Ащеулов Алексей Андреевич
Джакишев Амир Маратович
студенты
Научный руководитель: **Владимир Сергеевич Греков**
старший преподаватель кафедры безопасности информационных технологий
РГУ нефти и газа (НИУ) им. И.М. Губкина

ИЗОЛЯЦИЯ И ЗАЩИТА КРИПТОГРАФИЧЕСКОЙ СЛУЖБЫ С ГОСТ-АЛГОРИТМАМИ В КОНТЕЙНЕРНОЙ СРЕДЕ (DOCKER)

В статье исследуется проблема защиты криптографических ключей ГОСТ в эфемерной среде Docker на базе ОС Alt Linux. Предложена архитектура, реализующая концепцию «Data-in-Motion» для нейтрализации угроз остаточной информации (Data Remanence) и «нулевого секрета» (Secret Zero). Решение базируется на интеграции системы HashiCorp Vault с моделью аутентификации AppRole и механизмом временного хранения данных в оперативной памяти (tmpfs). Описана разработка программного агента для автоматизации жизненного цикла ключей: от перехвата до гарантированного уничтожения. Экспериментально подтверждена эффективность защиты и соответствие системы требованиям российского законодательства.

Ключевые слова: Docker, Alt Linux, ГОСТ Р 34.10-2012, HashiCorp Vault, Secret Management, Secret Zero, Data Remanence, tmpfs, AppRole, импортозамещение.

The article explores GOST cryptographic key protection in the ephemeral Docker environment based on Alt Linux OS. An architecture implementing the "Data-in-Motion" concept is proposed to mitigate data remanence and "Secret Zero" threats. The solution is based on the integration of HashiCorp Vault with the AppRole authentication model and a temporary data storage mechanism in RAM (tmpfs). The development of a software agent for automating the key lifecycle — from interception to guaranteed destruction — is described. Experimental results confirm the effectiveness of the protection and compliance with Russian regulatory standards.

Key words:

Docker, Alt Linux, GOST R 34.10-2012, HashiCorp Vault, Secret Management, Secret Zero, Data Remanence, tmpfs, AppRole, import substitution.

Введение

Широкое распространение микросервисной архитектуры существенно повлияло на подходы к обеспечению информационной безопасности корпоративных систем. Контейнерные технологии, в частности Docker и платформы оркестрации Kubernetes/OpenShift, позволили ускорить разработку и развертывание программного обеспечения, однако одновременно усложнили защиту конфиденциальных данных. Основная сложность связана с эфемерной природой контейнеров: сервисы могут существовать ограниченное время, что делает неприменимыми традиционные статические методы управления криптографическими ключами, такие как ручной ввод паролей, привязка к аппаратным идентификаторам или использование физических носителей.

В условиях перехода на отечественное программное обеспечение и обеспечения технологического суверенитета особое значение приобретает интеграция российских криптографических стандартов (ГОСТ Р 34.10-2012, ГОСТ Р 34.11-2012) в контейнерные среды на базе ОС Alt Linux. При этом одной из ключевых проблем автоматизированных инфраструктур остается «Secret Zero» — необходимость передачи первичного секрета приложению без участия пользователя. На практике это часто приводит к небезопасным решениям, таким как хранение учетных данных в Docker-образах или их

передача через переменные окружения, что повышает риск компрометации на этапах сборки и эксплуатации.

Дополнительную угрозу представляет проблема остаточной информации (Data Remanence), обусловленная особенностями работы современных SSD-накопителей и механизмов виртуализации. Из-за алгоритмов выравнивания износа и использования снапшотов стандартные операции удаления файлов не гарантируют полного уничтожения данных. В результате фрагменты закрытых ключей могут сохраняться на физических носителях даже после завершения работы контейнера.

Целью данной работы является разработка и экспериментальная проверка архитектуры защищенного криптографического сервиса в среде Docker под управлением ОС Alt Linux. В исследовании реализуется концепция «Data-in-Motion», при которой чувствительные данные существуют исключительно в оперативной памяти и только в течение минимально необходимого времени, что обеспечивает изоляцию процессов генерации ключей и их гарантированное уничтожение без записи на энергонезависимые носители.

