шифрования:

- ввод ключа пользователем;
- генерация и хранение ключей на отдельном сервере, которому доверяют все участики процесса обмена информацией.

При вводе ключа пользователем он должен передать этот ключ всем адресатам каким-либо образом. Данный способ прост в реализации, так как не требует отдельного сервера для распределения и хранения ключей. Но у него есть существенный минус, так как пользователи должны передавать друг другу ключи безопасным способом.

Для реализации автоматической генерации ключей, необходимо создать для этого сервер, которому должны доверять все участники обмена информацией. Все пользователи предварительно должны быть авторизованы на этом сервере. В этом случае, перед шифрованием сообщения пользователь делает запрос на сервер для получения ключа. Сервер генерирует ключ и идентификационный номер сообщения, сохраняет их в базе данных и передает пользователю через защищенный канал, использующий протокол HTTPS или асимметричный алгоритм шифрования на уровне приложения. Пользователь, получив ключ и идентификационный номер сообщения, шифрует сообщение, добавляет к нему идентификационный номер и отправляет адресатам, а список адресатов отправляет на сервер. Адресаты, получив сообщение, читают из него идентификационный номер и отправляют запрос на сервер для получения ключа для расшифровки этого сообщения. Сервер, получив запрос на получение ключа, проверяет, имеет ли данный пользователь право на получение данного ключа, если проверка прошла успешно, ключ отправляется пользователю. Пользователь, получив ключ, использует его для расшифровки сообщения.

Данный подход сложнее в реализации, но проще для пользования. Он подходит для корпоративного использования, так как сервер для генерациии и хранения ключей можно разместить на компьютерах в локальной сети, что решает проблему доверия к этому серверу.

Немаловажным фактором является удобность использования программного средства для шифрования информации на клиенте. Обычно люди пользуются определенной программной-клиентом при работе с электронной почтой. Их выбор основан на таких факторах, как функциональность, удобство интерфейса, соответствие другим персональным требованиям. При реализации программного средства как отдельного приложения пользователю придется отказаться от использования привычного почтового клиента, если он хочет воспользоваться функциями шифрования. Для пользователя это неудобно, так как он привык работать с определенным клинтом, он можен сомневаться в качестве реализации функций, присущих обычному почтовому клиенту, в новом приложении.

Таким образом, наиболее удобным для пользователя способом реализации программного средства шифрования на клиенте являются расширения для уже существующих почтовых клиентов. Кроме удобства такой подход также удобен тем, что не нужно реализовывать функционал обычного почтового клиента. Это позволяет упростить разработку и не заниматься поддержкой функциональности, не связанной с шифрованием.

В настоящее время наиболее широко используемыми почтовыми клиентами являются клиенты, реализованные как веб-интерфейсы. Поэтому наиболее целесообразным является разработка расширений для браузеров, которые бы внедряли функции шифрования в почтовые веб-клиенты. При таком подходе существует еще одно немаловажное преимущество – расширения для браузеров разабатываются на языке JavaScript. Таким образом, можно разработать ядро программного средства, которое будет осуществлять шифрование, в виде библиотеки на языке JavaScript и использовать при реализации расширений для различных браузеров.

Список использованных источников:

1. Ярмолик В.Н., Портянко С.С., Ярмолик С.В. Криптография, стеганография и охрана авторского права. – Мн.:Издательский центр БГУ, 2007. – 242 с.
2. Habrahabr [Электронный ресурс]. – Режим доступа: http://habrahabr.ru/company/mailru/blog/230743/, свободный. – Загл. с экрана. – Яз. Русский

МОДУЛЬ ВЫЗОВА И ПРЕДОСТАВЛЕНИЯ УДАЛЕННЫХ СЕРВИСОВ ДЛЯ ПЛАТФОРМЫ RUBY ON RAILS

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Перминов В. В.

Бахтизин В. В. – к-т. техн. наук, профессор

Сервис-ориентированная архитектура (СОА) – относительно новый архитектурный стиль в разработке программного обеспечения. Множество стандартов по передаче и обработки информации основаны на сервис-ориентированной архитектуре. СОА разделяет задачи на отдельные модули, называемые сервисами, которые могут быть распределены по сети. Сервис – общий интерфейс, который предоставляет

доступ к каким-либо данным или отдельной функциональности и может быть задействован одновременно в различных приложениях. Таким образом, повышается стабильность системы в целом, улучшается кроссплатформенность, снижается стоимость и повышается скорость разработки.

