершения протокола, является менее строгим. Более строгий запрос с фиксацией непосредственного получения данных рекомендуется применять к тем данным, которые сохранят ценность за пределами конкретного сеанса. К такой информации относятся долговременные ключи и секретные сообщения сторон.

На примере протокола STS проверим конфиденциальность долговременного личного ключа `sk_self`, а также сеансового ключа `K`, получаемого из общего секрета. Для этого зафиксируем событие `PartyKnowsLTkey(sk_self)` в самом начале процесса стороны и `PartyFinishes(K)` в конце процесса стороны.

Результат проверки:

```
Query not (event(PartyKnowsLTkey(sk)) && attacker(sk)) is
true.
Query not event( PartyFinishes(id_self_2, id_peer_2,
eph_sk_self_2, K_2)) && attacker_p1(K_2)) is true.
```

Указание фазы `phase 0` продиктовано проверкой свойства защиты от «чтения назад», которое будет рассмотрено далее.

Следует отметить, что ослабленное свойство для долговременного ключа получает вердикт `true`, тогда как усиленное свойство для сеансовых ключей возвращает результат `false`. Это объясняется возможностью злоумышленника подменить сеансовый ключ одного из легитимных участников своим значением. В результате атакующий и легитимная сторона вычисляют одинаковый ключевой материал, на основе которого формируются идентичные секретные ключи, фиксируемые ранним событием получения знания о ключе.

4. Аутентификация стороны

Аутентификация стороны представляет собой свойство протокола, обеспечивающее для одного из участников возможность анализа полномочий второго участника, а также подтверждение факта непосредственного участия второй стороны в текущем сеансе протокола.

В соответствии с данным определением, свойство формулируется так: «Если сторона A успешно завершила протокол, то сторона B действительно принимала в нем участие». Требование участия стороны B в протоколе может интерпретироваться различным образом.

Наименее строгим требованием будет то, что сторона B вступила в протокол, тогда как наиболее строгим – что последнее сообщение, полученное стороной A, действительно было отправлено стороной B (т.е. выполнение стороной B наиболее позднего шага, требующегося стороне A для успешного завершения протокола).

Такое соотношение между требованиями выполняется в связи с тем, что если сторона выполнила некоторый шаг протокола, то она выполнила и предыдущие шаги, тогда как выполнение последующих не гарантировано.

Наиболее строгое требование мы выделим в отдельное свойство «аутентификация данных» и рассмотрим его в следующем разделе. Здесь же остановимся на наименее строгом свойстве.

Запрос на его проверку в ProVerif имеет следующий вид:

```
query event(PartyFinishes(..., id_peer)) ==>
event(PartyStarted(id_peer)).
```

Здесь `PartyFinishes(..., id_peer)` это событие в запросе означает что текущая сторона `id_self` успешно закончила протокол с (как она считает) `id_peer`. Событие `PartyStarted(id_peer)` в запросе означает что сторона `id_peer` начала протокол, событие в процессе стороны фиксирует собственный идентификатор т.е. записывается как

`PartyStarted(id_self)`. Равенство идентификаторов в запросе подчеркивает, что события выполнены разными сторонами протокола.

Результат выполнения запроса в протоколе STS:

```
Query event(PartyFinishes(id_self_2, id_peer_2,
eph_sk_self_2, K_2)) ==> event(PartyStarted(id_peer_2 )) is
true.
```

5. Аутентификация данных

Свойство аутентификации данных обеспечивает гарантию того, что полученное сообщение или его компоненты сформированы конкретным участником протокола в определенный момент времени в прошлом, и что эти данные не подверглись модификации или фальсификации. При этом гарантии уникальности и актуальности данных могут не предоставляться.

Запрос для проверки данного свойства может быть сформулирован следующим образом: «если участник проверил данные и принял их, то легитимный участник действительно отправил эти данные». Необходимо подчеркнуть, что событие принятия данных должно фиксироваться непосредственно после успешной проверки полученной информации, а не просто после получения сообщения. Без проверки обеспечение свойства аутентификации данных невозможно.

Запросы аутентификации данных в ProVerif могут быть сформулированы следующим образом:

```
query id_peer:identificator, data_peer: datatype;
event(PartyAcceptsData(id_peer, data_peer)) ==>
event(PartySendsData(id_peer, data_peer))
```

Здесь событие `PartySendsData(id_self, data_self)` фиксируется непосредственно после отправки своих данных, а событие `PartyAcceptsData(id_peer, data_peer)` фиксируется сразу после проверки полученных данных. Одинаковые идентификаторы в событиях запроса указывают на то, что события фиксируются на разных сторонах протокола.