1. Литературный обзор и анализ предметной области

1.1. Эволюция подходов к безопасности в микросервисной архитектуре

Переход к микросервисной архитектуре привёл к размыванию традиционного периметра безопасности, при котором защита строилась вокруг изолированного сервера или сегмента сети. В современных распределённых системах основное внимание смещается на изоляцию отдельных компонентов и контроль взаимодействий между сервисами внутри инфраструктуры (East–West трафик). Контейнерная платформа Docker реализует изоляцию процессов с использованием механизмов ядра Linux, в частности пространств имён (namespaces) и групп управления ресурсами (cgroups).

Для криптографических сервисов особое значение имеет изоляция файловой системы, обеспечиваемая MNT namespace, которая позволяет ограничить доступ процессов к данным и реализовать локальные области хранения. Однако использование ГОСТ-алгоритмов в эфемерных контейнерных средах сопровождается рядом принципиальных ограничений.

Во-первых, возникает проблема «Secret Zero»: отсутствие возможности ручного ввода пароля или применения аппаратных токенов требует наличия безопасного механизма передачи первичного секрета для аутентификации сервиса в системе управления ключами. Во-вторых, архитектура Docker-образов предполагает сохранение данных в слоях файловой системы, вследствие чего любой секрет, записанный на диск на этапе сборки или выполнения, потенциально может быть извлечён даже после его удаления.

Наличие угроз остаточной информации (Data Remanence) делает подобные подходы неприемлемыми для хранения закрытых криптографических ключей. Это обусловливает необходимость перехода к модели «Data-in-Motion», при которой чувствительные данные существуют исключительно в оперативной памяти и только в течение ограниченного времени, необходимого для выполнения криптографической операции.

1.2. Специфика применения российских криптографических стандартов (ГОСТ)

Использование российских криптографических алгоритмов в среде Linux традиционно реализуется с применением либо сертифицированных проприетарных средств, либо открытых программных библиотек. В системах с повышенными требованиями к безопасности обязательным является применение сертифицированных криптопровайдеров, таких как КриптоПро CSP. Однако в контексте контейнеризации и DevOps-подходов их использование сопряжено с рядом практических сложностей.

Ключевыми ограничениями являются необходимость лицензирования каждого узла, сложность автоматизированной активации в эфемерных средах, а также проблемы с использованием аппаратных носителей в контейнерах. В

результате архитектура системы усложняется за счёт дополнительных сервисов, например серверов лицензирования или прокси-компонентов, что снижает надёжность и масштабируемость решения.

Альтернативный подход основан на использовании OpenSSL с поддержкой GOST Engine, который доступен в репозиториях Alt Linux. Данный вариант не требует лицензирования, легко интегрируется в конвейеры CI/CD и позволяет автоматизировать сборку защищённых контейнерных образов. Модульная архитектура OpenSSL и нативная поддержка ГОСТ-алгоритмов в Alt Linux делают данный стек практически применимым для микросервисных систем, что и определило его выбор в рамках настоящего исследования.

1.3. Угрозы хранения и современные методы управления секретами

Проблема остаточной информации (Data Remanence) является одной из наиболее серьёзных угроз в виртуализированных и контейнерных инфраструктурах. Использование твердотельных накопителей с алгоритмами выравнивания износа, а также механизмы снапшотов гипервизоров и систем резервного копирования приводят к тому, что данные могут сохраняться на физических носителях даже после их программного удаления. В таких условиях стандартные средства очистки файловой системы не обеспечивают требуемого уровня безопасности.

Наиболее надёжным способом противодействия данной угрозе является отказ от хранения криптографических ключей на энергонезависимых носителях и реализация концепции «Data-in-Motion», при которой все чувствительные данные размещаются исключительно в оперативной памяти и уничтожаются по завершении операции.

Для централизованного управления ключами и их безопасной доставки в контейнерные приложения применяются специализированные системы управления секретами. Одним из наиболее распространённых решений является HashiCorp Vault, обеспечивающий шифрование данных при хранении и доступ к ним через API на основе строгих политик. Использование метода

аутентификации AppRole, основанного на связке RoleID и SecretID, позволяет реализовать контролируемый процесс загрузки секретов в контейнеры, минимизируя участие администратора и снижая риски компрометации в эфемерных средах.

