МАТЕМАТИЧЕСКИЕ ОСНОВЫ КРИПТОГРАФИИ

УДК 681.322

ПОИСК УПРОЩЕННОЙ МОДЕЛИ ПРОТОКОЛОВ ИНФРАСТРУКТУРЫ ЦИФРОВОЙ ПОДПИСИ С ИСПОЛЬЗОВАНИЕМ ВЕРИФИКАТОРОВ МОДЕЛЕЙ

С. Е. Прокопьев

г. Москва

E-mail: prsz@bk.ru

Важнейшим способом повышения безопасности информационных систем, использующих криптографические методы защиты информации, является эффективная декомпозиция кода криптографических сервисов на программные модули. Под эффективной декомпозицией понимается: 1) изоляция программных модулей разной степени критичности (т. е. имеющих разную степень опасности последствий от ошибок или программных закладок) друг от друга; 2) применимость одних и тех же программных модулей без модификации (и, как следствие, повторной верификации) в различных криптографических сервисах; 3) получение компактных, детерминированных, криптографически полных упрощенных моделей программных модулей. Одним из возможных подходов к решению проблемы эффективной декомпозиции является предварительный анализ набора протоколов, реализующих криптографический сервис, в рамках т. н. *UC-моделей*. В настоящей статье рассмотрена проблема декомпозиции криптографического сервиса «Цифровая подпись на базе PKI» в рамках указанного подхода и исследована возможность частичной автоматизации процедуры поиска его упрощенной модели с использованием *верификатора моделей NuSMV*.

Ключевые слова: *UC-модели, верификаторы моделей, NuSMV, цифровая подпись, PKI.*

Введение

К программным реализациям криптографических алгоритмов предъявляются особые требования, поэтому прикладная и криптографическая части приложения обычно реализуются разными разработчиками. Для повышения безопасности приложений, использующих криптографические методы защиты информации (снижения ущерба от ошибок прикладного разработчика и трудоемкости анализа безопасности), криптографический код необходимо [1] размещать внутри т. н. *криптографического периметра (криптоядра)*, изолированного от прикладного кода и решающего, помимо реализации криптографических алгоритмов, задачу создания защищенного канала ввода ключевой информации. Известно, что слабые реализации криптографических протоколов не менее опасны, чем слабые реализации криптографических примитивов (функции шифрования, вычисления и проверки имитовставки и цифровой подписи и т. д.), поэтому в криптоядро обычно помещаются не только примитивы, но и целые криптографические сервисы. Под криптографическим сервисом здесь понимается реализация одного или нескольких криптографических протоколов, совместно решающих некоторую задачу, вместе с реализацией служебных функций. Например, сервис «Проверка цифровой подписи», кроме самого алгоритма верификации цифровой подписи, будет включать

реализацию целого семейства протоколов инфраструктуры открытых ключей (PKI), а также функции управления ключевой информацией и настройки правил политики безопасности.

Размещение кода криптографических протоколов внутри криптоядра, очевидно, приводит к сильному увеличению размера последнего. В то же время в криптографических протоколах существуют части, не являющиеся сильно критичными с точки зрения безопасности. Например, известна атака типа downgrade, когда злоумышленник понижает стойкость используемого участниками криптокомплекта (ciphersuite) в протоколе TLS [2]. Очевидно, что эта угроза менее опасна, чем угроза компрометации ключей, так как участники изначально были согласны на использование любых криптокомплектов из числа ими заявленных. С учетом этого, задачу накопления сообщений фазы handshake и вычисления значения хэш-функции от их конкатенации можно рассматривать как слабокритичную часть протокола и вынести из криптоядра. Таким образом, перед программной реализацией необходим анализ криптографических протоколов на предмет выявления в них частей разной степени критичности.

Другая проблема, возникающая при размещении кода протоколов в криптоядре, — это необходимость модульной декомпозиции сложных криптографических протоколов. Часто разные сложные протоколы используют в качестве вспомогательных одни и те же более простые. Необходимо провести декомпозицию кода разных криптографических серисов на модули так, чтобы одни и те же программные модули криптоядра могли без модификации (и, как следствие, повторной верификации) быть применены при конструировании различных криптографических сервисов. Для этого свойства безопасности протоколов должны быть проанализированы в как можно более нефиксированных условиях, в предельном случае — в условиях неизвестного окружения.

Далее, для того чтобы сформулировать свойства безопасности всей системы или сложного сервиса (обычно в виде списка предотвращаемых угроз), разработчикам нужна некоторая простая и адекватная модель, имитирующая сложные реальные алгоритмы сервисов, — их упрощенные модели (абстракции). Внешнее поведение абстракции должно быть неотличимым (в некоторой заданной степени) от внешнего поведения реального протокола (т. н. *принцип симуляции*). При этом абстракции должны быть криптографически полными (cryptographically sound): из отсутствия уязвимостей в модели информационной системы на базе абстракций криптографических сервисов должно следовать отсутствие уязвимостей в модели информационной системы на базе протоколов, реализующих эти сервисы.

С учетом приведенных выше рассуждений, из всех существующих подходов к анализу безопасности криптографических протоколов нам наиболее интересны модели, в которых изначально заложен модульный подход к анализу безопасности сложных составных криптографических протоколов с использованием криптографически полных абстракций.

Такие модели есть. Это модели *с определениями безопасности симуляцией в неизвестном окружении*, далее их будем называть *UC-моделями* (universally composable) (термин из [3]). Среди формально-логических — это модель Линкольна—Митчелла—Митчелла—Скедрова (ЛММС) [4, 5], среди теоретико-сложностных — Канетти [3, 6–8] и Пфитцманн—Шунтера—Вайднера (ПШВ) [9, 10]. Связь между этими моделями безопасности установлена в [4]; фактически, они отличаются способами задания модели вычислений.

