БЕЗОПАСНАЯ ОБРАБОТКА ИНФОРМАЦИОННЫХ ЗАПРОСОВ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ УПРАВЛЕНИЯ НА ОСНОВЕ СИНТАКСИСА КРИПТОГРАФИЧЕСКИХ СООБЩЕНИЙ (CMS)

Асратян Р.Э.
Институт проблем управления им. В.А. Трапезникова РАН
rea@ipu.ru

Аннотация: Рассмотрены методы реализации сетевой Службы защищенных сообщений, предназначенной для безопасной обработки информационных запросов в распределенных информационных системах. Отличительными особенностями службы являются тесная интеграция функций информационной защиты данных с функциями информационного взаимодействия в сети. Описаны особенности архитектурного построения службы на основе средств поддержки Синтаксиса защищенных сообщений (CMS) в Windows.

Ключевые слова: распределенные системы, Web-сервисы, Интернет-технологии, информационный обмен, защита данных, информационная безопасность.

Введение

Появление сетевой архитектуры .Net и технологии Web-сервисов [1] стало важным событием для разработчиков распределенных систем, но не устранило всех трудностей обеспечения информационной защиты в сети. Эти трудности чаще всего бывают связаны с отсутствием в архитектуре .NET встроенных средств защиты и аутентификации сетевых сообщений и более всего проявляются в разработках систем, предназначенных для работы в сложных, мульти-серверных и мульти-сетевых средах в условиях высоких требований к информационной безопасности [2,3].

В работе [4] описана новая сетевая служба PMS (Protected Message Service), разработанная с целью преодоления вышеуказанного недостатка. Суть подхода заключается в тесной интеграции функций сетевого информационного обмена с функциями защиты и аутентификации данных. Внешне эта интеграция проявляется в том, что отмеченные функции входят в набор методов главного программного класса службы – класса «Защищенное сообщение» (PmsMessage), отображающего электронный документ (информационный запрос или ответ), снабженный одной или несколькими удостоверяющими электронными цифровыми подписями (ЭЦП). В отличие, например, от технологии Web-сервисов, описываемая служба опирается не на модель вызова методов удаленных объектов, а на модель обмена сообщениями. В данном случае это означает, что все сервисные обрабатывающие функции (методы) имеют одинаковую, жесткую спецификацию: они получают объект класса «Защищенное сообщение» в качестве параметра и возвращают объект того же класса. Эти обрабатывающие функции группируются в одну или несколько динамических библиотек, которые подключаются к серверу PMS в момент его запуска (каждая библиотека может рассматриваться как отдаленный аналог Web-сервиса в .NET), и становятся доступными для клиентских компонент.

Реализация PMS на основе криптосистемы «КриптоПро» версии 3.6 и проведенные лабораторные эксперименты показали достаточно высокое быстродействие новой службы, не уступающее, а в отдельных случаях превосходящее быстродействие Web-сервисов в одинаковых условиях. Однако, при данном подходе возникает жесткая «привязанность» PMS к определенной криптосистеме, что может создать неудобства для разработчиков распределенных систем.

В данной работе рассматривается новый подход к архитектурному построению PMS, основанный на применении стандарта Cryptographic Message Syntax (CMS) и его программной поддержки в среде Windows в качестве базисного средства реализации. Главное преимущество этого подхода заключается в том, что он позволяет PMS «унаследовать» способность гибкой настройки на использование любой криптосистемы, поддерживающей стандарт CMS, и, тем самым, устранить ту жесткую привязку к определенной криптосистеме, о которой говорилось выше.

1 Краткие сведения о PMS

Как и всякая сетевая служба, основанная на базовом сетевом протоколе TCP/IP [5], PMS поддержана клиентским и серверным программным обеспечением. Сервер PMS представляет

собой постоянно активную программу, обслуживающую запросы на обработку от клиентов (по умолчанию используется порт 8132). Клиентское программное обеспечение представляет собой библиотеку функций PmsBase.dll, реализующих прикладной программный интерфейс (API) к PMS. Этот интерфейс является «лицом» PMS с точки зрения пользователя.

