

Специализированное in-методу хранилище информации о TLS-сертификатах и TLS-узлах с быстрым поиском

Александр Венедюхин

Софья Проклашкина



Аннотация

В статье представлено описание архитектуры специализированного in-методу хранилища для обработки TLS-сертификатов. Разработанная высокопроизводительная система оптимизирована для выполнения специализированных поисковых запросов, учитывающих формат и свойства TLS-сертификатов. Высокая производительность достигается за счёт использования комбинации эффективных структур данных, включая синхронизированные хеш-таблицы (sync.Map), префиксные деревья (Radix Tree) и суффиксные автоматы. Реализация на языке Go обеспечивает конкурентную обработку данных при чтении и записи.

Ключевые слова:

TLS-сертификат, TLS-протокол, удостоверяющий центр, криптографическая безопасность, уязвимость, база данных, хранилище в памяти, поисковый запрос, хеш-таблица, Интернет.

Введение

Современные информационные системы требуют специализированных решений для хранения и обработки данных, особенно когда речь идёт о высоконагруженных сервисах с жёсткими параметрами скорости отклика. В данной работе рассматривается пример такой системы, оптимизированной для работы с TLS-сертификатами. Предлагаемый подход может быть адаптирован и для других типов и структур данных.

Сертификаты TLS (Transport Layer Security), которые иногда по традиции называют «SSL-сертификатами» (Secure Sockets Layer), используются для аутентификации узлов сети Интернет и создания защищённого соединения между клиентом, в качестве которого часто выступает браузер, и сервером. С помощью средств цифровой подписи сертификат обеспечивает привязку открытого ключа сервера к сетевому имени и, таким образом, подтверждает подлинность веб-ресурса, а также позволяет шифровать передаваемую информацию, защищая её от перехвата и подделки [1].

Практическое значение разработки хранилища именно для TLS-сертификатов заключается в возможности его применения в системах мониторинга сетевого трафика, инфраструктуре центров сертификации, а также в различных сервисах информационной безопасности, где скорость обработки запросов имеет критическое значение.

Актуальность подобного хранилища обусловлена отсутствием возможности реализации уникальных решений на базе

универсальных СУБД, которые не могут обеспечить необходимую производительность для узкоспециализированных задач, а также удовлетворить требованиям некоторых предметных областей к структурам данных и операциям над ними. В случае с TLS-сертификатами критически важными становятся операции быстрого поиска по различным атрибутам, интерпретация которых специфична именно для области применения сертификатов (по IP-адресу с учётом подсетей, по доменному имени, дате действия, публичным ключам с учётом их математических особенностей), что и определило алгоритмическую архитектуру разрабатываемого решения.

Протокол TLS

TLS (Transport Layer Security) — это протокол для защищённой передачи данных в сети Интернет. Он обеспечивает целостность данных с помощью кодов аутентификации сообщений («имитовставок»), конфиденциальность — посредством применения методов симметричного шифрования. Для проверки подлинности соединения, то есть соответствия источника информации ожидаемому имени, используются асимметричные криптосистемы цифровой подписи. TLS работает поверх уже установленного соединения между узлами и использует это соединение для передачи TLS-сообщений. Как правило, протокол TLS работает поверх протокола TCP [2].

Протокол TLS является одним из самых изученных в Интернете, исследования по поиску уязвимостей проходят на постоянной основе и ведут к обновлению его архитектуры и устранению ошибок в новых версиях. Самой современной версией TLS является TLS 1.3, в которой значительное внимание уделено сокрытию метаданных.

Сертификат TLS

Сертификаты TLS позволяют решить проблему подмены узлов методом «атаки посредника», при которой злоумышленник выдаёт себя за конечный узел, передавая собственный открытый криптографический ключ.

Сертификат TLS — это публичный цифровой документ, аутентифицирующий конечные узлы и привязывающий открытый криптографический ключ к доменному имени при помощи цифровой подписи удостоверяющего центра (УЦ).

Сертификаты в TLS бывают трёх видов — корневые, промежуточные и серверные. Сертификаты сопоставляются по именам, указанным в них, и, таким образом, строится цепочка доверия, в которой с помощью ключа корневого сертификата можно проверить подлинность промежуточного сертификата, а с помощью ключа промежуточного — подлинность серверного (этот сертификат часто называют «оконечным» сертификатом). Корневые открытые ключи УЦ содержатся в базе данных проверяющего узла, а доверие к конкретному веб-серверу основывается на том, что веб-сервер знает секретный ключ, соответствующий открытому ключу серверного сертификата [3].