1. Модели анализа безопасности криптографических протоколов с определениями безопасности симуляцией в неизвестном окружении

В UC-моделях используются две модели вычислений: реальная (*real*) и идеальная (*ideal*). В реальной модели каждый участник Pi использует для решения требуемой задачи программу $Prot(i)$, реализующую алгоритм протокола. Доставку сообщений между программами $Prot(k)$ осуществляет злоумышленник A, который может произвольным образом модифицировать их. В идеальной модели существует специальный объект — доверенная идеальная функциональность (*ideal functionality*, ИФ), к которой каждый участник имеет конфиденциальный и аутентифицированный доступ и которая решает для них эту задачу некоторым идеальным образом. Т. е. ИФ реализует идеальные спецификации задачи. Например, задачу создания защищенного соединения (протоколы TLS, IPSec, SSH) можно специфицировать следующей идеальной моделью: участники передают свои сообщения вместе с идентификаторами адресатов сообщений идеальной функциональности (назовем ее «Защищенное соединение»), которая просто передает их по назначению, используя свои каналы связи с участниками. Протокол считается безопасным, если для любого обладающего заданными вычислительными возможностями злоумышленника, пытающегося извлечь информацию из процесса взаимодействия сторон в реальной модели, существует злоумышленник, получающий эту же информацию в идеальной модели. В модели ПШВ это обеспечивается требованием вычислительной неразличимости выходных распределений злоумышленников реальной и идеальной моделей. В модели Канетти используется специальный участник — *окружение (environment)*, который пытается угадать, с каким из злоумышленников он взаимодействует. При этом окружение может произвольным образом взаимодействовать со злоумышленником и реализациями сервиса в обеих моделях (в реальной — с программами протокола, в идеальной — с ИФ). Схема определения безопасности в модели Канетти отображена на рис. 1.

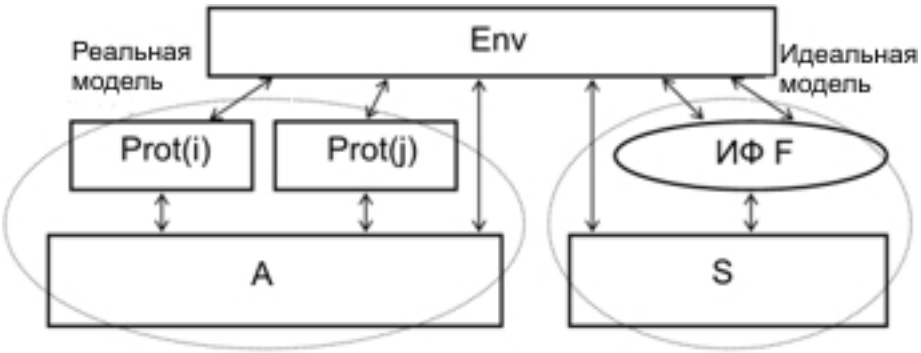

Рис. 1. Схема определения безопасности в модели Канетти

На схеме $Prot(i)$ и $Prot(j)$ — это программы, реализующие протокол на машинах участников i и j.

Немного более строгая формулировка определения безопасности в модели Канетти следующая:

Определение 1. *Протокол $Prot$ (совокупность $Prot(k)$ для всех участников $k = 1, 2, \ldots$) безопасно реализует ИФ F*, если для любого алгоритма A существует

алгоритм S, такой, что любое окружение Env не может отличить реальную модель от идеальной с вероятностью, существенно большей, чем $1/2$ [1].

Взаимодействие между ИФ F и злоумышленником идеальной модели S нужно для того, чтобы S мог получить информацию, которая с точки зрения безопасности решения задачи не является критической и получения которой злоумышленником A в реальных условиях не избежать. Например, в протоколах, реализующих задачу «Защищенное соединение», злоумышленник A всегда будет знать факт передачи, а также кому и кем направлены сообщения (так как он сам эти сообщения доставляет). Поэтому для обеспечения возможности симуляции необходимо, чтобы ИФ «Защищенное соединение» передавала злоумышленнику S эту информацию.

В дальнейшем при анализе сложных протоколов используется т. н. гибридная (*hybrid*) модель, в которой присутствуют как программы $Prot(k)$, так и ИФ. В гибридной модели программы $Prot(k)$ для решения вспомогательных задач обращаются к ИФ. При этом безопасность их композиции — сложносоставного протокола, полученного замещением ИФ протоколом, безопасно ее реализующим в «чистой» идеальной модели, — гарантируется теоремой о безопасности композиции безопасных протоколов.

Теорема 1 типовая для UC-подходов, неформальная формулировка. Если протокол $Prot1$ безопасно реализует ИФ $F1$ в $F2$-гибридной модели, а протокол $Prot2$ безопасно реализует ИФ $F2$ в реальной модели, то их композиция $Prot1(Prot2)$ безопасно реализует ИФ $F1$ в реальной модели.

Возможность такого результата обусловлена очень сильным определением безопасности — в неизвестном окружении, что приводит к сложности получения результатов в рамках UC-моделей. Поэтому необходимо развивать подходы, связанные с автоматизацией получения результатов.

2. Подготовка сервиса «Цифровая подпись на базе PKI» к автоматизированному анализу безопасности в рамках UC-моделей

Итак, UC-модели дают нам универсально применимую, компактную, детерминированную, криптографически полную абстракцию совокупности криптографических протоколов, решающих некоторую криптографическую задачу, — ИФ.

Определение безопасности в условиях неизвестного окружения и криптографическая полнота UC-моделей делают их наиболее мощными из всех существующих на сегодня подходов к анализу безопасности протоколов, т. е. покрывающими наибольшее число угроз безопасности. Обратной стороной этого является сложность получения результатов в рамках данного подхода, в частности автоматизации этого процесса. На вычислительной машине компактно и адекватно специфицировать такой объект как криптографический примитив, стойкий в смысле некоторого теоретико-сложностного определения безопасности, практически невозможно. В то же время одним из главных преимуществ UC-моделей является то, что формулировка ИФ криптографически полно симулирует поведение реального протокола, являясь при этом детерминированным алгоритмом. Поэтому, если примитивы будут заключены внутри вспомогательных под-

[1] Для большинства криптографических задач найдены протоколы, UC-безопасно их реализующие [3, 6, 7, 9]. Однако не для всех криптографических задач, для которых существует безопасный с точки зрения здравого смысла протокол, существует UC-безопасный протокол. Например, для нерешаемых в рамках UC-подхода задач привязки к биту, групповой подписи и некоторых других можно построить безопасные с точки зрения здравого смысла протоколы, безопасные в рамках т. н. игрового подхода к определению безопасности (эксперимента с оракулом).