2. Методы исследования и архитектура решения

В основе исследования лежит метод экспериментального моделирования. Был разработан и развернут стенд, воспроизводящий работу криптографического микросервиса в потенциально недоверенной контейнерной среде. Архитектура решения построена на использовании операционной системы Alt Linux, платформы контейнеризации Docker и системы управления секретами HashiCorp Vault, что позволило оценить практическую реализуемость предложенных механизмов защиты.

2.1. Операционная система Alt Linux как доверенная платформа

В качестве базовой платформы использована операционная система Alt Linux (ветка p11). Выбор обусловлен использованием независимого репозитория «Сизиф», инфраструктура сборки которого локализована на территории Российской Федерации, что снижает риски компрометации цепочек поставок программного обеспечения. Дополнительным преимуществом является наличие штатных механизмов управления криптографическими компонентами.

Настройка криптографической подсистемы реализована с использованием утилиты control, позволяющей централизованно управлять альтернативами системных библиотек. Такой подход исключает необходимость ручного редактирования конфигурационных файлов и снижает вероятность ошибок администрирования, в том числе при активации поддержки ГОСТ-алгоритмов в OpenSSL.

2.2. Архитектура изоляции и защиты памяти

Нейтрализация угрозы остаточной информации достигается за счёт размещения чувствительных данных в файловой системе tmpfs, смонтированной в каталоге /dev/shm. Данная файловая система размещается в

оперативной памяти и не использует энергонезависимые носители, что обеспечивает физическое уничтожение данных при завершении работы контейнера или отключении питания.

Для ограничения потенциальных атак область tmpfs монтируется с флагами noexec и nosuid, что предотвращает исполнение кода и игнорирует биты смены привилегий. Дополнительно задаётся ограничение на максимальный размер раздела, позволяющее снизить риск атак, связанных с исчерпанием ресурсов оперативной памяти. Таким образом, закрытые ключи существуют только в энергозависимой памяти и не сохраняются на дисковых устройствах в каком-либо виде.

2.3. Алгоритм управления секретами с использованием HashiCorp Vault

Доставка и обработка секретов организованы таким образом, чтобы исключить их постоянное хранение внутри контейнера. Первичные идентификаторы аутентификации Vault (RoleID и SecretID) передаются контейнеру при запуске и обрабатываются инициализационным скриптом entrypoint.sh. Сразу после чтения значения переменных окружения переносятся в защищённую область /dev/shm, после чего удаляются из окружения процесса.

Очистка переменных исключает возможность их извлечения через стандартные инструменты анализа контейнеров, включая docker inspect и файловую систему procfs. Дальнейшее взаимодействие с Vault осуществляется через краткоживущий токен, сформированный на основе механизма AppRole, что снижает последствия возможной компрометации контейнера.

2.4. Программный агент мониторинга (Watcher)

Для автоматизации обработки и уничтожения криптографических ключей разработан агент Watcher, реализующий событийную модель управления файлами. Агент использует подсистему inotify ядра Linux и утилиту inotifywait для отслеживания событий в каталоге /dev/shm.

При завершении записи закрытого ключа криптографической утилитой фиксируется событие close_write, после чего агент инициирует передачу зашифрованного файла в HashiCorp Vault по защищённому каналу. После подтверждения успешной записи локальная копия ключа немедленно удаляется. Использование событийного механизма позволяет минимизировать временной интервал, в течение которого ключ существует в незашифрованном виде, и исключает задержки, характерные для опросных (polling) схем.

Компонент	**Функция в системе защиты**	**Технология реализации**
Хост-ОС	Доверенная среда, управление ресурсами	Alt Linux Workstation 11
Контейнеризация	Изоляция процессов и сети	Docker CE 24.x + Namespaces
Криптография	Генерация ключей ГОСТ	OpenSSL + gost-engine
Хранилище секретов	Централизованное управление ключами	HashiCorp Vault (KV v2)
Изоляция данных	Защита от Data Remanence	tmpfs (/dev/shm)
Агент Watcher	Автоматическое уничтожение ключей	Bash + inotify-tools

Таблица 1 – Технологический стек и функции компонентов защищенного сервиса

3. Практическая реализация и результаты исследования

В данном разделе детально описывается процесс развертывания стенда, конфигурации компонентов и анализа результатов экспериментов. Все работы

проводились в изолированной виртуальной среде, исключающей влияние внешних сетевых факторов.

