Internet Engineering Task Force (IETF)
Request for Comments: 6063
Category: Standards Track
ISSN: 2070-1721

A. Doherty
RSA, The Security Division of EMC
M. Pei
VeriSign, Inc.
S. Machani
Diversinet Corp.
M. Nystrom
Microsoft Corp.

December 2010

Протокол обеспечения динамического обмена симметричными ключами

Dynamic Symmetric Key Provisioning Protocol (DSKPP)

Аннотация

DSKPP представляет собой клиент-серверный протокол для инициализации (и настройки) симметричных ключей на локальных и удалённых криптографических модулях. Протокол может работать с использованием или без использования функциональности секретных ключей в криптомодулях, а также с организацией инфраструктуры открытых ключей или без таковой.

Два варианта протокола поддерживают множество сценариев применения. В четырехпроходном варианте ключи выполняется взаимная генерация ключей обеспечивающим сервером и криптомодулем - созданные ключи не передаются по проводам или эфиру. Двухпроходный вариант обеспечивает безопасную и эффективную загрузку и установку в криптомодули заранее созданных симметричных ключей.

Статус документа

Этот документ содержит проект стандарта Internet (Internet Standards Track).

Документ является результатом работы IETF и выражает согласованное мнение сообщества IETF. Документ был представлен на публичное рассмотрение и одобрен для публикации IESG. Дополнительная информация о стандартах Internet доступна в разделе 2 RFC 5741.

Информацию о текущем статусе документа, обнаруженных ошибках и способах обратной связи можно получить, воспользовавшись ссылкой http://www.rfc-editor.org/info/rfc6063.

Оглавление

1. Введение

Криптографические системы с симметричными ключами (т. е., системы, обеспечивающие засвидетельствования типа однократных паролей или механизмов запрос-отклик) обеспечивают преимущества с точки зрения производительности и эксплуатации по сравнению с системами на базе открытых ключей. Для использования таких систем требуется механизм обеспечения симметричных ключей, обеспечивающий функциональность, эквивалентную возможностям протокола управления сертификатами CMP[1] [RFC4210] и CMC[2] [RFC5272] в инфраструктуре открытых ключей PKI[3].

Традиционно криптографические модули обеспечиваются ключами в процессе производства и ключи импортируются в криптографический сервер с использованием, например, компакт-диска (CD-ROM) из комплекта поставки устройства. Некоторые производители предлагают также фирменные протоколы обеспечения ключами, которые зачастую не документируются в открытом виде (протокол CT-KIP[4] [RFC4758] является единственным исключением).

В документе описан DSKPP - клиент-серверный протокол обеспечения симметричными ключами, используемый между криптомодулем (соответствует клиенту DSKPP) и сервером обеспечения ключами (соответствует серверу DSKPP).

DSKPP обеспечивает открытый и совместимый механизм инициализации и настройки симметричных ключей в криптомодулях, доступных через Internet. Описание основано на информации, приведённой в [RFC4758], и включает специфические расширения типа засвидетельствования пользователей и поддержки формата [RFC6030] для передачи ключевого материала.

DSKPP поддерживает два основных варианта. 4-проходный вариант обеспечивает возможность предоставления симметричных ключей, включающую вклад в генерацию случайных значений на стороне клиента и сервера. 2-проходному варианту нужен один круговой обход вместо двух и он позволяет обеспечить заданный сервером ключ.

1.1. Ключевые слова

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе должны интерпретироваться в соответствии с документом [RFC2119].

1.2. Поддержка версий

Обеспечение выполняется с использованием синтаксиса явного указания номера версии. В настоящее время поддерживается только версия «1.0».

Целью введения номера версии протокола является обеспечение механизма, с помощью которого изменения, требуемые для криптографических алгоритмов (например, SHA-256) и атрибутов (например, размер ключа) могут быть внесены без нарушения работы существующих реализаций. Кроме того, это позволит выводить устаревшие реализации из эксплуатации без нарушения работы новых версий протокола.

Номер версии DSKPP имеет вид <major>.<minor>. Значения старшего и младшего номеров **должны** трактоваться как раздельные целые числа и каждое из этих чисел **может** инкрементироваться более чем на 1. Таким образом, DSKPP 2.4 будет иметь более низкую (младшую) версию по сравнению с DSKPP 2.13, а та, в свою очередь, будет ниже DSKPP 12.3. Ведущие нули (например, DSKPP 6.01) **должны** игнорироваться получателями, а установка их **недопустима**.

Значение старшего номера версии следует увеличивать только для случаев, когда алгоритмы защиты или форматы данных изменены столь существенно, что старые версии не могут взаимодействовать с новой версией (например, отказ от поддержки считавшегося ранее обязательным алгоритма, который был сочтён небезопасным). Значение младшего номера версии показывает наличие новых возможностей (например, введение новой опции алгоритма) и **должно** игнорироваться объектами с меньшим значением младшего номера версии, но использоваться для информационных целей объектами с большим значением младшего номера версии протокола.

1.3. Идентификаторы пространства имён

В документе применяются идентификаторы ресурсов URI[5] [RFC3986] для указания ресурсов, алгоритмов и семантики.

1.3.1. Определённые здесь идентификаторы

Пространство имён XML [XMLNS] URI для версии 1.0 протокола DSKPP представляет собой

```
"urn:ietf:params:xml:ns:keyprov:dskpp"
```

Ссылки на элементы схемы DSKPP используют префикс dskpp, но может применяться и другой префикс.

1.3.2. Идентификаторы, определённые в связанных спецификациях

В этом документе используются элементы, определённые в пространстве имён PSKC[6] [RFC6030], которые представлены с префиксом pskc и объявлены как

```
xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
```

1.3.3. Ссылочные идентификаторы

Кроме того, представленный в документе синтаксис DSKPP использует идентификаторы алгоритмов, определённые в пространстве имён XML Signature [XMLDSIG]:

```
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
```

Ссылки на идентификаторы алгоритмов в пространстве имён XML Signature используют префикс ds.

[1]Certificate Management Protocol.
[2]Certificate Management over CMS - управление сертификатами через CMS.
[3]Public Key Infrastructure.
[4]Cryptographic Token Key Initialization Protocol - протокол ключей криптомаркеров.
[5]Uniform Resource Identifier.
[6]Portable Symmetric Key Container - контейнер переносимого симметричного ключа.

2. Терминология

2.1. Определения

Ниже приведены определения терминов, используемых в данном документе. Использование этих терминов в других документах может быть иным.

Authentication Code (AC) - код засвидетельствования
Код аутентификации пользователя представляет собой строку шестнадцатеричных символов, известную устройству и серверу и содержащую, по крайней мере, идентификатор клиента и пароль. Комбинация ClientID/password используется лишь однократно и может иметь ограниченный срок действия, по истечении которого она отбрасывается.

Authentication Data (AD) - данные засвидетельствования
Данные засвидетельствования пользователя, полученные из кода засвидетельствования (AC).

Client ID - идентификатор клиента
Идентификатор, используемый сервером DSKPP для нахождения на сервере реального имени пользователя или идентификатора учётной записи. Идентификатор может представлять собой короткое случайное значение, никак не связанное с реальным именем пользователя.

Cryptographic Module - криптографический модуль (криптомодуль)
Компонента приложения, которая обеспечивает функциональность шифрования с симметричным ключом.

Device - устройство
Физическая часть оборудования или программной системы, содержащая криптомодули с симметричным ключом.

Device ID (DeviceID) - идентификатор устройства
Уникальный идентификатор для устройства, содержащего криптомодуль (например, мобильный телефон).

DSKPP Client - клиент DSKPP
Управляет обменом данными между криптомодулем с симметричным ключом и сервером DSKPP.

DSKPP Server - сервер DSKPP
Сервер обеспечения симметричными ключами, участвующий в работе протокола DSKPP.

DSKPP Server ID (ServerID) - идентификатор сервера DSKPP
Уникальный идентификатор сервера DSKPP.

Key Agreement - согласование ключа
Протокол организации ключа, в котором две или более стороны могут согласовать ключ так, чтобы обе могли влиять на результат.

Key Confirmation - подтверждение ключа
Обеспечение правомочным участникам протокола организации ключа гарантии того, что предполагаемый получатель разделяемого ключа действительно владеет разделяемым ключом.

Key Issuer - эмитент ключа
Организация, выпускающая симметричные ключи для конечных пользователей.

Key Package (KP) - ключевой пакет
Объект, инкапсулирующий в себе симметричный ключ и его конфигурационные параметры.

Key ID (KeyID) - идентификатор ключа
Уникальный идентификатор симметричного ключа.

Key Protection Method (KPM) - метод защиты ключа
метод транспортировки ключа в процессе двухпроходного DSKPP.

Key Protection Method List (KPML) - список методов защиты ключей
Список методов защиты ключей, поддерживаемых криптомодулем.

Key Provisioning Server - сервер обеспечения ключами
Система управления жизненным циклом ключей, которая обеспечивает эмитенту возможность обеспечения ключей для криптомодулей, размещённый на устройствах конечных пользователей.

Key Transport - транспортировка ключей
Процедура организации ключа, с помощью которой сервер DSKPP выбирает и шифрует ключевой материал, а потом передаёт его клиенту DSKPP [NIST-SP800-57].

Key Transport Key - ключ для транспортировки ключа
Секретный ключ, хранящийся в криптомодуле. Этот ключ образует пару с открытым ключом клиента DSKPP, применяемым сервером DSKPP для шифрования ключевого материала при доставке ключа [NIST-SP800-57]

Key Type - тип ключа
Тип криптографических методов симметричного ключа, для которых этот ключ будет использоваться (например, аутентификация OATH HOTP[1] или RSA SecurID, шифрование AES и т. п.).

Key Wrapping - упаковка ключа
Метод шифрования ключей для транспортировки [NIST-SP800-57].

Key Wrapping Key - ключ для упаковки ключа
Ключ шифрования симметричного ключа для упаковки последнего [NIST-SP800-57]

Keying Material - ключевой материал
Данные (например, ключ и параметры конфигурации), требуемые для организации и поддержки отношений с криптографическими преобразованиями [NIST-SP800-57]

Manufacturer's Key - ключ производителя
Уникальный первичный ключ (например, смарт-карта), установленный в оборудование при его производстве. При наличии такого ключа он может использоваться криптомодулем для создания секретных ключей.

Protocol Run - цикл работы протокола
Полное исполнение DSKPP, включающее один (двухпроходный) или два (четырехпроходный) обмена.

Security Attribute List (SAL) - список атрибутов защиты
Данные, включающие версию DSKPP, вариант DSKPP (двух- или четырехпроходный), форматы ключевых пакетов, типы ключей и криптографические алгоритмы, которые может поддерживать криптомодуль.

2.2. Обозначения

|| конкатенация (слияние) строк.

[1]Open AUTHentication HMAC-Based One-Time Password - аутентификация с одноразовым ключом на базе HMAC.

[x]	необязательный элемент x.
A ^ B	операция исключающее-ИЛИ над строками A и B, имеющими одинаковые размеры.
<XMLElement>	типографическое соглашение, используемое в теле текста.
DSKPP-PRF(k,s,dsLen)	псевдослучайная функция с ключом.
E(k,m)	шифрование m с использованием ключа k.
K	ключ, используемый для шифрования R_C (K_SERVER или K_SHARED) при расчёте MAC или DSKPP_PRF.
K_AC	секретный ключ, получаемый из кода засвидетельствования (AC) и используемый в целях засвидетельствования.
K_MAC	секретный ключ, получаемый в процессе обмена DSKPP, для использования при подтверждении ключа.
K_MAC'	второй секретный ключ, используемый для засвидетельствования сервера.
K_PROV	первичный ключ обеспечения, из которого создаются два ключа - K_TOKEN и K_MAC
K_SERVER	открытый ключ сервера DSKPP, который используется для шифрования R_C в четырехпроходном варианте.
K_SHARED	секретный ключ, который заранее известен клиенту и серверу DSKPP; используется для шифрования R_C в четырехпроходном варианте.
K_TOKEN	секретный ключ, организуемый в криптомодуле с использованием DSKPP.
R	псевдослучайное значение, выбранное клиентом DSKPP и используемое для расчёта MAC.
R_C	псевдослучайное значение, выбранное клиентом DSKPP и используемое в качестве входных данных для генерации K_TOKEN.
R_S	псевдослучайное значение, выбранное сервером DSKPP и используемое в качестве входных данных для генерации K_TOKEN.
URL_S	адрес сервера DSKPP в формате URL.

2.3. Сокращения

AC	Authentication Code - код засвидетельствования.
AD	Authentication Data - данные засвидетельствования.
DSKPP	Dynamic Symmetric Key Provisioning Protocol - протокол динамического обеспечения симметричными ключами.
HTTP	Hypertext Transfer Protocol - протокол передачи гипертекста.
KP	Key Package - ключевой пакет.
KPM	Key Protection Method - метод защиты ключа.
KPML	Key Protection Method List - список методов защиты ключа.
MAC	Message Authentication Code - код засвидетельствования ключа.
PC	Personal Computer - персональный компьютер.
PDU	Protocol Data Unit - модуль данных протокола.
PKCS	Public Key Cryptography Standards - стандарты шифрования с открытым ключом.
PRF	Pseudorandom Function - псевдослучайная функция.
PSKC	Portable Symmetric Key Container - переносимый контейнер с симметричным ключом.
SAL	Security Attribute List - список атрибутов защиты (см. параграф 2.1).
TLS	Transport Layer Security - защита транспортного уровня.
URL	Uniform Resource Locator - однотипный указатель ресурсов.
USB	Universal Serial Bus - универсальная последовательная шина.
XML	eXtensible Markup Language - расширенный язык разметки (текста).

3. Обзор DSKPP

В последующих параграфах дан обзор внутреннего устройства протокола и взаимодействия с внешними программами обеспечения. Сценарии использования рассмотрены в Приложении A.

3.1. Элементы протокола

Транзакция обеспечения ключами DSKPP включает три элемента:

Server - сервер
Сервер обеспечения DSKPP.

Cryptographic Module - криптомодуль
Криптографический модуль, которому обеспечиваются симметричные ключи (например, маркер аутентификации).

Client - клиент

Клиент DSKPP, управляющий обменом информацией между криптомодулем и сервером обеспечения ключами. Основным синтаксисом является XML [XML] и его уровни для транспортных механизмов, таких, как HTTP [RFC2616] и HTTP Over TLS [RFC2818]. Несмотря на то, что крайне желательна защита всего обмена данными между клиентом и сервером DSKPP с помощью транспортных механизмов, обеспечивающих защиту конфиденциальности и целостности (таких, как HTTP over TLS), такой защиты не достаточно для обмена симметричными ключами между сервером и криптомодулем. Протокол DSKPP создан для того, чтобы обеспечить разработчикам возможность выполнения требований по защите обмена ключами.

Сервер обменивается информацией только с клиентом. При рассмотрении серверов клиента и криптомодуль можно считать единым целым.

С точки зрения защиты клиента, однако, клиент и криптомодуль представляют собой разные логические объекты, которые в отдельных реализациях могут быть разделены и на физическом уровне.

Предполагается, что на устройстве будет размещаться приложение, вышележащее по отношению к криптомодулю, и это приложение будет управлять обменом данными между клиентом DSKPP и криптомодулем. Способ передачи элементов DSKPP между приложением и криптомодулем прозрачен для сервера DSKPP. Один из методов такой передачи описан в [CT-KIP-P11].

3.2. Базовый обмен DSKPP

3.2.1. Засвидетельствование пользователя

В потоке сообщений DSKPP пользователь обретает новое оборудование или программное приложение со встроенным криптомодулем. Целью DSKPP является обеспечение единого симметричного ключа и связанной с ним информации криптомодулю и серверу управления ключами, а также привязка этого ключа к корректному имени (или учётной записи) пользователя на сервере. Для выполнения этих операций сервер DSKPP **должен** аутентифицировать (получить засвидетельствование) пользователя, чтобы обрести уверенность в том, что тот уполномочен на получение нового ключа.

Засвидетельствование пользователя выполняется в рамках самого протокола «после» того, как клиент DSKPP инициирует первое сообщение. В этом случае клиент DSKPP **должен** иметь доступ к URL сервера DSKPP.

Кроме того, засвидетельствование пользователя может выполняться с помощью web-сервиса DSKPP или иного web-приложения «до» первого обмена сообщениями. В этом случае сервер DSKPP должен Server MUST запускать механизм (триггер) инициирования клиентом DSKPP первого сообщения в рамках протокольного взаимодействия.

3.2.2. Протокол, инициированный клиентом DSKPP

В приведённом ниже примере клиент DSKPP сначала инициирует DSKPP, а потом выполняется аутентификация пользователя с применением Client ID и кода засвидетельствования (AC).

Перед началом работы DSKPP:

- Код засвидетельствования (AC) генерируется сервером DSKPP и доставляется через отдельный доверенный канал (например, в форме бумажного документа сотрудником департамента ИТ).
- Пользователь обычно вводит значения Client ID и AC вручную (возможно, что устройство имеет лишь цифровую клавиатуру). По этой причине идентификатор и код зачастую представляют собой короткие числовые значения (например, 8 десятичных цифр). Однако сервер DSKPP может генерировать любые значения на своё усмотрение.
- Клиенту DSKPP требуется URL [RFC3986] сервера DSKPP (этот указатель не является специфическим для пользователя или секретным и может быть задан заранее) и набор доверенных привязок для верификации сертификата сервера.
- Для пользователя должна иметься учётная запись с идентификатором и долгосрочное имя в системе (или иной идентификатор учётной записи), с которым будет связан маркер. Сервер DSKPP будет использовать значение Client для поиска соответствующего кода AC, позволяющего аутентифицировать пользователя.

На этапе 1 клиент организует соединение TLS идентифицирует сервер (т. е., проверяет сертификат и сравнивает имя хоста в URL с полученным сертификатом), как описано в параграфе 3.1 [RFC2818].

После этого клиент и сервер DSKPP обмениваются сообщениями DSKPP, которые передаются по протоколу HTTPS. В этих сообщениях:

- Клиент и сервер согласуют криптографические алгоритмы, которые они хотят применять, алгоритмы, поддерживаемые для защиты сообщений DSKPP, и другие детали DSKPP.
- Клиент передаёт серверу значение Client ID и подтверждает, что ему известен соответствующий код AC.
- Клиент и сервер согласуют секретный ключ (ключ маркера или K_TOKEN); в зависимости от согласованного варианта протокола это будет свежий ключ, полученный в процессе работы DSKPP (4-проходный вариант, включающий 4 сообщения DSKPP) или сгенерированный сервером (возможно, существующий на сервере) и переданный клиенту (2-проходный вариант, в котором передаётся 2 сообщения DSKPP).
- Сервер передаёт ключевой пакет клиенту. Пакет в 2-проходном варианте включает только ключ, а в 4-проходном добавляются атрибуты, влияющие на то, как предоставленный ключ будет впоследствии применяться криптомодулем и криптографическим сервером. Точное содержимое пакета зависит от криптографического алгоритма (например, для алгоритма с одноразовым паролем, поддерживающего значения OTP[1] переменного размера одним из атрибутов в ключевом пакете будет размер OTP).

[1]One-time password - одноразовый пароль.

После успешного завершения работы протокола криптомодули хранят содержимое ключевого пакета. Кроме того, обеспечивающий сервер DSKPP хранит содержимое ключевого пакета с указанием криптосервера и связывает это с корректным именем пользователя. Пользователь теперь может применять своё устройство для операций с симметричным ключом.

Точное распределение операций между парой криптомодуль - клиент DSKPP и парой сервер обеспечения - сервер DSKPP не задаётся этим документом. На рисунке 1 показан один из возможных вариантов, но он предназначен лишь для иллюстрации.

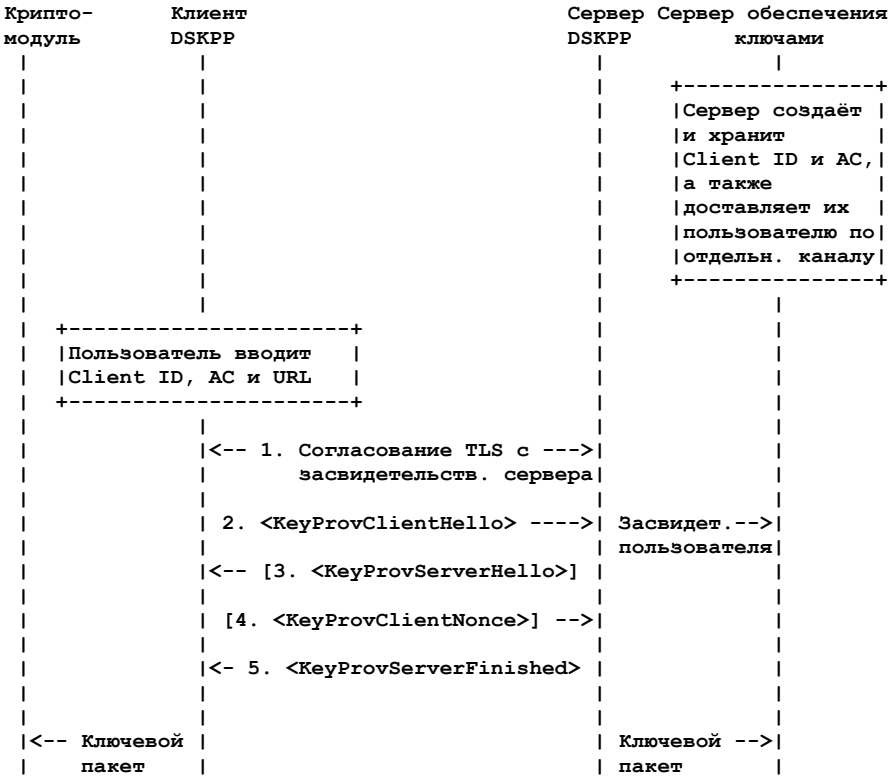

Рисунок 1. Базовый обмен DSKPP.

3.2.3. Протокол, включенный сервером DSKPP

В первом потоке сообщений (предыдущий параграф) значения Client ID и AC доставлялись клиенту с использованием некого независимого канала (например, на бумаге).

Во втором потоке сообщений пользователь сначала аутентифицирует web-сервер (например, страница самообслуживания на внутреннем сервере департамента ИТ), используя обычный web-браузер и некие существующие «верительные грамоты».

После этого пользователь запрашивает (выбирая ссылку или заполняя форму) предоставление нового ключа для криптомодуля. Web-сервер будет возвращать в ответ сообщение <KeyProvTrigger>, содержащее значения Client ID, AC и URL сервера DSKPP. Эта информация требуется также серверу DSKPP; взаимодействие серверов web и DSKPP выходит за пределы этого документа.

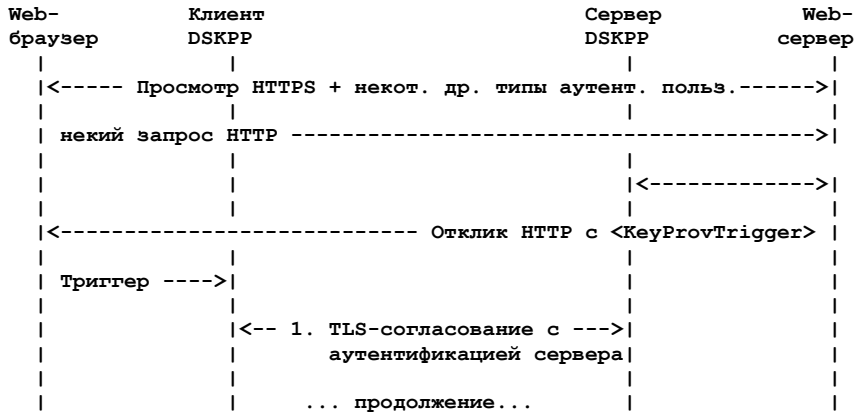

Рисунок 2. Обмен DSKPP с Web-аутентификацией.

