Руководство пользователя

КриптоАРМ Server 1.0

ООО «Цифровые технологии», 2025 г.

Содержание

Назначение и условия применения

КриптоАРМ Server — программный продукт, предназначенный для интеграции и автоматизации криптографических операций в информационных системах организаций. Он включает в себя набор криптографических библиотек и средства разработки (SDK), позволяющие встраивать функции шифрования данных, проверки электронных подписей и их создания в автоматизированных процессах обработки информации.

Функциональные возможности

Проверка электронной подписи

- Автоматизируйте проверку электронной подписи для поступающих документов
- Создавайте и храните протоколы проверки электронной подписи
- Визуализируйте электронную подпись для PDF документов

Автоматизация электронной подписи

- Создайте свой собственный сервис электронной подписи
- Поддержка стандартов CMS, CAdES, PAdES
- Используйте КриптоАРМ на рабочих станциях и мобильных устройствах как клиент для ЭП

Защита исходящих документов

- Попробуйте сервис для отправки конфиденциальных документов
- Автоматическое шифрование файлов перед отправкой по e-mail
- Автоматическая обработка входящих писем: расшифрование, проверка ЭП и категоризация

Функциональные модули

- **trusted-crypto** - формирование электронной подписи, проверка подписи, шифрование и расшифрование, сведения о сертификатах, генерация ключевой

пары и создание запросов на получение сертификата, подключение защищенных носителей.
- **trusted-pdf** - подпись и проверка электронной подписи встроенной в pdf, разметка областей для оттисков подписи, добавление графических штампов с информацией о подписи, пакетная подпись, сертификация подписанных документов с защитой от изменений, конвертация pdf-файлов в формат pdf-a/2b.
- **trusted-mailcore** - сбор, разбор и отправка электронных писем, поддержка IMAP, SMTP, POP3, S/MIME, контакты и адресные книги vCard, CardDAV.
- **trusted-curl** - обеспечение защищенного соединения к веб-серверам с поддержкой ГОСТ TLS.
- **trusted-xml-crypto** - проверка электронной подписи формата XMLDSig (XML Digital Signature).

Поддерживаемые криптопровайдеры

В приложении осуществляется поддержка криптопровайдера КриптоПро CSP версии 5.0 R3 и выше.

Поддерживаемые ключевые носители

В приложении поддерживается работа с ключевыми носителями Рутокен ЭЦП 2.0 USB, Рутокен ЭЦП 2.0 Type-C, Рутокен ЭЦП 3.0 NFC, Рутокен ЭЦП 3.0, Рутокен ЭЦП Type-C 3.0, JaCarta-2 ГОСТ через криптопровайдер КриптоПро CSP.

Автоматизация проверки электронной подписи

Автоматизация проверки электронной подписи является одной из наиболее востребованных задач в ИТ-инфраструктуре, связанной с электронным документооборотом. С использованием КриптоАРМ Сервер и модуля trusted-crypto вы можете быстро интегрировать проверку электронной подписи в серверные приложения, обеспечивая надежную проверку подписанных документов в соответствии с законодательством. В этой статье рассмотрим, как с помощью продукта КриптоАРМ Сервер и модуля trusted-crypto можно автоматизировать проверку электронной подписи.

КриптоАРМ Сервер — это продукт, состоящий из: - КриптоАРМ SDK (набор модулей для Node.js), среди прочего обеспечивающий простую интеграцию проверки подписей. - Полной документации для разработчиков. - Примера сервиса для проверки подписей и визуализации штампов в PDF-документах.

Модуль trusted-crypto входит в состав КриптоАРМ SDK. В части проверки подписи поддержаны стандарты от CAdES-BES до CAdES-A, методы для построения цепочек и проверки сертификатов, работа с TSP и OCSP и многое другое.

Для работы модуля необходимо следующее окружение и лицензии: - Node.js; - КриптоПро CSP; - Лицензии на КриптоАРМ и КриптоПро (серверные); - Лицензии на КриптоПро TSP Client и КриптоПро OCSP Client, если нужна работа с усовершенствованной подписью.

Проверка электронной подписи на сервере

Процесс проверки электронной подписи с помощью модуля trusted-crypto включает несколько этапов: загрузка подписанного файла или импорт буфера, проверка подписи и сертификатов подписчиков. Рассмотрим каждый этап подробнее на примере.

1. Загрузка подписанного файла

Первый шаг в процессе — это чтение подписанного файла. Для этого используем метод load или его асинхронный аналог loadAsync.

```
// Инициализация объекта для работы с подписанными данными
const cms = new trusted.cms.SignedData();

// Чтение файла с электронной подписью
cms.load("./outfile.sig");
```

Метод загружает файл с расширением .sig, содержащий электронную подпись. Возможна работа с разными форматами кодировками, такими как DER или BASE64.