Удостоверяющие центры считаются доверенной стороной, чей открытый ключ широко известен. К функциям УЦ относятся подпись сертификатов, получение и проверка информации, необходимой для подписи сертификата, отзыв сертификата по требованию владельца имени и секретного ключа, соответствующих сертификату.

Сертификат — это структура данных, которая включает в себя следующую информацию:

- Serial Number — серийный номер;
- Subject — имя субъекта (например, доменное имя);
- Issuer — имя издателя, обычно — УЦ;
- SubjectPublicKeyInfo — информация об открытом ключе;
- Signature Algorithm — алгоритм подписи сертификата;
- Not Before Date — начало действия сертификата;
- Not After Date — окончание действия сертификата;
- и другие поля [4].

Пример содержимого некоторых полей TLS-сертификата, расшифрованного с помощью утилиты openssl, показан на рисунке 1.

```
$ openssl x509 -in cacert.tls.pem -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 11485830970703032316 (0x9f65de69ceef2ffc)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=US, ST=MD, L=Baltimore, CN=Test CA/emailAddress=test@example.com
    Validity
      Not Before: Jan 24 14:24:11 2017 GMT
      Not After : Feb 23 14:24:11 2018 GMT
    Subject: C=US, ST=MD, L=Baltimore, CN=Test CA/emailAddress=test@example.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (4096 bit)
      Modulus:
        00:b1:7f:29:be:78:02:b8:56:54:2d:2c:ec:ff:6d:
        ...
        39:f9:1e:52:cb:8e:bf:8b:9e:a6:93:e1:22:09:8b:
```

Рис. 1. Пример содержимого некоторых полей TLS-сертификата, расшифрованного с помощью утилиты openssl.

Хранилища сертификатов TLS

Эффективность TLS-сертификатов напрямую зависит от правильного управления и хранения. В связи с этим появляется необходимость в специализированных системах, которые позволяют не только хранить TLS-сертификаты, но и анализировать их на предмет уязвимостей, а также оперативно реагировать на возможные инциденты безопасности.

Сервисы, подобные разрабатываемому, могут выполнять функцию центрального хранилища TLS-сертификатов, собранных с различных оконечных узлов. При этом такие данные, как IP-адреса, на которых был обнаружен TLS-сертификат; время сбора; информация о владельце и другие метаданные, сохраняются вместе с самим TLS-сертификатом. Это позволяет строить эффективную систему мониторинга и анализа состояния криптографической безопасности в сетях.

Примеры сервисов, работающих с анализом и хранением TLS-сертификатов:

1. Certificate Transparency (CT) — представляет собой инициативу в области публичного аудита, направленную на повышение подотчётности удостоверяющих центров посредством ведения неизменяемых журналов регистрации всех выпущенных TLS-сертификатов.
2. Crt.sh — поисковый интерфейс к данным журналов CT, предоставляющий возможность осуществлять запросы к сформированному по данным CT реестру сертификатов на основе специальных атрибутов, таких как доменное имя или криптографический отпечаток.
3. Censys — платформа для комплексного анализа сетевой инфраструктуры. В результате проведения активных сканирований IPv4-адресного пространства сервис агрегирует данные о доступных хостах, включая метаданные представленных TLS-сертификатов.
4. Shodan — специализируется на обнаружении и инвентаризации сетевых устройств и Интернета вещей (IoT). В процессе сканирования система также извлекает и каталогизирует информацию о TLS-сертификатах, что позволяет проводить анализ их параметров и выявлять потенциально уязвимые реализации.

5. ZMap — представляет собой высокопроизводительный инструмент для сканирования сетей, предназначенный для выполнения полного опроса IPv4-адресации в исследовательских целях. ZMap позволяет проводить крупномасштабные измерения, направленные в том числе на сбор статистических данных о распространении и конфигурации TLS в глобальной сети.