На рис. 1 представлен фрагмент кода в нотации C#, иллюстрирующий простое обращение к обрабатывающей функции с применением средств защиты данных. Две первые строки кода определяют две переменные типа PmsMessage, представляющего собой главный класс PMS («Защищенное сообщение»). Первой переменной (Request) присваивается значение: объект класса PmsMessage, инициированный символьной строкой (например, содержащей XML-документ информационного запроса). Вторая переменная (Reply) предназначена для хранения результата обработки. В третьей строке определяется и инициируется переменная класса PmsConnection, предназначенного для создания и прекращения сетевого соединения с сервером.

```
PmsMessage Request=  new PmsMessage("<request> … </request>");
PmsMessage Reply;
PmsConnection MyConn = new PmsConnection();
PmsCertList  SenderCerts = PmsCertList( new string [] {"Иванов", "Петров"});
Request.AddSignatures(SenderCerts);

MyConn.Connect("MyServer");
PmsCertList  ReceiverCert = MyConn.GetServerCertificate();

Reply=Request.Process (MyConn, "MyLib.MyFunc  param", ReceiverCert);
if(Reply != null)
{
     string Signer;
     for(int i=0; Reply.Signatures.Length; i++)
        if(Reply.VerifySignature(i, out Signer) >= 0)
              Console.WriteLine ("Ответ подписал: " + Signer);
     Console.WriteLine (Reply.GetString());
}
else
     Console.WriteLine ("Ошибка: " +MyConn.ErrMsg);
MyConn.Disconnect();
```

Рис. 1. Пример использования PMS

В четвертой и пятой строках кода выполняется формирование подписей в запросе. Сначала определяется переменная SenderCerts класса PmsCertList. Этот класс предназначен для хранения в памяти списков сертификатов с открытыми ключами в стандарте X509 и содержит несколько конструкторов для загрузки сертификатов из файлов или из системных хранилищ с поиском по имени владельца или по серийному номеру. В переменную SenderCerts загружаются два сертификата, соответствующих именам владельцев «Иванов» и «Петров» для последующего формирования двух ЭЦП в запросе. (Такой способ использования означает, что с этими сертификатами обязательно должен быть связаны парные им закрытые ключи, иначе формирование ЭЦП закончится неудачно.) Вызов метода AddSignatures позволяет сформировать две ЭЦП в сообщении Request.

Собственно вызов обрабатывающей функции начинается с вызова метода Connect, устанавливающего сетевое соединение с сервером с указанным сетевым именем или адресом. Далее выполняется первая сетевая операция – запрос сертификата сервера (метод GetServerCertificate) с занесением результата в переменную ReceiverCert уже знакомого нам класса PmsCertList для последующего шифрования информационного запроса. Сразу же после успешного получения сертификата сервера выполняется вызов удаленной функции с помощью применения метода Process к переменной Request с использованием все того же соединения и полученного сертификата. Предполагается, что к серверу PMS подключена библиотека MyLib.dll, содержащая код функции MyFunc. При запуске функции на сервере ей передаются значение переменной Request и опциональный строковый параметр param в качестве фактических параметров. Результат обработки

заносится в переменную Reply. Подчеркнем, что шифрование запроса и дешифрование ответа выполняются автоматически в методе Process.

Последующие строки обеспечивают последовательную проверку всех ЭЦП в полученном ответе сервера (вызов метода VeifySignature) и запись в стандартный вывод сведений о подписантах.

Пример заканчивается записью результата обращения в стандартный вывод (предполагается, что результат, как и запрос, имеет форму символьной строки) и закрытием соединения с сервером с помощью метода Disconnect, так как в данном примере оно больше не нужно.

Необходимо отметить, что из фрагмента кода намеренно удалены операторы обработки исключений.

2. Методы реализации PMS

На рис. 2 проиллюстрированы два архитектурных подхода к реализации PMS:

- на основе прямого подключения определенной криптосистемы к программным модулям PMS без использования CMS (архитектура «PMS-Криптосистема»),
- на основе CMS (архитектура «PMS-CMS-Криптосистема») с возможностью использования различных криптосистем.