протоколов нижнего уровня, то при анализе сложных протоколов эти подпротоколы можно заменить детерминированными ИФ без потери криптографической полноты (в отличие от алгебраических подходов к анализу безопасности криптографических протоколов на базе модели Долева—Яо). В итоге мы получаем детерминированный алгоритм сложного протокола, при анализе безопасности которого уже можно применять средства автоматизации.

В работе [11] в рамках модели ПШВ анализ безопасности протокола, реализующего задачу «Защищенное соединение с контролем порядка сообщений», был частично автоматизирован путем применения верификатора теорем PVS [12]. Данный протокол анализировался в гибридной модели, в которой в качестве вспомогательной ИФ выступала ИФ «Защищенное соединение без контроля порядка сообщений».

В настоящей работе в качестве объекта исследования было взято семейство протоколов инфраструктуры открытых ключей (PKI), реализующих сервис «Цифровая подпись на базе PKI». В качестве средства автоматизации был выбран верификатор моделей (*model checker*) NuSMV [13].

В [7] была сформулирована ИФ $F(Sig)$, привязывающая сообщения к открытому ключу. Наша цель — сформулировать надстройку над ней — функциональность, которая будет покрывать дополнительную специфику инфраструктуры PKI: привязку сообщений к идентификаторам субъектов, отзыв ключа, изменение действительности ключа во времени. Обозначим ее как ИФ $F(SigPKI)$. Вся вероятностная специфика реальных алгоритмов цифровой подписи скрыта внутри детерминированной $F(Sig)$, поэтому работа протоколов PKI в $F(Sig)$-гибридной модели будет детерминированной и может быть автоматизирована. Другая цель — оценить сложность протокола, для которого поиск формулировки ИФ может быть автоматизирован с помощью верификатора моделей NuSMV.

Одна из главных проблем при использовании средств формальной верификации — это обеспечение корректности спецификаций по отношению к анализируемой реальной системе. Одним из вариантов ее решения является написание спецификаций сначала на языке высокого уровня, а затем автоматическая трансляция его в спецификации на языке верификатора моделей. Как было показано в [14], такой подход неэффективен. Более эффективным способом повышения гарантий корректности спецификаций представляется декомпозиция спецификаций на независимые компоненты, взаимодействующие друг с другом посредством передачи сообщений. Такой подход, во-первых, существенно облегчает проверку корректности спецификаций за счет независимости модулей, во-вторых, позволяет использовать одни и те же модули в спецификациях разных систем. В [15] была предложена адаптация модели Канетти, которая задает эффективный способ декомпозиции криптографических приложений на программные модули с возможностью независимого анализа безопасности каждого из них. В настоящей работе мы внедрим этот подход в процесс описания моделей на языке спецификаций верификатора моделей NuSMV.

В соответствии с определением безопасности UC-моделей для любого алгоритма злоумышленника реальной модели A необходимо сконструировать такой алгоритм злоумышленника идеальной модели S, что окружение не сможет отличить A от S. Будем использовать распространенный подход на базе «черного ящика»: S запускает внутри себя копию A и эмулирует для него среду реальной модели.

Будем использовать следующий порядок действий:

1) Моделируется протокол (реальная модель).

2) Фиксируется предположительная формулировка ИФ и алгоритма симуляции злоумышленника S на базе «черного ящика A» (идеальная модель).

3) Используя возможности верификатора, перебираются алгоритмы «черного ящика A» и проверяется (с помощью предикатов верификатора) неразличимость моделей для окружения. Если верификатор обнаруживает контрпример, то либо идеальная модель (формулировки ИФ и алгоритма симуляции S), либо реальная модель (протокол) корректируются и симуляция проверяется заново. (Какая модель корректируется, зависит от того, какую задачу мы решаем: поиск формулировки ИФ для протокола (семейства протоколов) или построение протокола, безопасно реализующего заданную ИФ. В нашем случае — это поиск формулировки ИФ.)

4) Если модель не просчитывается (либо по времени, либо по памяти), то модель сокращается и просчитывается снова в соответствии с указанными выше шагами.

О полноте и корректности при сокращении модели

Формулировки функциональностей в UC-моделях предназначены для доказательства безопасности, т. е. доказывается только криптографическая полнота (cryptographic soundness) формулировки. И хотя авторы стремятся сделать их максимально корректными (см., например, трансформации ИФ «Цифровая подпись» ($F(Sig)$) от [3] к [6] в модели Канетти), корректность (*correctness*) формулировки не является конечной целью. Злоумышленнику идеальной модели позволено получить информации больше, чем злоумышленнику реальной модели, т.е. в обратную сторону симуляции нет. Поэтому, если в модели Канетти с точки зрения интуитивного представления о сервисе есть уязвимость в формулировке ИФ, это не означает, что она есть и в реальном протоколе, который ее безопасно реализует. Например, в реальных протоколах цифровой подписи вероятность того, что открытые ключи разных участников совпадут, пренебрежимо мала, в то время как в формулировке ИФ $F(Sig)$ из [3] открытый ключ генерируется злоумышленником идеальной модели; очевидно, что последний может сгенерировать один и тот же открытый ключ для разных участников с вероятностью 1.

При сокращении модели ситуация обратная: теряется криптографическая полнота. Однако смысл автоматизации построения функциональности заключается не в автоматическом получении доказательства симуляции, а в проверке алгоритма симуляции злоумышленника идеальной модели S. Поэтому при сокращении модели важно стремиться к сохранению корректности: найденные автоматизированным средством отклонения симуляции не должны быть ложными, т.е. должны присутствовать и в несокращенной модели. В дальнейшем они будут учтены при ручном построении функциональности. Цель — исследовать интересующий «срез» модели, повысить вероятность корректности доказательства, частично верифицировать утверждение о симуляции.