3.1. Характеристика экспериментального стенда

Стенд был развернут на аппаратной платформе с процессором архитектуры x86_64.

Программная конфигурация:

ОС: ALT Workstation 11.1 (ядро 6.1.x).

Docker: версия 28.3.3.

OpenSSL: с поддержкой GOST Engine

```
[root@AltLinux ~]# cat /etc/os-release
NAME="ALT Workstation"
VERSION="11.1"
ID=altlinux
VERSION_ID=11.1
PRETTY_NAME="ALT Workstation 11.1 (Prometheus)"
ANSI_COLOR="1;33"
CPE_NAME="cpe:/o:alt:workstation:11.1"
BUILD_ID="ALT Workstation 11.1"
ALT_BRANCH_ID="p11"
HOME_URL="https://basealt.ru/"
BUG_REPORT_URL="https://bugs.altlinux.org/"
DOCUMENTATION_URL="https://docs.altlinux.org/"
SUPPORT_URL="https://support.basealt.ru/"
LOGO=alt-distro-logo
```

```
[root@AltLinux ~]# docker version
Client:
 Version:           28.3.3
 API version:       1.51
 Go version:        go1.24.6
 Git commit:        980b856
 Built:             Thu Jul 31 09:36:02 2025
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          28.3.3
  API version:      1.51 (minimum version 1.24)
  Go version:       go1.24.6
```

Рисунок 1 — Идентификация версии операционной системы и среды контейнеризации

Для обеспечения сетевой безопасности была создана пользовательская сеть Docker gost-net типа bridge.

```
[root@AltLinux ~]# docker network inspect gost-net
[
    {
        "Name": "gost-net",
        "Id": "c3b2212ae69c5fbffea57cfedba440ce24a09c04652fc5ac8498f46124e32bfa",
        "Created": "2025-12-26T03:44:23.624477407+03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16"
                }
            ]
        },
        "Internal": true,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
```

Рисунок 2 — Параметры сетевой конфигурации изолированного сегмента gost-net

Флаг --internal критически важен: он запрещает контейнерам в этой сети доступ во внешнюю сеть Интернет, предотвращая эксфильтрацию (выгрузку) украденных ключей на серверы злоумышленников (C&C servers). Контейнеры могут общаться только друг с другом и с сервером Vault внутри периметра.

3.2. Настройка системы управления секретами HashiCorp Vault

Инициализация Vault производилась с использованием движка секретов Key-Value версии 2 (KV v2), который поддерживает версионирование секретов. Это позволяет восстановить предыдущую версию ключа в случае ошибочной перезаписи, повышая отказоустойчивость системы.

Политика безопасности (ACL):

Была разработана политика gost-policy.hcl, реализующая принцип Write-Only (Только запись) для генератора ключей.

```
path "secret/data/keys/*" {
  capabilities = ["create", "update"]
}
```

Рисунок 3 — Конфигурация политики доступа (ACL) с ограничением прав на чтение

Данная конфигурация является критическим элементом защиты. Она гарантирует, что даже если злоумышленник захватит контроль над контейнером генерации ключей и получит доступ к токену Vault, он сможет только *создавать* новые ключи, но не сможет *прочитать* уже существующие в базе. Это эффективно предотвращает утечку базы данных ключей.[1]

Настройка AppRole:

Для аутентификации была создана роль gost-role со следующими параметрами безопасности:

```
Key                     Value
---                     -----
bind_secret_id          true
local_secret_ids        false
secret_id_bound_cidrs   <nil>
secret_id_num_uses      0
secret_id_ttl           10m
token_bound_cidrs       []
token_explicit_max_ttl  0s
token_max_ttl           30m
token_no_default_policy false
token_num_uses          0
token_period            0s
token_policies          [gost-policy]
token_ttl               20m
token_type              default
```

Рисунок 4 — Команды конфигурации и параметры ролевой модели доступа в Vault

Полученные RoleID (статический) и SecretID (динамический) использовались для запуска клиентского контейнера.