Таким образом, можно выделить список основных функций подобных сервисов: сбор и хранение TLS-сертификатов; сохранение сопутствующих метаданных — IP-адресов, временных меток и информации о владельце; обеспечение возможности быстрого поиска и фильтрации TLS-сертификатов по различным критериям; проведение автоматизированного анализа криптографических параметров TLS-сертификатов с целью выявления потенциально скомпрометированных или слабых ключей.

Проектирование базы данных

В процессе анализа спецификаций и стандартов, предписывающих нормы и правила использования TLS-сертификатов, были выбраны следующие сценарии поиска данных в разрабатываемой системе.

1. Поиск TLS-сертификатов по IP-адресу. Позволяет находить все сертификаты, полученные с узлов под данными IP-адресами. (В данной разработке используется только IPv4.) Этот сценарий полезен при расследовании инцидентов безопасности, когда известен IP-адрес, связанный с подозрительной активностью. Для реализации было выбрано два способа поиска — по точному совпадению IP-адреса и по сетевому префиксу. Точное совпадение может использоваться, если злоумышленник использует определённый сервер. Поиск по префиксу может применяться при внутреннем аудите инфраструктуры; чтобы проверить адресно «близкие» узлы; определить, какие именно TLS-сертификаты установлены на различных серверах одной информационной системы; а также при исследовании активности вредоносных сетей или CDN.
2. Поиск TLS-сертификатов по издателю. Позволяет находить все TLS-сертификаты, выпущенные указанным удостоверяющим центром. В случае его компрометации или выявления недобросовестного поведения со стороны УЦ этот сценарий позволяет быстро найти все выпущенные им TLS-сертификаты и оценить масштаб возможного риска. Также он может использоваться для внутреннего контроля — например, чтобы убедиться, что используют только доверенные УЦ.
3. Поиск TLS-сертификатов по серийному номеру. Номер TLS-сертификата — это уникальный идентификатор, который может быть использован для отслеживания конкретного экземпляра. Номер должен быть уникальным, чтобы избежать коллизий при управлении TLS-сертификатами. Также с номерами сертификатов связаны некоторые особенности: например, предсказуемость номера может свидетельствовать о наличии уязвимости в системах УЦ и создать конфликт в системах мониторинга — так, ранее из-за использования нестойких хеш-функций УЦ (MD5) злоумышленник мог на практике угадать серийный номер и получить «дублирующий», поддельный сертификат с валидной подписью. Для разрабатываемого хранилища были выбраны два сценария поиска сертификатов с заданными серийными номерами — по точному совпадению и по подстроке в символьной записи номера. Точное совпадение позволяет быстро искать узлы и TLS-сертификаты по их серийному номеру или получать список различных TLS-сертификатов с одним номером. Реализация поиска по подстроке обеспечивает возможность обнаружения паттернов в создании серийных номеров TLS-сертификатов при их выпуске.
4. Поиск по значению открытого ключа. Позволяет находить сертификаты с ключами, обладающими некоторыми математическими особенностями. Важным аспектом в обеспечении безопасности является нахождение TLS-сертификатов, использующих конкретный открытый ключ, а также нахождение ключей, криптографические элементы которых математически зависят друг от друга. Необходимо обеспечить возможность расширения доступных математических операций над значениями ключей путём универсального хранения необходимых данных для исследований. Для реализации были выбраны следующие сценарии: поиск по точному совпадению элементов публичных ключей разных типов; поиск TLS-сертификатов, для которых значения модуля RSA-ключа не взаимно просты. Последний сценарий позволит обнаружить ключи, уязвимые к факторизации тривиальными методами.
5. Поиск по шестнадцатеричному значению представления открытого ключа. Шестнадцатеричное представление ключа часто используется в отчётах, логах и исследованиях. Возможность поиска по нему может помочь в сопоставлении данных между различными системами, а также в поиске TLS-сертификатов, использующих аналогичные или похожие ключи. Так же, как и для серийных номеров, были выбраны реализации сценария с частичным и полным совпадением.

Оптимизация процессов поиска

В данном проекте основным способом обеспечения большой производительности является хранение всех данных в оперативной памяти, что и обозначается термином «in-memoory хранилище».