Сообщение <KeyProvTrigger> передаётся в отклике HTTP и этот отклик маркируется MIME-типом application/dskpp+xml. Предполагается, что web-браузер настроен на распознавание этого типа MIME; браузер запускается клиентом DSKPP и обеспечивает того сообщением <KeyProvTrigger>.

После этого клиент DSKPP обращается к серверу DSKPP и использует значения Client ID и AC (из сообщения <KeyProvTrigger>) так же, как было описано для первого потока сообщений.

3.2.4. Варианты

Как отмечено в предыдущем параграфе, после старта протокола клиент и сервер **могут** выбрать 2-х или 4-проходный обмен сообщениями. Эти варианты протокола подходят для разных сценариев развёртывания. Основная разница между этими вариантами заключается в том, что 2-проходный вариант поддерживает доставку существующего ключа в

криптомодуль, а 4-проходный включает генерацию ключа и его согласование «на лету». В любом случае оба варианта протокола поддерживают возможности алгоритма за счёт согласования механизмов шифрования и типов ключей на начальном этапе работы.

3.2.4.1. Критерии использования четырехпроходного варианта

Четырехпроходный вариант требуется использовать при выполнении одного или нескольких перечисленных условий.

- Политика требует, чтобы обе участвующие в работе протокола стороны вносили вклад в энтропию для ключа. Реализация такой политики снижает риск раскрытия ключа в процессе обеспечения, поскольку ключ генерируется на основе взаимного соглашения сторон без передачи через кабель или эфир. Снижается также риск раскрытия после обеспечения ключа, поскольку ключ не имеет уязвимости для атак на одну из точек системы.
- Криптомодуль не поддерживает работы с секретным ключом[1].
- Криптомодуль размещается на устройстве, которое не имеет ни заданного производителем, ни какого-либо иного заранее известного сторонам ключа (например, смарт-карты или SIM[2]-карты), ни клавиатуры, которая может послужить для ввода пароля (типа клавиатуры мобильного телефона).

3.2.4.2. Критерии использования двухпроходного варианта

Двухпроходный вариант требуется использовать при выполнении одного или нескольких перечисленных условий.

- Должны быть обеспечены имеющиеся (унаследованные) ключи за счёт транспортировки их в криптомодуль.
- Криптомодуль размещается на устройстве, которое имеет заданный производителем ключ (такой ключ может находиться на смарт-карте) или иной вариант заранее известного сторонам ключа (такой ключ может размещаться на SIM-карте) и поддерживает операции с приватным ключом.
- Криптомодуль размещается на устройстве, которое имеет встроенную клавиатуру, позволяющую пользователю ввести парольную фразу, полезную при создании ключа для упаковки ключей при распространении ключевого материала.

3.3. Коды состояния

При отправке или получении сообщения, в котором значение атрибута Status отлично от Success или Continue, принятое по умолчанию поведение (если ниже явно не указано иное) заключается в том, что клиент и сервер DSKPP **должны** незамедлительно прервать работу DSKPP. Сервер и клиент DSKPP **должны** удалить все секретные значения, сгенерированные в процессе работы отказавшего DSKPP. Идентификаторы сессий после успешного или аварийного завершения работы **могут** сохраняться в целях детектирования повторного использования, но эти сохранённые идентификаторы **недопустимо** использовать для последующих сеансов работы протокола.

По возможности клиенту DSKPP **следует** передавать пользователю подходящее сообщение об ошибке.

Приведённые ниже коды состояния применимы для всех сообщений DSKPP Response, если явно не указано иное.

Continue - продолжение
Сервер DSKPP готов к приёму следующего запроса от клиента DSKPP. Этот код не может передаваться в финальном сообщении сервера.

Success - успех
Успешное завершение сеанса DSKPP. Этот код может передаваться только в финальном сообщении сервера.

Abort - прерывание
Сервер DSKPP отверг запрос клиента DSKPP без указания причины.

AccessDenied - отказ в доступе
Клиент DSKPP не имеет полномочий на контакт с этим сервером DSKPP.

MalformedRequest - некорректно сформирован запрос
Сервер DSKPP столкнулся с отказом при разборе запроса от клиента DSKPP.

UnknownRequest - непонятный запрос
Клиент DSKPP сделал запрос, который не известен серверу DSKPP.

UnknownCriticalExtension - непонятное критичное расширение
Расширение DSKPP, помеченное, как критичное (Critical), не удалось интерпретировать на приёмной стороне.

UnsupportedVersion - неподдерживаемая версия
Клиент DSKPP использует версию протокола DSKPP, не поддерживаемую сервером DSKPP. Этот код допустим только в первом сообщении-отклике сервера DSKPP.

NoSupportedKeyTypes - неподдерживаемые типы ключей
Этот код показывает, что клиент DSKPP предложил только те типы ключей, которые не поддерживаются сервером DSKPP. Этот код допустим только в первом сообщении-отклике сервера DSKPP.

NoSupportedEncryptionAlgorithms - нет поддерживаемых алгоритмов шифрования
Этот код показывает, что клиент DSKPP предложил только те алгоритмы шифрования, которые не поддерживаются сервером DSKPP. Этот код допустим только в первом сообщении-отклике сервера DSKPP.

NoSupportedMacAlgorithms - нет поддерживаемых алгоритмов MAC
Этот код показывает, что клиент DSKPP предложил только те алгоритмы MAC, которые не поддерживаются сервером DSKPP. Этот код допустим только в первом сообщении-отклике сервера DSKPP.

NoProtocolVariants - нет вариантов протокола
Клиент DSKPP не предложил требуемый вариант протокола (2-х или 4-проходный). Этот код допустим только в первом сообщении-отклике сервера DSKPP.

NoSupportedKeyPackages - нет поддерживаемых ключевых пакетов
Этот код показывает, что клиент DSKPP предложил только те форматы ключевого пакета, которые не поддерживаются сервером DSKPP. Этот код допустим только в первом сообщении-отклике сервера DSKPP.

[1]Для несимметричного алгоритма. *Прим. перев.*
[2]Subscriber Identity Module - модуль идентификации абонента.

AuthenticationDataMissing - отсутствуют данные для аутентификации
Клиент DSKPP не предоставил серверу DSKPP требуемых для засвидетельствования данных.

AuthenticationDataInvalid - некорректны данные аутентификации
Клиент DSKPP предложил данные засвидетельствования пользователя, которые сервер DSKPP не мог проверить.

InitializationFailed - отказ при инициализации
Сервер DSKPP не может сгенерировать корректный ключ на основе предоставленных данных. При получении этого кода клиенту DSKPP **следует** перезапустить DSKPP, поскольку в новой сессии данные могут быть приняты.

ProvisioningPeriodExpired - закончилось время обеспечения
Время обеспечения, установленное сервером DSKPP, истекло. При получении этого кода клиенту DSKPP **следует** сообщить пользователю причину отказа при инициализации ключа, а пользователь **должен** зарегистрироваться на сервере DSKPP для инициализации нового ключа.

3.4. Базовые элементы

Приведённые ниже расчёты применимы для обоих вариантов DSKPP.

3.4.1. Данные засвидетельствования пользователя

Данные засвидетельствования пользователя (AD) создаются на основе значений Client ID и AC, которые пользователь вводит до передачи первого сообщения DSKPP.

Примечание. Пользователь обычно вводит значения Client ID и AC вручную (возможно, что устройство имеет лишь цифровую клавиатуру). Поэтому идентификатор и код зачастую представляют собой короткие числовые значения (например, 8 десятичных цифр). Однако сервер DSKPP может генерировать любые значения на своё усмотрение.

3.4.1.1. Формат кода засвидетельствования

AC представляется в формате TLV[1]. Код включает по крайней мере два TLV и может также включать дополнительные TLV, в зависимости от реализации.

Поля TLV определены ниже

Type (1 символ) - тип
шестнадцатеричный символ, идентифицирующий тип информации в поле Value.

Length (2 символа) - размер
Два шестнадцатеричных символа, задающих размер следующего поля Value, который **может** достигать 255 символов. Length = 00 **может** использоваться для тегов, не включающих поля значения.

Value (переменный размер) - значение
Строка шестнадцатеричных символов переменного размера, содержащая информацию для данного TLV.

В таблице приведены TLV, определённые в этом документе. Производители могут использовать дополнительные TLV для решения специфических задач с учётом того, что старший бит **должен** быть установлен для указания фирменного (vendor-specific) типа. В последующих версиях протокола могут быть определены дополнительные TLV.

Тип	*Имя TLV*	*Поддержка*	*Пример использования*
1	`Client ID`	Обязательно	`{ "AC00000A" }`
2	`Password`	Обязательно	`{ "3582AF0C3E" }`
3	`Checksum`	Необязательно	`{ "4D5" }`

Client ID является обязательным TLV и представляет собой идентификатор запрашивающей стороны с максимальным размером 255. Значение представляется в форме строки шестнадцатеричных символов, которая идентифицирует запрос ключа. В качестве примера предположим, что Client ID имеет значение AC00000A. Тогда Client ID TLV в AC будет представлено в форме 108AC00000A.

Обязательное TLV для пароля (Password) содержит одноразовый разделяемый секрет, который известен пользователю и серверу обеспечения. Значение Password является уникальным и **следует** использовать случайную последовательность символов, поскольку это затруднит угадывание AC. Строка **должна** только содержать шестнадцатеричные символы. Например, для пароля 3582AF0C3E Password TLV будет иметь вид 20A3582AF0C3E.

Необязательное TLV для контрольной суммы (Checksum) генерируется сервером-эмитентом и передаётся пользователю, как часть кода AC. Если TLV используется, значение контрольной суммы **должно** рассчитываться с использованием алгоритма CRC16 [ISO3309]. Когда пользователь вводит AC, для набранной строки символов AC вычисляется контрольная сумма и полученное значение сравнивается с заданным в TLV. Для AC с тегами Client ID и Password, имеющими вид 108AC00000A20A3582AF0C3E, контрольная сумма CRC16 будет иметь значение 0x356, что даёт Checksum TLV = 334D5. Полная строка AC в этом случае будет иметь вид 108AC00000A20A3582AF0C3E3034D5.

Хотя данная спецификация рекомендует использовать шестнадцатеричные символы только для AC на уровне пользовательского интерфейса приложений и делать триплеты TLV непрозрачными для пользователя, как показано в приведённом выше примере, разработчики **могут** дополнительно выбрать использование других печатных символов Unicode [UNICODE] на уровне пользовательского интерфейса приложений с учётом специфических местных требований контекста и удобства использования. Когда на уровне пользовательского интерфейса желательно использовать отличные от шестнадцатеричных символы (например, другие печатные символы US-ASCII или символы иных языков), **требуется** использовать SASLprep [RFC4013] для нормализации пользовательского ввода перед тем, как преобразовать его в строку шестнадцатеричных символов. Например, если приложение позволяет использовать любые печатные символы US-ASCII и символы расширенного кода ASCII для полей Client ID и Password, для поля Client ID устанавливается значение myclient!D, а для поля Password - mYpas&#rD, пользователь будет вводить с клавиатуры (или иным способом) Client ID = myclient!D и Password = mYpas&#rD в раздельных полях формы или в соответствии с инструкциями провайдера. Прикладной уровень, обрабатывающий пользовательский ввод, **должен** преобразовать введённые пользователем строки для использования в протоколе. В результате получится строка 1146D79636C69656E7421442126D5970617326237244 (отметим, что в этом примере Checksum TLV не используется).

[1]Type-Length-Value - тип-размер-значение.

Ниже приведено более подробное описание примера.

Предположим, что необработанное поле Client ID имеет значение myclient!D[1].

Значение Client ID в форме имён символов будет иметь вид:

```
U+006D строчная латинская буква M
U+0079 строчная латинская буква Y
U+0063 строчная латинская буква C
U+006C строчная латинская буква L
U+0069 строчная латинская буква I
U+0065 строчная латинская буква E
U+006E строчная латинская буква N
U+0074 строчная латинская буква T
U+0021 восклицательный знак (!)
U+0044 прописная латинская буква D
```

В кодировке UTF-8 поле Client ID имеет вид 6D 79 63 6C 69 65 6E 74 21 44.

Размер значения Client ID в шестнадцатеричном формате равен 14 (20 шестнадцатеричных символов).

TLV-представление для поля Client ID имеет вид:

```
1146D79636C69656E742144
```

Необработанное значение Password имеет вид mYpas&#rD.

Значение Password в форме имён символов будет иметь вид:

```
U+006D строчная латинская буква M
U+0059 прописная латинская буква Y
U+0070 строчная латинская буква P
U+0061 строчная латинская буква A
U+0073 строчная латинская буква S
U+0026 символ амперсанда (&)
U+0023 знак фунта (#)
U+0072 строчная латинская буква R
U+0044 прописная латинская буква D
```

В кодировке UTF-8 поле Password имеет вид 6D 59 70 61 73 26 23 72 44.

Размер значения Password в шестнадцатеричном формате равен 12.

TLV-представление для поля Password имеет вид

```
2126D5970617326237244
```

После объединения полей Client ID и Password поле AC будет иметь вид

```
1146D79636C69656E7421442126D5970617326237244
```

3.4.1.2. Расчёт данных засвидетельствования пользователя

Данные засвидетельствования (Authentication Data) включают Client ID (извлекается из AC) и значение, получаемое из AC на основе преобразования (общее описание функции DSKPP-PRF приведено в параграфе 3.4.2, а в Приложении D дано описание DSKPP-PRF-AES)

```
MAC = DSKPP-PRF(K_AC, AC->ClientID||URL_S||R_C||[R_S], 16)
```

В 4-проходном варианте DSKPP криптомодуль использует R_C, R_S и URL_S для расчёта MAC. URL_S - указатель URL, который клиент DSKPP использует для контакта с сервером DSKPP. В 2-проходном варианте DSKPP криптомодуль не имеет доступа к R_S, поэтому для вычисления MAC используется только R_C в комбинации с URL_S. В обоих случаях значение K_AC **должно** получаться из AC->password с помощью преобразования [PKCS-5]:

```
K_AC = PBKDF2(AC->password, R_C || K, iter_count, 16)
```

Должно использоваться одно из следующих значений K:

a. 4-проходный вариант:

   - открытый ключ сервера DSKPP (K_SERVER) или (в варианте с заранее известным разделяемым ключом) ключ, известный клиенту и серверу (K_SHARED).

b. 2-проходный вариант:

   - открытый ключ клиента DSKPP или открытый ключ устройства, если доступен сертификат устройства;
   - известный клиенту и серверу ключ (K_SHARED);
   - ключ, созданный из парольной фразы.

Счётчик итераций iter_count **должен** иметь значение не менее 100 000 за исключением двух последних вариантов 2-проходного режима (когда в качестве K используется K_SHARED или ключ, созданный из парольной фразы), когда **должно** использоваться значение iter_count = 1.

3.4.2. Необратимая псевдослучайная функция DSKPP - DSKPP-PRF

Независимо от реализованного варианта протокола существует требование к криптографическому примитиву, который определяет детерминированное преобразование секретного ключа k и строки октетов переменного размера в битовую строку заданного размера dsLen.

Этот примитив должен удовлетворять тем же требованиям, что и функция хэширования с ключом - он **должен** принимать на входе данные произвольного размера и генерировать результат, который не может быть обращён и не имеет коллизий (определения этих терминов можно найти в документе [FAQ]). Более того, результат работы примитива

[1] В оригинале ошибочно сказано myclient!ID. *Прим. перев.*

должен быть непредсказуемым даже в тех случаях, когда известны результаты преобразований других входных данных с тем же ключом.

С точки зрения этой спецификации DSKPP-PRF представляет собой функцию-»чёрный ящик», которая принимает входные данные, генерирует псевдослучайное значение и **может** быть реализована любыми подходящими криптографическими методами. В Приложении D даны два примера реализации DSKPP-PRF.

```
DSKPP-PRF(k, s, dsLen)
```

Вход:

k секретный ключ в форме строки октетов;

s строка октетов переменной длины, содержащая произвольные данные, определяющие строку на выходе;

dsLen желаемый размер результата.

Выход:

DS псевдослучайная строка размером dsLen октетов.

Для целей этого документа размер секретного ключа k **должен** быть не менее 16 октетов.

3.4.3. Алгоритм хэширования сообщений DSKPP

Пр передаче своего последнего сообщения в сеансе работы протокола сервер DSKPP генерирует код MAC, который используется клиентом для подтверждения ключа. Расчёт MAC **должен** включать хэш-значения для всех сообщений DSKPP, переданных клиентом и сервером в данной транзакции. Чтобы рассчитать хэш-значения для кода MAC данной последовательности сообщений DSKPP msg_1, ..., msg_n, **должны** быть выполнены следующие операции:

a. Берётся последовательность сообщений DSKPP, содержащая все сообщения Request и Response, не включая данное сообщение.

b. Переданные повторно сообщения исключаются из последовательности[1].

c. Сообщения объединяются в одну строку (конкатенация).

d. Полученная строка хэшируется с использованием алгоритма SHA-256 в соответствии с [FIPS180-SHA].

4. Использование четырехпроходного варианта

В это разделе описаны методы и поток сообщений, которые образуют 4-проходный вариант. Работа DSKPP в этом варианте зависит от механизма согласования ключей между клиентом и сервером.

4.1. Механизм согласования ключа

Для 4-проходного DSKPP симметричный ключ, являющийся целью обеспечения, генерируется «на лету» без передачи его между клиентом и сервером DSKPP. Поток данных и расчёты описаны ниже.

4.1.1. Поток данных

Простой поток данных при генерации ключа в 4-проходном варианте протокола показан на рисунке 3.

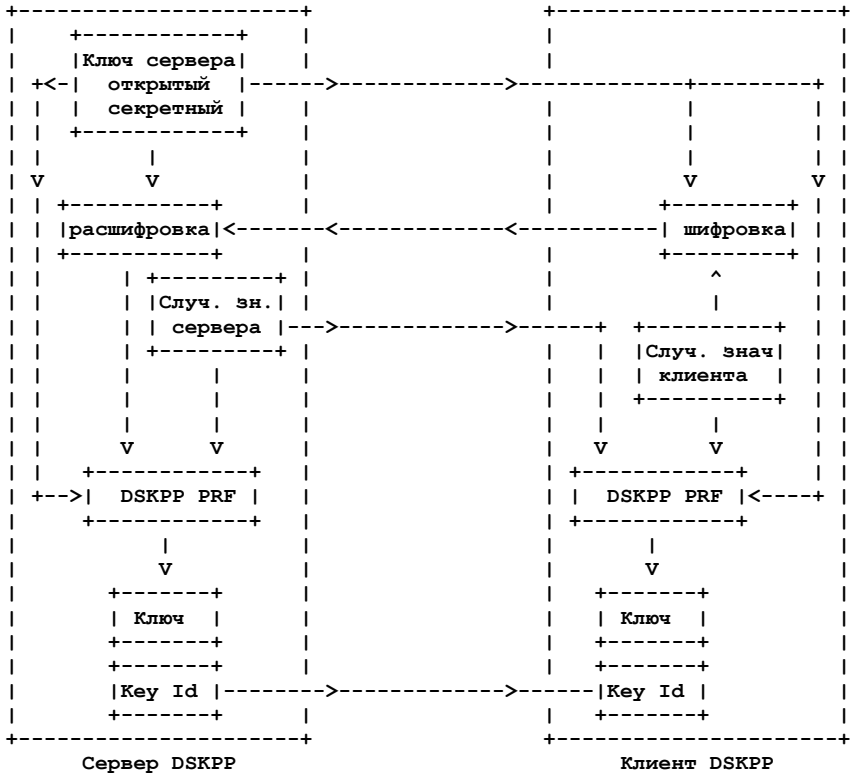

Рисунок 3. Схема потоков данных при генерации ключа DSKPP с использованием открытого ключа сервера.

[1]Полученная в результате последовательность **должна** чередовать сообщения DSKPP Request и DSKPP Response.

Включение двух случайных значений R_S и R_C в процесс генерации ключа обеспечивает обеим сторонам (криптомодулю и серверу DSKPP) гарантию того, что каждый будет вносить фактор случайности в создаваемый ключ и ключ будет уникальным. Включение ключа шифрования (K) гарантирует предотвращение MITM-атак[1], а также отличие ключа, получаемого криптографическим модулем от ключа, хранящегося на легитимном сервере DSKPP.

Хотя R_C и является псевдослучайной строкой, концептуально её можно рассматривать, как состоящую из двух частей - R_C1 и R_C2, где R_C1 генерируется в процессе работы протокола, а R_C2 может быть создана заранее и загружена в криптомодуль до его передачи пользователю. В этом случае **следует** обеспечивать уникальность второй части (R_C2) для каждого криптомодуля.

Человек, участвующий в MITM-атаке (с помощью повреждённой клиентской программы или при ошибочном подключении к другому серверу), может представить криптомодулю свой открытый ключ. Это позволит атакующему узнать клиентскую версию ключа K_TOKEN. Однако злоумышленник не сможет получить от легитимного сервера то же самое значение для K_TOKEN, поскольку K_TOKEN вычисляется с применением открытого ключа, а этот ключ будет отличаться у атакующего и сервера (или же атакующий не сможет расшифровать информацию, полученную от клиента). Поскольку атакующий уже «не находится между клиентом и сервером», те смогут определить, что они «рассинхронизированы», как только попытаются воспользоваться своими ключами. В случае расшифровки R_C с ключом K_SERVER важно проверить, что K_SERVER действительно является ключом легитимного сервера. Одним из способов является независимая проверка вновь созданного ключа K_TOKEN через соответствующую службу на сервере (например, с использованием канала, отличного от того, который применяется для генерации ключа).

4.1.2. Расчёт

В 4-проходном DSKPP клиент и сервер будут генерировать ключи K_TOKEN и K_MAC на основе ключа обеспечения (K_PROV), используя DSKPP-PRF (см. параграф 3.4.2), как показано ниже

```
K_PROV = DSKPP-PRF(k,s,dsLen)
```

где

```
k = R_C
```

(т. е. секретное случайное значение, выбранное клиентом DSKPP);

```
s = "Key generation" || K || R_S
```

K - ключ, применяемый для шифрования R_C и R_S, - случайное значение, выбранное сервером DSKPP

```
dsLen = (желаемый размер K_PROV, в котором первая половина задаёт K_MAC, а вторая - K_TOKEN)
```

Значения K_TOKEN и K_MAC создаются на основе K_PROV

```
K_PROV = K_MAC || K_TOKEN
```

При расчёте K_PROV созданные ключи K_MAC и K_TOKEN **могут** подвергаться определяемым алгоритмом преобразованиям перед тем, как использовать в качестве ключа выбранного типа. Одним из примеров такого преобразования является приведение к чётности для ключей DES.

Отметим, что этот расчёт выполняется только при 4-проходном варианте DSKPP.

4.2. Поток сообщений

Поток сообщений 4-проходного варианта включает два обмена:

1: Проход 1 = <KeyProvClientHello>, Проход 2 = <KeyProvServerHello>

2: Проход 3 = <KeyProvClientNonce>, Проход 4 = <KeyProvServerFinished>

В первой паре сообщений согласуются криптографические алгоритмы и выполняется обмен nonce. Вторая пара сообщений служит для организации симметричного ключа с использованием взаимного соглашения сторон о ключах.

Назначение и структура каждого сообщения описаны ниже. Формат XML и примеры даны в разделе 8 и Приложении B.

4.2.1. KeyProvTrigger

```
Клиент DSKPP                          Сервер DSKPP
------------                          ------------
                          [<---]      AD, [DeviceID],
                                      [KeyID], [URL_S]
```

«Триггерное» сообщение является необязательным. Сервер DSKPP передаёт такое сообщение после выполнения по автономному каналу перечисленных ниже действий.