Поддержка сервис-ориентированной архитектуры очень важна для текущих и будущих проектов на платформе Ruby on Rails, т.к. в современных веб-приложениях конкретные модули и сервисы могут располагаться удаленно, т.е. вне сервера с самим приложением. В настоящий момент Ruby on Rails имеет некоторые ограничения в области сервис-ориентированной архитектуры. Среди них – отсутствующая функциональность для вызова и предоставления сервисов как в самой среде Rails по умолчанию, так и в сторонних решениях в виде одного модуля. В результате появляется необходимость пользоваться несколькими сторонними решениями, которые, ко всему прочему, имеют плохую интеграцию между собой.

Решением данной проблемы может стать разработка единой системы, обеспечивающей расширяемый механизм вызова и предоставления удаленных сервисов по протоколу HTTP и HTTPS для Ruby on Rails. Исходя из требований к разрабатываемой системе, она должна состоять из двух модулей:

1. ServiceInvoker – выполняет запросы к удаленным сервисам и обработку ответов, обеспечивает механизм, который может быть использован для реализации любого протокола, выполняющего запросы к удаленным сервисам через HTTP.
2. ServiceProvider – выполняет прием запросов от удаленных сервисов и их обработку, улучшает существующие возможности среды Rails для предоставления сервисов по протоколу HTTP.

Данная система встраивается в платформу Ruby on Rails в виде подключаемой библиотеки и используется для связи между собой распределенных по сети ресурсов. Достоинством использования данной системы является повышение модульности и стабильности веб-приложения в целом, улучшение его кроссплатформенности, дальнейшее снижение стоимости и повышение скорости разработки.

ПРОГРАММНО-АППАРАТНОЕ СРЕДСТВО ИССЛЕДОВАНИЯ СВОЙСТВ ФИЗИЧЕСКИ НЕКЛОНИРУЕМОЙ ФУНКЦИИ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Пучков А. В.

Иванюк А. А. – доктор технических наук, профессор

Одной из ключевых проблем, возникающих при проектировании цифровых устройств, в том числе реализуемых на основе программируемых логических интегральных схем, на современном этапе развития технологий и рынка, является их идентификация. Перспективным направлением разрешения обозначенной проблемы является применение так называемых физически неклонируемых функций. Рассмотренное в работе программно-аппаратное средство даёт возможность провести экспериментальные исследования, позволяющее сделать выводы об эффективности решения задачи идентификации с помощью физически неклонируемых функций для цифровых устройств на базе программируемой логики.

Методы физической криптографии, в основе которой лежит структурная сложность электронных систем, всё чаще находят применение в области защиты цифровых систем от нелегального использования [1]. Согласно одному из современных формальных определений, которое было предложено П. Туилсом (P. Tuyls), физически неклонируемая функция (англ. Physically Unclonable Function, PUF) есть характеристика физической (цифровой) системы, которая не подлежит клонированию (копированию, воспроизведению) на других системах [2]. В процессе создания цифровых систем принципиально невозможно управлять величинами многих физических параметров, вследствие чего последние из-за физической вариации технологического процесса принимают случайные, но уникальные для каждой цифровой системы значения. Идея извлечения подобных уникальных параметров из цифровых систем и лежит в основе аппаратных PUF [1].

Задача PUF как цифрового устройства состоит в получении на выходных портах множества ответов R, соответствующих множеству запросов C таким образом, что пары (C_i, R_i) будут уникальными, непредсказуемыми и неклонируемыми на других аналогичных интегральных схемах [3]. Такие свойства позволяют использовать PUF для эффективного решения целого ряда задач, в первую очередь для уникальной идентификации цифровых систем.

Основой построения и функционирования многих PUF является измерение задержки сигналов в реконфигурируемых путях цифровых устройств. Классическим примером такого подхода является PUF типа арбитр, где на одном кристалле интегральной схемы осуществляется построение двух топологически и функционально идентичных путей, имеющих близкие, но принципиально различные из-за физической вариации технологического процесса, величины времени распространения сигналов по ним. Симметричные пути проектируются как пары двухвходовых мультиплексоров, управляющие входы которых образуют шину входных запросов PUF. Измерение разницы во времени распространения сигналов между двумя путями может быть достигнуто одновременной подачей фронта импульса и определением на выходах, какой из них оказался длиннее [1]. Последнее может в простейшем случае быть достигнуто при помощи синхронного D-триггера, сбрасываемого до выполнения измерений. В этом случае один из путей поступа-