Таким образом, в общем случае реализация сокращенной модели на языке спецификаций верификатора может быть и не полной, и не корректной. Очевидно, что при сокращении модели нужно стараться, чтобы одновременно вероятность нахождения реальной уязвимости была высока, а ложной уязвимости — мала. Необходимо выбирать оптимальное (относительно вычислительных мощностей) соотношение между компактностью формулировки и степенью ее полноты и корректности.

Анализ критичности компонентов сервиса «Цифровая подпись на базе PKI»

Различные варианты политики безопасности (ПБ) фактически означают использование в реализации сервиса разных криптографических протоколов, поэтому ИФ,

соответствующие каждому варианту ПБ, будут отличаться. Таким образом, сначала необходимо зафиксировать возможные операции участников PKI.

Пусть участник, подписывающий сообщения, оперирует следующими командами: «сгенерировать новую пару», «выбрать текущий рабочий ключ подписи», «подписать сообщение», «обновить сертификат», «отозвать ключ».

Пусть участник, проверяющий подпись, оперирует командами: «загрузить список доверенных участников», «загрузить открытый ключ из сертификата», «выбрать текущий рабочий открытый ключ», «загрузить интервал доверия списка отозванных сертификатов (CRL)», «проверить подпись», «загрузить CRL».

Необходимо определить, какие команды следует отдать злоумышленнику. Напомним, что один из критериев эффективности декомпозиции — это изолирование частей кода приложения разной степени криптографической критичности друг от друга. Необходимо максимально сокращать объем критичного кода путем выделения из него некритичного кода. Поэтому, чем больше функций мы сможем безопасно отдать злоумышленнику, тем легче будет верифицировать критичный код. При этом нельзя «перестараться»: спецификации должны соответствовать нашему интуитивному представлению о задачах сервиса. Например, очевидно, что определение списка ключей участников, которым участник доверяет как удостоверяющим центрам, отдавать злоумышленнику нельзя. Отметим, что передача части функций злоумышленнику не означает, что в реальной информационной системе они в действительности передаются злоумышленнику, — еще раз подчеркнем, что это просто способ эффективной декомпозиции кода на модули разной степени критичности.

1. Правила ПБ вводятся с учетом криптографических аспектов реализации сервиса «цифровая подпись на базе PKI», однако все это носит приблизительный характер. Поэтому стойкость протоколов PKI должна сохраняться даже при «неправильном» задании правил ПБ. Таким образом, при анализе безопасности следующие функции ПБ можно отдать злоумышленнику:

1) определение интервала обновления CRL;

2) определение срока истечения сертификата;

3) определение интервала, в течение которого запросы считаются свежими.

2. В инфраструктуре PKI отрицательный результат верификации сообщения не обязательно означает, что участник X не подписывал сообщение M. Пользователи одновременно могут обладать несколькими действительными сертификатами, и верификация действительной подписи неправильно выбранным ключом даст ошибку верификации. Вообще, на задаваемый сервису «цифровая подпись на базе PKI» вопрос: «Подписывал ли участник X сообщение M?» — могут быть получены следующие ответы: 1) да; 2) искажено сообщение или подпись, либо неправильно выбран открытый ключ; 3) срок годности открытого ключа истек; 4) срок годности сертификата открытого ключа истек; 5) использованный при верификации ключ содержится в CRL; 6) срок действия сертификата ключа, которым подписан CRL, истек; 7) срок доверия к CRL истек; 8) срок действия сертификата удостоверяющего центра (УЦ) истек, и т. д. Тогда, например, при получении ответа «срок годности открытого ключа истек» пользователь сервиса может запустить процедуру получения и верификации более свежего сертификата. С точки зрения эффективности разработки приложений (принцип максимальной независимости компонентов) часть приложения, использующая сервис, не должна подвергаться изменениям при изменении (развитии) инфраструктуры PKI, поэтому управление ключами целесообразно заключить внутри сервиса. И тогда от-

вет на заданный вопрос будет либо «да», либо «не знаю». Учитывая это, следующие функции ПБ также отдадим злоумышленнику:

1) инициация генерации новой пары;
2) выбор ключа верификации;
3) обновление сертификата чужого открытого ключа;
4) обновление сертификата своего открытого ключа.

Итоговые формулировки компонентов UC-модели

1. В [6] была сформулирована ИФ «Цифровая подпись» — $F(Sig)$, привязывающая сообщение к открытому ключу. Специфика ЭЦП на базе PKI заключается в изменении действительности открытых ключей во времени. Эта специфика не связана с манипуляциями со значениями подписей, особенностями генерации открытого ключа, поэтому формулировку ИФ «Цифровая подпись» в нашей экспериментальной модели упростим до следующей:

ИФ «Цифровая подпись», $F(Sig)$:

Содержит поля $owner = Pi$, $fid = SIGN\_FID$, $inst = pk$, значение которых фиксируется при запуске, и массив m одноразово-записываемых переменных, в которых сохраняются сообщения.

Генерация подписи: После получения во внешнем входе от субъекта Pi сообщения (M_SIGN, msg) сохраняет сообщение msg в свободной переменной массива m и возвращает во внешний выход субъекту Pi сообщение $(M\_SIGN\_R\_OK)$.

Проверка подписи: После получения во внешнем входе субъекта Pj сообщения (M_VERIFY, msg): если сообщение msg было ранее сохранено, то возвращает во внешний выход субъекту Pj ответ $(M\_VERIFY\_R\_OK)$, иначе $(M\_VERIFY\_R\_FAIL)$.

---

2. Протокол, реализующий сервис «Цифровая подпись на базе PKI».

В представленной ниже формулировке аспекты политики безопасности, связанные с самими алгоритмами работы участников протокола, зафиксированы следующим образом:

— Владелец ключа не устанавливает срок годности своих ключей.

— Владелец не проверяет истечение срока годности своего сертификата.