3.3. Реализация защищенного контейнера на базе Alt Linux

Сборка образа производилась на основе официального базового образа registry.altlinux.org/alt/alt:p11

```
FROM registry.altlinux.org/alt/alt:p11

RUN apt-get update && apt-get install -y openssl openssl-engines openssl-gost-engine curl jq inotify-tools
 systemd-sysvinit && apt-get clean

RUN control openssl-gost enabled

WORKDIR /usr/local/bin

COPY entrypoint.sh .
COPY watcher.sh .
RUN chmod +x entrypoint.sh watcher.sh
COPY vault-init.service /etc/systemd/system/
COPY vault-watcher.service /etc/systemd/system/

RUN systemctl enable vault-init.service vault-watcher.service

CMD ["/sbin/init"]
```

Рисунок 5 — Спецификация сборки контейнера (Dockerfile) с активацией ГОСТ

Особенностью данной реализации является использование systemd внутри контейнера (/sbin/init как PID 1). Хотя философия Docker ("один процесс на контейнер") обычно рекомендует избегать этого, в задачах

безопасности использование полноценного init-процесса оправдано. Systemd позволяет корректно управлять зависимостями служб (Watcher должен стартовать строго после инициализации секретов), автоматически перезапускать упавшие процессы и, самое главное, корректно обрабатывать сигналы завершения (SIGTERM) для очистки памяти перед остановкой контейнера.

Скрипт Entrypoint (Миграция секретов):

Выполняет критическую функцию скрытия секретов.

```
#!/bin/bash
echo "export ROLE_ID='$ROLE_ID'" > /dev/shm/.vault_creds
echo "export SECRET_ID='$SECRET_ID'" >> /dev/shm/.vault_creds
echo "export VAULT_ADDR='$VAULT_ADDR'" >> /dev/shm/.vault_creds
chmod 600 /dev/shm/.vault_creds
```

Рисунок 6 — Реализация алгоритма безопасной миграции секретов в скрипте entrypoint.sh

Перенос секретов в /dev/shm (который смонтирован в RAM) и установка прав 600 гарантирует, что прочитать их сможет только процесс с правами root внутри контейнера.

```bash
#!/bin/bash

WATCH_DIR="/dev/shm"
CRED_FILE="/dev/shm/.vault_creds"
PAYLOAD_FILE="/tmp/vault_payload.json"

echo "[Watcher] Запуск демона. Мониторинг каталога $WATCH_DIR..."
while [ ! -f "$CRED_FILE" ]; do
  echo "[Watcher] Ожидание инициализации секретов..."
  sleep 2
done
source "$CRED_FILE"
inotifywait -m -e close_write --format '%f' "$WATCH_DIR" | while read FILENAME; do

    if [[ "$FILENAME" == .* ]]; then continue; fi
    FILEPATH="$WATCH_DIR/$FILENAME"
    echo "[EVENT] Обнаружен файл ключа: $FILENAME"

    VAULT_TOKEN=$(curl -s --fail --request POST \
        --data "{\"role_id\":\"$ROLE_ID\",\"secret_id\":\"$SECRET_ID\"}" \
        "$VAULT_ADDR/v1/auth/approle/login" | jq -r .auth.client_token)

    if [ -z "$VAULT_TOKEN" ] || [ "$VAULT_TOKEN" == "null" ]; then
        echo "[ERROR] Не удалось получить токен. Проверьте VAULT_ADDR ($VAULT_ADDR) и ID."
        continue
    fi

    jq -n --arg k "$(cat "$FILEPATH")" --arg n "$FILENAME" \
       '{"data": {"key_name": $n, "content": $k}}' > "$PAYLOAD_FILE"

    HTTP_CODE=$(curl -s -o /dev/null -w "%{http_code}" \
        --header "X-Vault-Token: $VAULT_TOKEN" \
        --header "Content-Type: application/json" \
        --request POST \
        --data @"$PAYLOAD_FILE" \
        "$VAULT_ADDR/v1/secret/data/keys/$FILENAME")

    if [ "$HTTP_CODE" == "200" ]; then
        echo "[SUCCESS] Ключ успешно передан в Vault."
        rm -f "$FILEPATH"
        echo "[SECURE] Локальная копия ключа уничтожена."
    else
        echo "[ERROR] Ошибка загрузки. HTTP код: $HTTP_CODE"
    fi
    rm -f "$PAYLOAD_FILE"
done
```

Рисунок 7 — Алгоритм перехвата ключей в watcher.sh

Скрипт watcher.sh реализует логику "наблюдения и уничтожения".