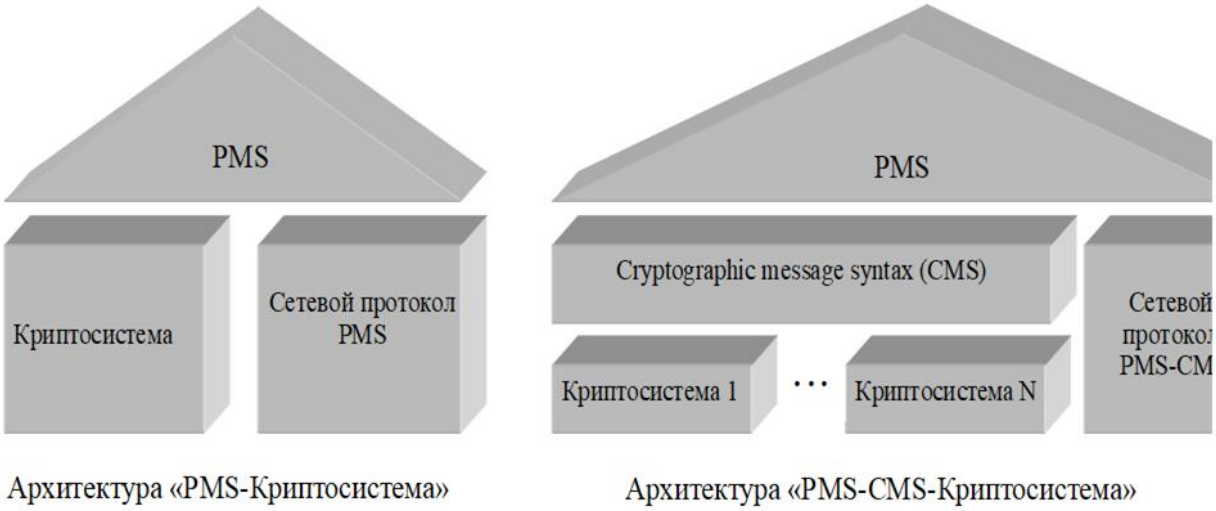

Рис. 2. Архитектурные решения для реализации PMS

Первый подход основан на прямом вызове функций определенной криптосистемы из кода программных модулей PMS для выполнения операций формирования ЭЦП, шифрования и т.п. Очевидно, что этот подход обеспечивает наименьше «накладные расходы», но реализация PMS оказывается жестко привязана к выбранной криптосистеме.

Второй подход основано на использовании средств поддержки CMS в качестве промежуточной прослойки между программными модулями PMS и используемой криптосистемой. Главное преимущество - возможность гибкой настройки на применение любой криптосистемы, поддерживающей стандарт CMS без изменения кода программных модулей службы, которые используют универсальный программный интерфейс CMS для выполнения крипто-функций.

Как видно из рис. 2, важной составной частью каждого из двух архитектурный решений является используемый сетевой протокол. Независимо от выбранного решения, этот протокол организуется по следующим общим принципам.

- PMS в полной мере использует двоичную природу TCP/IP [5]. Взаимодействие между клиентом и сервером PMS осуществляется по специальному, достаточно простому протоколу, ориентированному на передачу двоичных сетевых сообщений (PMS-сообщений) в обоих направлениях (никакие преобразования двоичных данных в текстовую форму типа base64 не применяются). Каждое такое сообщение в общем случае содержит два массива байтов: заголовок сообщения и тело сообщения (см. рис. 3). Первые 4 байта заголовка или тела сообщения содержат целое число – его длину.
- При передаче запроса от клиента к серверу в заголовок сетевого сообщения помещается строка, содержащая полное имя вызываемой функции, а в тело сообщения упаковывается структура PmsMessage в открытой или зашифрованной форме, содержащая

информационный запрос. Строка заголовка используется сервером для организации вызова соответствующей обрабатывающей функции.

- При передаче результата обработки от сервера к клиенту в заголовок сетевого сообщения помещается строка диагностического сообщения (значение параметра Msg, сформированное обрабатывающей функцией), а в тело сообщения упаковывается структура PmsMessage, содержащая ответ сервера в открытой или зашифрованной форме, предварительно подписанный собственным закрытым ключом сервера. Никакие двоично-текстовые преобразования (типа base64) не применяются. Полученное от сервера диагностическое сообщение автоматически присваивается члену ErrMsg объекта класса PmsSrvLibraries на стороне клиента (см. рис. 1).