— Обновление сертификата: Владелец подписывает запрос на обновление любого своего ключа любым своим ключом. Получив запрос, УЦ проверяет, что оба ключа не отозваны и их сертификаты действительны.

— УЦ подписывает все корректные запросы без POP (proof of possession, доказательство знания секретного ключа).

— Срок годности CRL определяет УЦ.

— Срок доверия к сообщениям о статусе ключей верифицирующий участник устанавливает сам.

Программы протокола PKI работают в $F(Sig)$-гибридной модели.

2.1. Владелец открытого ключа, **PKIo**:

Работает в $FSig$-гибридной модели. Содержит поля $owner = Pi$ (на практике присваивается при создании, здесь — фиксирован сразу), $fid = PKIO\_FID$, $inst$ (здесь будет только один экземпляр, поэтому поле не будет использоваться).

Генерация подписи. После получения во внешнем входе команды (M_SIGN,msg) передает в коммуникационный выход сообщение $(M\_GETMYPK)$. Ожидает в коммуникационном входе сообщение $(M\_GETMYPK\_R\_OK, pk)$, после чего передает $FSig[inst = pk]$ команду (M_SIGN, msg). После получения от $FSig[inst = pk]$ ответа (M_SIGN_OK) возвращает во внешний выход сообщение (M_SIGN_R_OK).

Выдача запроса на получение сертификата. После получения в коммуникационном входе команды $(M\_UPDMYCERT, pk1, pk2)$ передает $FSig[inst = pk2]$ сообщение $(M\_SIGN, msg = (CERTREQ\_ID, pk1, owner))$. Ожидает от $FSig[inst = pk2]$ ответ $(M\_SIGN\_R\_OK)$ и возвращает в коммуникационный выход сообщение $(M\_UPDMYCERT\_R\_OK, msg)$.

Выдача запроса на отзыв ключа. После получения в коммуникационном входе команды $(M\_RVKMYPK, pk)$ передает $FSig[inst = pk]$ сообщение $(M\_SIGN, msg = (RVKREQ\_ID, pk)$. Ожидает ответ $(M\_SIGN\_R\_OK)$ и возвращает в коммуникационный выход сообщение $(M\_RVKMYPK\_R\_OK, msg)$.

2.2. Субъект, проверяющий подписанное сообщение, $PKIv$:

Работает в $F(Sig)$-гибридной модели. Содержит переменные $owner = Pj$, $fid = PKIV\_FID$, $inst$, ca_pk — открытый ключ УЦ, которому Pj доверяет, массив записей $(pk, subj, notafter, rvkst, rvkstexpt)$, где pk — ключ субъекта $subj, notafter$ — срок годности ключа, $rvkst$ — состояние отзыва ключа ($REVOKED$ или OK), $rvkstexpt$ — срок годности значения состояния отзыва.

Обновление сертификата. После получения в коммуникационном входе команды $(M\_UPDCERT, msg = (CERT\_ID, pk, subj, notafter))$ передает $FSig[inst = ca\_pk]$ команду (M_VERIFY, msg). Если получен ответ $(M\_VERIFY\_R\_OK)$, то находит запись вида $(pk, subj, *, *)$ и обновляет поле $notafter$, после чего возвращает в коммуникационный выход сообщение $(M\_UPDCERT\_R\_OK)$. Если получен ответ $(M\_VERIFY\_R\_FAIL)$, то возвращает в коммуникационный выход сообщение $(M\_UPDCERT\_R\_FAIL)$.

Обновление состояния отзыва ключа. После получения в коммуникационном входе команды $(M\_UPDRVKST, msg = (RVKST\_ID, pk, rvkst, rvkstexpt))$ передает $FSig[inst = ca\_pk]$ сообщение (M_VERIFY, msg). Если от $FSig[inst = ca\_pk]$ получен ответ $(M\_VERIFY\_R\_FAIL)$, либо получен $(M\_VERIFY\_R\_OK)$, но ($curtime > rvkstexpt$ или $rvkst = OK$ и $pk.rvkst = REVOKED$), то возвращает $(M\_UPDRVKST\_R\_FAIL)$. Иначе обновляет поле $pk.rvkst := rvkst$ и возвращает в коммуникационный выход сообщение $(M\_UPDRVKST\_R\_OK)$.

Проверка подписи. После получения во внешнем входе команды $(M\_VERIFY, msg, subj)$ передает в коммуникационный выход сообщение $(M\_GETPK, subj)$. После получения в коммуникационном входе ответа (M_GETPK_R, pk) находит запись вида $(pk, subj, *, *)$, проверяет, что $curtime \leqslant notafter$, $rvkst \neq REVOKED$ и $curtime \leqslant rvkstexpt$. Если какое-либо из условий не выполнено, то возвращает во

внешний выход сообщение ($M\_VERIFY\_R\_FAIL$). Иначе передает $FSig[inst = = pk]$ сообщение (M_VERIFY, msg). Если от $FSig[inst = pk]$ получен ответ $(M\_VERIFY\_R\_OK)$, то возвращает во внешний выход $(M\_VERIFY\_R\_OK)$, если $(M\_VERIFY\_R\_FAIL)$, то $(M\_VERIFY\_R\_FAIL)$.

2.3. Удостоверяющий центр, **PKIca**:

Работает в $FSig$-гибридной модели. Содержит поля $cert\_dur$ — длительность действия сертификатов, $rvkst\_dur$ — длительность действия сообщений о состоянии отзыва, $mypk$ — свой открытый ключ, массив записей $(pk, subj, notafter, rvkst)$, где $rvkst$ принимает значения OK и $REVOKED$.