Оптимизация алгоритмов поиска в системе хранения TLS-сертификатов, размещённой в оперативной памяти, тоже играет ключевую роль в обеспечении высокой производительности и эффективности анализа данных. Поскольку объём хранимых TLS-сертификатов может быть очень большим — особенно при сборе информации из Интернета, — простые переборные или линейные методы поиска становятся неприемлемыми с точки зрения времени отклика. Для оптимизации и ускорения были выбраны следующие структуры:

1. Radix Tree — представляет собой сжатое дерево префиксов, в котором ключи хранятся в виде строк, а пути между узлами определяются общими префиксами. В отличие от обычного префиксного дерева, Radix Tree объединяет узлы с единственным дочерним элементом, что позволяет значительно сократить объём используемой памяти и повысить эффективность поиска. В контексте разрабатываемого хранилища TLS-сертификатов Radix Tree применяется для поиска TLS-сертификатов по префиксу IP-адреса, поскольку хорошо соответствует принципам организации IP-адресного пространства. Это особенно полезно при анализе подсетей или при обработке запросов, связанных с группами IP-адресов. Основными преимуществами использования Radix Tree в данном случае являются низкая вычислительная сложность выполнения операций вставки, поиска и удаления, а также возможность эффективной реализации диапазонных запросов на основе IP-префикса.
2. Suffix Automaton — это компактная структура данных, представляющая собой минимальный детерминированный конечный автомат, распознающий элементы множества суффиксов базовой строки. Одним из ключевых свойств суффиксного автомата является линейная зависимость количества состояний и количества переходов от длины входной строки, позволяющая работать с большими объёмами данных. В данной системе суффиксный автомат используется для поиска TLS-сертификатов по частичному совпадению в шестнадцатеричном представлении открытого ключа и серийного номера. Это позволяет находить TLS-сертификаты, содержащие заданную последовательность символов в этих полях, что имеет важное значение при выявлении потенциально скомпрометированных или слабых криптографических параметров. Суффиксный автомат обеспечивает линейную временную сложность построения и логарифмическую сложность поиска.
3. sync.Map — это конкурентно-безопасная реализация ассоциативных массивов из стандартной библиотеки языка Go, которая оптимизирована для случаев, когда один набор ключей читается многократно, но изменяется редко. В данном проекте sync.Map применяется для хранения TLS-сертификатов в привязке к различным индексным множествам: IP-адрес, издатель, серийный номер, шестнадцатеричное представление открытого ключа. Это позволяет находить точные совпадения за практически постоянное количество базовых операций и при этом обеспечивать потокобезопасный (конкурентный) доступ к данным без внешних механизмов блокирования и синхронизации. Данный тип структуры выбран из-за своей производительности и безопасности использования при параллельных вычислениях.

Описанные решения позволяют существенно снизить временную сложность выполнения запросов, обеспечивая логарифмическую или даже константную скорость доступа к данным. Поиск эффективен не только для простых «пользовательских» запросов, но и для автоматизированного анализа метрик безопасности, например, мониторинга скомпрометированных или слабых ключей. Благодаря оптимизации механизмов поиска система является масштабируемой,

отзывчивой и пригодной для решения задач информационной безопасности в режиме реального времени.

Разработка программного приложения

В качестве языка реализации был выбран язык Go, предоставляющий встроенную поддержку параллелизма через go-рутины и каналы, что позволяет эффективно управлять множеством одновременных операций ввода-вывода и обработки данных. Кроме того, стандартная библиотека Go содержит мощные инструменты работы с криптографическими примитивами, что существенно упрощает работу с TLS-сертификатами.

Проект реализован по модульной архитектуре, включающей слои доставки (delivery), бизнес-логики (usecase) и хранения данных (repository). Основное хранилище TLS-сертификатов представлено структурой Repository (рис. 2), которая содержит несколько специализированных контейнеров: sync.Map для точного поиска по IP, издателю, серийному номеру и шестнадцатеричному представлению публичного ключа; Radix Tree для префиксного поиска IP-адресов; Suffix Automaton для поиска частичного совпадения строк серийных номеров и шестнадцатеричных представлений ключей; KeyStorage для хранения и сравнения публичных ключей. Все эти структуры взаимодействуют между собой и обеспечивают эффективный доступ к данным по различным критериям поиска.

```
type Repository struct { 10 usages  ⚡ Софья *
    ipRadixTree          *radixTree.RadixTree
    ipMap                sync.Map
    issuerMap            sync.Map
    serialSuffixAutomaton *suffixAutomaton.SuffixAutomaton
    serialMap            sync.Map
    HexPubKeyMap         sync.Map
    HexPubKeySuffixAutomaton *suffixAutomaton.SuffixAutomaton
    Keys                 *KeyStorage
    loadWg               sync.WaitGroup
    loadWorkerCh         chan struct{}
```

Рис. 2. Скриншот содержимого структуры Repository.