1. Пользователь направляется его браузером на web-приложение, обеспечивающее ключ и подпись для него (аутентификация).
2. Пользователь запрашивает ключ.
3. Web-приложение обрабатывает запрос и возвращает пользователю код AC (например, в ответ на запрос регистрации в защищённой web-сессии).
4. Web-приложение получает код AC от пользователя (возможно, посредством запроса на ввод кода в с использованием web-формы или выбора пользователем указателя URL, в который встроен код AC).
5. Web-приложение получает данные аутентификации (AD) из AC, как описано в параграфе 3.4.1.
6. Web-приложение передаёт AD и, возможно, DeviceID (идентифицирует конкретное устройство, для которого будет предоставляться ключ) и/или KeyID (идентифицирует ключ, который будет заменён) серверу DSKPP.

Назначение сообщения

[1]Man in the middle - букв., «человек посередине». Перехват и изменение данных с участием человека. *Прим. перев.*

Начало сеанса DSKPP. Сервер DSKPP использует это сообщение для того, чтобы инициировать передачу первого сообщения DSKPP приложением на клиентской стороне. Для предоставления системе обеспечения ключей способа передачи URL сервера DSKPP клиенту DSKPP.

Таким путём система обеспечения ключей может указать клиенту DSKPP на конкретный криптомодуль, который был заранее сконфигурирован на сервере обеспечения ключами .

При обновлении ключей для идентификации заменяемого ключа.

Что содержится в сообщении

Должно обеспечиваться значение AD, чтобы обеспечить серверу DSKPP возможность засвидетельствования клиента до завершения сеанса работы протокола.

Может включаться значение DeviceID, позволяющее программе обеспечения ключами связать предоставляемый ключ с конкретным устройством.

Может включаться значение KeyID, позволяющее программе обеспечения ключами идентифицировать заменяемый ключ (например, в случае обновления ключа).

Может включаться значение Server URL, позволяющее программе обеспечения ключами указать клиенту DSKPP сервер, с которым нужно контактировать.

4.2.2. KeyProvClientHello

```
Клиент DSKPP                        Сервер DSKPP
------------                        ------------
SAL, [AD],
[DeviceID], [KeyID]      --->
```

При первом подключении клиента DSKPP к серверу DSKPP клиент должен передать сначала сообщение <KeyProvClientHello>. Это сообщение может быть также передано клиентом в ответ на <KeyProvTrigger>.

Что содержится в сообщении

Список атрибутов безопасности (SAL[1]), включаемый в <KeyProvClientHello>, содержит комбинации версий и вариантов DSKPP, форматов ключевых пакетов, типов ключей и криптографических алгоритмов, которые поддерживает клиент DSKPP, указывающие предпочтения клиента (желаемый вариант указывается первым).

Если сообщению <KeyProvClientHello> предшествовало сообщение <KeyProvTrigger>, в <KeyProvClientHello> **должны** быть включены значение AD, DeviceID и/или KeyID из сообщения-триггера.

Если сообщению <KeyProvClientHello> не было инициировано сообщением <KeyProvTrigger>, в <KeyProvClientHello> **может** быть включено значение DeviceID, которое известно и серверу DSKPP, а также идентификатор ключа, ранее предоставленного сервером обеспечения DSKPP.

Примечания для приложений

Если этому сообщению предшествует триггер <KeyProvTrigger>, у приложения уже имеется доступное значение AD (см. параграф 4.2.1). Однако, если перед сообщением не было передано триггера<KeyProvTrigger>, приложение **должно** получить значение кода засвидетельствования пользователя (возможно, запрашивая у того ввод AC вручную - например, с использованием цифровой клавиатуры).

Приложение **должно** также создать данные аутентификации (AD) из кода AC, как описано в параграфе 3.4.1, и сохранить эти данные для использования в следующем сообщении <KeyProvClientNonce>.

Как сервер DSKPP использует это сообщение

Сервер DSKPP будет искать в сообщении подходящую комбинацию версии DSKPP, варианта (в данном случае 4-проходного), формата ключевого пакета, типа ключа и набора криптографических алгоритмов. Если атрибуты SAL клиента DSKPP не соответствуют возможностям сервера DSKPP или не удовлетворяют требованиям к обеспечению ключами, сервер DSKPP будет устанавливать для атрибута Status значение, отличное от Continue. При обнаружении соответствия устанавливается Status = Continue.

При включении в <KeyProvClientHello> значений AD, DeviceID и KeyID сервер DSKPP будет проверять их. Для сервера DSKPP **недопустимо** принимать значение DeviceID, если этот сервер не передавал DeviceID в предшествующем триггерном сообщении. Отметим также, что клиент DSKPP вправе инициировать сессию DSKPP без получения от сервера триггерного сообщения <KeyProvTrigger>, но в этом случае восприятие любого представленного значения DeviceID **недопустимо** для сервера DSKPP, если у того нет доступа к уникальному ключу (который будет использоваться протоколом) для идентификации устройства.

4.2.3. KeyProvServerHello

```
Клиент DSKPP                        Сервер DSKPP
------------                        ------------
                          <---   SAL, R_S, [K], [MAC]
```

Сервер DSKPP будет передавать это сообщение в ответ на <KeyProvClientHello> после того, как будут просмотрены комбинации версии DSKPP, варианта (в данном случае 4-проходного), формата ключевого пакета, типа ключа и набора криптографических алгоритмов. Если подходящей комбинации не будет найдено, серверу следует передавать сообщение со статусом отказа.

Назначение сообщения

С помощью этого сообщения задаётся контекст для протокольной сессии. Кроме того, сервер DSKPP использует это сообщение для передачи случайного маркера nonce, который требуется на каждой стороне для согласования общего симметричного ключа (K_TOKEN).

[1]Security Attribute List.

Что содержится в сообщении

Атрибут статуса эквивалентен коду, возвращаемому сервером в ответ на <KeyProvClientHello>. Если сервер нашёл в списке SAL подходящий набор атрибутов, он устанавливает статус Continue и возвращает список SAL (выбирается из списка SAL, полученного в сообщении <KeyProvClientHello>). Список SAL от сервера задаёт версию DSKPP и вариант протокола (в данном случае 4-проходный), тип ключа, криптографические алгоритмы а формат ключевого пакета, которые клиент **должен** использовать в оставшейся части протокольной сессии.

Случайный маркер nonce (R_S) для использования при генерации симметричного ключа путём согласования. Размер R_S может зависеть от выбранного типа ключа.

Ключ (K) будет использоваться клиентом DSKPP для шифрования клиентского значения nonce, включаемого в <KeyProvClientNonce>. K является открытым ключом сервера (K_SERVER) или известным обеим сторонам секретным ключом (K_SHARED).

Если ключ будет обновляться, **должно** присутствовать значение MAC, чтобы клиент DSKPP мог убедиться, что ключ на замену получен от доверенного сервера. Это значение MAC **должно** рассчитываться с использованием функции DSKPP-PRF (см. параграф 3.4.2) и входной параметр k **должен** представлять иметь значение существующего ключа MAC - K_MAC' (т. е., значение ключа MAC, которое существовало до этого протокольного сеанса; реализация **может** задавать в качестве K_MAC' значение заменяемого K_TOKEN), а в качестве входного параметра dsLen **должно** использоваться значение размера R_S.

Как клиент DSKPP использует это сообщение

Значение атрибута Status, отличающееся от Continue, говорит об отказе и клиент DSKPP **должен** прервать сеанс.

Если успешное выполнение протокола приводит к замене существующего ключа на заново сгенерированный, клиент DSKPP **должен** проверить значение кода MAC, представленное в сообщении <KeyProvServerHello>. Клиент DSKPP **должен** прервать сеанс DSKPP, если значение MAC не удалось проверить, и **должен** в этом случае удалить все маркеры nonce, ключи и/или секреты, связанные с этим сеансом.

Если Status = Continue, криптографический модуль генерирует случайное значение R_C, используя криптографический алгоритм, заданный в SAL. Размер nonce R_C будет зависеть от выбранного типа ключа.

Значение R_C шифруется с использованием ключа K и алгоритма, заданного в SAL.

Метод, который клиент DSKPP **должен** использовать для шифрования R_C

Если K эквивалентен K_SERVER (т. е. открытому ключу сервера DSKPP), **может** использоваться схема шифрования RSA из PKCS #1 [PKCS-1]. При K эквивалентном K_SERVER криптографическому модулю **следует** проверить сертификат сервера до начала его использования для шифрования R_C, как описано в параграфе 3.1 [RFC2818] и [RFC5280].

Если K эквивалентен K_SHARED, клиент DSKPP **может** использовать функцию DSKPP-PRF для обеспечения независимости от других алгоритмов. В этом случае клиент использует K_SHARED в качестве входного параметра k (K_SHARED **следует** использовать только для этого).

```
dsLen = len(R_C)
```

где len - размер R_C.

```
DS = DSKPP-PRF(K_SHARED, "Encryption" || R_S, dsLen)
```

Это выражение будет давать псевдослучайную строку DS размером R_C. Шифрование R_C **может** быть обеспечено путём использования операции XOR для DS и R_C

```
E(DS, R_C) = DS ^ R_C
```

Сервер DSKPP в этом случае будет выполнять обратное преобразование для получения R_C из E(DS, R_C).

4.2.4. KeyProvClientNonce

```
Клиент DSKPP                          Сервер DSKPP
------------                          ------------
E(K,R_C), AD          --->
```

Клиент DSKPP будет передавать это сообщение сразу же после получения сообщения <KeyProvServerHello> со статусом Continue.

Назначение сообщения

В этом сообщении клиент DSKPP передаёт данные засвидетельствования пользователя (AD) и случайный маркер nonce, зашифрованный с использованием ключа сервера DSKPP (K). Клиентское случайное значение nonce требуется обеим сторонам для согласования общего симметричного ключа (K_TOKEN).

Что содержится в сообщении

Данные аутентификации (AD), которые были созданы на основе кода AC, вводятся пользователем до отправки сообщения <KeyProvClientHello> (см. параграф 3.2).

Случайное значение nonce (R_C) клиента DSKPP, зашифрованное, как описано в параграфе 4.2.3.

Как сервер DSKPP использует это сообщение

Сервер DSKPP **должен** использовать AD для аутентификации пользователя. Если засвидетельствование завершилось отказом, сервер DSKPP **должен** возвратить код состояния, обусловленный причиной отказа.

Если аутентификация прошла, сервер DSKPP дешифрует R_C, используя свой ключ (K). Метод расшифровки зависит от ключа K, переданного клиенту в сообщении <KeyProvServerHello> - открытый ключ сервера (K_SERVER) или заранее известный обеим сторона ключ (K_SHARED) (описание зашифровки клиентом DSKPP значения R_C приведено в параграфе 4.2.3).

После расшифровки R_C сервер DSKPP рассчитывает ключ K_TOKEN, используя комбинацию двух случайных значений nonce R_S и R_C, а также ключа шифрования K, как описано в параграфе 4.1.2. Конкретная реализация функции DSKPP-PRF (например, описанная в Приложении D) зависит от алгоритма MAC, заданного в сообщении <KeyProvServerHello>. После этого сервер DSKPP генерирует ключевой пакет, который содержит атрибуты применения (такие, как срок действия и размер). В ключевой пакет **недопустимо** включать ключ K_TOKEN, поскольку в 4-проходном варианте K_TOKEN никогда не передаётся между клиентом и сервером DSKPP. Сервер сохраняет K_TOKEN и ключевой пакет вместе с учётной записью пользователя на криптографическом сервере.

В заключение сервер генерирует код MAC для подтверждения ключа, который клиент будет использовать для предотвращения ложных сообщений Commit, которые могут переводить криптомодуль в состояние, когда сервер не распознает сохранённый ключ.

Код MAC, используемый для подтверждения ключа, **должен** рассчитываться, как показано ниже.

```
msg_hash = SHA-256(msg_1, ..., msg_n)
dsLen = len(msg_hash)
MAC = DSKPP-PRF (K_MAC, "MAC 1 computation" || msg_hash, dsLen)
```

где

MAC
Для расчёта MAC используется псевдослучайная функция DSKPP, определённая в параграфе 3.4.2. Конкретная реализация DSKPP-PRF (пример описан в Приложении D) зависит от алгоритма MAC, заданного в сообщении <KeyProvServerHello>. Значение MAC **должно** рассчитываться с использованием имеющегося ключа MAC (K_MAC) и строки, формируемой путём конкатенации ASCII-строки «MAC 1 computation» и значения msg_hash.

K_MAC
Ключ, созданный из K_PROV, как описано в параграфе 4.1.2.

msg_hash
Хэш-сумма (см. определение в параграфе 3.4.3) для сообщений msg_1, ..., msg_n.

4.2.5. KeyProvServerFinished

```
Клиент DSKPP                    Сервер DSKPP
------------                    ------------
                        <---          KP, MAC
```

Сервер DSKPP будет передавать это сообщение после аутентификации пользователя и, при успешной аутентификации, генерации ключа K_TOKEN и ключевого пакета, а также связывания их с учётной записью пользователя на криптографическом сервере.

Назначение сообщения

С помощью этого сообщения сервер DSKPP подтверждает генерацию ключа K_TOKEN, а также передаёт клиенту связанный с ключом идентификатор и зависящие от приложения атрибуты (но не сам ключ) в ключевом пакете для завершения протокольной сессии.

Что содержится в сообщении

Атрибут статуса эквивалентен коду, возвращаемому сервером в ответ на <KeyProvClientNonce>. Если засвидетельствование пользователя прошло успешно, сервер рассчитал ключ K_TOKEN, сгенерировал ключевой пакет и связал его с учётной записью пользователя на криптографическом сервере, в поле Status сервер устанавливает значение Success. При установке для атрибута Status значения Success это сообщение действует, как сообщение Commit, говорящее криптомодулю. Что нужно сохранить сгенерированный ключ (K_TOKEN) и связать идентификатор данный идентификатор с этим ключом. По этой причине в данное сообщение **должен** быть помещён ключевой пакет (KP), который содержит идентификатор для сгенерированного ключа (но не сам ключ) и дополнительные конфигурационные параметры (например, отождествление сервера DSKPP, атрибуты использования ключа и т. п.). Используемый по умолчанию формат ключевого пакета **должен** основываться на контейнере PSKC[1], определённом в [RFC6030]. В качестве дополнительных **могут** использоваться форматы [RFC6031], PKCS #12 [PKCS-12] или PKCS #5 XML [PKCS-5-XML].

Вместе с KP сервер включает MAC-код подтверждения ключа, который позволяет клиенту избавиться от ложного сообщения Commit. Алгоритм MAC использует ту же функцию DSKPP-PRF, которая применяется для сообщения <KeyProvServerHello>.

Как клиент DSKPP использует это сообщение

Значение атрибута Status, отличное от Success, говорит об отказе и клиент DSKPP **должен** прервать сессию.

После получения сообщения <KeyProvServerFinished> с кодом Status = Success клиент DSKPP **должен** проверить код MAC подтверждения ключа, который был передан в этом сообщении. Клиент DSKPP **должен** прервать сессию DSKPP, если значение MAC не прошло верификацию, а также **должен** в этом случае удалить все маркеры nonce, ключи и/или секреты, связанные с этим протокольным сеансом.

Если <KeyProvServerFinished> имеет Status = Success и код MAC прошёл верификацию, клиент DSKPP **должен** рассчитать ключ K_TOKEN, используя комбинацию двух случайных значений nonce R_S и R_C, а также серверного ключа шифрования K, как описано в параграфе 4.1.2. Для расчёта используется та же функция DSKPP-PRF, которая служит для расчёта кода MAC. Клиент DSKPP связывает ключевой пакет, содержащийся в <KeyProvServerFinished>, с созданным ключом K_TOKEN и постоянно хранит эти данные в криптомодуле.

После этой операции **не допускается** возможность замены ключа, пока не будет обеспечена информация о ключе авторизации с помощью кода MAC в последующем сообщении <KeyProvServerHello> (и <KeyProvServerFinished>).

[1]Portable Symmetric Key Container - переносимый контейнер с симметричным ключом.

5. Использование двухпроходного варианта

В этом разделе описаны методы и поток сообщений для 2-проходного варианта протокола. Двухпроходный DSKPP важен для доставки ключевого материала от сервера DSKPP к клиенту DSKPP. Сервер DSKPP передаёт ключевой материал в ключевом пакете, отформатированном в соответствии с [RFC6030], [RFC6031], PKCS #12 [PKCS-12] или PKCS #5 XML [PKCS-5-XML].

Ключевой материал включает первичный ключ K_PROV, на основе которого клиент DSKPP создаёт два ключа: симметричный для криптомодуля (K_TOKEN) и ключ, используемый для подтверждения (K_MAC). Ключевой материал включает также атрибуты использования ключей (такие, как срок действия и размер).

Сервер DSKPP шифрует K_PROV для предотвращения доступа к этому ключу всем, кроме сервера и криптомодуля. Сервер DSKPP использует для шифрования K_PROV любой из трёх методов защиты - Key Transport, Key Wrap или Passphrase-Based Key Wrap Key Protection[1].

Хотя клиент и сервер DSKPP могут согласовать используемый метод защиты ключа, фактическая защита осуществляется в KeyPackage. Формат KeyPackage задаёт способ защиты ключа с использованием трёх методов. Ниже перечислены форматы KeyPackage, определённые для DSKPP:

- Контейнер PSKC [RFC6030]

  urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
- Контейнер SKPC [RFC6031]

  urn:ietf:params:xml:ns:keyprov:dskpp:skpc-key-container
- Контейнер PKCS12 [PKCS-12]

  urn:ietf:params:xml:ns:keyprov:dskpp:pkcs12-key-container
- Контейнер PKCS5-XML [PKCS-5-XML]

  urn:ietf:params:xml:ns:keyprov:dskpp:pkcs5-xml-key-container

Каждый из перечисленных методов защиты описан ниже.

5.1. Методы защиты ключей

В этом разделе описаны три метода защиты ключей для 2-проходного варианта. Дополнительные методы защиты **могут** быть определены через процесс IETF или внешними путями.

5.1.1. Транспортировка ключа

Этот метод предназначен для поддерживающих PKI[2] устройств. Сервер DSKPP шифрует ключевой материал и доставляет его клиенту DSKPP. Сервер шифрует ключевой материал, используя открытый ключ клиента DSKPP, секретный ключ которого хранится в криптомодуле. Клиент DSKPP расшифровывает ключевой материал и использует его для создания симметричного ключа K_TOKEN.

URN для идентификации метода

urn:ietf:params:xml:schema:keyprov:dskpp:transport

Клиент и сервер DSKPP **должны** поддерживать механизм шифрования

http://www.w3.org/2001/04/xmlenc#rsa-1_5,

определённый в [XMLENC].

5.1.2. Упаковка ключа

Этот метод идеален для устройств с заранее созданными ключами (например, SIM-карт). Сервер DSKPP шифрует ключевой материал, используя заранее известный ключ упаковки, и доставляет его клиенту DSKPP. Клиент DSKPP расшифровывает ключевой материал и использует его для создания симметричного ключа K_TOKEN.

URN для идентификации метода

urn:ietf:params:xml:schema:keyprov:dskpp:wrap

Клиент и сервер DSKPP **должны** поддерживать перечисленные ниже механизмы упаковки ключей.

AES128 KeyWrap

Основан на id-aes128-wrap из [RFC3394] и http://www.w3.org/2001/04/xmlenc#kw-aes128 из [XMLENC]

AES128 KeyWrap with Padding

Основан на id-aes128-wrap-pad из [RFC5649] и http://www.w3.org/2001/04/xmlenc#kw-aes128 из [XMLENC]

AES-CBC-128

Основан на [FIPS197-AES] и http://www.w3.org/2001/04/xmlenc#aes128-cbc из [XMLENC]

5.1.3. Защита ключа с использованием парольной упаковки

Этот метод является вариантом упаковки ключа, применимым для устройств с ограниченными возможностями, цифровую клавиатуру (например, мобильных телефонов). Сервер DSKPP шифрует ключевой материал, используя упаковку ключа на основе представленной пользователем парольной фразы, и доставляет зашифрованный ключевой материал клиенту DSKPP. Клиент DSKPP расшифровывает ключевой материал и использует его для создания симметричного ключа K_TOKEN.

Для предотвращения раскрытия ключа K_TOKEN кому-либо, кроме сервера DSKPP и криптомодуля, этот метод **следует** применять только в тех случаях, когда устройство непосредственно включает средства ввода парольной фразы (например, клавиатуру).

URN для идентификации метода

urn:ietf:params:xml:schema:keyprov:dskpp:passphrase-wrap

[1]Транспортировка ключа, упаковка ключа, защита ключа с использованием парольной упаковки.
[2]Public Key Infrastructure - инфраструктура открытых ключей. *Прим. перев.*

Клиент и сервер DSKPP **должны** поддерживать перечисленные ниже механизмы:

- Парольная схема шифрования PBES2, определённая в [PKCS-5] (указана как http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbes2 в [PKCS-5-XML]).
- Парольная функция создания ключей PBKDF2, определённая в [PKCS-5] (указана как http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2 в [PKCS-5-XML]).
- Все перечисленные ниже механизмы упаковки ключей.

AES128 KeyWrap
Основан на id-aes128-wrap из [RFC3394] и http://www.w3.org/2001/04/xmlenc#kw-aes128 из [XMLENC]
AES128 KeyWrap with Padding
Основан на id-aes128-wrap-pad из [RFC5649] и http://www.w3.org/2001/04/xmlenc#kw-aes128 из [XMLENC]
AES-CBC-128
Основан на [FIPS197-AES] и http://www.w3.org/2001/04/xmlenc#aes128-cbc из [XMLENC]

5.2. Поток сообщений

В 2-проходном варианте протокола используется один этап обмена сообщениями:

```
1:  Проход 1 = <KeyProvClientHello>, Проход 2 = <KeyProvServerFinished>
```

Хотя здесь нет обмена сообщениями <ServerHello> или <ClientNonce>, клиент DSKPP сохраняет возможность задания своих предпочтений в части алгоритма и поддерживаемых типов ключей с помощью сообщения <KeyProvClientHello>.

Назначение и содержимое каждого сообщения описаны ниже. Формат XML и примеры приведены в разделе 8 и Приложении B.

5.2.1. KeyProvTrigger

Триггерное сообщение используется точно так же, как в 4-проходном варианте (см. параграф 4.2.1).

5.2.2. KeyProvClientHello

```
Клиент DSKPP                          Сервер DSKPP
------------                          ------------
SAL, AD, R_C,
[DeviceID], [KeyID],
KPML                        --->
```

Когда клиент DSKPP в первый раз подключается к серверу DSKPP, он должен передать в первую очередь сообщение <KeyProvClientHello>. Клиент может также отправить <KeyProvClientHello> в ответ на <KeyProvTrigger>.

Назначение сообщения

С помощью этого сообщения клиент DSKPP показывает свои предпочтения в части алгоритма и поддерживаемых типов ключей, а также поддерживаемые версии DSKPP, варианты протокола (в данном случае 2-проходный), форматы ключевого пакета и методы защиты ключей. Кроме того, клиент DSKPP упрощает засвидетельствование пользователей за счёт передачи аутентификационных данных (AD), которые пользователь предоставил до отправки первого сообщения DSKPP.

Примечания для приложений

В этом сообщении серверу DSKPP **должны** передаваться данные (AD). Если сообщению предшествовал триггер <KeyProvTrigger>, у приложения уже имеются доступные данные AD (см. параграф 4.2.1). Однако, если триггера <KeyProvTrigger> не было, приложение **должно** найти код аутентификации пользователя (AC), возможно запрашивая ввод этого кода пользователем (например, с цифровой клавиатуры устройства). Приложение **должно** также создать данные AD на основе кода AC, как описано в параграфе 3.4.1, и сохранить их для использования в следующем сообщении <KeyProvClientNonce>.

Что содержится в сообщении

Список атрибутов защиты (SAL), включаемый в <KeyProvClientHello>, содержит комбинацию поддерживаемых клиентом DSKPP версий и вариантов протокола DSKPP, форматов ключевых пакетов, типов ключей и криптографических алгоритмов (предпочтительные варианты указываются сначала).

Данные засвидетельствования (AD), включённые в <KeyProvTrigger> или сгенерированные, как описано в «Примечания для приложений» выше.