Результат выполнения запроса в протоколе STS:

```
Query event(PartyAcceptsData(id, pk, cert, ctext)) ==>
event(PartySendsData(id, pk, cert, ctext)) is true.
```

Отметим на примере протокола STS, что корректность второго сообщения означает, что и заявленный сеансовый ключ из первого сообщения был отправлен той же легитимной стороной, которая отправила второе сообщение. То, что из более позднего события (отправка второго сообщения) следует более раннее события этой же стороны (отправка первого сообщения) с согласующимися параметрами, является тривиальным свойством,

которое выполняется независимо от протокола и его фактических свойств. В связи с этим оказывается, что построение корректного запроса в виде следствия из проверки более позднего сообщения к корректности более раннего не проще, чем проведение полноценной аналитической работы.

6. Защита от повтора

Защита от повтора означает, что данные, полученные стороной протокола, были отправлены второй стороной протокола в том же сеансе.

Следует отметить два ключевых момента относительно данного свойства:

Во-первых, в ProVerif запросы конфиденциальности сразу учитывают возможность различных атак, в том числе и атак повтора (если стороны протокола реплицируются), но отдельная проверка своевременности данных все еще желательна.

Во-вторых, данные, которые могут быть признаны актуальными, должны быть созданы только легитимным участником. В этом смысле защита от повтора представляет собой усиленное требование к аутентификации данных, т.е. данные не только сформированы легитимным участником протокола, но и в конкретном сеансе.

Запрос для проверки данного свойства может быть сформулирован следующим образом: «если участник в некотором сеансе верифицировал данные и принял их, то легитимный участник действительно отправил эти данные в рамках этого же сеанса».

Здесь свойство требует контролировать события разных сторон и их принадлежность к одному и тому же сеансу протокола.

Универсальным решением будет использование инъективных событий:

```
query inj-event(PartyAcceptsData(id, data)) ==>
inj-event(PartySendsData(id, data)).
```

Альтернативным подходом является привязка к волатильным (генерируемым в каждом сеансе протокола) данным. Сложность заключается в том, что их волатильность и согласованность для двух сторон должна быть доказана. В этом смысле для протоколов выработки общего ключа (в том числе STS) было бы естественно использовать привязку к сеансовому ключу сторон, а свойства обновления и согласованности ключа (рассмотренные далее) проверяются явно. Однако этот подход не универсален. Более универсальным будет доказать волатильность аутентифицируемых данных. Таким образом, запрос, усиливающий свойство аутентификации данных до защиты от повтора, будет следующим:

```
query event(PartyAcceptsVolatileData(data_peer, SID_self1))
&& event(PartyAcceptsVolatileData(data_peer, SID_self2)) ==>
SID_self1 = SID_self2.
```

Здесь `SID_self` представляет собой некоторые гарантированно волатильные данные, генерируемые самой стороной и применяемые для отличия одного сеанса от другого.

Результат выполнения запросов в протоколе STS:

```
Query inj-event(PartyAcceptsData(id, pk, cert, ctext)) ==>
inj-event(PartySendsData(id, pk, cert, ctext)) is true.
Query event(AttributedVolatileData(ctext, eph_sk_self1)) &&
event(AttributedVolatileData(ctext, eph_sk_self2)) ==>
eph_sk_self1 = eph_sk_self2 is true.
```

Таким образом, подтверждается свойство защиты от повтора в протоколе STS.

7. Подтверждение ключа

Подтверждение ключа означает, что один из участников протокола получает гарантию того, что второй участник действительно обладает конкретным секретным ключом (или ключевым материалом, необходимым для вычисления ключа).

Данное свойство формализуется так: «если участник A завершает протокол с общим ключом, то участник B владеет этим ключом».

Запросы подтверждения ключа в ProVerif могут быть сформулированы следующим образом:

```
Query id_peer:identifier, k1:key;
event(PartyFinishes(..., id_peer, k)) ==>
event(PartyKnowsKey(id_peer, k)).
```

Результат выполнения запроса в протоколе STS:

```
Query event(PartyFinishes(id_self_2, id_peer_2,
eph_sk_self_2, K_2)) ==>
event(PartyKnowsSessionKey(id_peer_2, K_2)) is true.
```

Таким образом, свойство подтверждения ключа подтверждается для протокола STS.

8. Обновление ключа

Протокол обеспечивает обновление ключа, если в каждом сеансе протокола используется уникальный сеансовый ключ, что обеспечивает защиту от компрометации данных, переданных в предыдущих сеансах, даже при компрометации текущего сеансового ключа.

