

Министерство науки и высшего образования Российской Федерации
**Федеральное государственное бюджетное образовательное учреждение
высшего образования**

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ»**

Кафедра телекоммуникаций и основ радиотехники (ТОР)

Горелкин Богдан Константинович

**АНАЛИЗ УЯЗВИМОСТЕЙ КРИПТОАЛГОРИТМА RSA ПРИ ЕГО
АППАРАТНОЙ РЕАЛИЗАЦИИ**

Магистерская программа

11.04.02 – «Инфокоммуникационные системы беспроводного широкополосного
доступа»

Диссертация на соискание академической степени магистра

Научный Руководитель:

Кандидат технических наук

Доцент Е.В. Рогожников

Консультант:

Sébastien Pillement

Professor of Embedded Real Time Systems

Research at the IETR laboratory

Томск 2020

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ»

Кафедра телекоммуникаций и основ радиотехники (ТОР)

УТВЕРЖДАЮ

Зав. кафедрой ТОР

_____ К.Ю. Попова

«___» _____ 2020 г.

ЗАДАНИЕ НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Тема: Анализ уязвимостей криптоалгоритма RSA при его аппаратной реализации.

2. Цель и исходные данные: изучение атаки с использованием данных побочного температурного канала на криптографический алгоритм RSA

2.1 Объект исследования: алгоритм RSA реализованный на микроконтроллере STM32F070Rb.

2.2 Источники информации:

1) Michael Hutter and Jorn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks

2) Rivest R. L., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems // Commun ACM. 1978. P. 120–126. DOI:10.1145/359340.359342

3) Zhou Y., Feng D. G.. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing // Information Security Seminar WS 0607. 2006

4) Coutinho, S.C. The Mathematics of Ciphers: Number Theory and RSA Cryptography / A K Peters/CRC Press; 1 edition (January 15, 1999)

3. Основные вопросы, подлежащие разработке:

- Литературный обзор угроз безопасности передаваемых данных;
- Исследование атак по сторонним каналам;
- Математическая модель и принцип работы алгоритма RSA;
- Реализация экспериментальной установки на STM32;
- Анализ полученных результатов.

4. Срок сдачи диссертации для определения готовности к защите –

8 июля 2020 г.

5. Руководитель канд. техн. наук,

доцент каф. TOP

(уч.степень, звание)

Рогожников Е.В.

(подпись) (ФИО)

6. Консультант ст. преп.

Professor of Embedded Real Time Systems

Research at the IETR laboratory

(уч.степень, звание)

Sébastien Pillement

(подпись) (ФИО)

8. Студент гр. 168-М



Горелкин Б.К.
(подпись) (ФИО)

Дата составления задания " 30 " мая 2020 г.

Реферат

Диссертационная работа 99 страниц, 63 рисунка, 7 таблиц, 47 источников 4 приложения.

RSA, STM32F070RB, STM32, SIDE CHANNEL ATTACK, TEMPERATURE ATTACK, КРИПТОГРАФИЯ, АССИМЕТРИЧНЫЙ АЛОГОРИТМ. ШИФРОВАНИЕ

В данной работе представлены результаты аналитического обзора литературы посвященной спутниковой навигации и разработкам в данной сфере. Были рассмотрены несколько наиболее актуальных на текущий момент направлений исследований.

Цель работы – анализ полученных данных по побочному каналу от встроенного температурного сенсора в процессе выполнения дешифрования на микроконтроллере STM32F070Rb.

В работе были рассмотрены уязвимости криптоалгоритма RSA, выполнена его реализация на микроконтроллере, совершены атаки по сторонним каналам.

Разработан алгоритм определения ложного GPS сигнала

Пояснительная записка к данной работе выполнена в текстовом редакторе Microsoft Word 2013 и оформлена согласно ОС ТУСУР 01-2013.