Второй вариант, импорт буфера:

```
//Метод определения кодировку для буфера
export const getBufferEncoding = (cmsBuffer: Buffer) => {
  const firstTwoSymbols = cmsBuffer.toString("utf8", 0, 2);
  if (firstTwoSymbols === "--" || firstTwoSymbols === "MI") {
```

```
    return "PEM";
  } else {
    return "DER";
  }
};

//Получим кодировку прочитав первые байты сообщения
const encoding = getBufferEncoding(cmsBuffer);

//Импорт буфера в объект подписи
cms.import(cmsBuffer, trusted.DataFormat[encoding!]);
```

2. Проверка подписи

После загрузки файла следующим шагом будет проверка подписи с помощью метода verify. Этот метод возвращает булево значение: true — если подпись действительна, и false — если возникла ошибка верификации.

```
//Используем метод проверки подписи
const res = cms.verify();
console.log(`Результат проверки: ${res}`);
```

Для асинхронной работы можно использовать метод verifyAsync.

```
cms.verifyAsync((error, result) => {
    if (error) {
        console.error(`Ошибка проверки подписи: ${error}`);
    } else {
        console.log(`Результат проверки: ${result}`);
    }
});
```

3. Работа с отсоединенной подписью

Отсоединенная подпись (detached signature) — это подпись, которая передается отдельно от подписываемого контента. Для проверки такой подписи необходимо загрузить как файл с подписью, так и сам подписанный документ.

```
cms.load("./outfile.sig");
cms.content = {
```

```
    type: trusted.cms.SignedDataContentType.url,
    data: "./data.docx"
};
const result = cms.verify();
console.log(`Открепленная подпись действительна: ${result}`);
```

4. Проверка конкретного подписчика

Если необходимо проверить подпись конкретного подписчика (соподпись), можно получить коллекцию подписчиков с помощью метода signers и затем передать конкретного подписчика в метод verify.

```
const signers = cms.signers();
const signer = signers.items(0); // Выбираем первого подписчика
const result = cms.verify(signer);
console.log(`Подпись подписчика действительна: ${result}`);
```

5. Проверка сертификатов подписчика

Для проверки сертификата подписчика можно извлечь его из объекта Signer или передать сертификат отдельно. Далее строим цепочку и проверяем сертификат. Для построения цепочки используется хранилище КриптоПро. При этом списки отзыва и промежуточные сертификаты будут скачаны автоматически (если не установлены в хранилище и есть доступ к сети).

```
function verifyCertificate(cert) {
  return new Promise((resolve, reject) => {
    trusted.utils.Csp.verifyCertificateChainAsync(cert, (errorMsg,
result) => {
        if (errorMsg) {
                return reject({
                    title: 'Не удалось проверить цепочку сертификатов
для ' + cert?.subjectFriendlyName,
                    text: 'verifyCertificateChainAsync: ' + errorMsg,
                    timestamp: new Date().toISOString(),
                });
        }

        resolve(result);
    });
  });
```

```
}


function buildChain(cert) {
  return new Promise((resolve, reject) => {
    trusted.utils.Csp.buildChainAsync(cert, (errorMsg,
certCollection) => {
        if (errorMsg) {
                return reject({
                    title: 'Не удалось построить цепочку сертификатов
для ' + cert?.issuerFriendlyName,
                    text: 'buildChainAsync: ' + errorMsg,
                    timestamp: new Date().toISOString(),
                });
        }

        resolve(certCollection);
    });
  });
}


const certificate = signer.certificate;
const certChain = await buildChain(certificate);

for (let i = 0; i < certChain?.length; i++) {
  const cert = certChain?.items(i);
  const result = await verifyCertificate(cert);
  console.log(`Статус сертификата: ${result}`);
}
```

Использование в микросервисной архитектуре

В микросервисной архитектуре проверка электронной подписи с помощью КриптоАРМ Сервер и trusted-crypto может быть вынесена в отдельный сервис, который будет отвечать исключительно за криптографические операции. Такой подход упрощает масштабирование системы и повышает безопасность, так как каждый микросервис имеет ограниченный доступ к различным компонентам системы.

Пример возможного REST API:

Проверка подписи документа

Метод: `POST /api/v1/signature/verify`
Описание: Проверяет электронную подпись документа.

Запрос: - Content-Type: application/json или multipart/form-data (если файлы передаются через форму). - Параметры: - document (обязательный, Base64, File): Документ, для которого нужно проверить подпись. - signature (обязательный, Base64, File): Подпись в формате CMS. - certificate (опциональный, Base64, File): Сертификат, используемый для проверки подписи (если не встроен в подпись). - type (опциональный, string): Тип подписи (CAdES, CMS, XLT1). По умолчанию — автоопределение. - detached (опциональный, boolean): Указывает, является ли подпись откреплённой от документа (detached).