Выдача сертификата. После получения в коммуникационном входе сообщения $(M\_CERTREQ, msg = (CERTREQ\_ID, pk1, subj), pk2)$ проверяет, что $pk1.subj = = subj, pk1.rvkst = OK$, $pk2.subj = subj$, $curtime \leqslant pk2.notafter$, $pk2.rvkst = OK$. Если какое-либо из условий не выполнено, то возвращает в коммуникационный выход сообщение $(M\_CERTREQ\_R\_FAIL)$. Иначе передает $FSig[inst = pk2]$ сообщение (M_VERIFY, msg). Если от $FSig[inst = pk2]$ получен ответ $(M\_VERIFY\_R\_OK)$, то обновляет поле $pk1.notafter := curtime + cert\_dur$ и передает $FSig[inst = mypk]$ сообщение $(M\_SIGN, msg' = (CERT\_ID, pk1, pk1.subj, curtime + cert\_dur))$. После получения ответа $(SIGN\_R\_OK)$ возвращает в коммуникационный выход ответ $(M\_CERTREQ\_R\_OK, msg')$.

Отзыв ключа. После получения в коммуникационном входе сообщения $(M\_RVKREQ, msg = (RVKREQ\_ID, pk))$ проверяет, что есть запись вида $(pk, *, *, rvkst)$, где $rvkst \neq REVOKED$. Если такой записи нет, то возвращает в коммуникационный выход сообщение $(M\_RVKREQ\_R\_FAIL)$. Иначе передает $FSig[inst = pk]$ сообщение (M_VERIFY, msg). Если получен ответ $(M\_VERIFY\_R\_FAIL)$, то возвращает в коммуникационный выход $(M\_RVKREQ\_R\_FAIL)$. Иначе присваивает $pk.rvkst := REVOKED$ и возвращает в коммуникационный выход сообщение $(M\_RVKREQ\_R\_OK, msg')$.

Выдача статуса отзыва. После получения в коммуникационном входе сообщения $(M\_RVKSTREQ, pk)$ передает $FSig[inst = mypk]$ сообщение $(M\_SIGN, msg' = = (RVKST\_ID, pk, rvkst, curtime + rvkst\_dur))$. После получения ответа $(M\_SIGN\_R\_OK)$ возвращает в коммуникационный выход сообщение $(M\_RVKSTREQ\_R\_OK, msg')$.

П р и м е ч а н и е: При выдаче сертификата УЦ PKIca проверяет, что ключ в запросе на сертификат ранее не был подписан другим субъектом. Это замена механизма проверки обладания секретным ключом (POP) для случая, когда УЦ в системе один. Если вводить POP, то это нужно делать путем модификации ИФ $FSig$: к команде (Verify) добавится команда (Get_POP).

3. Формулировка абстракции совокупности протоколов PKI — ИФ «Цифровая подпись на базе PKI», $F(SigPKI)$:

Содержит поля $owner = Pi$, $fid = SIGNPKI\_FID$, значение которых фиксируется при запуске, и массив одноразово-записываемых записей вида (m, pk).

Генерация подписи. После получения во внешнем входе от участника Pi сообщения (M_SIGN, m) записывает в коммуникационный выход сообщение $(M\_GETMYPK)$.

После получения ответа $(M\_GETMYPK\_R\_OK, pk)$ сохраняет запись (m, pk) и возвращает во внешний выход сообщение $(M\_SIGN\_R\_OK)$.

Генерация подписи сообщений PKI. После получения в коммуникационном входе сообщения (M_SIGN, m, pk) проверяет, что первое поле m равно либо $CERTREQ\_ID$, либо $RVKREQ\_ID$. Если нет, то возвращает в коммуникационный выход сообщение $(M\_SIGN\_R\_FAIL)$. Иначе сохраняет запись (m, pk) и возвращает в коммуникационный выход ответ $(M\_SIGN\_R\_OK)$.

Проверка подписи. После получения во внешнем входе сообщения (M_VERIFY, m) записывает в коммуникационный выход сообщение (M_GETPK). Если получен ответ $(M\_GETPK\_R\_FAIL)$, то возвращает во внешний выход ответ $(M\_VERIFY\_R\_FAIL)$. Если получен ответ $(M\_GETPK\_R\_OK, pk)$, то находит сохраненную запись (m, pk) и возвращает во внешний выход (M_VERIFY_OK). Если такой записи нет, то возвращает $(M\_VERIFY\_R\_FAIL)$.

Проверка подписи злоумышленником. После получения в коммуникационном входе сообщения (M_VERIFY, m, pk) находит сохраненную запись (m, pk) и возвращает в коммуникационный выход (M_VERIFY_OK). Если такой записи нет, то возвращает $(M\_VERIFY\_R\_FAIL)$.

Отметим, что можно было не включать в $FSigPKI$ для каждой сохраненной строки текущий открытый ключ, а возложить задачу запоминания того, какой ключ соответствует каждому сообщению на злоумышленника, и тогда формулировка ИФ была бы более компактной. Однако в этом случае значительно возрастет сложность легального алгоритма работы злоумышленника, без которого протокол не будет работать. Наша главная цель — свертывание большого протокола в компактную ИФ, однако легальный алгоритм работы злоумышленника, по возможности, тоже должен быть компактным.

4. Пусть A' — «черный ящик», реализующий алгоритм работы злоумышленника идеальной модели. Зададим следующий алгоритм симуляции злоумышленника идеальной модели, S:

При инициализации S имеет значения массивов переменных $pv0.(pk, expt, rvkst)$, $pca0.(pk, subj, expt)$, переменных $po0.owner$,$pca0.cert\_dur$, $pca0.rvkst\_dur$.

A' посылает $PKIo$ $(M\_UPDMYCERT, pk1, pk2)$: S посылает $FSigPKI$ сообщение $(M\_SIGN, msg = (CERTREQ\_ID, pk1, po0.owner), pk2)$, ожидает ответ $(M\_SIGN\_R\_OK)$, возвращает A' сообщение $(M\_UPDMYPK\_R\_OK, msg)$.