Оглавление

Реферат	4
ВВЕДЕНИЕ	6
1. Безопасность и конфиденциальность передаваемых/собираемых данных.....	12
2. Криптосистема с открытым ключом RSA	16
3. Атака по сторонним каналам.....	20
4. Виды атак по сторонним каналам	23
5. Атаки по данным со стороны температурного канала.....	28
6. Получение экспериментальных данных	31
6.1 Вычисление ключей шифрования на ПК.	33
6.2 Получение зашифрованного сообщения.	37
6.3 Получение данных температуры в процессе дешифровки.....	39
7. Анализ полученных данных.	41
Заключение	67
Список использованных источников	69
Приложение А.....	74
Приложение Б.....	77
Приложение В	79
Приложение Г	98

ВВЕДЕНИЕ

С каждым днем в мире становится все больше и больше устройств с возможностью выхода в интернет, а следовательно, по закону Меткалфа (полезность любой системы равняется квадрату элементов этой системы) «интернет вещей» с каждым днем становится перспективнее. Интернет вещей не просто связывает миллиарды устройств в одну сеть, как когда-то Интернет объединил все компьютеры. Реальная инновация и потенциал Интернета вещей в том, чтобы трансформировать бизнес-модели, позволять компаниям продавать продукты, по-новому принося дополнительную пользу как компании, так и клиенту.

В настоящее время Интернет вещей (IoT – Internet of Things) сильно изменяется в зависимости от многих жизненных факторов. На данные изменения может влиять, например, то, как мы передвигаемся, как мы принимаем решения, и даже как мы получаем энергию. Интернет вещей состоит из сложных датчиков, исполнительных механизмов и микросхем, встроенных в окружающие нас физические объекты, что делает их умнее, чем когда-либо. Все эти объекты соединены вместе и обмениваются большим количеством данных с другими цифровыми компонентами без какого-либо вмешательства человека [1].

Тем самым, технология Интернета вещей позволяет значительно облегчить функцию людей, повысить комфортабельность использования и снизить затраты на персонал. Технология интернета вещей может быть применена в различных областях, некоторые из которых проиллюстрированы на рисунке 1.1.



Рисунок 1.1 – Сферы применения IoT

Примеры использования IoT приведены ниже согласно порядку на рисунке, представленном выше.

1. Использование интернета вещей для **умных городов** подразумевает создание автоматизированных парковок, управления городским транспортом, контроля уровня освещенности, обеспечение эффективного обращения с отходами.
2. В сфере **агрокультуры** применение IoT обусловлено созданием встроенного мониторинга параметров окружающей среды, в экологических целях – для осуществления биоиндексации, например, через отслеживание поведения пчел, сохранения лесов, посредством предотвращения незаконной вырубки, контролем за переработкой отходов. Также технология делает возможным автоматическое управление запасами на производствах, автоматизацию процессов орошения, сбора урожая, контроля условий хранения, утилизации отходов и обработки урожая в целом.

3. На **производстве** технология Интернета вещей способна обеспечить встроенный автоматический мониторинг, управление жизненными циклами продукции, оптимизацию технических процессов, контроль качества продукции для предотвращения дефектов и брака. Также, IoT позволит усилить безопасность работников, посредством контроля за показателями состояния среды и предупреждения в случае аномальных значений показателей.
4. Применение технологии в сфере **энергетических сетей** заключается в необходимости роботизированного учета и мониторинга станций, процессов и производства, а также упрощает управление интеллектуальными сетями.
5. В сфере **умных домов** Internet of Things может использоваться в качестве элементов контроля за счетчиками водо- или электроснабжения, сохранение данных с датчиков и счетчиков для анализа данных и прогнозирования, автоматическое поддержание температуры, дистанционное управление светом в помещении, управление бытовыми приборами и приборами личного пользования. Также возможно осуществление удаленного контроля, например, с помощью мобильного устройства, над доступом к объектам, посредством регуляции состояния ворот, светового оснащения, для обеспечения пожарной или охранной сигнализации.
6. В **транспортной сфере** применение Интернета вещей может представлять собой автоматизацию управления автопарком, поддержка в просчётах логистических операций, технологии для автоматического отслеживания грузов и др.
7. Использование технологии Internet of Things для сферы **окружающей среды** позволит осуществлять удаленный

мониторинг состояния воздуха, прогнозирования состояния водоёмов, почвы и окружающей среды в целом. Датчики также позволят отслеживать степень загрязнённости, концентрацию вредных химических веществ, измерять такие параметры, как уровень радиоактивного излучения и шумового фона и другие параметры, в зависимости от необходимости и целей мониторинга.

8. Применение настоящей технологии также возможно в сфере **личных устройств**. Встраивание технологии в умные гаджеты позволит обеспечить автоматизированный мониторинг показателей здоровья, осуществить контроль местоположения, использовать тревожные кнопки и др.