Доступ к сервису реализуется через HTTP-сервер, запускаемый на номере порта, указанного в конфигурации (по умолчанию — 8080). Сервер реализован с использованием стандартной библиотеки Go net/http. Единственная точка входа — это энд-поинт /query, принимающий JSON-данные в POST-запросах. HTTP-запрос обрабатывается функцией HandleQuery, которая десериализует JSON-объект в структуру domain.Query, передаёт её в уровень usecase и формирует ответ в виде списка TLS-сертификатов или генерирует статус ошибки. Ответ сериализуется в JSON и отправляется обратно клиенту.

Загрузка базы данных TLS-сертификатов выполняется из TSV-файла, каждая строка которого содержит IP-адрес и

Base64-представление DER-кодированного TLS-сертификата. Загрузка выполняется параллельно. Перед добавлением в репозиторий входные данные проходят несколько этапов проверок: корректности формата строки, наличия требуемых полей, успешности декодирования Base64 и последующего разбора x509-сертификата. В случае ошибки на любом из этапов выводится соответствующее сообщение в лог, но обработка остальных строк продолжается.

Параллелизм при загрузке TLS-сертификатов обеспечивается с помощью пула рабочих go-рутин, ограниченной константой `maxLoadWorkers`. Необходимость параллельной обработки обусловлена в рамках задачи тем, что декодирование и парсинг сертификатов достаточно затратны по вычислительным ресурсам, при этом исходные данные базы (TSV) могут находиться на дисковом хранилище со сверхбыстрым доступом (SSD и пр.). Обновление исходных данных также может происходить с высокой частотой, а система планируется для использования на выборках, состоящих из десятков миллионов сертификатов. Для контроля количества одновременно работающих go-рутин используется буферизованный канал `loadWorkerCh`. При добавлении каждого TLS-сертификата в репозиторий задействуется механизм конкурентного добавления в разделяемые структуры данных. Для безопасного доступа к `sync.Map` используется метод `StoreToSyncMap`, который применяет мьютекс для модификации содержимого массива TLS-сертификатов, связанных с конкретным ключом. Также используются блокировки в структурах `KeyStorage`, `SuffixAutomaton`, `RadixTree` основанные на `sync.RWMutex`, что гарантирует целостность данных при одновременном чтении и записи.

Обработка TLS-сертификатов происходит в момент их добавления в репозиторий. Метод `AddCertificate` выполняет семантическое разложение (парсинг) TLS-сертификата на составляющие элементы и добавляет их в соответствующие структуры. Извлечение открытого ключа осуществляется с помощью метода `ExtractKeyInfo`, который поддерживает RSA и ECDSA-ключи. Для каждого найденного ключа вычисляется его уникальный хеш, используемый как идентификатор в `KeyStorage`. Шестнадцатеричное символьное представление открытого ключа рассчитывается с помощью `hex.EncodeToString`, а серийный номер преобразуется в строковое представление. Эти данные добавляются в `HexPubKeyMap` и `serialMap` соответственно, а также регистрируются в суффиксных автоматах для обеспечения возможности частичного поиска. IP-адреса заносятся в `ipMap` и дерево `Radix Tree`, что позволяет выполнять как точный, так и префиксный поиск.

Запросы к системе поступают методом HTTP POST на энд-поинт `/query` в виде JSON-объектов, содержащих два поля: `Action` и `Query`. Поле `Action` определяет тип действия, в рамках данной реализации оно всегда имеет значение «Select». Поле `Query` представляет собой вложенную структуру, где ключом является тип фильтрации (например, «IP», «Issuer», «Serial»), а значением — ещё один объект, содержащий поля «Func» и «Value». Поддерживаются следующие функции поиска: «Equal» — точное совпадение, «Prefix» — префиксный поиск, «Contains» — частичное совпадение. Запросы десериализуются с помощью `json.NewDecoder`, и проводится строгая проверка структуры данных. Если запрос не соответствует ожидаемому формату, возвращается ошибка.