Случайное значение маркера nonce клиента DSKPP (R_C), которое было использовано клиентом при генерации AD. За счёт включения R_C в сессию DSKPP клиент DSKPP получает возможность проверить жизнеспособность сервера DSKPP до передачи ключа.

Если сообщению <KeyProvClientHello> предшествовал триггер <KeyProvTrigger>, тогда это сообщение **должно** включать также значения DeviceID и/или KeyID из триггерного сообщения. Если триггера не было перед <KeyProvClientHello>, это сообщение **может** включать значение DeviceID, известное клиенту и серверу DSKPP, а также **может** содержать идентификатор ключа, связанный с ключом, который сервер обеспечения DSKPP представил ранее.

Список методов защиты ключей (KPML[1]), поддерживаемых клиентом DSKPP. Каждый элемент списка **может** включать «содержимое» ключа шифрования для сервера DSKPP, которое используется для защиты ключевого материала, возвращаемого сервером клиенту. Это содержимое **должно** иметь тип <ds:KeyInfoType> ([XMLDSIG]). Для каждого метода защиты ключа допустимыми вариантами для <ds:KeyInfoType> являются:

- Транспортировка ключа

[1]The list of key protection methods.

Только те варианты <ds:KeyInfoType>, которые идентичны открытому ключу (т. е., <ds:KeyName>, <ds:KeyValue>, <ds:X509Data> или <ds:PGPData>). **Рекомендуется** опция <ds:X509Certificate> варианта <ds:X509Data> для случаев, когда сертифицирован открытый ключ, соответствующий секретному ключу криптомодуля.

- Упаковка ключа

  Только те варианты <ds:KeyInfoType>, которые идентичны симметричному ключу (т. е., <ds:KeyName> и <ds:KeyValue>). **Рекомендуется** вариант <ds:KeyName>.

- Упаковка ключа на базе пароля

  Должна использоваться опция <ds:KeyName> и имя ключа **должно** идентифицировать парольную фразу, которая будет использоваться сервером для генерации ключа упаковки. Компоненты идентификатора и парольной фразы в <ds:KeyName> **должны** совпадать с компонентами Client ID и Authentication Code данных AD (тех же AD, что содержатся в данном сообщении).

Как сервер DSKPP использует это сообщение

Сервер DSKPP будет искать подходящую комбинацию версии DSKPP, варианта (в данном случае, 2-проходного) протокола, формата ключевого пакета, типа ключа и криптографических алгоритмов. Если список SAL клиента DSKPP не соответствует возможностям сервера DSKPP или не согласуется с политикой предоставления ключей, сервер DSKPP будет устанавливать для атрибута Status значение, отличное от Success. При соответствии будет устанавливаться Status = Success.

Сервер DSKPP будет проверять значения DeviceID и KeyID, если они включены в <KeyProvClientHello>. Серверу DSKPP **недопустимо** принимать DeviceID, если сам сервер не передавал DeviceID в предшествующем триггерном сообщении. Отметим, что клиент DSKPP может начать работу без получения от сервера триггерного сообщения <KeyProvTrigger>, но в этом случае любое, представленное клиентом значение DeviceID **недопустимо** воспринимать со стороны сервера, если у того нет доступа к уникальному ключу для идентифицируемого устройства, который будет использоваться в работе протокола.

Сервер DSKPP **должен** использовать значение AD для аутентификации пользователя. При отказе аутентификации сервер DSKPP **должен** установить в коде возврата значение отказа и удалить все маркеры nonce, ключи и или секреты, связанные с данным сеансом протокола.

Если аутентификация пользователя прошла успешно, сервер DSKPP генерирует ключ K_PROV. В 2-проходном варианте, когда клиент не имеет доступа к R_S, значение K_PROV генерируется случайным образом исключительно сервером DSKPP, при этом ключ K_PROV **должен** состоять из 2 частей равного размера

```
K_PROV = K_MAC || K_TOKEN
```

Размер ключа K_TOKEN (и, следовательно, - K_MAC) определяется типом K_TOKEN, который **должен** быть одним из поддерживаемых клиентом DSKPP. В случаях, когда желаемый размер ключа K_TOKEN отличается от размера K_MAC для используемого алгоритма MAC, для генерации K_PROV **должен** выбираться больший из двух размеров. Ключ MAC укорачивается для создания K_MAC в тех случаях, когда используется алгоритм MAC, для которого размер K_MAC больше того, который требуется для ключа желаемого K_TOKEN. Если размер K_TOKEN больше требуемого для обеспечения соответствия с размером K_MAC, сервер обеспечения и принимающий клиент должны определить актуальный размер секретного ключа из алгоритма целевого ключа и сохранять лишь усечённую до нужного размера часть K_TOKEN. При отсечке **должно** сохраняться требуемое число начальных байтов реального ключа K_TOKEN или K_MAC. Например, если сервер обеспечения работает на базе секретного ключа HOTP размером 20 и MAC-алгоритма DSKPP-PRF-SHA256 (Приложение D), размер K_PROV будет 64. Соответственно, ключи K_TOKEN и K_MAC будут содержать по 32 байта. Используемый ключ HOTP будет включать первые 20 байтов ключа K_TOKEN.

После расчёта K_PROV сервер DSKPP выбирает один из методов защиты ключей, предложенных в списке KPML клиентом DSKPP и использует этот метод вместе с соответствующими данными для шифрования K_PROV. Сервер DSKPP генерирует ключевой пакет для транспортировки информации метода шифрования ключа и зашифрованного ключа K_PROV. Формат зашифрованных данных определяется выбранным ключевым пакетом. Ключевой пакет **должен** задавать и использовать выбранный метод защиты ключа и ключевую информацию из сообщения <KeyProvClientHello>. Ключевой пакет включает также атрибуты использования (такие, как срок действия и размер). Сервер сохраняет ключевой пакет и K_TOKEN, связывая их с учётной записью пользователя на криптографическом сервере.

Сервер генерирует MAC для подтверждения ключа и клиент будет использовать этот код для предотвращения ложных сообщений Commit, которые могут переводить криптомодуль в состояние, когда сервер не распознает сохранённый ключ.

В дополнение к этому при обновлении существующего ключа сервер генерирует второй код MAC, который будет возвращаться клиенту, как аутентификационные данные (AD) сервера, чтобы клиент DSKPP мог убедиться, что замена ключа осуществляется доверенным сервером.

Сервер DSKPP **должен** использовать при расчёте MAC для подтверждения ключа описанный ниже метод.

```
msg_hash = SHA-256(msg_1, ..., msg_n)
dsLen = len(msg_hash)
MAC = DSKPP-PRF (K_MAC, "MAC 1 computation" || msg_hash || ServerID, dsLen)
```

где

MAC — Значение MAC **должно** рассчитываться с использованием уже принятого алгоритма MAC, а при расчёте **должна** применяться строка ASCII «MAC 1 computation», msg_hash и ServerID, а также существующий ключ MAC (K_MAC).

K_MAC — Ключ, полученный из K_PROV, который сервер DSKPP **должен** предоставить криптомодулю.

msg_hash — Хэш-сумма, определённая в параграфе 3.4.3, для сообщений msg_1, ..., msg_n.

ServerID Идентификатор, который сервер DSKPP **должен** включать в элемент <KeyPackage> сообщения <KeyProvServerFinished>.

Если в качестве алгоритма MAC используется функция DSKPP-PRF (определена в параграфе 3.4.2)входной параметр этой функции **должен** представлять собой конкатенацию строки ASCII «MAC 1 computation», msg_hash и ServerID, а параметр dsLen **должен** совпадать с размером msg_hash.

Для расчёта кода MAC, применяемого при аутентификации, сервера DSKPP сервер **должен** использовать описанный ниже метод.

Код MAC **должен** рассчитываться для строки ASCII «MAC 2 computation», идентификатора сервера ServerID и R с использованием уже существующего ключа MAC - K_MAC' (ключ MAC, который существовал до запуска протокольной сессии). Отметим, что реализация может выбрать в качестве K_MAC' значение ключа K_TOKEN, который будет меняться.

Если в качестве алгоритма MAC используется функция DSKPP-PRF, входной параметр s **должен** представлять собой конкатенацию строки ASCII «MAC 2 computation» со значениями ServerID и R. Параметр dsLen **должен** иметь значение не менее 16 (т. е., размер кода MAC **должен** быть не менее 16 октетов):

```
dsLen >= 16
MAC = DSKPP-PRF (K_MAC', "MAC 2 computation" || ServerID || R, dsLen)
```

Алгоритм MAC **должен** совпадать с алгоритмом, который сервер DSKPP использует для расчёта кода MAC при подтверждении ключа.

5.2.3. KeyProvServerFinished

```
Клиент DSKPP                    Сервер DSKPP
------------                    ------------
                    <---         KP, MAC, AD
```

Сервер DSKPP будет передавать это сообщение после засвидетельствования пользователя и, при положительном результате аутентификации, генерации ключа K_TOKEN и ключевого пакета, а также связывания их с учётной записью пользователя на криптосервере.

Назначение сообщения

С помощью этого сообщения сервер DSKPP доставляет ключевой пакет, содержащий зашифрованный ключ K_PROV и атрибуты его использования.

Что содержится в сообщении

Атрибут Status является эквивалентом кода, возвращаемого сервером в ответ на сообщение <KeyProvClientHello>. Если сервер нашёл подходящий набор атрибутов в клиентском списке SAL, он устанавливает Status = Success.

Подтверждающее сообщение **должно** включать ключевой пакет (KP[1]), содержащий идентификаторы сервера DSKPP и ключа, тип ключа, зашифрованный ключ K_PROV, метод шифрования и дополнительные конфигурационные параметры. Используемый по умолчанию ключевой пакет для симметричного шифрования **должен** основываться на контейнере PSKC, определённом в [RFC6030]. В число дополнительных форматов **могут** входить [RFC6031], PKCS #12 [PKCS-12], PKCS #5 XML [PKCS-5-XML].

Это сообщение **должно** включать код MAC, который клиент DSKPP будет использовать для подтверждения ключа. Это значение MAC рассчитывается с использованием строки «MAC 1 computation», как описано выше.

Если осуществляется замена существующего ключа, сообщение **должно** также включать код MAC для аутентификации сервера (рассчитывается с использованием строки «MAC 2 computation», как описано выше), который передаётся клиенту DSKPP, как AD.

Как клиент DSKPP использует это сообщение

После получения сообщения <KeyProvServerFinished> с Status = Success клиент DSKPP **должен** проверить оба кода MAC (поля MAC и AD). Если верификация любого из кодов MAC дала отрицательный результат, клиент DSKPP **должен** прервать протокольную сессию и удалить все маркеры nonce, ключи и/или секреты, связанные с этой сессией.

Если в сообщении <KeyProvServerFinished> содержится Status = Success и проверка кода MAC прошла успешно, клиент DSKPP **должен** выделить ключ K_PROV из полученного ключевого пакета и создать ключ K_TOKEN. В заключение клиент DSKPP инициализирует криптомодуль с ключом K_TOKEN и соответствующими атрибутами его использования. После выполнения этой операции **недопустимо** менять ключ, пока не будет получен ключ разрешения с использованием кода MAC в новом сообщении <KeyProvServerFinished>.

6. Расширения протокола

Протокол DSKPP рассчитан на возможность расширения. В последующих параграфах определены два расширения, включённые в схему DSKPP. Поскольку при использовании этих расширений могут возникать проблемы совместимости, разработчикам протоколов следует взвешенно подходить к решению вопроса об использовании этих разрешений. Например, если конкретная реализация основана на фирменных расширениях, она может оказаться несовместимой с независимыми реализациями, которые не знают об этих расширениях.

Расширения могут передаваться в любом сообщении DSKPP с использованием типа ExtensionsType. Этот тип представляет собой список расширений, содержащий пары «тип-значение», которые определяют дополнительные функции, поддерживаемые клиентом или сервером DSKPP. Каждое расширение **может** быть помечено, как критическое путём установки для атрибута Critical в Extension значение true. Если расширение не имеет маркировки Critical, принимающая сторона не обязана интерпретировать такое расширение; принимающая сторона всегда свободна в решении вопроса об отказе от некритических расширений.

[1]Key Package.

6.1. ClientInfoType

Расширение ClientInfoType **может** содержать любые специфичные для клиента данные, требуемые приложению. Это расширение **может** присутствовать в сообщениях <KeyProvClientHello> и <KeyProvClientNonce>. При наличии такого расширения **недопустимо** устанавливать для него флаг Critical.

Серверы DSKPP **должны** поддерживать это расширение. Для серверов DSKPP **недопустимы** попытки интерпретации содержащихся в расширении данных. При получении такого расширения сервер **должен** включить его в неизменном виде в следующий отклик для данного сеанса. От серверов DSKPP не требуется сохранение данных ClientInfoType.

6.2. ServerInfoType

Расширение ServerInfoType **может** содержать любые специфичные для сервера данные, требуемые приложению (например, информацию о состоянии). Это расширение допустимо только в сообщениях <KeyProvServerHello> с атрибутом Status = Continue. При наличии такого расширения **недопустимо** устанавливать для него флаг Critical.

Клиенты DSKPP **должны** поддерживать это расширение. Для клиентов DSKPP **недопустимы** попытки интерпретации содержащихся в расширении данных. При получении такого расширения клиент **должен** включить его в неизменном виде в следующий отклик для данного сеанса. От клиентов DSKPP не требуется сохранение данных ServerInfoType.

7. Привязки протокола

7.1. Общие требования

DSKPP предполагает транспорт с гарантией доставки.

7.2. Привязка HTTP/1.1 для DSKPP

В этом параграфе рассмотрена привязка описанных выше сообщений к протоколу HTTP/1.1 [RFC2616]. Эта привязка к HTTP является обязательной для реализации, хотя в последующих версиях спецификации могут быть определены дополнительные привязки. Отметим, что в общем случае клиент HTTP будет отличаться от клиента DSKPP (n. t., клиент HTTP будет выполнять функции посредника для сообщений DSKPP от клиента DSKPP к серверу DSKPP). Кроме того, на серверной стороне HTTP сервер DSKPP **может** получать сообщения DSKPP от сервера HTTP-front-end. Сервер DSKPP будет идентифицироваться специфическим указателем URL, который может быть задан заранее или установлен при инициализации клиента.

7.2.1. Идентификация сообщений DSKPP

Тип MIME для всех сообщений DSKPP **должен** быть

```
application/dskpp+xml
```

7.2.2. Заголовки HTTP

Для предотвращения кэширования откликов, содержащих сообщения DSKPP прокси-серверами рекомендуются перечисленные ниже меры.

- При использовании HTTP/1.1 запрашивающей стороне **следует**:
  - включить в заголовок поле Cache-Control со значением "no-cache, no-store";
  - включить в заголовок поле Pragma со значением "no-cache".
- При использовании HTTP/1.1 отвечающей стороне **следует**:
  - включить в заголовок поле Cache-Control со значением "no-cache, no-must-revalidate, private";
  - включить в заголовок поле Pragma со значением "no-cache";
  - **не** включать полей Validator типа заголовков Last-Modified или Etag.

Для обработки согласования контента запросы HTTP **могут** включать в заголовок поле HTTP Accept. Это поле **следует** идентифицировать с использованием типа MIME, заданного в параграфе 7.2.1. Заголовок Accept **может** включать дополнительные типы содержимого, которые будут определяться в новых версиях этого протокола.

На заголовки HTTP не накладывается каких-либо ограничений, кроме требования устанавливать для заголовка Content-Type значение типа MIME, указанное в параграфе 7.2.1.

7.2.3. Операции HTTP

Установившиеся соединения, определённые в HTTP/1.1, являются **опциональными**. Запросы DSKPP преобразуются в запросы HTTP с использованием метода POST. Отклики DSKPP преобразуются в отклики HTTP.

Для 4-проходного DSKPP сообщения в протокольной сессии «связываются воедино». В частности, сообщение <KeyProvServerHello> привязывается к предшествующему сообщению <KeyProvClientHello> путём передачи в соответствующем отклике HTTP. Сообщение <KeyProvServerHello> **должно** иметь атрибут SessionID и одноимённый атрибут в последующем сообщении <KeyProvClientNonce> **должен** быть идентичным. Сообщение <KeyProvServerFinished> снова привязывается к остальным через HTTP (и, возможно, SessionID).

7.2.4. Коды состояния HTTP

Запрашивающей стороне DSKPP HTTP, которой не удалось осуществить обмен сообщениями с отвечающей стороной DSKPP HTTP, **следует** возвращать отклик с кодом 403 (Forbidden - запрещено). В этом случае содержимое тела HTTP не имеет значения. При возникновении ошибки HTTP во время обработки запроса DSKPP сервер HTTP **должен** возвращать отклик 500 (Internal Server Error - внутренняя ошибка сервера). Этот код **следует** возвращать для связанных с HTTP ошибок, обнаруженных до передачи управления процессору DSKPP или в тех случаях, когда процессор DSKPP сообщает о внутренней ошибке (например, некорректное пространство имён DSKPP XML или

невозможно найти схему DSKPP). Если получен запрос, не являющийся сообщением клиента DSKPP, отвечающая сторона DSKPP **должна** возвращать код 400 (Bad request - некорректный запрос).

В тех случаях, когда возвращается отклик HTTP с кодом 4xx или 5xx, содержимое тела HTTP не имеет значения.

Коды перенаправления (3xx) применяются, как обычно.

Всякий раз при успешном выполнении HTTP POST отвечающая сторона DSKPP HTTP **должна** использовать код 200 и включать подходящее сообщение DSKPP (возможно, с информацией об ошибке DSKPP) в тело HTTP.

7.2.5. HTTP-засвидетельствование

Поддержка аутентификации HTTP/1.1 не предполагается.

7.2.6. Инициализация DSKPP

Если пользователь запрашивает инициализацию ключа в сеансе Web-просмотра и этот запрос имеет подходящий запрос Accept (например, с URL конкретного сервера DSKPP), сервер DSKPP **может** ответить передачей инициализационного сообщения DSKPP в отклике HTTP со значением Content-Type, установленным в соответствии с параграфом 7.2.1 и кодом 200 (OK). Инициализационное сообщение **может** содержать в своём теле данные типа URL для клиента DSKPP, которые могут использоваться при контакте с сервером DSKPP. Если сообщение содержит данные, они **должны** представлять собой корректный экземпляр элемента <KeyProvTrigger>.

Если пользовательский запрос был направлен какому-либо иному ресурсу, серверу DSKPP недопустимо отвечать на него, объединяя данные DSKPP с кодом отклика 200. В таких случаях серверу DSKPP **следует** отвечать путём передачи инициализационного сообщения DSKPP в отклике HTTP со значением Content-Type, установленным в соответствии с параграфом 7.2.1 и кодом отклика 406 (Not Acceptable - неприемлемо).

7.2.7. Примеры сообщений

a. Инициализация от сервера DSKPP:

```
HTTP/1.1 200 OK

Cache-Control: no-store
Content-Type: application/dskpp+xml
Content-Length: <некое значение>
```

Данные инициализации DSKPP в форме XML ...

b. Начальный запрос от клиента DSKPP:

```
POST http://example.com/cgi-bin/DSKPP-server HTTP/1.1

Cache-Control: no-cache, no-store
Pragma: no-cache
Host: www.example.com
Content-Type: application/dskpp+xml
Content-Length: <некое значение>
```

Данные DSKPP в форме XML (поддерживаемая версия, поддерживаемые алгоритмы, ...)

c. Начальный запрос от сервера DSKPP:

```
HTTP/1.1 200 OK

Cache-Control: no-cache, no-must-revalidate, private
Pragma: no-cache
Content-Type: application/dskpp+xml
Content-Length: <некое значение>
```

Данные DSKPP в форме XML (случайный маркер nonce от сервера, открытый ключ сервера, ...)

8. XML-схема DSKPP

8.1. Общие требования к обработке

Некоторые элементы DSKPP основываются на компонентах, которые способны сравнивать полученные значения с сохранёнными. Если явно не указано иное, все элементы, имеющие схему XML типа «xs:string» или производного от него типа, **должны** сравниваться с использованием точного бинарного сравнения. В частности для реализаций DSKPP **недопустимо** сравнение текстовых строк без учёта регистра символов, нормализация или игнорирование пробелов, а также преобразование определяемых локальными (языковыми) параметрами форматов (в частности, представления чисел).

Реализации, сравнивающие значения, которые представлены в разных кодировках символов, **должны** использовать метод сравнения, дающий такой же результат, как двоичное сравнение этих значений, преобразованных в кодировку Unicode [UNICODE].

Порядок сопоставления или сортировки для атрибутов или элементов не определяется. Следовательно, для реализаций DSKPP **недопустима** зависимость от какого-либо конкретного порядка сортировки значений.

8.2. Схема

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
   xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   targetNamespace="urn:ietf:params:xml:ns:keyprov:dskpp"
```

```
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
<xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
   schemaLocation=
   "http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>
<xs:import namespace="urn:ietf:params:xml:ns:keyprov:pskc"
   schemaLocation="keyprov-pskc-1.0.xsd"/>
<xs:complexType name="AbstractRequestType" abstract="true">
   <xs:annotation>
      <xs:documentation> Basic types[1] </xs:documentation>
   </xs:annotation>
   <xs:attribute name="Version" type="dskpp:VersionType" use="required"/>
</xs:complexType>
<xs:complexType name="AbstractResponseType" abstract="true">
   <xs:annotation>
      <xs:documentation> Basic types[1] </xs:documentation>
   </xs:annotation>
   <xs:attribute name="Version" type="dskpp:VersionType" use="required"/>
   <xs:attribute name="SessionID" type="dskpp:IdentifierType" />
   <xs:attribute name="Status" type="dskpp:StatusCode" use="required"/>
</xs:complexType>

<xs:simpleType name="VersionType">
   <xs:restriction base="xs:string">
      <xs:pattern value="\d{1,2}\.\d{1,3}" />
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="IdentifierType">
   <xs:restriction base="xs:string">
      <xs:maxLength value="128" />
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="StatusCode">
   <xs:restriction base="xs:string">
      <xs:enumeration value="Continue" />
      <xs:enumeration value="Success" />
      <xs:enumeration value="Abort" />
      <xs:enumeration value="AccessDenied" />
      <xs:enumeration value="MalformedRequest" />
      <xs:enumeration value="UnknownRequest" />
      <xs:enumeration value="UnknownCriticalExtension" />
      <xs:enumeration value="UnsupportedVersion" />
      <xs:enumeration value="NoSupportedKeyTypes" />
      <xs:enumeration value="NoSupportedEncryptionAlgorithms" />
      <xs:enumeration value="NoSupportedMacAlgorithms" />
      <xs:enumeration value="NoProtocolVariants" />
      <xs:enumeration value="NoSupportedKeyPackages" />
      <xs:enumeration value="AuthenticationDataMissing" />
      <xs:enumeration value="AuthenticationDataInvalid" />
      <xs:enumeration value="InitializationFailed" />
      <xs:enumeration value="ProvisioningPeriodExpired" />
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="DeviceIdentifierDataType">
   <xs:choice>
      <xs:element name="DeviceId" type="pskc:DeviceInfoType" />
      <xs:any namespace="##other" processContents="strict" />
   </xs:choice>
</xs:complexType>

<xs:simpleType name="PlatformType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="Hardware" />
      <xs:enumeration value="Software" />
      <xs:enumeration value="Unspecified" />
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="TokenPlatformInfoType">
   <xs:attribute name="KeyLocation" type="dskpp:PlatformType"/>
   <xs:attribute name="AlgorithmLocation" type="dskpp:PlatformType"/>
</xs:complexType>

<xs:simpleType name="NonceType">
   <xs:restriction base="xs:base64Binary">
      <xs:minLength value="16" />
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="AlgorithmsType">
   <xs:sequence maxOccurs="unbounded">
      <xs:element name="Algorithm" type="dskpp:AlgorithmType"/>
```

[1]Базовые типы.