С каждым днем количество собираемых и передаваемых данных увеличивается, и количество сфер, использующих технологии IoT, растет. В связи с этим, аспект безопасности данных становится актуальным и еще более важным. При аппаратной реализации шифрующих устройств, важно рассмотреть всевозможные уязвимости, а также найти способы предотвращения возможных атак на криптографические системы. Кроме основной информации от шифрующего или дешифрующего устройства, можно также извлечь и дополнительную информацию, полученную по сторонним каналам.

Актуальность работы:

Важным аспектом при аппаратной реализации криптографических устройств является защита от утечек информации по сторонним каналам. Тема атак на криптосистемы по данным температурного канала, на данный момент, недостаточно исследована, и следовательно, утечка по данному стороннему каналу может угрожать безопасности передаваемой информации. Из этого вытекает необходимость подробного изучения возможностей получения полезных данных по температуре криптографического устройства. В настоящее время защита данных от атак по побочным каналам является приоритетным

направлением исследований в области защиты информации от вмешательства в работу криптографической системы с попытками выкрасть данные.

Цели и задачи работы:

Целью данной работы является анализ полученных данных по побочному каналу от встроенного температурного сенсора в процессе выполнения дешифрования на микроконтроллере STM32F070Rb. Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Рассмотреть угрозы безопасности передаваемых данных.
2. Разобрать принцип работы алгоритма RSA.
3. Исследовать атаки по сторонним каналам.
4. Реализовать экспериментальную установку для получения данных со встроенного температурного датчика процессора STM32.
5. Проанализировать полученные результаты.

Научная новизна работы. Ключ криптографического алгоритма может быть вычислен при помощи данных, собранных по сторонним каналам. В данной работе получены новые экспериментальные данные температуры процессора при дешифровании сообщений микроконтроллером STM32F070Rb, Ранее извлечение полезных данных по температурному каналу производилось только при помощи дополнительного оборудования. В данной работе используется встроенный температурный сенсор микроконтроллера STM32F070Rb.

Практическая значимость полученных результатов. Результаты диссертационной работы могут повлиять на выбор процессоров, используемых при создании устройств в индустрии «интернета вещей».

Апробация работы. По результатам моделирования работы была опубликована статья на Международную научно-техническую конференцию «Научная сессия ТУСУР», г. Томск, 2020.

Публикации работы: По материалу диссертационной работы была публикация статьи в научном журнале «Доклады ТУСУР».

Личный вклад: Основные результаты диссертации получены лично автором. Совместно с научным руководителем Рогожниковым Евгением Васильевичем обсуждалась методология работы и выполнены некоторые этапы анализа данных. Моделирование и обработка данных проведены непосредственно автором данной работы в рамках работы в лаборатории на базе исследовательского центра Institut d'Électronique et des Technologies du numéRique по учебной специализации “Wireless Embedded Technologies” в рамках программы двойного диплома ТУСУР – Политех Нанта.

1 Безопасность и конфиденциальность передаваемых/собираемых данных

IoT вносит значительный вклад в улучшение нашей повседневной жизни, что проявляется в существовании умных приложений для различных отраслей, таких как smart города, умное строительство, здравоохранение, умные сети, промышленное производство и многое другое. В настоящее время одной из проблем, потенциально угрожающих устройствам Internet of Things, является безопасность и конфиденциальность передаваемых или собираемых данных, которые часто напрямую связаны с частной жизнью людей или компаний.

На данный момент многие организации и предприятия находятся в стадии внедрения решений по обеспечению безопасности для защиты их устройств IoT [2]. В настоящее время проблемы безопасности в IoT являются зачастую более сложными, чем существующие проблемы безопасности в привычном нам интернете. Это происходит ввиду того, что устройства IoT обычно достаточно ограничены в ресурсах вычислительной мощности, памяти и энергии. Эти ограничения означают, что многие существующие решения в обеспечении безопасности и конфиденциальности передаваемых данных абсолютно неприменимы. В целом, можно заключить, что безопасность систем состоит из[3]:

- Конфиденциальность (информация становится неразборчивой при несанкционированном доступе);
- Целостность (данные не должны быть изменены третьим лицом ни случайно, ни намеренно);
- Аутентификация (источник данных не должен являться притворным, необходима идентификация устройства);
- Неотрицание (отправитель сообщения не может отрицать, что отправил сообщение в будущем);
- Доступность (услуги системы должны быть доступны для законных пользователей);

- Приватность (личности пользователей не должны быть идентифицируемыми или отслеживаемыми по их поведению и выполняемым действиям в системе).

Более подробная информация о службах безопасности и механизмы для борьбы с различными видами угроз представлена в таблице 1.1.