A' посылает $PKIca(M\_CERTREQ, msg = (CERTREQ\_ID, pk1, subj), pk2)$: Если $pca0.pk1.subj \neq subj$, или $curtime > pca0.pk2.expt$, или $curtime \geqslant pca0.pk2.rvkst$, или $pca0.pk2.subj \neq subj$, то возвращает A' $(M\_CERTREQ\_R\_FAIL)$. Иначе посылает $FSigPKI$ сообщение $(M\_VERIFY, msg, pk2)$. Если от $FSigPKI$ получен ответ $(M\_VERIFY\_R\_OK)$, то возвращает A' сообщение $(M\_CERTREQ\_R\_OK, msg' = (CERT\_ID, pk1, subj, curtime + pca0.cert\_dur))$ и запоминает запись $(pk1, subj, curtime + pca0.cert\_dur)$. Если получен ответ $(M\_VERIFY\_R\_FAIL)$, то возвращает A' сообщение $(M\_CERTREQ\_R\_FAIL)$.

A' посылает $PKIca$ $(M\_RVKREQ, msg = (RVKREQ\_ID, pk))$: S посылает $FSigPKI$ сообщение (M_VERIFY, msg, pk). Если от $FSigPKI$ получен ответ $(M\_VERIFY\_R\_OK)$, то присваивает $pca0.pk.rvkst := REVOKED$ и воз-

вращает A' сообщение ($M\_RVKREQ\_R\_OK$). Иначе возвращает A' сообщение ($M\_RVKREQ\_R\_FAIL$).

A' посылает $PKIca$ ($M\_RVKSTREQ, pk$): S возвращает A' сообщение ($M\_RVKREQ\_R\_OK$, $RVKST\_ID$, $msg' = (pk, pca0.pk.rvkst, curtime + pca0.rvkst\_dur)$), запоминает msg' и возвращает A' сообщение ($M\_RVKSTREQ\_R\_OK, msg'$).

A' посылает $PKIv$ ($M\_UPDCERT, msg = (CERT\_ID, pk, subj, expt)$): Если у S запись $(pk, subj, expt)$ не была сохранена, либо $pv0.pk.subj \neq subj$, то возвращает A' ($M\_UPDCERT\_R\_FAIL$). Иначе обновляет значение $pv0.pk.expt := expt$ и возвращает A' ($M\_UPDCERT\_OK$).

A' посылает $PKIv$ ($M\_UPDRVKST$, $msg = (RVKST\_ID, pk, rvkst, rvkstexpt)$): Если у S запись $(pk, rvkst, rvkstexpt)$ не была сохранена, либо $pv0.pk.rvkst = OK$ и $rvkst = REVOKED$, либо $curtime > rvkstexpt$, то возвращает A' ($M\_UPDRVKST\_R\_FAIL$). Иначе обновляет значение $PKIv.pk.rvkst := rvkst$ и возвращает A' ($M\_UPDCERT\_OK$).

$FSigPKI$ посылает S (M_GETPK): S посылает A' (M_GETPK), ожидает ответ ($M\_GETPK\_R\_OK, pk$). Если $curtime \leqslant pk.expt$ и $pk.rvkst = OK$, то возвращает $FSigPKI$ ответ ($M\_GETPK\_R\_OK, pk$), иначе ($M\_GETPK\_R\_FAIL$).

$FSigPKI$ посылает S ($M\_GETMYPK$): S посылает A' ($M\_GETMYPK$), ожидает ответ ($M\_GETMYPK\_R\_OK, pk$) и возвращает его $FSigPKI$.

3. Эксперименты с верификатором моделей NuSMV

Сокращение мощности модели

Приведенные выше формулировки алгоритмов участников протокола, алгоритмов идеальных функциональностей и алгоритма симуляции злоумышленником не ограничены в количестве хранимых внутри себя сообщений. Для возможности задания спецификаций модели на верификаторе моделей необходимо привести модель в конечный вид. Для сокращения мощности модели были использованы следующие приемы:

1. Число копий $FSig$ — 4. Массив сообщений, хранимых каждой копией $FSig$, содержит три элемента. Вычисление модели останавливается, когда производится попытка записи сообщения в уже заполненную копию ИФ $FSig$.

2. Число копий $PKIo$ — 1. $PKIo$ содержит два собственных ключа.

3. Число копий $PKIca$ — 1. $PKIca$ имеет один собственный ключ.

4. Число копий $PKIv$ — 1. В $PKIv$ фиксирован открытый ключ УЦ.

5. Для сокращения вариантов выбора использовались следующие алгоритмы генерации сообщений злоумышленником реальной модели A (в нотации NuSMV):

alg1 (полный перебор, моделирование недетерминированного алгоритма работы A): $init(m[0]) := \{0,1\}$; $init(m[1]) := \{0,1,2,3\}$; $init(m[2]) := \{0,1\}$; $init(m[3]) := \{0,1,2,3\}$; $init(m[4]) := \{0,1\}$; $next(m[0]) := \{0,1\}$; $next(m[1]) := \{0,1,2,3\}$; $next(m[2]) := \{0,1\}$; $next(m[3]) := \{0,1,2,3\}$; $next(m[4]) := \{0,1\}$;

alg2: $init(m[0]) := 0$; $init(m[1]) := 0$; $init(m[2]) := 0$; $init(m[3]) := 0$; $init(m[4]) := 0$; $next(m[0]) := m[2]$; $next(m[1]) := \{0,1,2,3\})$; $next(m[2]) := \{0,1\}$; $next(m[3]) := m[1]$; $next(m[4]) := \{0,1\}$;

alg3: $init(m[0]) := 1$; $init(m[1]) := 0$; $init(m[2]) := 1$; $init(m[3]) := 0$; $init(m[4]) := 0$; $next(m[0]) := m[2]$; $next(m[1]) := \{0,1,2,3\}$; $next(m[2]) := \{0,1\}$; $next(m[3]) := m[1]$; $next(m[4]) := \{0,1\}$;

alg4: $init(m[0]) := \{0,1\}$; $init(m[1]) := \{0,1,2,3\}$; $init(m[2]) := \{0,1\}$; $init(m[3]) := \{0,1,2,3\}$; $init(m[4]) := \{0,1\}$; $next(m[0]) := m[2]$; $next(m[1]) := m[0] + m[2]$; $next(m[2]) := \{0,1\}$; $next(m[3]) := m[1]$; $next(m[4]) := \{0,1\}$;

alg5: $init(m[0]) := \{0,1\}$; $init(m[1]) := \{0,1,2,3\}$; $init(m[2]) := \{0,1\}$; $init(m[3]) := \{0,1,2,3\}$; $init(m[4]) := \{0,1\}$; $next(m[0]) := a.m[2]$; $next(a.m[1]) := a.m[3]$; $next(a.m[2]) := a.m[0]$; $next(a.m[4]) := \{0,1\}$;