Ответ: - Status 200: Подпись валидна.

```
{
  "status": "valid",
  "details": {
  "issuer": "Issuer CN",
  "subject": "Subject CN",
  "valid_from": "2024-01-01T00:00:00Z",
  "valid_to": "2026-01-01T00:00:00Z",
  "signing_time": "2024-10-14T12:30:00Z"
  }
}
```

- Status 400: Ошибка валидации подписи.

```
{
  "status": "invalid",
  "error": "Signature does not match the document"
}
```

Примеры кода и документация

Для более глубокого изучения возможностей модуля trusted-crypto и его применения для автоматической проверки электронной подписи в форматах CAdES-BES, CAdES-T, CAdES-X Long Type 1 и CAdES-A на сервере рекомендуем ознакомиться с документацией и примерами кода. Они помогут вам интегрировать проверку подписей в серверные и микросервисные архитектуры.

Документация на модуль trusted-crypto: trusted-crypto.

Попробуйте бесплатную демонстрацию возможностей КриптоАРМ Сервер и модуля trusted-crypto, загрузив SDK.

Запросите триальную лицензию заполнив форму на сайте cryptoarm.ru.

Если у вас есть вопросы по интеграции модуля trusted-crypto в ваши серверные решения или вы хотите узнать больше о возможностях автоматической проверки электронной подписи, свяжитесь с нами по электронной почте sales@cryptoarm.ru. Мы также готовы предложить услуги по интеграции решений «под ключ».

Автоматическое подписание обезличенным сертификатом

Обезличенная электронная подпись представляет собой подпись, использующую сертификат без указания персональных данных владельца, и предназначена для юридических лиц и индивидуальных предпринимателей (в сертификате для ИП все же указываются данные владельца). Она выполняет функцию, схожую с печатью организации, что позволяет автоматизировать процессы подписания в информационных системах. Сертификат, используемый для такой подписи, не закрепляется за конкретным сотрудником.

Пример кода для автоматического подписания

1. Чтение обезличенного сертификата

Загрузка файла сертификата осуществляется с помощью метода load или асинхронного loadAsync.

```
const certFile = "mycertificate.cer";
const cert = trusted.pki.Certificate.load(DEFAULT_RESOURCES_PATH + "/"
+ certFile, trusted.DataFormat.DER);
```

Этот метод загружает сертификат, который может быть закодирован в формате DER или BASE64. Сертификат должен быть установлен в хранилище сертификатов с привязкой к закрытому ключу, в момент подписи trusted-crypto получит контекст ключа. Альтернативный вариант работы с сертификатами — работа с классом хранилища

trusted.pkistore.PkiStore.

2. Подпись данных или файла

Подпись выполняется методом sign. После подписания данные можно экспортировать в буфер или записать в файл.

```javascript var signer; var policies;

sd = new trusted.cms.SignedData();

sd.policies = ["detached"];
```

```
 sd.content = {
        type: trusted.cms.SignedDataContentType.buffer,
        data: "Hello world"
  };
```

```
 sd.content = {
      type: trusted.cms.SignedDataContentType.buffer,
      data: "Hello world"
 };
```

Для асинхронной подписи можно использовать метод `signAsync`.

```javascript
function (done) {
                cms = new trusted.cms.SignedData();

                cms.policies = isDetached ? ["detached"] : [];

                cms.content = {
                    type: trusted.cms.SignedDataContentType.url,
                    data: DEFAULT_RESOURCES_PATH + "/" + plainFile
                };
                cms.signAsync(cert, done);
            })
```

3. Добавление подписи (соподпись)

```
const csd = new trusted.cms.SignedData();
```

```
 csd.import(signBuffer)

 const cert = trusted.pki.Certificate.load(DEFAULT_RESOURCES_PATH +
"/cert-with-pin.cer", trusted.DataFormat.DER);

csd.sign(cert);
```

4. Усовершенствованная подпись

```
var sdCades;
            var connSettings = new trusted.utils.ConnectionSettings();
            connSettings.Address = TSP_SERVICE_ADDRESS;

            sdCades = new trusted.cms.SignedData();

            sdCades.content = {
                type: trusted.cms.SignedDataContentType.buffer,
                data: "CAdES-" + cadesType + " test 1",
            };

            var cadesParams = new trusted.cms.CadesParams();
            cadesParams.cadesType = cadesTypeByName(
                cadesType,
                "Unknown type " + cadesType
            );
            cadesParams.connSettings = connSettings;
            cadesParams.tspHashAlg = "1.2.643.7.1.1.2.2";
            sdCades.signParams = cadesParams;

            assert.doesNotThrow(function () {
                sdCades.sign(cert);
            });

            assert.notStrictEqual(
                sdCades.export(),
                undefined,
                "Unable to export CAdES"
            );
            sdCades.save(
                DEFAULT_OUT_PATH + "/" + signCades1,
                trusted.DataFormat.PEM
            );
```

Работа с почтой в Node.js. Примеры и возможности КриптоАРМ Сервер

Интеграция электронной почты — одна из важных функций для серверных приложений, будь то отправка уведомлений, получение сообщений или автоматизация рассылки файлов. Если вы разрабатываете серверные приложения на платформе Node.js и вам требуется безопасная работа с почтовыми протоколами IMAP и SMTP, КриптоАРМ Сервер предлагает удобное решение — модуль trusted-mailcore2.