```
      </xs:sequence>
   </xs:complexType>

   <xs:simpleType name="AlgorithmType">
      <xs:restriction base="xs:anyURI" />
   </xs:simpleType>

   <xs:complexType name="ProtocolVariantsType">
      <xs:sequence>
         <xs:element name="FourPass" minOccurs="0" />
         <xs:element name="TwoPass" type="dskpp:KeyProtectionDataType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="KeyProtectionDataType">
      <xs:annotation>
         <xs:documentation xml:lang="en">
            This element is only valid for two-pass DSKPP[1].
         </xs:documentation>
      </xs:annotation>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="SupportedKeyProtectionMethod" type="xs:anyURI"/>
        <xs:element name="Payload" type="dskpp:PayloadType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="PayloadType">
      <xs:choice>
         <xs:element name="Nonce" type="dskpp:NonceType" />
         <xs:any namespace="##other" processContents="strict"/>
      </xs:choice>
   </xs:complexType>

   <xs:complexType name="KeyPackagesFormatType">
      <xs:sequence maxOccurs="unbounded">
         <xs:element name="KeyPackageFormat" type="dskpp:KeyPackageFormatType"/>
      </xs:sequence>
   </xs:complexType>

   <xs:simpleType name="KeyPackageFormatType">
      <xs:restriction base="xs:anyURI" />
   </xs:simpleType>

   <xs:complexType name="AuthenticationDataType">
      <xs:annotation>
         <xs:documentation xml:lang="en">
            Authentication Data contains a MAC[2].
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="ClientID"
            type="dskpp:IdentifierType" minOccurs="0"/>
         <xs:choice>
            <xs:element name="AuthenticationCodeMac" type="dskpp:AuthenticationMacType"/>
            <xs:any namespace="##other" processContents="strict" />
         </xs:choice>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="AuthenticationMacType">
      <xs:sequence>
         <xs:element minOccurs="0" name="Nonce" type="dskpp:NonceType"/>
         <xs:element minOccurs="0" name="IterationCount" type="xs:int"/>
         <xs:element name="Mac" type="dskpp:MacType" />
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="MacType">
      <xs:simpleContent>
         <xs:extension base="xs:base64Binary">
            <xs:attribute name="MacAlgorithm" type="xs:anyURI"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <xs:complexType name="KeyPackageType">
      <xs:sequence>
         <xs:element minOccurs="0" name="ServerID" type="xs:anyURI"/>
         <xs:element minOccurs="0" name="KeyProtectionMethod" type="xs:anyURI" />
         <xs:choice>
            <xs:element name="KeyContainer" type="pskc:KeyContainerType"/>
            <xs:any namespace="##other" processContents="strict"/>
         </xs:choice>
```

[1]Этот элемент подходит только для двухпроходного DSKPP.
[2]Данные засвидетельствования содержат MAC.

```
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="InitializationTriggerType">
      <xs:sequence>
         <xs:element minOccurs="0" name="DeviceIdentifierData"
            type="dskpp:DeviceIdentifierDataType" />
         <xs:element minOccurs="0" name="KeyID" type="xs:base64Binary"/>
         <xs:element minOccurs="0" name="TokenPlatformInfo"
            type="dskpp:TokenPlatformInfoType" />
         <xs:element name="AuthenticationData" type="dskpp:AuthenticationDataType" />
         <xs:element minOccurs="0" name="ServerUrl" type="xs:anyURI"/>
         <xs:any minOccurs="0" namespace="##other" processContents="strict" />
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="ExtensionsType">
      <xs:annotation>
         <xs:documentation> Extension types[1] </xs:documentation>
      </xs:annotation>
      <xs:sequence maxOccurs="unbounded">
         <xs:element name="Extension" type="dskpp:AbstractExtensionType"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="AbstractExtensionType" abstract="true">
      <xs:attribute name="Critical" type="xs:boolean" />
   </xs:complexType>

   <xs:complexType name="ClientInfoType">
      <xs:complexContent mixed="false">
         <xs:extension base="dskpp:AbstractExtensionType">
            <xs:sequence>
               <xs:element name="Data" type="xs:base64Binary"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="ServerInfoType">
      <xs:complexContent mixed="false">
         <xs:extension base="dskpp:AbstractExtensionType">
            <xs:sequence>
               <xs:element name="Data" type="xs:base64Binary"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:element name="KeyProvTrigger"
      type="dskpp:KeyProvTriggerType">
      <xs:annotation>
         <xs:documentation> DSKPP PDUs </xs:documentation>
      </xs:annotation>
   </xs:element>

   <xs:complexType name="KeyProvTriggerType">
      <xs:annotation>
      <xs:documentation xml:lang="en">
         Message used to trigger the device to initiate a DSKPP run[2].
      </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:choice>
            <xs:element name="InitializationTrigger"
               type="dskpp:InitializationTriggerType" />
            <xs:any namespace="##other" processContents="strict"/>
         </xs:choice>
      </xs:sequence>
      <xs:attribute name="Version" type="dskpp:VersionType"/>
   </xs:complexType>

   <xs:element name="KeyProvClientHello"
      type="dskpp:KeyProvClientHelloPDU">
      <xs:annotation>
         <xs:documentation>KeyProvClientHello PDU</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="KeyProvClientHelloPDU">
      <xs:annotation>
         <xs:documentation xml:lang="en">
            Message sent from DSKPP Client to DSKPP Server to initiate a DSKPP session[3].
```

[1]Типы расширений.
[2]Сообщение используется в качестве триггера для инициирования запуска DSKPP.
[3]Сообщение передаётся от клиента DSKPP серверу DSKPP для инициирования сеанса DSKPP.

```
            </xs:documentation>
         </xs:annotation>
         <xs:complexContent mixed="false">
            <xs:extension base="dskpp:AbstractRequestType">
               <xs:sequence>
                  <xs:element minOccurs="0" name="DeviceIdentifierData"
                     type="dskpp:DeviceIdentifierDataType" />
                  <xs:element minOccurs="0" name="KeyID" type="xs:base64Binary" />
                  <xs:element minOccurs="0" name="ClientNonce" type="dskpp:NonceType" />
                  <xs:element name="SupportedKeyTypes" type="dskpp:AlgorithmsType" />
                  <xs:element name="SupportedEncryptionAlgorithms"
                     type="dskpp:AlgorithmsType" />
                  <xs:element name="SupportedMacAlgorithms" type="dskpp:AlgorithmsType" />
                  <xs:element minOccurs="0" name="SupportedProtocolVariants"
                     type="dskpp:ProtocolVariantsType" />
                  <xs:element minOccurs="0" name="SupportedKeyPackages"
                     type="dskpp:KeyPackagesFormatType" />
                  <xs:element minOccurs="0" name="AuthenticationData"
                     type="dskpp:AuthenticationDataType" />
                  <xs:element minOccurs="0" name="Extensions" type="dskpp:ExtensionsType" />
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <xs:element name="KeyProvServerHello"
         type="dskpp:KeyProvServerHelloPDU">
         <xs:annotation>
            <xs:documentation>KeyProvServerHello PDU</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:complexType name="KeyProvServerHelloPDU">
         <xs:annotation>
            <xs:documentation xml:lang="en">
               Response message sent from DSKPP Server to DSKPP Client in four-pass DSKPP[1].
            </xs:documentation>
         </xs:annotation>
         <xs:complexContent mixed="false">
            <xs:extension base="dskpp:AbstractResponseType">
               <xs:sequence minOccurs="0">
                  <xs:element name="KeyType" type="dskpp:AlgorithmType"/>
                  <xs:element name="EncryptionAlgorithm" type="dskpp:AlgorithmType" />
                  <xs:element name="MacAlgorithm" type="dskpp:AlgorithmType"/>
                  <xs:element name="EncryptionKey" type="ds:KeyInfoType"/>
                  <xs:element name="KeyPackageFormat" type="dskpp:KeyPackageFormatType" />
                  <xs:element name="Payload" type="dskpp:PayloadType"/>
                  <xs:element minOccurs="0" name="Extensions" type="dskpp:ExtensionsType" />
                  <xs:element minOccurs="0" name="Mac" type="dskpp:MacType"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <xs:element name="KeyProvClientNonce" type="dskpp:KeyProvClientNoncePDU">
         <xs:annotation>
            <xs:documentation>KeyProvClientNonce PDU</xs:documentation>
         </xs:annotation>
      </xs:element>

      <xs:complexType name="KeyProvClientNoncePDU">
         <xs:annotation>
            <xs:documentation xml:lang="en">
               Response message sent from DSKPP Client to DSKPP Server in a four-pass DSKPP
               session[1].
            </xs:documentation>
         </xs:annotation>
         <xs:complexContent mixed="false">
            <xs:extension base="dskpp:AbstractRequestType">
               <xs:sequence>
                  <xs:element name="EncryptedNonce" type="xs:base64Binary"/>
                  <xs:element minOccurs="0" name="AuthenticationData"
                     type="dskpp:AuthenticationDataType" />
                  <xs:element minOccurs="0" name="Extensions" type="dskpp:ExtensionsType" />
               </xs:sequence>
               <xs:attribute name="SessionID" type="dskpp:IdentifierType" use="required"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <xs:element name="KeyProvServerFinished" type="dskpp:KeyProvServerFinishedPDU">
         <xs:annotation>
            <xs:documentation>KeyProvServerFinished PDU</xs:documentation>
         </xs:annotation>
      </xs:element>
```

[1]Отклик, передаваемый от сервера DSKPP клиенту DSKPP в 4-проходном варианте DSKPP.

```
    <xs:complexType name="KeyProvServerFinishedPDU">
       <xs:annotation>
          <xs:documentation xml:lang="en">
             Final message sent from DSKPP Server to DSKPP Client in a DSKPP session.
             A MAC value serves for key confirmation, and optional AuthenticationData
             serves for server authentication[1].
          </xs:documentation>
       </xs:annotation>
       <xs:complexContent mixed="false">
          <xs:extension base="dskpp:AbstractResponseType">
             <xs:sequence minOccurs="0">
                <xs:element name="KeyPackage" type="dskpp:KeyPackageType" />
                <xs:element minOccurs="0" name="Extensions" type="dskpp:ExtensionsType" />
                <xs:element name="Mac" type="dskpp:MacType" />
                <xs:element minOccurs="0" name="AuthenticationData"
                   type="dskpp:AuthenticationMacType" />
             </xs:sequence>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
 </xs:schema>
```

9. Требования по соответствию

Для обеспечения совместимости всех реализаций DSKPP сервер DSKPP:

a. **должен** реализовать 4-проходный вариант протокола (раздел 4);

b. **должен** реализовать 2-проходный вариант протокола (раздел 5);

c. **должен** поддерживать аутентификацию пользователей (параграф 3.2.1);

d. **должен** поддерживать функции порождения ключей:

- DSKPP-PRF-AES DSKPP-PRF (Приложение D);
- DSKPP-PRF-SHA256 DSKPP-PRF (Приложение D);

e. **должен** поддерживать указанный ниже механизм шифрования для защиты клиентских маркеров nonce в 4-проходном протоколе:

- механизм, описанный в параграфе 4.2.4;

f. **должен** поддерживать для операций (например, «заворачивания») с симметричными ключами один из перечисленных ниже алгоритмов шифрования:

- KW-AES128 без заполнения, указанный ссылкой http://www.w3.org/2001/04/xmlenc#kw-aes128 в [XMLENC];
- KW-AES128 без заполнения, указанный ссылкой http://www.w3.org/2001/04/xmlenc#kw-aes128 в [XMLENC] и [RFC5649];
- AES-CBC-128 (см. [FIPS197-AES]);

g. **должен** поддерживать указанный ниже алгоритм шифрования для операций с асимметричными ключами (например, транспортировки ключей):

- схема RSA Encryption [PKCS-1];

h. **должен** поддерживать следующие функции контроля целостности/KDF MAC:

- DSKPP-PRF-AES (Приложение D);
- DSKPP-PRF-SHA256 (Приложение D);

i. **должен** поддерживать ключевые пакеты PSKC [RFC6030]; **должны** быть реализованы все 3 метода защиты ключей PSKC (Key Transport, Key Wrap, Passphrase-Based Key Wrap);

j. **могут** поддерживаться ключевые пакеты ASN.1 в соответствии с определением [RFC6031].

Клиенты DSKPP **должны** поддерживать 2-проходный или 4-проходный вариант протокола. Клиенты DSKPP **должны** соответствовать всем требованиям пунктов (c) - (j).

Реализации DSKPP **должны** привязывать сообщения DSKPP к протоколу HTTP/1.1, как описано в параграфе 7.2.

DSKPP является защитным протоколом и одной из его основных функций является предоставление доступа к инициализации криптомодуля с новым симметричным ключом только уполномоченным сторонам. Следовательно, конфигурация любой конкретной реализации может включать произвольное число ограничений, относящихся к алгоритмам и доверенным агентствам, которые не будут обеспечивать универсальной интероперабельности.

10. Вопросы безопасности

10.1. Общие вопросы

Протокол DSKPP разработан для защиты генерируемого ключевого материала от раскрытия. Никакие элементы, за исключением сервера DSKPP и криптомодуля, не могут получить доступа к генерируемым значениям K_TOKEN, если используется достаточно сильный криптоалгоритм, а на клиентской стороне генерация и шифрование R_C, а также

[1]Финальное сообщение, переданное от сервера к клиенту DSKPP в сеансе DSKPP. Значение MAC служит для подтверждения ключа, а необязательные данные AuthenticationData - для засвидетельствования сервера.

генерация K_TOKEN выполняются в соответствии с требованиями криптомодуля. Это условие выполняется даже в случаях наличия враждебного кода на стороне клиента DSKPP. Однако, как отмечено в последующих параграфах, DSKPP не обеспечивает защиты от некоторых угроз, обусловленных MITM[1]-атаками и некоторыми другими формами атак. Следовательно для защиты от таких угроз, протокол DSKPP **должен** использовать транспорт, обеспечивающий защиту конфиденциальности и целостности, типа HTTP over TLS с подходящим шифрованием [RFC2818]. Отметим, что шифры TLS с анонимным обменом ключами не подходят для этого случая [RFC5246].

10.2. Активные атаки

10.2.1. Введение

Активная атака **может** представлять собой попытку изменения, удаления, вставки, повторного использования или смены порядка сообщений с разными целями, включая отказ служб и компрометацию ключевого материала.

10.2.2. Изменение сообщений

Модификация сообщений <KeyProvTrigger> может приводить к отказу службы (изменение любого из идентификаторов или Authentication Code) или вынуждать клиента DSKPP контактировать с подставным сервером DSKPP. Последний вариант относится к MITM-атакам, которые рассматриваются в параграфе 10.2.7.

Атакующий может изменить сообщение <KeyProvClientHello>. Это позволяет атакующему указать для клиента DSKPP другой ключ или устройство, а также предложить клиенту криптографические алгоритмы, отличающихся от предпочитаемых клиентом (например, более слабые). Атакующий может также предложить использование более ранних версий DSKPP, которые могут иметь уязвимости. Такие изменения могут позволить атакующему воздействовать (инициализировать или изменить) на другой криптомодуль (сервер выделит ключ для другого модуля) или получить доступ к сгенерированному ключу за счёт применения более слабого криптоалгоритма или уязвимой версии протокола. Реализации DSKPP могут обеспечить защиту от угроз второго типа за счёт строго контроля используемых версий и алгоритмов. Первая угроза (выделение сгенерированного ключа для другого - некорректного - модуля) не реализуема для случаев, когда используется вариант DSKPP с разделяемым ключом (предполагается, что существующие разделяемые ключи уникальны для каждого криптомодуля), но остаётся актуальной для вариантов с открытыми ключами. Следовательно, для серверов DSKPP **недопустимо** принимать в одностороннем порядке идентификаторы устройств при использовании вариантов с открытым ключом. Это указано и в описании протокола. Однако в вариантах с разделяемым ключом атакующий может получить возможность представления подложного идентификатора (что может также привести к некорректному связыванию пользователя с созданным ключом), если атакующий имеет в реальном масштабе времени доступ к криптомодулю с идентифицированным ключом. В результате такой атаки могут возникать ситуации, когда сгенерированный ключ будет связан с корректным криптомодулем, но модуль будет связываться с некорректным пользователем. Такие атаки и возможные их последствия, а также меры противодействия более подробно рассмотрены в параграфе 10.5.

Атакующий может также изменить сообщение <KeyProvServerHello>. Это означает, что атакующий может указать иные типы ключей, алгоритмы, версию протокола, нежели предлагает легитимный сервер (например, более слабые криптографически). Атакующий может также представить подменные маркеры nonce взамен передаваемых легитимным сервером. В первом случае клиент **может** защититься путём строгого выполнения правил в части выбора допустимых алгоритмов и версий протокола. Во втором случае (подмена nonce) проблем безопасности не возникает, поскольку сгенерированный ключ не будет соответствовать ключу, сгенерированному легитимным сервером. Кроме того, каждый раз, когда запуск DSKPP будет приводить к замене имеющегося ключа, элемент <Mac> будет обеспечивать защиту от изменения R_S.

Возможно также изменение сообщений <KeyProvClientNonce>. Если атакующий меняет атрибут SessionID, на сервере, по сути, будет происходить переход к другой сессии в предположении, что новое значение SessionID в данный момент действительно на этом сервере. Это по-прежнему не даёт злоумышленнику возможности узнать сгенерированное значение K_TOKEN, поскольку значение R_C было «упаковано» (wrapped) для легитимного сервера. Изменение элемента <EncryptedNonce> (например, замена значения на известное атакующему значение для базового R'C) не будет приводить к смене клиентом своего состояния до-DSKPP, поскольку сервер не сможет предоставить действительное значение MAC в своём финальном сообщении клиенту. Однако сервер **может** в конечном итоге сохранить K'TOKEN вместо K_TOKEN. Если криптографический модуль был связан с конкретным пользователем, это не будет приводить к нарушению защиты. Дополнительное рассмотрение этой угрозы и мер противодействия приведено ниже в параграфе 10.5. Отметим, что использование TLS не защищает от этой атаки, если злоумышленник имеет доступ к клиенту DSKPP (например, через вредоносную программу) [RFC5246].

Кроме того, атакующие могут также менять сообщения <KeyProvServerFinished>. Замена элемента <Mac> будет приводить лишь к отказу в обслуживании (denial of service). Замена любого другого элемента может приводить клиента DSKPP к связыванию, например, ложного (wrong) сервиса с созданным ключом. DSKPP **следует** использовать на основе транспорта, обеспечивающего защиту конфиденциальности и целостности, если такая угроза существенна.

10.2.3. Удаление сообщений

Удаление сообщений не причинит никакого вреда, кроме отказа в обслуживании, поскольку криптографическому модулю **недопустимо** менять своё состояние (т. е., «вводить в дело» созданный ключ), пока он не получит финального сообщения от сервера DSKPP и не обработает успешно это сообщение, включая проверку MAC в нем. Удалённое сообщение <KeyProvServerFinished> не вызовет перехода сервера в несогласованное с криптографическим модулем состояние, если на сервере реализованы предложения из параграфа 10.5.

10.2.4. Вставка сообщений

Активный атакующий может в любой момент инициировать запуск DSKPP и предложить любой идентификатор устройства. Реализации сервера DSKPP благодаря использованию <KeyProvTrigger> **могут** получить некоторую защиту от нечаянной инициализации ключа, непреднамеренной замены имеющегося ключа или назначения ключа криптомодулю путём запуска DSKPP. Элемент <AuthenticationData> позволяет серверу связать запуск DSKPP, например, с аутентифицированной пользователем сессией. Защита этого метода зависит от способности защитить

[1]Man-in-the-middle - атака с перехватом данных на пути при участии человека.

элемент <AuthenticationData> в инициализационном сообщении DSKPP. Если атакующий способен захватить это сообщение, он может соперничать с легитимным пользователем при инициализации ключа. DSKPP на основе транспорта с защитой конфиденциальности и целостности, вкупе с рекомендациями параграфа 10.5 **рекомендуется** применять в тех случаях, когда это актуально.

Вставка других сообщений в работающий протокол выглядит как изменение легитимно переданных сообщений.

10.2.5. Повторное использование сообщений

В 4-проходном DSKPP попытка повторить записанное ранее сообщение DSKPP будет обнаружена, поскольку использование nonce говорит об активности обеих сторон. Например, DSKPP Client знает, что сервер, с которым он взаимодействует «жив», поскольку сервер **должен** создать MAC для переданной клиенту информации.

То же самое верно и для 2-проходных DSKPP, благодаря требованию к клиенту передавать R в сообщении <KeyProvClientHello>. Сервер будет включать R в расчёт MAC.

10.2.6. Смена порядка сообщений

Атакующий может попытаться изменить порядок сообщений в 4-проходном DSKPP, но это будет обнаружен, поскольку каждое сообщение имеет уникальный тип. Отметим, что в 2-проходном DSKPP атаки с изменением порядка сообщений невозможны, поскольку каждая сторона передаёт не больше одного сообщения разом.

10.2.7. Атаки с участием человека (MITM)

В дополнение к другим активным этапам злоумышленник в MITM-атаке может предоставить клиенту DSKPP свой открытый ключ. Эта угроза и способы защиты от неё описаны в параграфе 4.1.1. Злоумышленник в MITM-атаке может выступать в качестве прокси, не мешая работе DSKPP, но получая нужную информацию (см. параграф 10.3).

10.3. Пассивные атаки

Пассивные атакующие могут перехватывать DSKPP для получения информации, которая может служит для подмены пользователей, организации активных атак и т. д.

Если DSKPP работает без защиты конфиденциальности на транспортном уровне, злоумышленник в пассивной атаке может узнать:

- криптографические модули, которыми обладает конкретный пользователь;
- идентификаторы ключей этих криптографических модулей и другие атрибуты, относящиеся к ключам (например, срок действия);
- версии DSKPP и криптографических алгоритмов, поддерживаемые конкретным клиентом и сервером DSKPP;
- любые значения, представленные в расширении <extension>, которое является частью <KeyProvClientHello>.

Всякий раз, когда перечисленные угрозы актуальны, для DSKPP **должен** использоваться транспорт с защитой конфиденциальности. Если для описанных выше случаев применимы MITM-атаки, транспорт **должен** также поддерживать аутентификацию на стороне сервера.

10.4. Криптографические атаки

Атакующий с неограниченным доступом к инициализированному криптомодулю может использовать этот модуль в качестве предсказатель заранее рассчитанных значений, которые позднее могут использоваться для того, чтобы представить себя сервером DSKPP. В параграфе 4.1.1 подробно рассматривается эта угроза и даны **рекомендации** по защите.

Разработчикам известно, что криптографические алгоритмы со временем стареют. По мере разработки новых криптографических методов и роста производительности вычислений, время, требуемое для взлома конкретного криптоалгоритма, будет уменьшаться. Поэтому реализации криптографических алгоритмов **следует** делать модульными с возможностью простой вставки новых алгоритмов. Т. е. разработчикам **следует** быть готовыми к регулярной замене алгоритмов в своих реализациях.

10.5. Взаимодействие DSKPP с засвидетельствованием пользователей

Если созданные DSKPP ключи будут связаны с конкретным пользователем на сервере DSKPP (или сервере, которому доверяет и с которым взаимодействует сервер DSKPP), для защиты от угроз, где атакующий заменяет предоставленный клиентом шифрованный R_C своим R'C (независимо от применения вариации открытого ключа или вариации общего секрета DSKPP для шифрования клиентского nonce), серверу **не следует** представлять созданный K_TOKEN для связывания с криптографическим модулем, если пользователь одновременно не подтвердит владение устройством, на котором установлен криптографический модуль, содержащий K_TOKEN и не представит ту или иную аутентификационную информацию (например, Authentication Code) по отдельному каналу. Например, если криптомодуль представляет собой маркер одноразового пароля, для аутентификации пользователя может потребоваться как одноразовый пароль, созданный криптомодулем, так и представленный по отдельному каналу код аутентификации (Authentication Code), чтобы сервер «представил» сгенерированное для данного пользователя значение OTP. В предпочтительном варианте пользователю **следует** выполнять эту операцию не с того хоста, который использовался для инициализации ключей в криптографическом модуле, чтобы минимизировать риск вмешательства в процесс вредоносных программ у клиента.

> Примечание. Сценарий, где атакующий заменяет представленный клиентом R_C своим R'C, не подходит для двухпроходного DSKPP, поскольку клиент не предоставляет энтропии для K_TOKEN. Атака как таковая (и меры противодействия) остаётся применимой для двухпроходного DSKPP, однако она становится по сути MITM-атакой.

Другая угроза связана с возможностью атакующего вынудить пользователя к аутентификации у него, а не у легитимной службы до запуска DSKPP. При успехе атакующий сможет выдать себя легитимному сервису за этого пользователя и затем получить действительный триггер DSKPP. При использовании варианта открытого ключа DSKPP это может

привести к тому, что атакующий сможет (после успешного запуска DSKPP) выдать себя за пользователя. Поэтому **должны** применяться обычные меры предосторожности, чтобы пользователи аутентифицировались только легитимными службами.

10.6. Прочие вопросы

10.6.1. Вклад клиента в энтропию K_TOKEN

В 4-проходном DSKPP клиент и сервер предоставляют случайный материал для K_TOKEN так, что обе стороны могут проверить свой вклад в результирующий ключ. В определённом здесь 2-проходном варианте DSKPP вклад в энтропию K_TOKEN вносит лишь сервер. Это означает, что повреждённый или взломанный генератор (псевдо)случайных чисел может привести к более серьёзным повреждениям, нежели в 4-проходном варианте. Поэтому реализациям серверов **следует** быть крайне осторожными, чтобы предотвратить такие ситуации.

10.6.2. Подтверждение ключа

4-проходные серверы DSKPP обеспечивают подтверждение ключей с помощью MAC в R_C сообщений <KeyProvServerFinished>. В 2-проходном варианте DSKPP, описанном здесь, подтверждение ключа обеспечивается MAC, включающим R, с использованием K_MAC.

10.6.3. Аутентификация сервера

Серверы DSKPP **должны** аутентифицировать себя всякий раз, когда успешный запуск 2-проходного протокола DSKPP приводит к замене имеющегося K_TOKEN на K_TOKEN' или возможна DoS-атака denial-of-service[1], когда несанкционированный сервер DSKPP может заменить K_TOKEN другим ключом. В 2-проходном DSKPP серверы аутентифицируются включением расширения AuthenticationDataType, содержащего MAC, как описано в разделе 5 для 2-проходного DSKPP.

Всякий раз, когда успешный запуск 2-проходного протокола DSKPP приводит к замене имеющегося K_TOKEN на K_TOKEN', клиент и сервер DSKPP должны выполнить указанные ниже процедуры для предотвращения DoS-атак, в который несанкционированный сервер DSKPP заменяет K_TOKEN другим ключом.

- Сервер DSKPP **должен** использовать расширение AuthenticationDataType для передачи второго MAC, рассчитанного в соответствии с параграфом 5.2.2.
- Клиент DSKPP **должен** проверить подлинность сервера с использованием MAC из расширения AuthenticationDataType, полученного от сервера DSKPP, к которому клиент подключён.

10.6.4. Аутентификация пользователя

Сервер DSKPP **должен** аутентифицировать клиента, чтобы убедиться в доставке K_TOKEN на нужное устройство. Для этого **следует** рассмотреть приведённые ниже аспекты.

- При использовании Authentication Code для проверки подлинности клиента возможны атаки по словарю на Authentication Data.
- Размер Authentication Code при работе по незащищённому каналу **следует** делать больше, чем при защищённом канале. Когда устройство, например, мобильный телефон с небольшим экраном, не может обрабатывать длинное значение Authentication Code с удобством для пользователя, DSKPP следует применять защищённые каналы.
- При использовании незащищённого канала Authentication Code **следует** передавать серверу с кодом MAC, как указано в параграфе 3.4.1. Authentication Code и nonce **должны** быть достаточно стойкими для предотвращения восстановления Authentication Code по хэшированным данным (HMAC[2]). С учётом передачи nonce в открытом виде с использованием незащищённого транспорта, криптостойкость Authentication Data определяется в основном качеством Authentication Code.
- Когда Authentication Code передаётся от сервера DSKPP на устройство в триггерном сообщении инициализации DSKPP, такое сообщение может быть перехвачено злоумышленником для захвата инициализации ключа. Для предотвращения этого транспортный уровень, используемый для передачи триггера DSKPP, **должен** обеспечивать защиту конфиденциальности и целостности (например, защищённая сессия в браузере).

10.6.5. Защита ключа в двухпроходном DSKPP

Для разных способов применения 2-проходного DSKPP определены три основных метода защиты, которые **должны** поддерживаться форматом ключей, таким как [RFC6030] и [RFC6031]. Поэтому защита ключа в 2-проходном DSKPP зависит от защиты формата пакетов ключей, выбранного для работы протокола. Ниже приведены некоторые соображения для метода передачи ключей с помощью пароля (Passphrase-Based Key Wrap).

Методу Passphrase-Based Key Wrap **следует** применять функцию PBKDF2 из [PKCS-5] для генерации ключа шифрования из парольной фразы и строки «затравки» (salt). Важно отметить, что шифрование на основе парольной фразы в общем случае обеспечивает ограниченную защиту, несмотря на применение затравки и счётчика итераций в PBKDF2 для осложнения атак. Поэтому реализациям **следует** принимать дополнительные меры по усилению защиты в методе Passphrase-Based Key Wrap. Для этого **следует** рассмотреть перечисленные ниже меры.

- Парольная фраза совпадает с компонентой одноразового пароля в Authentication Code (см. описание формата AC в параграфе 3.4.1). Парольную фразу **следует** выбирать осторожно, и **следует** принимать во внимание рекомендации, такие как приведённые в [NIST-PWD].

[1]Denial-of-service - отказ в обслуживании.
[2]Hashed MAC.

- **Следует** использовать разные парольные фразы для каждой инициализации ключа, если это возможно (например, **следует** избегать применения «глобальной» парольной фразы для пакетных криптографических модулей). Одним из вариантов является использование случайных парольных фраз.
- Парольные фразы **следует** надёжно защищать, если они хранятся на сервере и/или в криптографическом модуле, а доставлять на устройства **следует** с применением защищённых методов.
- **Следует** реализовать предварительную аутентификацию пользователей, чтобы предотвратить доставку K_TOKEN подставным получателям.
- Число итераций в PBKDF2 **следует** делать большим, чтобы навязать атакующему больший объем работы при использовании методов подбора - brute-force (см. рекомендации [PKCS-5]). Однако **необходимо** отметить, что увеличения числа итераций повышает объем работы и для легитимных криптомодулей при расшифровке недавно полученного K_TOKEN. Серверы **могут** применять сравнительно небольшое число итераций для способствования устройствам с ограниченными вычислительными возможностями, таким как PDA и сотовые телефоны, когда применяются другие меры защиты и безопасность Passphrase-Based Key Wrap не снижается.
- **Следует** по возможности применять TLS [RFC5246] для защиты 2-проходного режима работы протокола. Защита на транспортном уровне обеспечивает дополнительную безопасность недавно созданным K_TOKEN.

10.6.6. Гибкость алгоритмов

Для многих протоколов требуется гибкость в плане алгоритмов. Одна из причин этого заключается в том, что многие протоколы в прошлом использовали фиксированный размер полей, таких как выходные данные, ключи и т. п. Это не относится к DSKPP, за исключением размера ключа при расчёте DSKPP-PRF. Другой причиной является отсутствие согласования. Это тоже не относится к DSKPP, за исключением применения SHA-256 в подтверждающем MAC сообщении. Обновление размера ключей для DSKPP-PRF или алгоритма подтверждающих MAC сообщений потребует новой версии протокола, которая поддерживается атрибутом Version.

11. Поддержка других языков

DSKPP предназначен для межмашинных коммуникаций и его элементами являются маркеры (token), не предназначенные для прямого использования людьми. Обмен информацией в DSKPP использует XML. Все процессоры XML должны понимать кодировку UTF-8 [RFC3629], поэтому все клиенты и серверы DSKPP **должны** понимать XML в кодировке UTF-8. Кроме того, клиентам и серверам DSKPP **недопустимо** представлять XML в кодировке, отличной от UTF-8.

12. Взаимодействие с IANA

Этот документ требует нескольких регистраций в IANA, описанных ниже.

12.1. Регистрация субпространства имён URN

Этот параграф регистрирует пространство имён XML "urn:ietf:params:xml:ns:keyprov:dskpp" в соответствии с [RFC3688].

URI: urn:ietf:params:xml:ns:keyprov:dskpp

Регистрационные контакты:

IETF, рабочая группа KEYPROV (keyprov@ietf.org), Andrea Doherty (andrea.doherty@rsa.com)