6. Использовался следующий алгоритм изменения переменной текущего времени $curtime$:

$next(curtime) := case$

$(token = CHECK\_SIM\_E | token = CHECK\_SIM\_A) \& curtime = 0 : 1;$

$(token = CHECK\_SIM\_E | token = CHECK\_SIM\_A) \& curtime = 1 : 2;$

$1 : curtime; esac;$

Результаты экспериментов

В результате просчитывания модели на верификаторе моделей NuSMV (в указанных выше вариантах сокращения мощности) нарушений симуляции, обеспечиваемой предложенным алгоритмом работы злоумышленника A, выявлено не было. Ниже в таблице представлена зависимость времени вычисления и объема модели в памяти от алгоритмов работы злоумышленника реальной модели A (для AMD Sempron 2800+, 512 Mb RAM, Linux 2.4.18):

№	A	NumAll	NumReach	MemGo	MemCheck	TimeGo	TimeCheck
1	alg1	$2^{265.343}$	*	*	*	*	*
2	alg2	$2^{265.343}$	348982	196	410	25:22	45:32
3	alg3	$2^{265.343}$	180033	157	380	12:28	23:28
4	alg4	$2^{265.343}$	15487	56	90	1:14	1:41
5	alg5	$2^{265.343}$	9708	53	81	0:53	1:15

Здесь A — алгоритм работы злоумышленника реальной модели, NumAll — число всех состояний модели, NumReach — число достижимых состояний модели, MemGo — размер модели в памяти (фаза *go*, в мегабайтах), MemCheck — размер модели в памяти(фаза $compute\_reachable + check\_spec$, в мегабайтах), TimeGo — время вычисления (фаза *go*, мин:с), TimeCheck — время вычисления (фаза $compute\_reachable + check\_spec$, мин:с), * — исчерпан объем оперативной памяти через 1 час.

Выводы

1. Был сформулирован вариант компактной детерминированной модели (идеальной функциональности) семейства криптографических протоколов, реализующих сервис «Цифровая подпись на базе PKI», а также алгоритм симуляции злоумышленника идеальной модели, необходимый для проверки корректности данной формулировки в рамках UC-моделей.

2. Эксперименты с верификатором моделей NuSMV позволили оценить сложность протокола, для которого поиск формулировки ИФ может быть автоматизирован с помощью современных верификаторов моделей.

3. По результатам экспериментов, сокращение модели вычислений для верификаторов моделей рекомендуется проводить в 2 этапа. На первом этапе из реальной модели убираются все функции, которые, на взгляд исследователя, не связаны с интересующими его свойствами полной модели. Полученная сокращенная модель описывается на языке спецификаций верификатора. Цель сокращения модели на первом этапе

— уместить ее символическое представление в памяти (не должно занимать больше, чем 1/2 объема оперативной памяти — см. таблицу результатов экспериментов). На втором этапе вводятся искусственные ограничения на перебираемые верификатором состояния модели. Например, устраняется логическое дублирование ветвей вычислений, когда изменение порядка некоторых действий в системе на результат анализа не влияет. Если в результате работы на втором этапе модель становится быстропросчитываемой, тогда в модель возвращаются некоторые функции, сокращенные на первом этапе, после чего снова проводится второй этап, и т. д.

ЛИТЕРАТУРА

1. *NSA Cross-Organizational CAPI Team.* Security Service API: Cryptographic API Recommendation. Second Edition. // Government Information Technology Issues, 1996.
2. RFC 2246. The TLS Protocol Version 1.0. January 1999.
3. *Canetti R.* Universally Composable Security: A New paradigm for Cryptographic Protocols // 42nd FOCS, 2001.
4. *Lincoln P., Mitchell J., Mitchell M., Scedrov A.* Probabilistic Polynomial-Time Equivalence and Security Analysis. `http://citeseer.ist.psu.edu.` 1999.
5. *Mateus P., Mitchell J., Scedrov A.* Composition of Cryptographic Protocols in a Probabilistic Polynomial-Time Process Calculus. `http://citeseer.ist.psu.edu.` 2002.
6. *Canetti R.* Universally Composable Signature, Certification, and Authentication. `http://citeseer.ist.psu.edu.` 2004.
7. *Canetti R., Krawczyk H.* Universally Composable Notions of Key Exchange and Secure Channels. `http://citeseer.ist.psu.edu.` 2002.
8. *Canetti R., Rabin T.* Universal Composition with Join State. `http://eprint.iacr.org.` 2002.
9. *BackesM., Pfitzmann B., Waidner M.* A Universally Composable Cryptographic Library. `http://citeseer.ist.psu.edu.` 2003.
10. *Pfitzmann B., Schunter M., Waidner M.* Secure Reactive Systems. // IBM Research Report RZ 3206(93252), IBM Research Division, Zurich, Feb. 2000.
11. *Backes M., Jacobi C.* Cryptographically Sound and Machine-Assisted Verification of Security Protocols // In Proc. 20th Annual STACS, volume 2607 of Lecture Notes in Computer Science, pages 675-686. Springer, 2003.
12. `http://pvs.csl.sri.com` — Prototype Verification System (PVS).
13. *Cimatti A., Clarke E., Giunchiglia F., Roveri M.* NuSMV: a new symbolic model checker. // Software Tools for Technology Transfer, 1998.
14. *Cheetancheri S.* Our experiments with NuSMV. `http://shasta.cs.ucdavis.edu/eas.` 2004.
15. *Прокопьев С. Е.* Адаптация модели безопасности Канетти для использования в качестве архитектуры подсистемы криптографических сервисов // Проблемы информационной безопасности. Компьютерные системы. 2005. № 5.