Использование inotifywait позволяет реагировать мгновенно. В отличие от поллинга (проверки по таймеру раз в N секунд), событийная модель исключает ситуации, когда файл лежит на диске несколько секунд в ожидании цикла проверки. Время реакции определяется скоростью планировщика ядра Linux и составляет микросекунды.

3.4. Экспериментальная проверка и анализ результатов

Для проверки спроектированной системы была проведена серия тестов, имитирующих цикл работы криптографического сервиса и проверку механизмов защиты от утечек данных.

Тест 1: Проверка готовности криптографического движка. На первом этапе проверяется корректность активации поддержки российских алгоритмов внутри изолированного контейнера. Выполнение команды openssl engine -t позволяет подтвердить, что библиотека OpenSSL успешно загрузила динамический движок gost и он готов к выполнению операций .

```
[root@AltLinux gost-project]# docker exec -it gost-client bash
[root@4bff5ee08018 bin]# openssl engine -t
(rdrand) Intel RDRAND engine
     [ available ]
(dynamic) Dynamic engine loading support
     [ unavailable ]
(gost) Reference implementation of GOST engine
     [ available ]
```

Рисунок 8 — Верификация доступности и статуса криптографического движка GOST Engine

Тест 2: Генерация ключевой пары и работа агента мониторинга. В ходе теста инициируется создание закрытого ключа по стандарту ГОСТ Р 34.10-2012 с параметром paramset:A, что соответствует требованиям для формирования квалифицированной электронной подписи . Агент Watcher, используя событийную модель ядра, должен мгновенно зафиксировать появление файла в оперативной памяти (tmpfs), передать его в Vault и уничтожить локальную копию.

```
Jan 08 16:17:35 4bff5ee08018 watcher.sh[30438]: [EVENT] Обнаружен файл ключ
a: gost_key.pem
Jan 08 16:17:35 4bff5ee08018 watcher.sh[30438]: [SUCCESS] Ключ успешно пере
дан в Vault.
Jan 08 16:17:35 4bff5ee08018 watcher.sh[30438]: [SECURE] Локальная копия кл
юча уничтожена.
```

```
[root@4bff5ee08018 bin]# openssl genpkey -algorithm gost2012_256 -pkeyopt
paramset:A -out /dev/shm/gost_key.pem
[root@4bff5ee08018 bin]# ls /dev/shm
[root@4bff5ee08018 bin]#
```

Рисунок 9 — Журнал событий Watcher: фиксация обнаружения, передачи и удаления ключа

Тест 3: Контроль отсутствия остаточной информации и проверка в хранилище. Завершающий этап подтверждает реализацию концепции Data-in-

Motion. Проверяется фактическое отсутствие секретов на файловой системе сервиса и их наличие в защищенном хранилище Vault .