```
XML:
   BEGIN
      <?xml version="1.0"?>
      <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
      <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
      <head>
         <title>DSKPP Messages[1]</title>
      </head>
      <body>
         <h1>Namespace for DSKPP Messages[2]</h1>
         <h2>urn:ietf:params:xml:ns:keyprov:dskpp</h2>
         <p>See RFC 6063</p>
      </body>
      </html>
   END
```

12.2. Регистрация XML-схемы

Этот параграф регистрирует схему XML в соответствии с [RFC3688].

URI: urn:ietf:params:xml:ns:keyprov:dskpp

Регистрационные контакты:

IETF, рабочая группа KEYPROV (keyprov@ietf.org), Andrea Doherty (andrea.doherty@rsa.com)

Схема:

XML для этой схемы можно найти в разделе 8 данного документа.

12.3. Регистрация MIME Media Type

Этот параграф регистрирует тип носителя MIME "application/dskpp+xml".

[1]Сообщения DSKPP.
[2]Пространство имён сообщений DSKPP.

```
   To:  ietf-types@iana.org
   Subject:  Registration of MIME media type application/dskpp+xml
   MIME media type name:  application
   MIME subtype name:  dskpp+xml
   Required parameters:  (none)
   Optional parameters:  charset
      Indicates the character encoding of enclosed XML.[1]
   Encoding considerations:  Uses XML, which can employ 8-bit characters, depending on the character
encoding used.  See [RFC3023], Section 3.2.  Implementations need to support UTF-8 [RFC3629].[2]
   Security considerations:  This content type is designed to carry protocol data related to key
management.  Security mechanisms are built into the protocol to ensure that various threats are dealt
with.  Refer to Section 10 of RFC 6063 for more details[3]
   Interoperability considerations:  None
   Published specification:  RFC 6063.
   Applications that use this media type:  Protocol for key exchange.[4]
   Additional information:
      Magic Number(s): (none)
      File extension(s): .xmls
      Macintosh File Type Code(s): (none)
   Person & email address to contact for further information:
      Andrea Doherty (andrea.doherty@rsa.com)
   Intended usage:  LIMITED USE[5]
   Author/Change controller:  The IETF
   Other information:  This media type is a specialization of application/xml
      [RFC3023], and many of the considerations described there also apply to
      application/dskpp+xml.[6]
```

12.4. Регистрация кодов состояния

Этот параграф регистрирует коды состояний, включаемые в каждое сообщение с откликом DSKPP. Коды состояний заданы в определении типа <StatusCode>, содержащемся в схеме XML (раздел 8). Реестр кратко описан ниже.

```
Related Registry:
      KEYPROV DSKPP Registries, Status codes for DSKPP
   Defining RFC:
      RFC 6063.
   Registration/Assignment Procedures:
      Following the policies outlined in [RFC3575], the IANA policy for
      assigning new values for the status codes for DSKPP MUST be
      "Specification Required" and their meanings MUST be documented in
      an RFC or in some other permanent and readily available reference,
      in sufficient detail that interoperability between independent
      implementations is possible.  No mechanism to mark entries as
      "deprecated" is envisioned.  It is possible to update entries from the registry.[7]
Registrant Contact:
      IETF, KEYPROV working group (keyprov@ietf.org),
      Andrea Doherty (andrea.doherty@rsa.com).
```

12.5. Регистрация версии DSKPP

Этот параграф регистрирует номер версии DSKPP, как показано ниже.

Версия DSKPP	*Спецификация*
1.0	Этот документ

Для определения новой версии DSKPP требуется стандартизация. Отмена, удаление или изменение имеющихся версий DSKPP не предусмотрены.

12.6. Субреестр PRF Algorithm ID

Эта спецификация основана на криптографическом примитиве DSKPP-PRF, которые обеспечивает детерминированное преобразование секретного ключа k и строки октетов переменного размера s в битовую строку заданного размера dsLen. С точки зрения этой спецификации DSKPP-PRF является функциональным «чёрным ящиком», который на основе входных данных создаёт псевдослучайное значение, которое может быть реализовано любым подходящим и правомочным криптографическим методом. В параграфе 3.4.2 представлены две реализации DSKPP-PRF - DSKPP-PRF-AES и DSKPP-PRF-SHA256.

Этот параграф регистрирует идентификаторы, связанные с этими реализациями. Для субреестров PRF Algorithm ID применяется процедура Specification Required в соответствии с RFC 5226 [RFC5226]. Обновления **должны** документироваться в RFC или других постоянно и легко доступных документах с описанием, достаточно подробным для обеспечения возможности взаимодействия независимых реализаций.

Для отмены субреестра требуется одобрение эксперта. После признания PRF Algorithm ID устаревшим, алгоритм **не следует** применять в новых реализациях.

[1]Указывает кодировку символов вложенного XML.
[2]Используется XML с возможностью поддержки 8-битовых символов в зависимости от применяемой кодировки (см. параграф 3.2 в [RFC3023]). Реализации должны поддерживать UTF-8 [RFC3629].
[3]Этот тип носителя предназначен для переноса протокольных данных, связанных с управлением ключами. Механизмы защиты встроены в протокол для предотвращения различных угроз. Дополнительную информацию можно найти в разделе 10 RFC 6063.
[4]Протокол для обмена ключами.
[5]Ограниченное применение.
[6]Этот тип носителя является специализацией application/xml [RFC3023] и многие вопросы, рассмотренные в упомянутом документе, применимы и к application/dskpp+xml.
[7]В соответствии с политикой, указанной в [RFC3575], процедура IANA для выделения новых значений кодов состояния DSKPP **должна** быть Specification Required и смысл новых значений **должен** быть документирован в RFC или ином постоянно и легко доступном документе с подробностями, достаточными для обеспечения взаимодействия независимых реализаций. Механизма маркировки записей как устаревших (deprecated) не предусмотрено. Записи в реестре можно обновлять.

12.6.1. DSKPP-PRF-AES

Этот параграф регистрирует приведённый ниже реестр пространства имён IETF XML.

```
Common Name:
   DSKPP-PRF-AES
URI:
   urn:ietf:params:xml:ns:keyprov:dskpp:prf-aes-128
Identifier Definition:
   The DSKPP-PRF-AES algorithm realization is defined in
   Appendix D.2.2 of this document.[1]
Registrant Contact:
   IETF, KEYPROV working group (keyprov@ietf.org), Andrea Doherty (andrea.doherty@rsa.com)
```

12.6.2. DSKPP-PRF-SHA256

Этот параграф регистрирует приведённый ниже реестр пространства имён IETF XML.

```
Common Name:
   DSKPP-PRF-SHA256
URI:
   urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256
Identifier Definition:
   The DSKPP-PRF-SHA256 algorithm realization is defined in
   Appendix D.3.2 of this document[2].
Registrant Contact:
   IETF, KEYPROV working group (keyprov@ietf.org), Andrea Doherty (andrea.doherty@rsa.com)
```

12.7. Регистрация ключевого контейнера

Этот параграф регистрирует тип Key Container.

```
Key Container:
   The registration name for the Key Container.[3]
Specification:
   Key Container defines a key package format that specifies how a
   key should be protected using the three key protection methods
   provided in Section 5.1.[4]
Registration Procedure:
   Following the policies outlined in [RFC3575], the IANA policy for
   assigning new values for the status codes for DSKPP MUST be
   "Specification Required" and their meanings MUST be documented in
   an RFC or in some other permanent and readily available reference,
   in sufficient detail that interoperability between independent
   implementations is possible.[5]
Deprecated:
   TRUE if based on expert approval this entry has been deprecated
   and SHOULD NOT be used in any new implementations.  Otherwise,
   FALSE.[6]
Identifiers:
   The initial URIs for the Key Container defined for this version of
   the document are listed here:[7]
   Name:  PSKC Key Container
   URI:  urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
   Specification:  [RFC6030]
   Deprecated:  FALSE
   Name:  SKPC Key Container
   URI:  urn:ietf:params:xml:ns:keyprov:dskpp:skpc-key-container
   Specification:  [RFC6031]
   Deprecated:  FALSE
   Name:  PKCS12 Key Container
   URI:  urn:ietf:params:xml:ns:keyprov:dskpp:pkcs12-key-container
   Specification:  [PKCS-12]
   Deprecated:  FALSE
   Name:  PKCS5-XML Key Container
   URI:  urn:ietf:params:xml:ns:keyprov:dskpp:pkcs5-xml-key-container
   Specification:  [PKCS-5-XML]
   Deprecated:  FALSE
Registrant Contact:
   IETF, KEYPROV working group (keyprov@ietf.org), Andrea Doherty (andrea.doherty@rsa.com)
```

13. Интеллектуальная собственность

RSA и RSA Security являются (зарегистрированными) торговыми знаками RSA Security, Inc. В США и/или других странах. Названия другой продукции и услуг могут быть торговыми знаками соответствующих владельцев.

[1]Реализация алгоритма DSKPP-PRF-AES определена в приложении D.2.2 к этому документу.
[2]Реализация алгоритма DSKPP-PRF-SHA256 определена в приложении D.3.2 к этому документу.
[3]Регистрационное имя для контейнера ключей.
[4]Key Container определяет формат упаковки ключей, который задаёт способ защиты ключей с применением трёх методов, представленных в параграфе 5.1.
[5]В соответствии с политикой, указанной в [RFC3575], процедура IANA для выделения новых значений кодов состояния DSKPP **должна** быть Specification Required и смысл новых значений **должен** быть документирован в RFC или ином постоянно и легко доступном документе с подробностями, достаточными для обеспечения взаимодействия независимых реализаций.
[6]TRUE, если с одобрения эксперта эта запись признана устаревшей и её **не следует** применять в новых реализациях. В остальных случаях FALSE.
[7]Начальные URI для Key Container, определённые в этой версии документа, перечислены ниже.

14. Участники работы

Эта работа основана на информации из [RFC4758], созданного Magnus Nystrom, с усовершенствованиями, заимствованными из отдельного документа, авторами которого являются Mingliang Pei и Salah Machani (например, аутентификация пользователей и поддержка разных форматов упаковки ключей).

Авторы признательны Philip Hoyer за его работу по согласованию схем DSKPP и PSKC.

Спасибо Hannes Tschofenig и Phillip Hallam-Baker за их рецензии, отклики и вклад в текст документа.

15. Благодарности

Авторы признательны перечисленным ниже людям за рецензирование предыдущих версий документа DSKPP.

- Dr. Ulrike Meyer (июнь 2007);
- Niklas Neumann (июнь 2007);
- Shuh Chang (июнь 2007);
- Hannes Tschofenig (июнь и август 2007);
- Sean Turner (август 2007 и июль 2008);
- John Linn (август 2007);
- Philip Hoyer (сентябрь 2007);
- Thomas Roessler (ноябрь 2007);
- Lakshminath Dondeti (комментарии в декабре 2007);
- Pasi Eronen (комментарии в декабре 2007);
- Phillip Hallam-Baker (обзор и редактирование в ноябре 2008 и январе 2009);
- Alexey Melnikov (май 2010);
- Peter Saint-Andre (май 2010).

Авторы благодарны перечисленным ниже людям за их вклад в отдельные аспекты разработки DSKPP.

- Anders Rundgren (формат упаковки ключей и данные аутентификации клиентов);
- Thomas Roessler (привязка HTTP);
- Hannes Tschofenig (привязка HTTP);
- Phillip Hallam-Baker (реестр алгоритмов);
- N. Asokan (первым отметил слабость Authentication Data).

В заключение авторы хотели бы поблагодарить Robert Griffin за то, что он открыл для них каналы связи с IEEE P1619.3 Key Management Group и помог оставаться в курсе возможных направления сотрудничества (особенно в части представления ключей и глобальны идентификаторов ключей).

16. Литература

16.1. Нормативные документы

[FIPS180-SHA] National Institute of Standards and Technology, "Secure Hash Standard", FIPS 180-2, February 2004, <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf>.

[FIPS197-AES] National Institute of Standards and Technology, "Specification for the Advanced Encryption Standard (AES)", FIPS 197, November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[ISO3309] International Organization for Standardization, "ISO Information Processing Systems - Data Communication - High-Level Data Link Control Procedure - Frame Structure", ISO 3309, 3rd Edition, October 1984.

[PKCS-1] RSA Laboratories, "RSA Cryptography Standard", PKCS #1 Version 2.1, June 2002, <http://www.rsasecurity.com/rsalabs/pkcs/>.

[PKCS-5] RSA Laboratories, "Password-Based Cryptography Standard", PKCS #5 Version 2.0, March 1999, <http://www.rsasecurity.com/rsalabs/pkcs/>.

[PKCS-5-XML] RSA Laboratories, "XML Schema for PKCS #5 Version 2.0", PKCS #5 Version 2.0 Amd.1 (FINAL DRAFT), October 2006, <http://www.rsasecurity.com/rsalabs/pkcs/>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, September 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, September 2005.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, June 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, September 2009.

[RFC6030] Hoyer, P., Pei, M., and S. Machani, "Portable Symmetric Key Container (PSKC)", RFC 6030, October 2010.

[UNICODE] Davis, M. and M. Duerst, "Unicode Normalization Forms", March 2001, <http://www.unicode.org/unicode/reports/tr15/tr15-21.html>.

[XML] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation, November 2008, <http://www.w3.org/TR/2006/REC-xml-20060816/>.

[XMLDSIG] W3C, "XML Signature Syntax and Processing", W3C Recommendation, February 2002, <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/>.

[XMLENC] W3C, "XML Encryption Syntax and Processing", W3C Recommendation, December 2002, <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/>.

16.2. Дополнительная литература

[CT-KIP-P11] RSA Laboratories, "PKCS #11 Mechanisms for the Cryptographic Token Key Initialization Protocol", PKCS #11 Version 2.20 Amd.2, December 2005, <http://www.rsasecurity.com/rsalabs/pkcs/>.

[FAQ] RSA Laboratories, "Frequently Asked Questions About Today's Cryptography", Version 4.1, 2000.

[NIST-PWD] National Institute of Standards and Technology, "Password Usage", FIPS 112, May 1985, <http://www.itl.nist.gov/fipspubs/fip112.htm>.

[NIST-SP800-38B] International Organization for Standardization, "Recommendations for Block Cipher Modes of Operation: The CMAC Mode for Authentication", NIST SP800-38B, May 2005, <http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf>.

[NIST-SP800-57] National Institute of Standards and Technology, "Recommendation for Key Management - Part I: General (Revised)", NIST 800-57, March 2007, <http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf>.

[PKCS-11] RSA Laboratories, "Cryptographic Token Interface Standard", PKCS #11 Version 2.20, June 2004, <http://www.rsasecurity.com/rsalabs/pkcs/>.

[PKCS-12] "Personal Information Exchange Syntax Standard", PKCS #12 Version 1.0, 2005, <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4758] Nystroem, M., "Cryptographic Token Key Initialization Protocol (CT-KIP) Version 1.0 Revision 1", RFC 4758, November 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6031] Turner, S. and R. , "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type", RFC 6031, December 2010.

[XMLNS] W3C, "Namespaces in XML", W3C Recommendation, January 1999, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

Приложение A. Варианты применения

Предполагается, что DSKPP будет применяться для предоставления симметричных ключей криптографическим модулям в различных вариантах, каждый из которых имеет свои требования, как описано ниже. Это приложение является информационной частью документа.

A.1. Запрос одного ключа

Обычный сценарий заключается в том, что криптографический модуль запрашивает симметричный ключ у сервера обеспечения, расположенного в локальной сети или в Internet. В зависимости от варианта развёртывания. сервер обеспечения может генерировать новый ключ «на лету» или использовать созданный заранее ключ, например, предоставленный традиционным сервером эмиссии ключей. Сервер обеспечения назначает симметричному ключу уникальный идентификатор и предоставляет его криптографическому модулю.

A.2. Запрос множества ключей

Криптографический модуль передаёт множество запросов для симметричных ключей одному серверу обеспечения. Симметричные ключи не обязательно должны быть однотипными, т. е. они могут применяться с разными криптографическими алгоритмами, включая алгоритмы одноразовых ключей аутентификации и алгоритм шифрования AES .

A.3. Засвидетельствование пользователя

В некоторых вариантах развёртывания. эмитент ключей может полагаться на стороннюю службу обеспечения. В таких случаях эмитент направляет запросы на предоставления от криптомодулей службе обеспечения. Таким образом, эмитент отвечает за аутентификацию пользователей тем или иным автономным (out-of-band) способом до предоставления пользователю права получить ключ. После предоставления эмитентом такого права он выдаёт пользователю код аутентификации (Authentication Code) и делает его доступным службе обеспечения, чтобы пользователь мог подтвердить своё право на получение ключей.

A.4. Политика тайм-аутов предоставления

Эмитент может предоставить пользователю в процессе регистрации временный код аутентификации, который пользователь будет вводить в криптомодуль для представления себя серверу обеспечения. Сервер будет разрешать предоставление ключа криптомодулю на пользовательском устройстве при необходимости проверить подлинность пользователя лишь в случае ввода пользователем корректного Authentication Code с фиксированным сроком действия, заданным эмитентом.

A.5. Обновление ключа

Криптомодуль запрашивает обновление симметричного ключевого материала, присоединённого к идентификатору ключа в отличие от сохранения постоянного значения ключа и обновления метаданных. Такая потребность может возникнуть в случае, когда пользователь хочет обновить своё устройство, на котором размещён криптомодуль, или при завершении срока действия ключа. Когда пользователь применяет один и тот же криптомодуль, например, для выполнения строгой аутентификации на множестве Web-сайтов со входом в систему (login), сохранение идентификатора ключа избавляет пользователя от необходимости регистрировать на каждом сайте новый ключ.

A.6. Замена предустановленного ключа

Этот вариант представляет особый случай обновления симметричного ключа, когда локальный администратор может аутентифицировать пользователя процедурно без инициирования процесса обеспечения. Это также позволяет эмитенту устройств заранее загружать ключ в криптомодуль с требованием замены этого ключа до начала использования модуля. В другом варианте это может применяться организациями, где устройства могут переходить от одного пользователя к другому. В этом случае эмитент ключей предоставляет новый симметричный ключ для криптомодуля, размещённого на устройстве, который до этого владел другой пользователь.

Отметим, что этот вариант по существу совпадает с предыдущим если при обновлении применяется тот же идентификатор ключа.

A.7. Заранее известный заводской ключ

Криптомодуль загружается в смарт-карту после её выдачи пользователю. Симметричный ключ для криптомодуля обеспечивается с использованием защищённого канала, имеющегося в большинстве платформ для смарт-карт. Это позволяет напрямую организовать защищённый канал между микросхемой смарт-карты и сервером обеспечения. Например, команды карт (т. е. APDU[1]) шифруются с заранее введённым производителем карты ключом и передаются напрямую в микросхему карты, обеспечивая защищённую послепродажную подготовку в «полевых» условиях. Этот защищённый поток может проходить через TLS [RFC5246] и другие границы защищённого транспорта.

Отметим, что для этого варианта имеется два предварительных условия - протокол должен быть туннелируемым, а сервер обеспечения должен знать заданный производителем ключ.

A.8. Сквозная защита ключевого материала

В этом варианте TLS не обеспечивает сквозной защиты ключевого материала, передаваемого от сервера обеспечения в криптомодуль. Например, TLS может завершаться на приложении, размещённом в PC, а не в криптомодуле (т. е. конечной точке), расположенном в устройстве хранения данных [RFC5246]. Согласование ключа с взаимной аутентификацией обеспечивает сквозную защиту, которую не может предоставить TLS.

Приложение B. Примеры

В этом приложении даны примеры сообщений, иллюстрирующие параметры, кодирование и семантику для 2- и 4-проходного обмена DSKPP. Примеры написаны с использованием XML и синтаксически корректны. Однако значения MAC и шифров являются вымышленными. Приложение является информационной частью документа.

B.1. Триггерное сообщение

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvTrigger Version="1.0"
  xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
  xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc">
  <dskpp:InitializationTrigger>
    <dskpp:DeviceIdentifierData>
        <dskpp:DeviceId>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
            <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
```