Тем не менее, в архитектуре «PMS-Криптосистема» детали реализации сетевого протокола PMS могут различаться в зависимости от выбранной криптосистемы, т.к. от последней зависит состав и структура крипто-данных, включенных в подписанный и/или зашифрованный объект PmsMessage в теле сетевого сообщения. Важнейшее преимущество архитектуры «PMS-CMS-Криптосистема» заключается в том, что в этом случае реализация сетевого протокола (протокол «PMS-CMS») не зависит от применяемых криптосистем и целиком основывается на универсальном стандарте представления защищенных данных в CMS (см. RFC 5652 в https://tools.ietf.org/html/rfc5652).

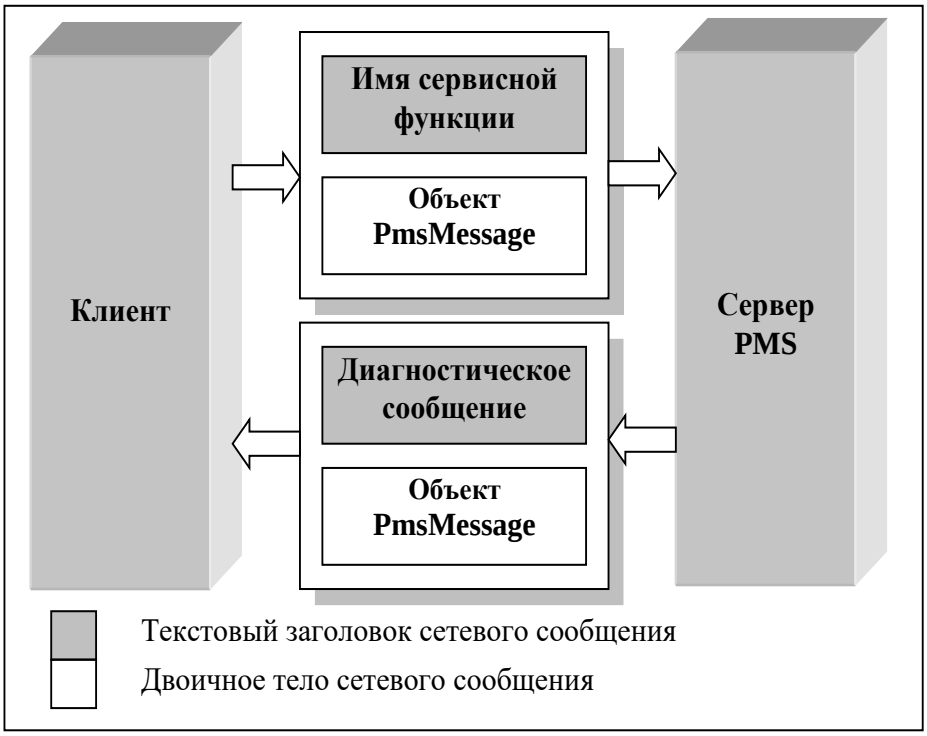

Рис. 3. Сетевой протокол Службы защищенных сообщений

3 Реализация PMS на базе CMS

Данный подход к реализации PMS основан на функциональном сходстве ее главного класса (PmsMessage) с главным классом CMS (SignedCms): оба класса представляют контейнер для хранения произвольных данных, оснащенный необходимыми методами для формирования и проверки электронных подписей. Вместе с тем CMS не содержит классов и методов для удаленной обработки данных в сети (аналогов класса PmsConnection или метода PmsMessage.Process). Фактически, описываемый подход можно рассматривать как создание своего рода «надстройки» над CMS, направленной на сетевую обработку данных.

На рис. 4 проиллюстрировано соотношение основных классов и методов PMS и CMS, представляющее собой основу описываемого подхода. Стрелки обозначают прямой вызов одного метода другим. В частности, метод AddSignatures класса PmsMessage (см. рис. 1) выполняет вызов метода ComputeSignature класса SignedCms для формирования каждой ЭЦП в защищенном сообщении, а метод Process опирается в своей работе на методы класса EnvelopedCms для

выполнения шифрования отправляемых в сеть данных и дешифрования данных, принятых из сети (методы Encrypt и Decrypt соответственно).