Таблица 1.1 – Механизмы борьбы с угрозами

Цель кибератаки	Механизм безопасности	Пример
Конфиденциальность	Шифрование сообщений и знаков	<ul style="list-style-type: none"> • Симметричные криптографические механизмы (AES, CBC); • Ассиметричные механизмы (RSA, DSA, IBE, ABE и др.).
Целостность	Хеш-функции, подпись сообщения	<ul style="list-style-type: none"> • Хеш-функции (SHA-256, MD5 и др.); • Коды аутентификации сообщений (HMAC).
Аутентификация	Цепочка хэшей, код аутентификации сообщений	<ul style="list-style-type: none"> • HMAC; • CBC-MAC; • ECDSA.
Неотрицание	Подпись сообщения	<ul style="list-style-type: none"> • ECDSA; • HMAC.
Доступность	Псевдослучайное скачкообразное изменение частоты, контроль доступа, вторжение профилактические системы, межсетевые экраны	Обнаружение вторжений на основе сигнатур, на основе статистических аномалий обнаружения вторжений
Приватность	Псевдонимность, отсутствие связи, k-анонимность, Zero Knowledge Proof (ZKP)	EPID, DAA, обязательства Педерсена (Pedersen Commitment Scheme)

Кибератаки в системах IoT создают опасения и сдерживают развитие интернета вещей, поскольку они могут способствовать краже конфиденциальных данных, причинить физический ущерб и даже угрожать жизни людей.

Криптография в основном обеспечивает конфиденциальность данных. Реализация криптографических алгоритмов основывается на сложности вычислений, при отсутствии некоторых данных. Основы некоторых криптографических алгоритмов приведены в таблице 1.2.

Таблица 1.2 – Механизмы борьбы с угрозами

	Пример	Основан
Симметричный криптографический механизм	AES	На сложности решения некоторых типов уравнений [4,5].
	DES	На комбинации нелинейных (S-блоков) и линейных (перестановок E, IP, IP-1) преобразователей.
	GOST block cipher (Magma)	На сетях Фейстеля[6].
Ассиметричный криптографический алгоритм	RSA	На сложности факторизации полупростых чисел
	Elgamal	На сложности вычисления дискретных логарифмов в конечном поле
	DSA	На сложности вычисления дискретных логарифмов в конечном поле

Следует классифицировать методы обеспечения безопасности по двум основным категориям:

- Новые появляющиеся методы для предотвращения кибератак и сохранения целостности структур, такие как SDN, Blockchain;
- Классические подходы.

Классические подходы к осуществлению безопасности в основном разработанные специально для IoT сетей, обеспечивающие конфиденциальность, приватность и доступность услуг. Данный подход актуален в основном для централизованных систем, поскольку централизованные системы могут позволить себе такую роскошь, как вычислительные ресурсы и высокий расход энергии. Однако для обеспечения безопасности не все типы шифрования можно использовать.

При использовании симметричных шифров, таких как AES, который использовало Агентство национальной безопасности США для защиты сведений, составляющих государственную тайну. До уровня SECRET было разрешено использовать ключи длиной 128 бит, для уровня TOP SECRET требовались ключи длиной 192 и 256 бит [5], также как и для ГОСТ 28147-89 (блочный шифр магма) созданный группой криптографов КГБ 8го управления [7,8].

В AES шифрование и дешифрование выполняется по одному и тому же ключу, это означает, что передача ключа должна происходить по защищенному каналу для всех устройств. Если злоумышленник получит ключ, то все устройства системы, использующие этот ключ, могут оказаться под его контролем. Для того чтобы организовать защищенный канал и безопасно передать симметричный ключ, часто используются ассиметричные методы шифрования, называемые также односторонними.

2 Криптосистема с открытым ключом RSA

Криптосистемы с открытым ключом можно использовать в трех назначениях:

1. Как самостоятельные средства защиты передаваемых и хранимых данных.
2. Как средства для распределения ключей.
3. Средства аутентификации пользователей.

Наиболее популярной криптосистемой с открытым ключом является алгоритм RSA, названный по первым буквам фамилий своих изобретателей – Ривеста (Rivest), Шамира (Samir) и Адлемана (Adleman) [9]. Криптосистема RSA – первая практическая реализация криптографии с открытым ключом на основе понятия однонаправленной функции с секретом, предложенного Диффи и Хеллманом [10, 11].