[1]Application Protocol Data Unit - блок данных прикладного протокола.

```
            <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
          </dskpp:DeviceId>
        </dskpp:DeviceIdentifierData>
        <dskpp:KeyID>SE9UUDAwMDAwMDAx</dskpp:KeyID>
        <dskpp:TokenPlatformInfo KeyLocation="Hardware"
          AlgorithmLocation="Software"/>
        <dskpp:AuthenticationData>
          <dskpp:ClientID>31300257</dskpp:ClientID>
          <dskpp:AuthenticationCodeMac>
            <dskpp:IterationCount>512</dskpp:IterationCount>
            <dskpp:Mac>4bRJf9xXd3KchKoTenHJiw==</dskpp:Mac>
          </dskpp:AuthenticationCodeMac>
        </dskpp:AuthenticationData>
        <dskpp:ServerUrl>keyprovservice.example.com
          </dskpp:ServerUrl>
      </dskpp:InitializationTrigger>
    </dskpp:KeyProvTrigger>
```

B.2. 4-проходный протокол

B.2.1. <KeyProvClientHello> без предшествующего триггера

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvClientHello
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0">
    <dskpp:DeviceIdentifierData>
        <dskpp:DeviceId>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
            <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
            <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
        </dskpp:DeviceId>
    </dskpp:DeviceIdentifierData>
    <dskpp:SupportedKeyTypes>
        <dskpp:Algorithm>
            urn:ietf:params:xml:ns:keyprov:pskc:hotp
        </dskpp:Algorithm>
        <dskpp:Algorithm>
http://www.rsa.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES
        </dskpp:Algorithm>
    </dskpp:SupportedKeyTypes>
    <dskpp:SupportedEncryptionAlgorithms>
        <dskpp:Algorithm>
            http://www.w3.org/2001/04/xmlenc#aes128-cbc
        </dskpp:Algorithm>
    </dskpp:SupportedEncryptionAlgorithms>
    <dskpp:SupportedMacAlgorithms>
        <dskpp:Algorithm>
            urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256
        </dskpp:Algorithm>
    </dskpp:SupportedMacAlgorithms>
    <dskpp:SupportedProtocolVariants>
        <dskpp:FourPass/>
    </dskpp:SupportedProtocolVariants>
    <dskpp:SupportedKeyPackages>
        <dskpp:KeyPackageFormat>
            urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
        </dskpp:KeyPackageFormat>
    </dskpp:SupportedKeyPackages>
</dskpp:KeyProvClientHello>
```

B.2.2. <KeyProvClientHello> с предшествующим триггером

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvClientHello
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0">
    <dskpp:DeviceIdentifierData>
        <dskpp:DeviceId>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
            <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
            <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
        </dskpp:DeviceId>
    </dskpp:DeviceIdentifierData>
    <dskpp:KeyID>SE9UUDAwMDAwMDAx</dskpp:KeyID>
    <dskpp:SupportedKeyTypes>
        <dskpp:Algorithm>
            urn:ietf:params:xml:ns:keyprov:pskc:hotp
        </dskpp:Algorithm>
```

```
            <dskpp:Algorithm>
    http://www.rsa.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES
            </dskpp:Algorithm>
        </dskpp:SupportedKeyTypes>
        <dskpp:SupportedEncryptionAlgorithms>
            <dskpp:Algorithm>http://www.w3.org/2001/04/xmlenc#aes128-cbc</dskpp:Algorithm>
        </dskpp:SupportedEncryptionAlgorithms>
        <dskpp:SupportedMacAlgorithms>
            <dskpp:Algorithm>urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256</dskpp:Algorithm>
        </dskpp:SupportedMacAlgorithms>
        <dskpp:SupportedProtocolVariants>
          <dskpp:FourPass/>
        </dskpp:SupportedProtocolVariants>
        <dskpp:SupportedKeyPackages>
            <dskpp:KeyPackageFormat>
                urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
            </dskpp:KeyPackageFormat>
        </dskpp:SupportedKeyPackages>
    </dskpp:KeyProvClientHello>
```

B.2.3. <KeyProvServerHello> без предшествующего триггера

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvServerHello
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0"
    Status="Continue"
    SessionID="4114">
    <dskpp:KeyType>urn:ietf:params:xml:ns:keyprov:pskc:hotp</dskpp:KeyType>
    <dskpp:EncryptionAlgorithm>
        http://www.w3.org/2001/04/xmlenc#aes128-cbc
    </dskpp:EncryptionAlgorithm>
    <dskpp:MacAlgorithm>
        urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256
    </dskpp:MacAlgorithm>
    <dskpp:EncryptionKey>
      <ds:KeyName>Example-Key1</ds:KeyName>
    </dskpp:EncryptionKey>
    <dskpp:KeyPackageFormat>
        urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
    </dskpp:KeyPackageFormat>
    <dskpp:Payload>
        <dskpp:Nonce>EjRWeJASNFZ4kBI0VniQEg==</dskpp:Nonce>
    </dskpp:Payload>
</dskpp:KeyProvServerHello>
```

B.2.4. <KeyProvServerHello> в предположении обновления ключа

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvServerHello
  xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
  xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  Version="1.0"
  SessionID="4114"
  Status="Continue">
  <dskpp:KeyType>
    urn:ietf:params:xml:schema:keyprov:otpalg#SecurID-AES
  </dskpp:KeyType>
  <dskpp:EncryptionAlgorithm>
     http://www.w3.org/2001/04/xmlenc#aes128-cbc
  </dskpp:EncryptionAlgorithm>
  <dskpp:MacAlgorithm>
     urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256
  </dskpp:MacAlgorithm>
  <dskpp:EncryptionKey>
    <ds:KeyName>Example-Key1</ds:KeyName>
  </dskpp:EncryptionKey>
  <dskpp:KeyPackageFormat>
    urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
  </dskpp:KeyPackageFormat>
  <dskpp:Payload>
    <dskpp:Nonce>qw2ewasde312asder394jw==</dskpp:Nonce>
  </dskpp:Payload>
  <dskpp:Mac
    MacAlgorithm="urn:ietf:params:xml:ns:keyprov:dskpp:prf-aes-128">
    cXcycmFuZG9tMzEyYXNkZXIzOTRqdw==
  </dskpp:Mac>
</dskpp:KeyProvServerHello>
```

B.2.5. <KeyProvClientNonce> с принятым по умолчанию шифрованием

Это сообщение содержит значение nonce, выбранное криптомодулем, R_C шифруется с заданным ключом и алгоритмом.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvClientNonce
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    SessionID="4114"
    Version="1.0">
    <dskpp:EncryptedNonce>
        oTvo+S22nsmS2Z/RtcoF8CTwadRa1PVsRXkZnCihHkU1rPueggrd0NpEWVZR
        16Rg16+FHuTg33GK1wH3wffDZQ==
    </dskpp:EncryptedNonce>
</dskpp:KeyProvClientNonce>
```

B.2.6. <KeyProvServerFinished> - принятое по умолчанию шифрование

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvServerFinished
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0"
    Status="Success"
    SessionID="4114">
    <dskpp:KeyPackage>
        <dskpp:KeyContainer Version="1.0" Id="KC0001">
            <pskc:KeyPackage>
                <pskc:DeviceInfo>
                    <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
                    <pskc:SerialNo>987654321 </pskc:SerialNo>
                    <pskc:StartDate>2009-09-01T00:00:00Z </pskc:StartDate>
                    <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
                </pskc:DeviceInfo>
                <pskc:CryptoModuleInfo>
                    <pskc:Id>CM_ID_001</pskc:Id>
                </pskc:CryptoModuleInfo>
                <pskc:Key
                   Id="MBK000000001"
                   Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
                   <pskc:Issuer>Example-Issuer</pskc:Issuer>
                   <pskc:AlgorithmParameters>
                       <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
                    </pskc:AlgorithmParameters>
                    <pskc:Data>
                        <pskc:Counter>
                            <pskc:PlainValue>0</pskc:PlainValue>
                        </pskc:Counter>
                    </pskc:Data>
                    <pskc:Policy>
                        <pskc:KeyUsage>OTP</pskc:KeyUsage>
                    </pskc:Policy>
                </pskc:Key>
            </pskc:KeyPackage>
        </dskpp:KeyContainer>
    </dskpp:KeyPackage>
    <dskpp:Mac
        MacAlgorithm="urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
        151yAR2NqU5dJzETK+SGYqN6sq6DEH5AgHohra3Jpp4=
    </dskpp:Mac>
</dskpp:KeyProvServerFinished>
```

B.3. 2-проходный протокол

B.3.1. Пример использования метода транспортировки ключей

Клиент указывает поддержку всех методов защиты ключей - Key Transport, Key Wrap, Passphrase-Based Key Wrap.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvClientHello
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0">
    <dskpp:DeviceIdentifierData>
        <dskpp:DeviceId>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
            <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
            <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
        </dskpp:DeviceId>
    </dskpp:DeviceIdentifierData>
    <dskpp:SupportedKeyTypes>
        <dskpp:Algorithm>urn:ietf:params:xml:ns:keyprov:pskc:hotp</dskpp:Algorithm>
        <dskpp:Algorithm>
            http://www.rsa.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES
        </dskpp:Algorithm>
```

```
        </dskpp:SupportedKeyTypes>
        <dskpp:SupportedEncryptionAlgorithms>
            <dskpp:Algorithm>http://www.w3.org/2001/04/xmlenc#rsa_1_5</dskpp:Algorithm>
        </dskpp:SupportedEncryptionAlgorithms>
        <dskpp:SupportedMacAlgorithms>
            <dskpp:Algorithm>urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256</dskpp:Algorithm>
        </dskpp:SupportedMacAlgorithms>
        <dskpp:SupportedProtocolVariants>
            <dskpp:TwoPass>
                <dskpp:SupportedKeyProtectionMethod>
                    urn:ietf:params:xml:schema:keyprov:dskpp:transport
                </dskpp:SupportedKeyProtectionMethod>
                <dskpp:Payload>
                    <ds:KeyInfo>
                        <ds:X509Data>
                            <ds:X509Certificate>
    MIIB5zCCAVCgAwIBAgIESZp/vDANBgkqhkiG9w0BAQUFADA4MQ0wCwYDVQQKEwRJRVRGM
    RMwEQYDVQQLEwpLZXlQcm92IFdHMRIwEAYDVQQDEwlQU0tDIFRlc3QwHhcNMDkwMjE3MD
    kxMzMyWhcNMTEwMjE3MDkxMzMyWjA4MQ0wCwYDVQQKEwRJRVRGMRMwEQYDVQQLEwpLZXl
    Qcm92IFdHMRIwEAYDVQQDEwlQU0tDIFRlc3QwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
    AoGBALCWLDa2ItYJ6su80hd1gL4cggQYdyyKK17btt/aS6Q/eDsKjsPyFIODsxeKVV/uA
    3wLT4jQJM5euKJXkDajzGGOy92+ypfzTX4zDJMkh61SZwlHNJxBKilAM5aW7C+BQ0RvCx
    vdYtzx2LTdB+X/KMEBA7uIYxLfXH2Mnub3WIh1AgMBAAEwDQYJKoZIhvcNAQEFBQADgYE
    Ae875m84sYUJ8qPeZ+NG7REgTvlHTmoCdoByU0LBBLotUKuqfrnRuXJRMeZXaaEGmzY1k
    LonVjQGzjAkU4dJ+RPmiDlYuHLZS41Pg6VMwY+03lhk6I5A/w4rnqdkmwZX/NgXg06aln
    c2pBsXWhL4O7nk0S2ZrLMsQZ6HcsXgdmHo=
                            </ds:X509Certificate>
                        </ds:X509Data>
                    </ds:KeyInfo>
                </dskpp:Payload>
            </dskpp:TwoPass>
        </dskpp:SupportedProtocolVariants>
        <dskpp:SupportedKeyPackages>
            <dskpp:KeyPackageFormat>
                urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
            </dskpp:KeyPackageFormat>
        </dskpp:SupportedKeyPackages>
        <dskpp:AuthenticationData>
            <dskpp:ClientID>AC00000A</dskpp:ClientID>
            <dskpp:AuthenticationCodeMac>
                <dskpp:Nonce>ESIzRFVmd4iZqrvM3e7/ESIzRFVmd4iZqrvM3e7/ESI=</dskpp:Nonce>
                <dskpp:IterationCount>100000</dskpp:IterationCount>
                <dskpp:Mac
                    MacAlgorithm="urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
                    3eRz51ILqiG+dJW2iLcjuA==
                </dskpp:Mac>
            </dskpp:AuthenticationCodeMac>
        </dskpp:AuthenticationData>
    </dskpp:KeyProvClientHello>
```

В этом примере сервер отвечает на предыдущий запрос, возвращая пакет ключей, в котором ключ обеспечения зашифрован с использованием метода защиты ключа Key Transport.

```
    <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <dskpp:KeyProvServerFinished
        xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
        xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
        xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:dkey="http://www.w3.org/2009/xmlsec-derivedkey#"
        xmlns:pkcs5=
           "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"
        Version="1.0"
        Status="Success"
        SessionID="4114">
        <dskpp:KeyPackage>
            <dskpp:KeyContainer Version="1.0" Id="KC0001">
                <pskc:EncryptionKey>
                    <ds:X509Data>
                        <ds:X509Certificate>
    MIIB5zCCAVCgAwIBAgIESZp/vDANBgkqhkiG9w0BAQUFADA4MQ0wCwYDVQQKEwRJRVRGM
    RMwEQYDVQQLEwpLZXlQcm92IFdHMRIwEAYDVQQDEwlQU0tDIFRlc3QwHhcNMDkwMjE3MD
    kxMzMyWhcNMTEwMjE3MDkxMzMyWjA4MQ0wCwYDVQQKEwRJRVRGMRMwEQYDVQQLEwpLZXl
    Qcm92IFdHMRIwEAYDVQQDEwlQU0tDIFRlc3QwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
    AoGBALCWLDa2ItYJ6su80hd1gL4cggQYdyyKK17btt/aS6Q/eDsKjsPyFIODsxeKVV/uA
    3wLT4jQJM5euKJXkDajzGGOy92+ypfzTX4zDJMkh61SZwlHNJxBKilAM5aW7C+BQ0RvCx
    vdYtzx2LTdB+X/KMEBA7uIYxLfXH2Mnub3WIh1AgMBAAEwDQYJKoZIhvcNAQEFBQADgYE
    Ae875m84sYUJ8qPeZ+NG7REgTvlHTmoCdoByU0LBBLotUKuqfrnRuXJRMeZXaaEGmzY1k
    LonVjQGzjAkU4dJ+RPmiDlYuHLZS41Pg6VMwY+03lhk6I5A/w4rnqdkmwZX/NgXg06aln
    c2pBsXWhL4O7nk0S2ZrLMsQZ6HcsXgdmHo=
                        </ds:X509Certificate>
                    </ds:X509Data>
                </pskc:EncryptionKey>
                <pskc:KeyPackage>
                    <pskc:DeviceInfo>
                        <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
                        <pskc:SerialNo>987654321</pskc:SerialNo>
```

```
                    <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
                    <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
                </pskc:DeviceInfo>
                <pskc:Key
                    Id="MBK000000001"
                    Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
                    <pskc:Issuer>Example-Issuer</pskc:Issuer>
                    <pskc:AlgorithmParameters>
                        <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
                    </pskc:AlgorithmParameters>
                    <pskc:Data>
                        <pskc:Secret>
                            <pskc:EncryptedValue>
                                <xenc:EncryptionMethod
                                 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa_1_5"/>
                                <xenc:CipherData>
                                    <xenc:CipherValue>
eyjr23WMy9S2UdKgGnQEbs44T1jmX1TNWEBq48xfS20PK2VWF4ZK1iSctHj/u3uk+7+y8
uKrAzHEm5mujKPAU4DCbb5mSibXMnAbbIoAi2cJW60/18FlzwaU4EZsZ1LyQ1GcBQKACE
eylG5vK8NTo47vZTatL5UxmbmOX2HvaVQ=
                                    </xenc:CipherValue>
                                </xenc:CipherData>
                            </pskc:EncryptedValue>
                        </pskc:Secret>
                        <pskc:Counter>
                            <pskc:PlainValue>0</pskc:PlainValue>
                        </pskc:Counter>
                    </pskc:Data>
                    <pskc:Policy>
                        <pskc:KeyUsage>OTP</pskc:KeyUsage>
                    </pskc:Policy>
                </pskc:Key>
            </pskc:KeyPackage>
        </dskpp:KeyContainer>
    </dskpp:KeyPackage>
    <dskpp:Mac
        MacAlgorithm="urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
        GHZ0H6Y+KpxdlVZ7zgcJDiDdqc8Gcmlcf+HQi4EUxYU=
    </dskpp:Mac>
</dskpp:KeyProvServerFinished>
```

B.3.2. Пример использования метода упаковки ключа

Клиент передаёт запрос, который указывает общий ключ для защиты K_TOKEN, а сервер отвечает с использованием метода защиты ключей Key Wrap. Authentication Data в этом примере базируются на Authentication Code, а не на сертификате устройства.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvClientHello
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0">
    <dskpp:DeviceIdentifierData>
        <dskpp:DeviceId>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
            <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
            <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
        </dskpp:DeviceId>
    </dskpp:DeviceIdentifierData>
    <dskpp:SupportedKeyTypes>
        <dskpp:Algorithm>urn:ietf:params:xml:ns:keyprov:pskc:hotp</dskpp:Algorithm>
        <dskpp:Algorithm>
 http://www.rsa.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES
        </dskpp:Algorithm>
    </dskpp:SupportedKeyTypes>
    <dskpp:SupportedEncryptionAlgorithms>
        <dskpp:Algorithm>http://www.w3.org/2001/04/xmlenc#aes128-cbc</dskpp:Algorithm>
    </dskpp:SupportedEncryptionAlgorithms>
    <dskpp:SupportedMacAlgorithms>
        <dskpp:Algorithm>urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256</dskpp:Algorithm>
    </dskpp:SupportedMacAlgorithms>
    <dskpp:SupportedProtocolVariants>
        <dskpp:TwoPass>
            <dskpp:SupportedKeyProtectionMethod>
                urn:ietf:params:xml:schema:keyprov:dskpp:wrap
            </dskpp:SupportedKeyProtectionMethod>
            <dskpp:Payload>
                <ds:KeyInfo>
                    <ds:KeyName>Pre-shared-key-1</ds:KeyName>
                </ds:KeyInfo>
            </dskpp:Payload>
        </dskpp:TwoPass>
    </dskpp:SupportedProtocolVariants>
    <dskpp:SupportedKeyPackages>
```

```
            <dskpp:KeyPackageFormat>
                urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
            </dskpp:KeyPackageFormat>
        </dskpp:SupportedKeyPackages>
        <dskpp:AuthenticationData>
            <dskpp:ClientID>AC00000A</dskpp:ClientID>
            <dskpp:AuthenticationCodeMac>
                <dskpp:Nonce>
                    ESIzRFVmd4iZqrvM3e7/ESIzRFVmd4iZqrvM3e7/ESI=
                </dskpp:Nonce>
                <dskpp:IterationCount>1</dskpp:IterationCount>
                <dskpp:Mac
                    MacAlgorithm="urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
                    3eRz51ILqiG+dJW2iLcjuA==
                </dskpp:Mac>
            </dskpp:AuthenticationCodeMac>
        </dskpp:AuthenticationData>
    </dskpp:KeyProvClientHello>
```