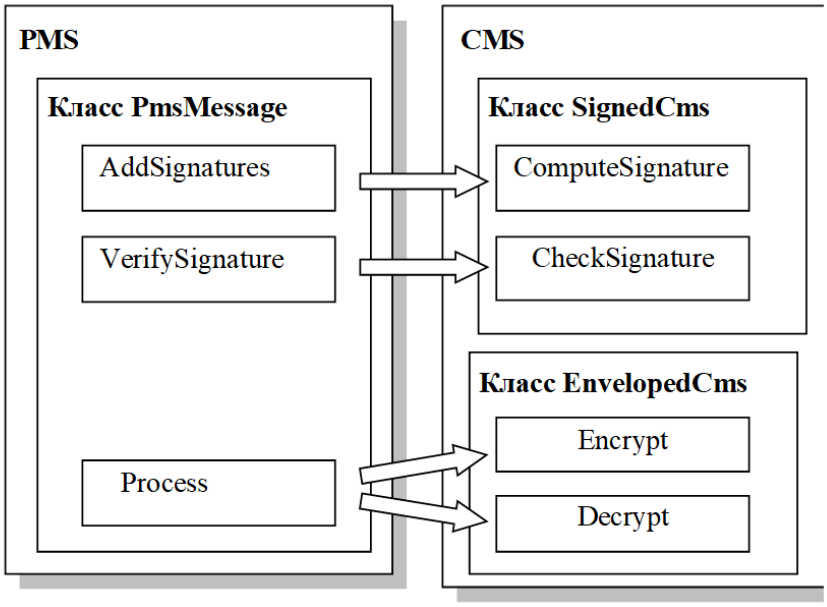

Рис. 4. Основные классы и методы PMS и CMS

4 Временные оценки

Для исследования свойств описанного подхода была проведена серия экспериментов с новой версии PMS. Основная цель этих экспериментов заключалась в сравнении быстродействия двух методов ее реализации (на основе архитектур «PMS-Криптосистема» и «PMS-CMS-Криптосистема») между собой, а также с быстродействием Web-сервисов в одинаковых условиях. Главное внимание уделялось вызовам сервисных функций с относительно малым (от нескольких миллисекунд до нескольких сотен миллисекунд) временем выполнения (при более длительной обработке разница между двумя технологиями практически нивелируется) с применением средств ЭЦП и шифрования сообщений на основе криптосистемы «КриптоПро» версии 3.6, соответствующей требованиям действующих в России ГОСТов в области криптографической защиты информации. В экспериментах с PMS использовались средства криптозащиты «КриптоПро», интегрированные в клиентскую библиотеку PmsBase.dll и сервер PMS или «напрямую» или посредством средств CMS. В экспериментах с Web-сервисами средства криптосистемы «КриптоПро» подключалась непосредственно к программе клиента и программе Web-сервиса. И серверы PMS с модельными библиотечными функциями и Internet Information Server с модельными Web-сервисами были установлены на одном и том же четырехъядерном сервере приложений с тактовой частотой 2.4 ГГц в операционной среде Window 2003 Server, а в качестве клиентской рабочей станции использовался одноядерный компьютер с тактовой частотой 2.8 ГГц.

На рис. 5 показаны характерные результаты экспериментов с очень быстрой сервисной функцией, выполняющей простое перекодирование полученного строчного сообщения в верхний регистр и возврат результата клиенту, при длине сообщения в 2 Кбайт, 50 Кбайт и 100 Кбайт соответственно. На рисунке приведена диаграммы времен выполнения операции на сервере в реализации «PMS-КриптоПро» (черный столбик), в реализации «PMS-CMS-КриптоПро» (серый столбик) и с помощью Web-сервиса (белый столбик). В каждом режиме время выполнения вычислялось, как среднее значение для 100 последовательных вызовов сервисной функции.