Обновление ключа формализуется следующим образом: «если участник протокола завершил сеанс протокола N с определенным ключом, и участник завершил сеанс M с тем же ключом, то это была один и тот же сеанс, т.е. $N = M$». Номер сеанса может быть эквивалентно заменен произвольными волатильными данными, сгенерированными данным

участником протокола; совпадение волатильных данных гарантирует принадлежность событий к одному и тому же сеансу.

Запросы обновления ключа в ProVerif могут быть сформулированы следующим образом:

```
Query id_self: identifier, k1, k2: keys, v_data1, v_data2:
volatileData;
event(PartyFinishes(id_self, v_data1, k1)) &&
event(PartyFinishes(id_self, v_data2, k2)) ==>
v_data1 = v_data2.
```

Результат выполнения запроса в протоколе STS:

```
Query event(PartyFinishes(id_self_2, id_peer1, eph_sk_self1,
K_2)) && event(PartyFinishes(id_self_2, id_peer2,
eph_sk_self2, K_2)) ==> eph_sk_self1 = eph_sk_self2 is true.
```

Необходимо подчеркнуть значимость собственного идентификатора, необходимого для отличия штатной ситуации, где один и тот же ключ получают две стороны одного сеанса, от ситуации, когда одна сторона получает одинаковый ключ дважды.

Отметим, что использование события, фиксирующего сеансовые ключи в момент их генерации, может не отражать целевое свойство. Так, в контексте протокола STS такой запрос даст отрицательный результат. Это обусловлено тем, что два репликанта (клона) участника A могут взаимодействовать между собой, генерируя идентичный сеансовый ключ, но поскольку репликанты моделируют различные сеансы протокола, возникает ситуация, при которой одним участником в различных сеансах фиксируется идентичный сеансовый ключ.

9. Защита от «чтения назад»

Под защитой от «чтения назад» понимается свойство протокола, которое гарантирует, что компрометация долговременных ключей, используемых в протоколе, не приводит к компрометации ранее сформированных сеансовых ключей.

Для проверки данного свойства необходимо смоделировать ситуацию, где стороны протокола проводят протокол, после чего атакующему передаются значения всех долговременных ключей, и проводится запрос: может ли атакующий при этих условиях вычислить значения сеансовых ключей.

Для моделирования такой ситуации в ProVerif применяются фазы, где нулевая фаза – это нормальное проведение протокола, а в фазе 1 раскрываются все долговременные параметры всех сторон протокола. Вследствие этого при проверке конфиденциальности долговременных параметров необходимо явно указывать, что проверяется свойство именно в фазе 0 (до раскрытия значения).

В виде запроса ProVerif проверку защиты от «чтения назад» можно записать как:

```
query data: datatype; event(PartyKnowsData(data)) &&
attacker(data) phase 1 ==> false.
```

Результаты анализа протокола STS:

```
Query not (event(PartyFinishes(id_self_2, id_peer_2,
eph_sk_self_2, K_2)) && attacker_p1(K_2)) is true.
```

Таким образом, подтверждается, что протокол STS обладает свойством защиты от «чтения назад».

10. Заключение

В данной работе представлен подход к формированию запросов в ProVerif для проверки основных свойств безопасности криптографических протоколов. Применение запросов на примере протокола Station-to-Station продемонстрировало эффективность формализации для строгой математической проверки таких свойств, как конфиденциальность, аутентификация, защита от атак повтора, подтверждение ключа, обновление ключа и защита от «чтения назад». Результаты анализа подтвердили соответствие протокола STS заявленным свойствам безопасности.

Следует отметить, что в настоящей работе не рассматривались свойства, требующие проверки через «эквивалентность наблюдений» (observational equivalence), такие как анонимность сторон или невозможность отслеживания действий участников. Анализ этих свойств с использованием ProVerif, а также расширение предложенного подхода на более сложные протоколы представляют собой перспективные направления для дальнейших исследований.

Библиографические ссылки

1. ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial / B. Blanchet [et al.]. URL: https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf (date of access: 22.09.2025).

2. *Diffie W., van Oorschot P. C., Wiener M. J.* Authentication and authenticated key exchanges // Designs, Codes and Cryptography. 1992. Vol. 2, iss. 2, P. 107–125.

3. *Дасько Ф. Б.* ProVerif-STS-model // GitHub. URL: https://github.com/HassiumPotassium/ProVerif-STS-model (date of access: 22.09.2025).

4. *Blanchet, B.* Modeling and verifying security protocols with the applied pi calculus and ProVerif // Foundations and Trends in Privacy and Security. 2016. Vol. 1, iss. 1–2. P. 1–135.

5. *Basin D., Cremers C., Meadows C.* Model checking security protocols // Handbook of Model Checking / eds.: E. M. Clarke [et al.]. Springer, Cham, 2018. P. 727–762.