Основной принцип RSA шифрования построен на больших простых числах $(p; q)$ и математических операциях над ними. В ассиметричном, в отличие от симметричного, генерируется не один, а пара ключей: приватный служит для дешифровки сообщений адресатом и остаётся в тайне, а публичный передаётся адресанту. Генерация ключей выполняется по следующему алгоритму [12, 13].

1. Выбирается два случайных, больших простых числа p и q .

Числа p и q остаются в секрете и должны находиться в диапазоне от $(2^{2047} - 1)$ до $(2^{2048} - 1)$ или в двоичном представлении: от 100...001 до 111...111 (2047 бит).

2. Вычисляется модуль p и q чисел: $n = p * q$.

n является частью открытого и закрытого ключей и сообщается адресантам по незащищенным каналам. Разложение числа n на простые множители позволит дешифровать сообщение. На сегодняшний день смогли факторизовать RSA-232 криптосистему (17.02.2020г) с использованием суперкомпьютера "Ломоносов" МГУ им М. В. Ломоносова и суперкомпьютера "Жорес"[14].

3. Вычисляется функция Эйлера.

Функция Эйлера показывает количество натуральных чисел взаимно простых с n $\varphi(n) = (p - 1) * (q - 1)$

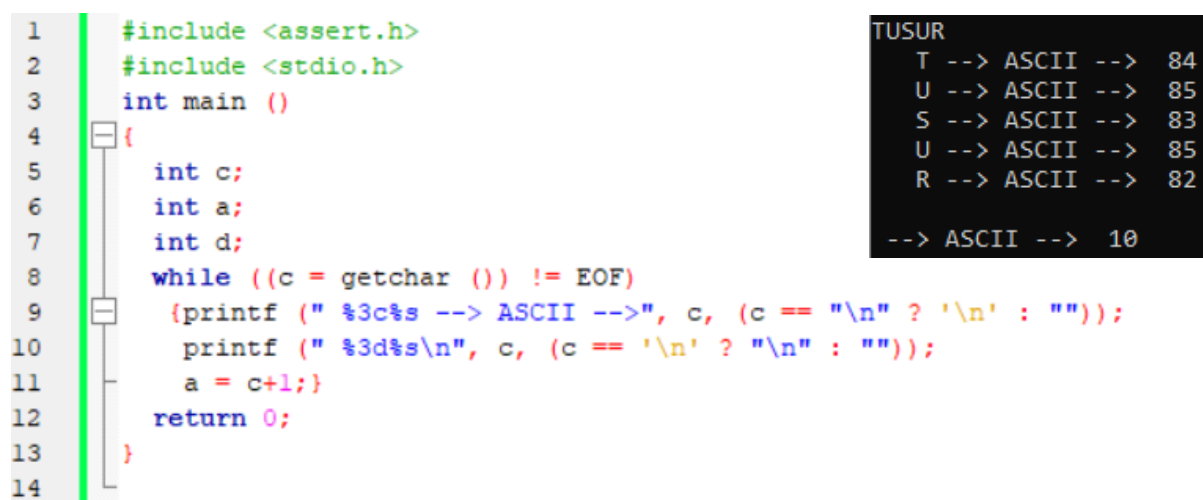
4. Генерируется открытый ключ $\{e, n\}$. Этот ключ сообщается оппоненту по незащищенным каналам связи.

Выбирается целое число e , взаимно простое с $\varphi(n)$, $e \in (1; \varphi(n))$. Следует отметить, что для повышения скорости шифрования, чаще выбираются числа Ферма [15].

5. Генерируется закрытый ключ $\{d, n\}$. Данный ключ остается в секрете и работает в качестве дешифратора.

Выбирается число d , удовлетворяющее условию: $(d * e) \bmod \varphi(n) = 1$
Для рациональной работы с памятью используется алгоритм быстрого возведения в степень [16].

Если есть необходимость передавать не только цифры, но и текст, тогда каждый символ необходимо трансмогрифицировать в целочисленное представление. Эту процедуру сделать не сложно, например воспользоваться порядковым номером в алфавите, кодом ASCII [17] символов (Рисунок 2.1) или другим способом, важно помнить, что целочисленное представление символа не должно превышать n . Однако этого будет недостаточно, для безопасной работы системы, т.к. один и тот же символ будет зашифрован одинаковым числом.



```

1  #include <assert.h>
2  #include <stdio.h>
3  int main ()
4  {
5      int c;
6      int a;
7      int d;
8      while ((c = getchar ()) != EOF)
9      {
10         printf (" %3c%s --> ASCII -->", c, (c == "\n" ? "\n" : ""));
11         printf (" %3d%s\n", c, (c == '\n' ? "\n" : ""));
12         a = c+1;
13     }
14     return 0;