В этом примере сервер отвечает на предыдущий запрос возвратом пакета ключа, в котором ключ обеспечения зашифрован с использованием метода защиты Key Wrap.

```
    <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <dskpp:KeyProvServerFinished
        xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
        xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
        xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:dkey="http://www.w3.org/2009/xmlsec-derivedkey#"
        xmlns:pkcs5="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"
        Version="1.0"
        Status="Success"
        SessionID="4114">
        <dskpp:KeyPackage>
             <dskpp:KeyContainer Version="1.0" Id="KC0001">
                 <pskc:EncryptionKey>
                    <ds:KeyName>Pre-shared-key-1</ds:KeyName>
                 </pskc:EncryptionKey>
                 <pskc:MACMethod
                     Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
                     <pskc:MACKey>
                         <xenc:EncryptionMethod
                             Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                         <xenc:CipherData>
                             <xenc:CipherValue>
      2GTTnLwM3I4e5IO5FkufoMUBJBuAf25hARFv0Z7MFk9Ecdb04PWY/qaeCbrgz7Es
                             </xenc:CipherValue>
                         </xenc:CipherData>
                     </pskc:MACKey>
                 </pskc:MACMethod>
                 <pskc:KeyPackage>
                     <pskc:DeviceInfo>
                         <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
                         <pskc:SerialNo>987654321</pskc:SerialNo>
                         <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
                         <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
                     </pskc:DeviceInfo>
                     <pskc:CryptoModuleInfo>
                         <pskc:Id>CM_ID_001</pskc:Id>
                     </pskc:CryptoModuleInfo>
                     <pskc:Key
                         Id="MBK000000001"
                         Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
                         <pskc:Issuer>Example-Issuer</pskc:Issuer>
                         <pskc:AlgorithmParameters>
                           <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
                         </pskc:AlgorithmParameters>
                         <pskc:Data>
                             <pskc:Secret>
                                 <pskc:EncryptedValue>
                                   <xenc:EncryptionMethod
                                   Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                                     <xenc:CipherData>
                                         <xenc:CipherValue>
                                             oTvo+S22nsmS2Z/RtcoF8AabC6vr
                                             09sh0QIU+E224S96sZjpV+6nFYgn
                                             6525OoepbPnL/fGuuey64WCYXoqh
                                             Tg==
                                         </xenc:CipherValue>
                                     </xenc:CipherData>
                                 </pskc:EncryptedValue>
                                 <pskc:ValueMAC>o+e9xgMVUbYuZH9UHe0W9dIo88A=</pskc:ValueMAC>
                             </pskc:Secret>
                             <pskc:Counter><pskc:PlainValue>0</pskc:PlainValue></pskc:Counter>
                         </pskc:Data>
                         <pskc:Policy><pskc:KeyUsage>OTP</pskc:KeyUsage></pskc:Policy>
                     </pskc:Key>
```

```
            </pskc:KeyPackage>
        </dskpp:KeyContainer>
    </dskpp:KeyPackage>
    <dskpp:Mac
        MacAlgorithm="urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
        l53BmSO6qUzoIgbQegimsKk2es+WRpEl0YFqaOp5PGE=
    </dskpp:Mac>
</dskpp:KeyProvServerFinished>
```

B.3.3. Пример использования упаковки ключа на базе пароля

Клиент передаёт запрос, аналогичный описанному в приложении B.3.1 с Authentication Data на основе Authentication Code, а сервер отвечает с использованием метода Passphrase-Based Key Wrap для шифрования ключа обеспечения (отметим, что шифрование выводится из парольной компоненты Authentication Code). Authentication Data передаются в открытом виде при использовании защищённого канала, такого как TLS [RFC5246].

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvClientHello
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Version="1.0">
    <dskpp:DeviceIdentifierData>
        <dskpp:DeviceId>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
            <pskc:StartDate>2009-09-01T00:00:00Z</pskc:StartDate>
            <pskc:ExpiryDate>2014-09-01T00:00:00Z</pskc:ExpiryDate>
        </dskpp:DeviceId>
    </dskpp:DeviceIdentifierData>
    <dskpp:SupportedKeyTypes>
        <dskpp:Algorithm>
            urn:ietf:params:xml:ns:keyprov:pskc:hotp
        </dskpp:Algorithm>
        <dskpp:Algorithm>
 http://www.rsa.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES
        </dskpp:Algorithm>
    </dskpp:SupportedKeyTypes>
    <dskpp:SupportedEncryptionAlgorithms>
        <dskpp:Algorithm>
            http://www.w3.org/2001/04/xmlenc#rsa_1_5
        </dskpp:Algorithm>
    </dskpp:SupportedEncryptionAlgorithms>
    <dskpp:SupportedMacAlgorithms>
        <dskpp:Algorithm>
            urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256
        </dskpp:Algorithm>
    </dskpp:SupportedMacAlgorithms>
    <dskpp:SupportedProtocolVariants>
        <dskpp:TwoPass>
            <dskpp:SupportedKeyProtectionMethod>
             urn:ietf:params:xml:schema:keyprov:dskpp:passphrase-wrap
            </dskpp:SupportedKeyProtectionMethod>
            <dskpp:Payload>
                <ds:KeyInfo>
                    <ds:KeyName>Passphrase-1</ds:KeyName>
                </ds:KeyInfo>
            </dskpp:Payload>
        </dskpp:TwoPass>
    </dskpp:SupportedProtocolVariants>
    <dskpp:SupportedKeyPackages>
        <dskpp:KeyPackageFormat>
            urn:ietf:params:xml:ns:keyprov:dskpp:pskc-key-container
        </dskpp:KeyPackageFormat>
    </dskpp:SupportedKeyPackages>
    <dskpp:AuthenticationData>
        <dskpp:ClientID>AC00000A</dskpp:ClientID>
        <dskpp:AuthenticationCodeMac>
            <dskpp:Nonce>
                ESIzRFVmd4iZqrvM3e7/ESIzRFVmd4iZqrvM3e7/ESI=
            </dskpp:Nonce>
            <dskpp:IterationCount>1</dskpp:IterationCount>
            <dskpp:Mac
                MacAlgorithm=
               "urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
               K4YvLMN6Q1DZvtShoCxQag==
            </dskpp:Mac>
        </dskpp:AuthenticationCodeMac>
    </dskpp:AuthenticationData>
</dskpp:KeyProvClientHello>
```

В этом примере сервер отвечает на предыдущий запрос, возвращая пакет ключа в котором ключ обеспечения зашифрован с использованием метода защиты Passphrase-Based Key Wrap.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dskpp:KeyProvServerFinished
    xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
```

```
      xmlns:dskpp="urn:ietf:params:xml:ns:keyprov:dskpp"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:dkey="http://www.w3.org/2009/xmlsec-derivedkey#"
      xmlns:pkcs5=
         "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"
      Version="1.0"
      Status="Success"
      SessionID="4114">
      <dskpp:KeyPackage>
          <dskpp:KeyContainer Version="1.0" Id="KC0002">
              <pskc:EncryptionKey>
                  <dkey:DerivedKey>
                      <dkey:KeyDerivationMethod
                      Algorithm=
                      "http://www.rsasecurity.com/rsalabs/pkcs/schemas/
                          pkcs-5v2-0#pbkdf2">
                          <pkcs5:PBKDF2-params>
                              <Salt>
                                  <Specified>Ej7/PEpyEpw=</Specified>
                              </Salt>
                              <IterationCount>1000</IterationCount>
                              <KeyLength>16</KeyLength>
                          </pkcs5:PBKDF2-params>
                      </dkey:KeyDerivationMethod>
                      <xenc:ReferenceList>
                          <xenc:DataReference URI="#ED"/>
                      </xenc:ReferenceList>
                      <dkey:MasterKeyName>
                         Passphrase1
                      </dkey:MasterKeyName>
                  </dkey:DerivedKey>
              </pskc:EncryptionKey>
              <pskc:MACMethod
                  Algorithm=
                     "http://www.w3.org/2000/09/xmldsig#hmac-sha1">
                  <pskc:MACKey>
                      <xenc:EncryptionMethod
                          Algorithm=
                        "http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                      <xenc:CipherData>
                          <xenc:CipherValue>
       2GTTnLwM3I4e5IO5FkufoOEiOhNj91fhKRQBtBJYluUDsPOLTfUvoU2dStyOwYZx
                          </xenc:CipherValue>
                      </xenc:CipherData>
                  </pskc:MACKey>
              </pskc:MACMethod>
              <pskc:KeyPackage>
                  <pskc:DeviceInfo>
                      <pskc:Manufacturer>
                         TokenVendorAcme
                      </pskc:Manufacturer>
                      <pskc:SerialNo>
                         987654321
                      </pskc:SerialNo>
                      <pskc:StartDate>
                         2009-09-01T00:00:00Z
                      </pskc:StartDate>
                      <pskc:ExpiryDate>
                         2014-09-01T00:00:00Z
                      </pskc:ExpiryDate>
                  </pskc:DeviceInfo>
                  <pskc:CryptoModuleInfo>
                      <pskc:Id>CM_ID_001</pskc:Id>
                  </pskc:CryptoModuleInfo>
                  <pskc:Key
                      Id="MBK000000001"
                      Algorithm=
                         "urn:ietf:params:xml:ns:keyprov:pskc:hotp">
                      <pskc:Issuer>Example-Issuer</pskc:Issuer>
                      <pskc:AlgorithmParameters>
                         <pskc:ResponseFormat Length="6"
                           Encoding="DECIMAL"/>
                      </pskc:AlgorithmParameters>
                      <pskc:Data>
                          <pskc:Secret>
                              <pskc:EncryptedValue>
                                  <xenc:EncryptionMethod
                                      Algorithm=
                                      "http://www.w3.org/2001/04/
                                      xmlenc#aes128-cbc"/>
                                  <xenc:CipherData>
                                      <xenc:CipherValue>
                                        oTvo+S22nsmS2Z/RtcoF8HX385uMWgJ
                                        myIFMESBmcvtHQXp/6T1TgCS9CsgKtm
                                        cOrF8VoK254tZKnrAjiD5cdw==
```

```
                                    </xenc:CipherValue>
                                </xenc:CipherData>
                            </pskc:EncryptedValue>
                            <pskc:ValueMAC>
                                pbgEbVYxoYs0x41wdeC7eDRbUEk=
                            </pskc:ValueMAC>
                        </pskc:Secret>
                        <pskc:Counter>
                            <pskc:PlainValue>0</pskc:PlainValue>
                        </pskc:Counter>
                    </pskc:Data>
                    <pskc:Policy>
                        <pskc:KeyUsage>OTP</pskc:KeyUsage>
                    </pskc:Policy>
                </pskc:Key>
            </pskc:KeyPackage>
        </dskpp:KeyContainer>
    </dskpp:KeyPackage>
    <dskpp:Mac MacAlgorithm=
        "urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256">
        Jc4VsNODYXgfbDmTn9qQZgcL3cKoa//j/NRT7sTpKOM=
    </dskpp:Mac>
</dskpp:KeyProvServerFinished>
```

Приложение C. Интеграция с PKCS #11

Клиент DSKPP, которому нужно взаимодействовать с подключённым криптомодулем для обмена DSKPP, **может** использовать PKCS #11 [PKCS-11] в качестве программного интерфейса, как описано здесь. Это приложение является информационной частью документа.

C.1. 4-проходный вариант

При 4-проходном обмене DSKPP с криптомодулем при использовании программного интерфейса PKCS #11 **рекомендуется** процедура, описанная в приложении B к [CT-KIP-P11].

C.2. 2-проходный вариант

Предлагаемая процедура для 2-проходного обмена DSKPP с криптомодулем при использовании программного интерфейса PKCS #11, определённого в [CT-KIP-P11], описана ниже.

a. На стороне клиента.

   1. Клиент выбирает подходящий слот и маркер (например, используя элемент <DeviceIdentifier> или <PlatformInfo> в триггерном сообщении DSKPP).
   2. Генерируется nonce R, например, путём вызова C_SeedRandom и C_GenerateRandom.
   3. Клиент передаёт серверу первое сообщение, включающее nonce R.

b. На стороне сервера.

   1. Генерируется базовый ключ K_PROV = K_TOKEN | K_MAC (| указывает конкатенацию), например, путём вызова C_GenerateKey (с использованием типа ключа CKK_GENERIC_SECRET). Шаблон для K_PROV **должен** разрешать экспорт (но только в упакованном виде, т. е. для CKA_SENSITIVE **должно** быть установлено значение CK_TRUE, а для CKA_EXTRACTABLE **должно** устанавливаться значение CK_TRUE), а также использование для последующего вывода ключей. Из K выводится ключ маркера K_TOKEN подходящего типа путём вызова C_DeriveKey с использованием механизма PKCS #11 CKM_EXTRACT_KEY_FROM_KEY и установкой CK_EXTRACT_PARAMS в качестве первого бита базового секретного ключа (т. е. 0). Аналогично, ключ MAC K_MAC выводится из K_PROV путём вызова C_DeriveKey с использованием механизма CKM_EXTRACT_KEY_FROM_KEY, а в качестве CK_EXTRACT_PARAMS устанавливается размер K_PROV (в битах), поделённый на 2.
   2. Сервер пакует (wrap) K_PROV с использованием открытого ключа клиента DSKPP или устройства, заранее известный секретный ключ или общий секрет, выведенный с использованием C_WrapKey. Если использование алгоритма упаковки ключей DSKPP согласовано, **должен** применяться механизм CKM_KIP_WRAP для упаковки K. При вызове C_WrapKey для дескриптора hKey в структуре CK_KIP_PARAMS **должно** устанавливаться значение NULL_PTR. Параметр pSeed в структуре CK_KIP_PARAMS **должен** указывать nonce R, предоставленное клиентом DSKPP, а параметр ulSeedLen **должен** указывать размер R. Параметр hWrappingKey при вызове C_WrapKey **должен** указывать ключ упаковки ключа.
   3. Затем сервер должен рассчитать MAC, используя K_MAC. Если применяемый алгоритм DSKPP MAC был согласован, MAC вычисляется путём вызова C_SignInit с механизмом CKM_KIP_MAC и последующим вызовом C_Sign. При вызове C_SignInit значением K_MAC **должен** быть ключ подписи, для параметра hKey в структуре CK_KIP_PARAMS **должно** быть установлено значение NULL_PTR, для pSeed в структуре CT_KIP_PARAMS - NULL_PTR, а параметр ulSeedLen **должен** иметь значение 0. При вызове C_Sign параметр pData **должен** быть конкатенацией строк ServerID и nonce R, а в качестве ulDataLen **должен** быть указан размер строки конкатенации. Желаемый размер MAC **должен** быть указан параметром pulSignatureLen и **должен** совпадать с размером R.
   4. Если серверу нужно также аутентифицировать своё сообщение (по причине замены K_TOKEN), он **должен** рассчитать второе значение MAC. Если согласовано использование алгоритма DSKPP MAC, значение MAC рассчитывается путём вызова C_SignInit с механизмом CKM_KIP_MAC и последующего вызова C_Sign. В этом вызове C_SignInit значение K_MAC', имевшееся до этого запуска DSKPP, **должно** быть ключом подписи (реализация может задать в качестве K_MAC' значение K_TOKEN, которое будет заменено, или K_MAC из предыдущего запуска протокола), для параметра hKey в структуре CK_KIP_PARAMS **должно** быть

установлено значение NULL, для pSeed в структуре CT_KIP_PARAMS **должно** быть установлено значение NULL_PTR, а для параметра ulSeedLen **должно** быть установлено значение 0. При вызове C_Sign в качестве значения параметра pData **должна** применяться конкатенация строк ServerID и nonce R, а значением параметра ulDataLen **должен** быть размер объединённой строки. Желательный размер MAC **должен** быть задан параметром pulSignatureLen и **должен** совпадать с размером R.

5. Сервер передаёт сообщение клиенту, включая упакованный ключ K_TOKEN, MAC и возможно MAC для аутентификации.

c. На стороне клиента.

1. Клиент вызывает C_UnwrapKey для получения дескриптора K. Затем клиент дважды вызывает C_DeriveKey - один раз для вывода K_TOKEN, другой для вывода K_MAC. Клиент **должен** применять тот же механизм (CKM_EXTRACT_KEY_FROM_KEY) и такие же параметры механизма, которые использовались сервером (см. выше). При вызове C_UnwrapKey и C_DeriveKey **должен** использоваться параметр pTemplate для установки дополнительных атрибутов ключа в соответствии с локальной политикой, как они были согласованы и выражены в протоколе. В частности, значение элемента <KeyID> из отклика сервера **может** применяться в качестве CKA_ID для K_TOKEN. Ключ K_PROV **должен** быть уничтожен после вывода K_TOKEN и K_MAC.

2. Значение MAC проверяется обращением процедуры его генерации на сервере. Если было согласовано применение механизма CKM_KIP_MAC, при вызове C_VerifyInit для параметра hKey в структуре CK_KIP_PARAMS **должно** быть установлено значение NULL_PTR, для pSeed - NULL_PTR, а для ulSeedLen - 0. Параметр hKey в C_VerifyInit **должен** указывать K_MAC. При вызове C_Verify в качестве значения pData **должна** использоваться конкатенация строк ServerID и nonce R, параметр ulDataLen **должен** указывать размер объединённой строки, pSignature - значение MAC, полученное от сервера, а ulSignatureLen - размер MAC. Если проверка MAC не проходит, протокольная сессия завершается отказом. Маркер **должен** создаваться так, чтобы новые значения K_TOKEN и K_MAC не представлялись до проверки MAC.

3. При получении MAC для аутентификации (**требуется**, если новый K_TOKEN будет заменять имеющийся ключ в маркере) код проверяется аналогичным способом, но с использованием K_MAC', связанного с этим сервером и имевшегося до запуска протокола (реализация может указать для K_MAC' значение заменяемого K_TOKEN или K_MAC из предшествующего запуска протокола). Если проверка MAC не проходит, протокольная сессия завершается отказом. Маркер **должен** создаваться так, чтобы новые значения K_TOKEN и K_MAC не представлялись до проверки MAC.

Приложение D. Примеры реализации DSKPP-PRF

D.1. Введение

В этом приложении определяется DSKPP-PRF в терминах AES [FIPS197-AES] и HMAC [RFC2104]. Приложение является информационной частью документа.

D.2. DSKPP-PRF-AES

D.2.1. Идентификация

Для криптомодулей, поддерживающих эту реализацию DSKPP-PRF, **должен** применяться приведённый ниже идентификатор URN, указывающий этот алгоритм в DSKPP.

```
urn:ietf:params:xml:ns:keyprov:dskpp:prf-aes-128
```

При использовании этого URN для указания алгоритма шифрования, **должен** применяться метод шифрования значений R_C, описанный в параграфе 4.2.4.

D.2.2. Определение

```
DSKPP-PRF-AES (k, s, dsLen)
```

Вход:

k используемый ключ шифрования;

s строка октетов, содержащая случайные данные размером sLen;

dsLen желаемый размер результата.

Выход:

DS псевдослучайная строка размером dsLen октетов.

Этапы:

1. Пусть bLen указывает размер выходного блока AES в октетах

   ```
   bLen = (размер выходного блока AES в октетах)
   ```
   (обычно bLen = 16)

2. Если dsLen > (2^{32} - 1) * bLen, выводится сообщение «derived data too long[1]» и работа прекращается.

3. Пусть n - число блоков размером bLen в выходных данных, округлённое в большую сторону, а j - число октетов в последнем блоке

   ```
   n = CEILING( dsLen / bLen)
   j = dsLen - (n - 1) * bLen
   ```

4. Для каждого блока псевдослучайной строки DS применяется определённая ниже функция F с ключом k, строкой s и индексом рассчитываемого блока

   ```
   B1 = F (k, s, 1) ,
   ```

[1]Слишком много данных на выходе.

```
B2 = F (k, s, 2) ,
...
Bn = F (k, s, n)
```

Функция F определяется в терминах конструкции CMAC из [NIST-SP800-38B] с использованием блочного шифра AES

```
F (k, s, i) = CMAC-AES (k, INT (i) || s)
```

где INT (i) - 4-октетное представление целого числа i (сначала старший октет), а выходной размер CMAC равен bLen.

Блоки сливаются и выбираются первые dsLen октетов для создания желаемой строки данных DS

```
DS = B1 || B2 || ... || Bn<0..j-1>
```

Выводится строка данных DS.

D.2.3. Пример

Для dsLen = 16 будем иметь

```
n = 16 / 16 = 1
j = 16 - (1 - 1) * 16 = 16
DS = B1 = F (k, s, 1) = CMAC-AES (k, INT (1) || s)
```

D.3. DSKPP-PRF-SHA256

D.3.1. Идентификация

Для криптомодулей, поддерживающих эту реализацию DSKPP-PRF, **должен** применяться приведённый ниже идентификатор URN, указывающий этот алгоритм в DSKPP.

```
urn:ietf:params:xml:ns:keyprov:dskpp:prf-sha256
```

При использовании этого URN для указания алгоритма шифрования, **должен** применяться метод шифрования значений R_C, описанный в параграфе 4.2.4.

D.3.2. Определение

```
DSKPP-PRF-SHA256 (k, s, dsLen)
```

Вход:

k — используемый ключ шифрования;

s — строка октетов, содержащая случайные данные размером sLen;

dsLen — желаемый размер результата.

Выход:

DS — псевдослучайная строка размером dsLen октетов.

Этапы:

1. Пусть bLen указывает выходной размер SHA-256 в октетах [FIPS180-SHA] (без отсечки вывода HMAC)

   ```
   bLen = 32
   (normally, bLen = 16)
   ```

2. Если dsLen > (2^{32} - 1) * bLen, выводится сообщение «derived data too long[1]» и работа прекращается.

3. Пусть n - число блоков размером bLen в выходных данных, округлённое в большую сторону, а j - число октетов в последнем блоке

   ```
   n = CEILING( dsLen / bLen)
   j = dsLen - (n - 1) * bLen
   ```

4. Для каждого блока псевдослучайной строки DS применяется определённая ниже функция F с ключом k, строкой s и индексом рассчитываемого блока

```
B1 = F (k, s, 1),
B2 = F (k, s, 2),
...
Bn = F (k, s, n)
```

Функция F определяется в терминах конструкции HMAC из [RFC2104] с использованием алгоритма подписи SHA-256

```
F (k, s, i) = HMAC-SHA256 (k, INT (i) || s)
```

где INT (i) - 4-октетное представление целого числа i (сначала старший октет), а выходной размер HMAC равен bLen.

Блоки сливаются и выбираются первые dsLen октетов для создания желаемой строки данных DS

```
DS = B1 || B2 || ... || Bn<0..j-1>
```

Выводится строка данных DS.

D.3.3. Пример

Для sLen = 256 (два 128-октетных значения) и dsLen = 16 бедуем иметь

```
n = CEILING( 16 / 32 ) = 1
j = 16 - (1 - 1) * 32 = 16
B1 = F (k, s, 1) = HMAC-SHA256 (k, INT (1) || s)
DS = B1<0 ... 15>
```

Т. е. результатом будут первые 16 октетов вывода HMAC.

Адреса авторов

Andrea Doherty

RSA, The Security Division of EMC
174 Middlesex Turnpike
Bedford, MA 01730
USA
EMail: andrea.doherty@rsa.com

Mingliang Pei
VeriSign, Inc.
487 E. Middlefield Road
Mountain View, CA 94043
USA
EMail: mpei@verisign.com

Salah Machani
Diversinet Corp.
2225 Sheppard Avenue East, Suite 1801
Toronto, Ontario M2J 5C2
Canada
EMail: smachani@diversinet.com

Magnus Nystrom
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052
USA
EMail: mnystrom@microsoft.com

Перевод на русский язык

Николай Малых
nmalykh@protokols.ru