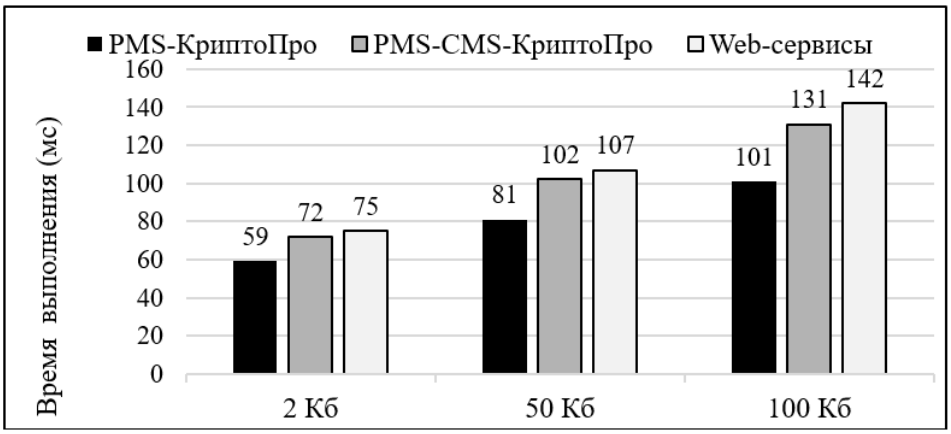

Рис. 5. Оценки быстродействия двух реализаций PMS и Web-сервиса

На рис 6 и 7 приведены результаты экспериментов с относительно медленными сервисными функциями со временем выполнения в несколько сотен миллисекунд. Главное внимание в этой серии экспериментов уделялось сравнению быстродействия двух реализаций PMS и Web-сервисов в условиях высокой нагрузки: при одновременной обработке пакетов информационных запросов и при применении криптозащиты. Скорость обработки вычислялась, как частное от деления числа запросов в пакете на полное время его выполнения. Характерные кривые, приведенные на рис.6 и 7, отражают зависимость скорости обработки от количества запросов в пакете для двух реализаций PMS и Web-сервисов при обращении к модельным сервисным функциям со временами выполнения 0.5 и 1 секунды соответственно с применением средств криптозащиты и длине входного и выходного сообщений 50 Кбайт.

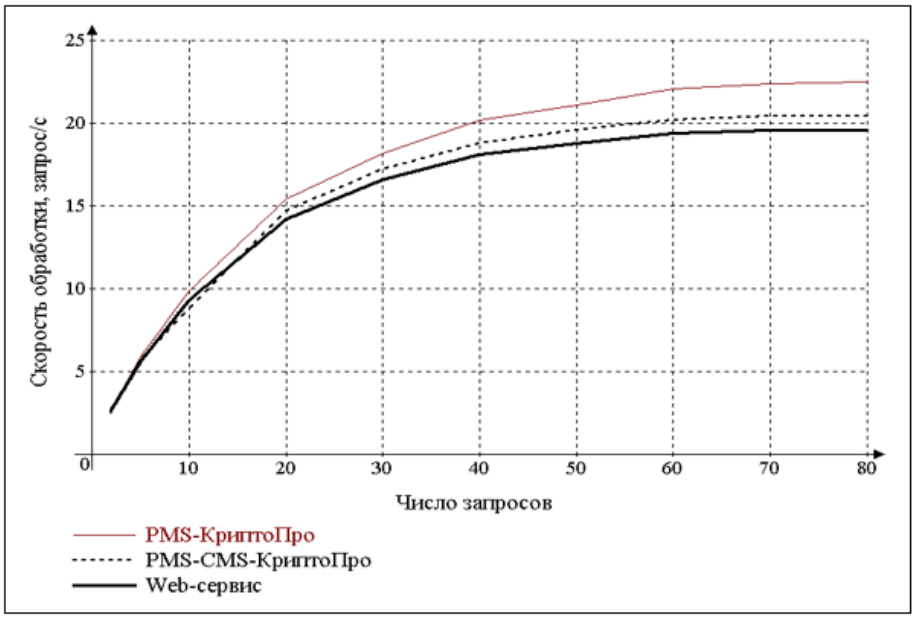

Рис. 6. Скорость обработки пакетов запросов (время обработки запроса 0.5 с.)

Как видно из рисунков, все кривые ведут себя в целом одинаково. При увеличении числа запросов в пакете растет и скорость обработки, что объясняется положительным эффектом от многопоточной обработки запросов и в IIS и в сервере PMS. Например, при 20-ти запросах в пакете и времени выполнения сервисной функции 500 мс скорость обработки достигает 15 запросов в секунду у PMS и 14 – у Web-сервиса (при последовательной обработке скорость не могла бы превысить значения 2). Однако при дальнейшем увеличении размеров пакета рост скорости обработки замедляется, а потом и вовсе останавливается вследствие достижения предельной производительности. Как видно из графиков, в этой серии экспериментов обе реализации PMS несколько превышают Web-сервисы по скорости обработки, но это превышение не является значительным.

