**Лабораторная работа №2**

**1. Общее описание алгоритма KDF\_TREE\_GOSTR3411\_2012\_256**

KDF\_TREE\_GOSTR3411\_2012\_256 — это алгоритм генерации производных ключей из одного исходного (мастер-) ключа, описанный в стандарте ГОСТ Р 50.1.113–2016. Он используется для криптографической диверсификации ключей с применением отечественной хеш-функции HMAC на базе Стрибог-256 (ГОСТ Р 34.11-2012).

**Основные особенности:**

* **Поддержка различных параметров:**
  + Исходный ключ может иметь любую длину, но используется только первые 32 байта.
  + Позволяет задавать произвольную длину выходного ключа L.
  + Число раундов (итераций) R регулирует объём выходных данных.
* **Логика работы:**
  + Если мастер-ключ короче 32 байт, работа прекращается с ошибкой.
  + Если требуется ключ длиной больше 32 байт, используется итерационный подход:  
    каждый раунд генерирует по 32 байта и добавляет их в выходной буфер.
* **Структура входных данных для HMAC:**
  + Порядковый номер итерации (4 байта)
  + Метка (label) — произвольная строка
  + Разделитель 0x00
  + Параметр seed (если задан)
  + Длина выходного ключа L (4 байта)
* **Пример генерации:**  
  При R = 2, L = 64 алгоритм выполнит две итерации, каждую из которых сгенерирует 32 байта, формируя итоговый ключ длиной 64 байта.

**2. Криптографические характеристики и структура алгоритма**

**2.1 Свойства стойкости:**

* **Безопасность мастер-ключа:** HMAC делает невозможным обратное восстановление оригинального ключа даже при наличии всех производных ключей.
* **Устойчивость к коллизиям:** Благодаря свойствам Стрибога, риск совпадения хешей у различных входных данных минимален.
* **Уникальность ключей:** Использование label и seed исключает дублирование выходных данных.
* **Повторяемость:** Результат зависит только от входных параметров, что важно для идентичного поведения в разных системах.
* **Масштабируемость:** Для длинных ключей просто увеличивается число итераций R.

**2.2 Принципы построения:**

* Алгоритм следует древовидной логике, где каждая итерация расширяет или развивает производный ключ.
* Использование HMAC с последовательным вводом данных обеспечивает надежность.
* Все промежуточные значения используются только один раз и не сохраняются, что снижает риски.

**2.3 Возможные ограничения:**

* **Полная зависимость от HMAC:** Нарушение стойкости Стрибога повлияет и на KDF.
* **Рост времени при больших L и R:** Увеличение параметров увеличивает вычислительные затраты.
* **Ограничение на размер входного ключа:** Менее 32 байт — недопустимо.

**2.4 Криптостойкость:**

Для мастер-ключа длиной 256 бит подбор по полному перебору требует 2²⁵⁶ операций, что на текущем уровне технологий невозможно.

**3. Анализ вычислительной сложности**

**3.1 Временная сложность:**

* Зависит линейно от количества итераций R и длины выходного ключа L.
* Пример:
  + R=1, L=32 → до 1.16 млн ключей/сек
  + R=2, L=64 → ~580 тыс ключей/сек

**3.2 Память:**

* Не используется динамическое выделение памяти.
* Все буферы фиксированы:
  + 32 байта на ключ
  + Временный буфер hash\_input (макс. ~300 байт)
* Пиковое потребление: ~1.5 КБ

**3.3 Рекомендации:**

* Для коротких ключей: L ≤ 32, R = 1
* Для длинных — предпочтительнее увеличивать R, чем L
* Не использовать избыточные label или seed в ресурсоограниченных системах

**4. Описание реализации**

**4.1 Структура проекта:**

* kdf\_tree.c — основная логика KDF
* kdf\_tree.h — заголовок с API
* main.c — демонстрация и тест производительности

**4.2 Ключевые функции:**

* KDF\_TREE\_GOSTR3411\_2012\_256(...) — функция генерации
* generate\_random\_bytes(...) — генерация входных данных
* print\_hex(...) — удобный вывод результата

**4.3 Особенности:**

* Использование libgcrypt для реализации HMAC(Streebog-256)
* Минимизация памяти
* Контроль ошибок (проверка длины ключа и корректности параметров)
* Строгая структура формирования входного буфера

**4.4 Пример:**

Мастер-ключ: c9 7a ... af  
Параметры: label = "test", seed = "abc", R = 2, L = 64  
Выход: 64 байта производного ключа

**4.5 Ограничения:**

* Требуется libgcrypt
* Без поддержки многопоточности
* Не оптимизирован для встроенных систем

**5. Результаты тестирования**

**5.1 Условия:**

* **ОС:** Ubuntu 22.04.4 LTS (WSL2 на Windows 11)
* **Архитектура:** x86\_64
* **Процессор:** Intel Core i5-12400H
* **ОЗУ:** 32 ГБ
* **Компилятор:** g++ 11.4

**5.2 Результаты:**

| **Кол-во ключей** | **Время (сек)** | **Скорость (ключ/сек)** |
| --- | --- | --- |
| 10 000 | 0.01 | 998 000 |
| 100 000 | 0.09 | 1 110 000 |
| 1 000 000 | 0.87 | 1 149 000 |

**5.3 Наблюдения:**

* Время прямо пропорционально количеству ключей
* Скорость увеличивается за счёт оптимизации кэширования
* Алгоритм способен достигать ~1.15 млн ключей/сек

**5.4 Влияние параметров:**

* seed снижает производительность на ~5%
* R = 2 снижает скорость примерно вдвое

**6. Тестовый стенд**

* **ОС:** Ubuntu 22.04.4 LTS (WSL2 на Windows 11)
* **Архитектура:** x86\_64
* **Процессор:** Intel Core i5-12400H
* **ОЗУ:** 32 ГБ
* **Компилятор:** g++ 11.4

**7. Мониторинг ресурсов (ЦП и ОЗУ)**

**Итоговые метрики:**

| **Показатель** | **Значение** |
| --- | --- |
| Общее время исполнения | 1.054 сек |
| Время ЦП | 1.053 сек |
| Утилизация ЦП | 99.99% |
| Использование RAM (max) | 1348 КБ |
| Утечки памяти | Не обнаружены |

**Вывод:**

* Полная загрузка CPU говорит о высокой эффективности вычислений
* Память остаётся стабильной во всех режимах
* Отсутствие аллокаций на куче повышает безопасность и стабильность
* Алгоритм подходит для использования в непрерывных процессах и системах с ограниченными ресурсами