Обработка запросов выполняется на уровне `usecase`, где реализован метод, вызывающий соответствующую функцию поиска в репозитории. После обработки результаты преобразуются в структуру, включающую информацию о TLS-сертификате в удобном для пользователя виде, в том числе в закодированном в Base64 виде и детализированном формате. Полученный список TLS-сертификатов сериализуется в JSON и отправляется обратно клиенту.

Таким образом, реализация приложения выполнена с учётом требований к высокой производительности, параллелизму и безопасности. Выбранный язык Go позволил эффективно использовать конкурентные конструкции, а именно — go-рутины, мьютексы и `sync.Map`. Проект предполагает разделение на уровни, что обеспечивает простоту расширяемости и тестирования. Система позволяет быстро находить TLS-сертификаты по различным критериям, включая точное и частичное совпадение, а также анализировать криптографические свойства, например, наличие общих множителей в RSA-модулях.

Заключение

Разработанная система может применяться в различных сферах, где требуется оперативный доступ к TLS-информации:

1. Мониторинг и выявление подозрительных TLS-сертификатов, связанных с известными угрозами, анализ временных меток и поведения удостоверяющих центров.
2. Анализ безопасности: обнаружение повторяющихся открытых ключей, предсказуемых серийных номеров, слабых RSA-ключей, использующих общие множители.
3. Исследования: сбор и анализ TLS-сертификатов в академических и научных целях, включая исследование распространённых практик выпуска TLS-сертификатов и выявления отклонений от стандартов безопасности.
4. Корпоративный аудит: организация может использовать систему для внутреннего контроля собственных TLS-ресурсов, включая мониторинг истечения срока действия, управление отзывом и анализ использования ключевых материалов.

Дальнейшее развитие системы может включать несколько перспективных направлений:

1. Добавление `persistence`: дополнение in-memoery хранилища средствами выгрузки данных долговременного хранения на дисковую подсистему, что позволит сохранять состояние системы и данные между запусками, работать с объёмами, превышающими доступную оперативную память.
2. Расширение типов поисковых запросов. В ходе проектирования построение системы происходило с опорой на возможные дальнейшие улучшения.

3. Интеграция с Certificate Transparency Logs: возможность автоматического получения новых TLS-сертификатов из публичных журналов CT и сравнения их с уже хранимыми данными для выявления неправильно выпущенных или потенциально опасных TLS-сертификатов.
4. Механизм подписки на события: добавление уведомлений о появлении новых TLS-сертификатов, истечении срока действия, обнаружении дублирования ключей и других событиях, важных для обеспечения информационной безопасности.
5. Поддержка дополнительных источников данных: расширение возможностей загрузки TLS-сертификатов за счёт непосредственной интеграции с сетевыми сканерами (например, ZMap), системами мониторинга трафика и сторонними API.
6. Веб-интерфейс: создание пользовательского интерфейса, позволяющего визуализировать с помощью графиков и таблиц результаты поиска, просматривать детали TLS-сертификатов и выполнять сложные выборки.

Таким образом, разработанная система представляет собой мощный и гибкий инструмент для анализа состояния TLS-инфраструктуры. Её текущая реализация уже позволяет выполнять широкий спектр поисковых и аналитических задач, а дальнейшее развитие открывает возможности применения в промышленных условиях, научных исследованиях и инструментах безопасности. Благодаря использованию современных алгоритмов и структур данных, а также внимательному подходу к организации параллелизма и безопасности, система является ценным инструментом в области обеспечения интернет-безопасности.

Список литературы:

- [1] ТЦИ. Сертификаты удостоверяющих центров TLS. URL: <https://tcinet.ru/press-centre/articles/7746/> (дата обращения: 25.05.2025).
- [2] TLS. Введение в TLS. URL: <https://tls.dxdт.ru/tls.html#intro> (дата обращения: 20.10.2023).
- [3] DXDT.RU TLS для DevOps. URL: <https://dxdт.ru/2021/11/28/9135/> (дата обращения: 25.05.2025).
- [4] IETF. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. URL: <https://www.ietf.org/rfc/rfc5280.txt> (дата обращения: 25.05.2025).

Об авторах

Венедюхин Александр Анатольевич,
ведущий специалист ФРСТ «ИнДата»

Проклашкина Софья Антоновна,
студентка МГТУ им. Н.Э. Баумана