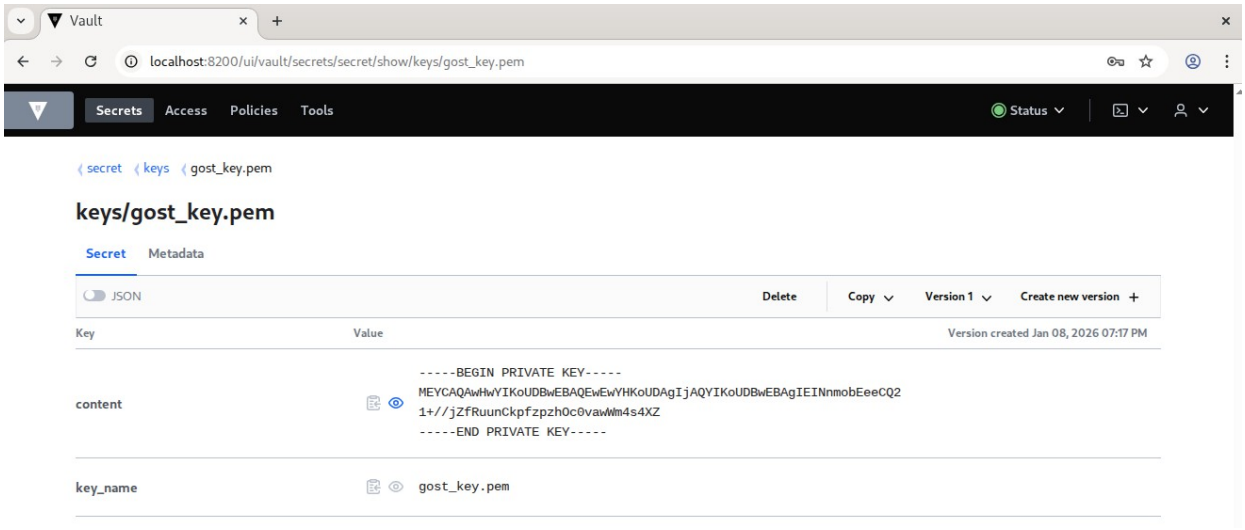

Рисунок 10 —Сохраненный ГОСТ-ключ в веб-интерфейсе HashiCorp Vault

Заключение

В ходе выполнения данной работы было проведено исследование особенностей обеспечения безопасности криптографических сервисов в контейнерных средах. Разработанная архитектура и результаты экспериментальной проверки подтвердили возможность построения защищённого микросервисного решения с использованием отечественного программного обеспечения и российских криптографических стандартов.

В рамках исследования показано, что использование операционной системы Alt Linux и её штатных механизмов управления криптографическими компонентами позволяет сформировать доверенную программную платформу без привлечения проприетарных и трудно масштабируемых средств. Применение OpenSSL с поддержкой ГОСТ-алгоритмов обеспечивает необходимую функциональность и совместимость с современными DevOps-подходами.

Рассмотренная архитектура устраняет проблему «Secret Zero» за счёт использования модели аутентификации AppRole системы HashiCorp Vault, что позволяет автоматизировать процесс доставки первичных секретов без их

жёсткого закрепления в коде или образах контейнеров. Это снижает риск компрометации и повышает общий уровень безопасности системы.

Экспериментально подтверждена эффективность реализации концепции «Data-in-Motion». Размещение закрытых ключей исключительно в оперативной памяти с применением файловой системы tmpfs и автоматического агента мониторинга обеспечивает их существование в незашифрованном виде только в течение минимально необходимого времени. Такой подход позволяет полностью исключить угрозу остаточной информации на физических носителях.

Предложенное решение не требует ручного вмешательства администратора в процессы генерации, хранения и уничтожения ключей, что снижает влияние человеческого фактора и делает архитектуру пригодной для масштабирования в распределённых контейнерных средах. В качестве дальнейшего направления развития рассматривается адаптация разработанных механизмов для использования в средах оркестрации Kubernetes, включая реализацию агента мониторинга в виде sidecar-контейнера.

Список использованных источников

1. ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. — М.: Стандартинформ, 2012. — 24 с.

2. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хэширования. — М.: Стандартинформ, 2012. — 20 с.

3. Об электронной подписи: Федер. закон от 06.04.2011 № 63-ФЗ (последняя редакция). — Текст: электронный // КонсультантПлюс: [сайт].

4. Определение отечественной технологической платформы в рамках создания киберполигона: «Сетевое и системное администрирование» / А. Г. Уймин // Текущие вызовы в подготовке кадров. Обучение специалистов по современным направлениям информационных технологий,

кибербезопасности и ИКТ-электроники, актуальным для экономики данных: сборник научных трудов. — Тверь, 2024. — С. 122–123.

5. Утилита управления альтернативами control в ОС Alt Linux. — Текст: электронный // ALT Linux Wiki: [сайт]. — URL: https://www.altlinux.org/Control (дата обращения: 26.12.2025).

6. HashiCorp Vault Documentation. AppRole Auth Method. — Текст: электронный // HashiCorp Developer: [сайт]. — URL: https://developer.hashicorp.com/vault/docs/auth/approle (дата обращения: 26.12.2025).

7. Docker Documentation. Runtime execution resources (Namespaces and Cgroups). — Текст: электронный // Docker Docs: [сайт]. — URL: https://docs.docker.com/engine/security/ (дата обращения: 26.12.2025).

8. Inotify: эффективный механизм мониторинга событий файловой системы в ядре Linux. — Текст: электронный // Linux Kernel Documentation. — URL: https://docs.kernel.org/admin-guide/index.html (дата обращения: 26.12.2025).

9. Gost-engine: OpenSSL extension implementing Russian GOST crypto algorithms. — Текст: электронный // GitHub Repository. — URL: https://github.com/gost-engine/engine (дата обращения: 26.12.2025).