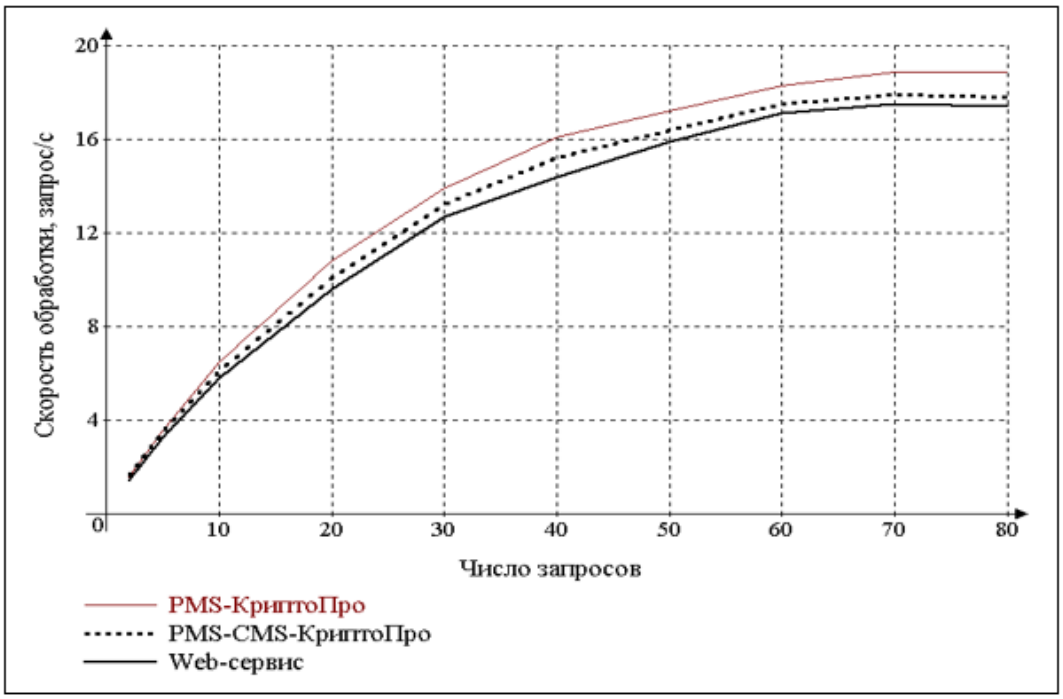

Рис. 7. Скорость обработки пакетов запросов (время обработки запроса 1 с.)

Заключение

В целом результаты экспериментов позволяют сформулировать следующие выводы.

- Реализация PMS на основе архитектуры «PMS-CMS-КриптоПро» в целом несколько уступает в быстродействии «прямой» реализации на основе архитектуры «PMS-КриптоПро», но в случае использования относительно медленных сервисных функций (с временем выполнения более 0.5 секунды) разница становится пренебрежимо малой (см. рис. 7).
- Обе реализации PMS не уступают в быстродействии Web-сервисам.
- Применение средств криптозащиты в обеих реализациях PMS не разрушает положительного эффекта от многопоточной обработки запросов.
- Как и Web-сервисы, обе реализации PMS вполне позволяют поддерживать скорость обработки до нескольких и даже нескольких десятков запросов в секунду даже при использовании средств криптозащиты, что обычно бывает достаточным для большинства информационных систем.

Литература

1. *Мак-Дональд М., Шпушта М.* Microsoft ASP.NET 3.5 с примерами на C# 2008 и Silverlight 2 для профессионалов. – М.: Вильямс, 2009. – 1408 с.
2. *Згоба А.И., Маркелов Д. В., Смирнов П. И.* Кибербезопасность: угрозы, вызовы, решения / Вопросы кибербезопасности, 2014, № 5. С.30 – 38.
3. *Козлов А. Д., Орлов В. Л.* Методы и средства обеспечения информационной безопасности распределенных корпоративных систем. – М.: ИПУ РАН, 2017. – 156 с.
4. *Асратян Р. Э.* Интернет-служба защищенной обработки информационных запросов в распределенных системах // Программная инженерия, 2016, № 11. – С.490 – 497.
5. *Хант К.* TCP/IP. Сетевое администрирование. – СПб.: Питер, 2007. – 816 с.