В этой статье мы рассмотрим основные возможности модуля trusted-mailcore2, его уникальные преимущества и продемонстрируем примеры кода, которые помогут вам быстро внедрить его в ваш проект.

Основные возможности trusted-mailcore2

Модуль trusted-mailcore2 создан для решения всех ключевых задач работы с почтой в Node.js. Он построен на основе opensource библиотеки MailCore2, но с важными доработками в части безопасности — нами добавлена поддержка российских криптографических алгоритмов.

Расширения безопасности:

- Криптографическая защита писем: Поддержка подписания и шифрования через модуль trusted-crypto, который обеспечивает защиту ГОСТ алгоритмами.
- Надёжные TLS соединения: Благодаря интеграции с cpcurl от КриптоПро обеспечена поддержка ГОСТ TLS для безопасных соединений с почтовыми серверами.
- Работа с сертифицированным криптопровайдером КриптоПро CSP.

Эти улучшения делают trusted-mailcore2 хорошим решением для проектов, требующих высокого уровня безопасности в работе с электронной почтой.

Примеры использования модуля

trusted-mailcore2 поддерживает все основные почтовые протоколы, что делает его универсальным инструментом для работы с почтовыми серверами. Вы можете работать с

IMAP, SMTP и POP3 в зависимости от потребностей вашего проекта. Классы по работе с письмом (MessageBuilder, MessageHeader, MessageParser) и вложениями (Attachment) позволяют реализовать любые задачи по формированию и чтению писем, например рассылку файлов по адресатам в автоматическом режиме.

1. Отправка электронных писем (SMTP)

С помощью trusted-mailcore2 можно легко отправлять письма через протокол SMTP. Доступны отправка писем с вложениями, использование разных форматов (текстовый или HTML) и настройка параметров отправки.

```
//Создаем объект сессии и зададим параметры подключения
const session = new SMTPAsyncSession();
session.hostname = "smtp-mail.outlook.com";
session.port = 587;
session.username = "test@hotmail.com";
session.password = "testpassword";
session.authType = AuthType.AuthTypeSASLLogin;
session.connectionType = ConnectionType.ConnectionTypeStartTLS;

//Создаем сообщение. Указываем заголовки и формируем тело
const createMessage = (): Buffer => {
    const testHeader = new MessageHeader();
    testHeader.from = "test@hotmail.com";
    testHeader.to = ["test1@hotmail.com", "test2@hotmail.com"];
    testHeader.subject = "Testing SMTP send ";

    const builder = new MessageBuilder();
    builder.header = testHeader;

    builder.htmlBody = "<div>Test message for send by SMTP.</div>";
    builder.textBody = "Test message for send by SMTP.\n-----\n";

    return builder.data;
};

const message = createMessage();

//Вызываем метод отправки со сформированным сообщением
session.sendMessageOperation(
    testConnection.EMAIL,
    [testConnection.MAIL_RECIPIENT],
    message
```

```
    ).then((result) => {
            try {
                    done();
            } catch (e) {
                    done(e);
            }
    }).catch((e) => {
    done(e);
});
```

2. Работа с протоколом IMAP (письма и папки)

trusted-mailcore2 поддерживает работу с протоколом IMAP, что позволяет получать письма с сервера, управлять папками, помечать письма прочитанными и удалять их. В примере вызовем некоторые из доступных методов.

```
const session = new IMAPSession();

session.hostname = "test.outlook.com";
session.port = 993;
session.username = "test@hotmail.com";
session.password = "test";
session.connectionType = ConnectionType.ConnectionTypeStartTLS;
session.checkCertificateEnabled = true;

//Получим все подписанные папки
const folders = session.fetchSubscribedFolders();

//Получим и выведем статус папки INBOX
const folderStatus = session.folderStatus("INBOX");
console.log(folderStatus.unseenCount);
console.log(folderStatus.messageCount);
console.log(folderStatus.recentCount);
console.log(folderStatus.uidNext);
console.log(folderStatus.uidValidity);
console.log(folderStatus.highestModSeqValue);

//Получим сообщения
const messages = sessionLogined.fetchMessagesByUID("INBOX");

//Скопируем сообщение из одной в папки в другую по его UID
const result = sessionLogined.copyMessages("INBOX", "PERSONAL", 1,
100);
```