```

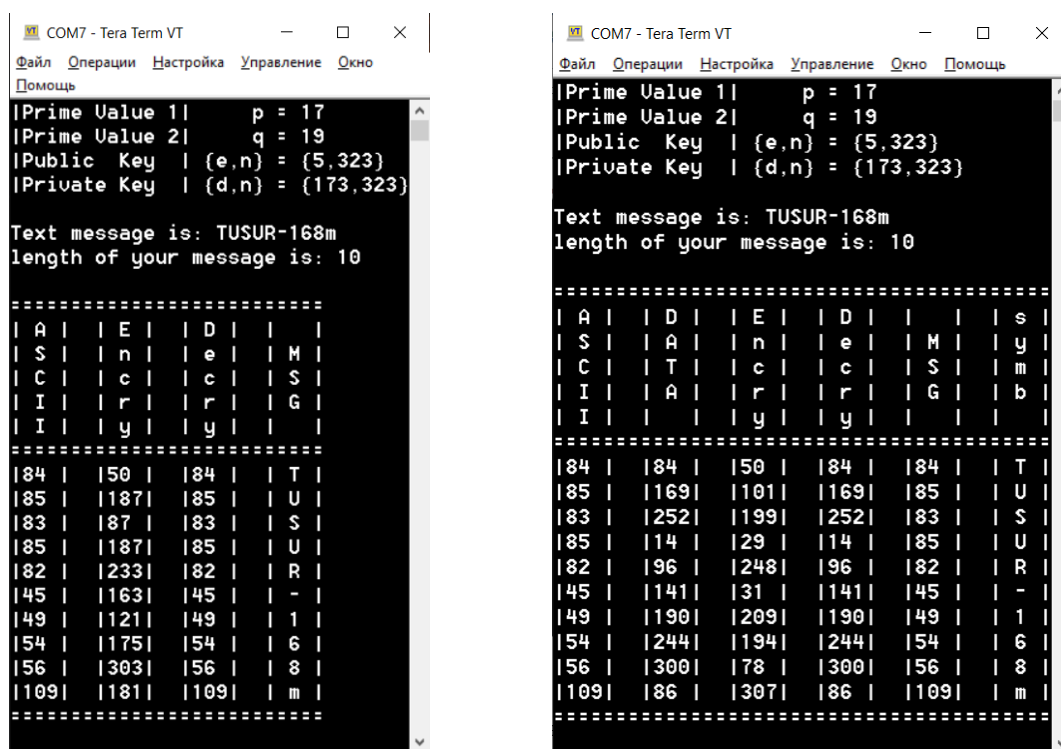
Symbol	ASCII
T	84
U	85
S	83
U	85
R	82
\n	10

Рисунок 2.1 – Получение ASCII символов.

При большой выборке зашифрованных данных, можно будет без использования ключей определить исходное сообщение (Рисунок 2.2а). Для предотвращения этого, следует прибегнуть к обратимому дополнительному алгоритму, где каждый блок шифруемых данных зависит от предыдущего (Рисунок 2.2б), как в последовательности Фибоначчи, или, например, к каждой следующей части передаваемого сообщения добавлять предыдущую и находить остаток от деления на n [18].

$$data[i] = (ASCII[i] + ASCII[i - 1]) \bmod(n).$$

Для получения исходного сообщения необходимо совершить обратную операцию.



а) без блочного шифра

б) с блочным шифром

Рисунок 2.2 – Результаты шифрования и дешифрования.

Алгоритм шифрования и дешифрования осуществляется по следующим формулам:

$$encrypt = data^e \bmod(n);$$

$$decrypt = encrypt^d \bmod(n);$$

При выборе числа, отправляемого на вход шифрующего устройства, необходимо учесть один очень важный момент. Если шифруемые данные меньше $n^{1/e}$, то злоумышленнику не составит труда найти исходное сообщение, найдя его по известному публичному ключу:

$$data = \sqrt[e]{encrypt};$$

Основная цель данной работы заключается в анализе работы RSA алгоритма и нахождении корреляции дешифруемых данных по побочным каналам. Т.к. это исследовательская работа и нет необходимости создавать устойчивую систему к атакам, блочный шифр будет опущен.

3 Атака по сторонним каналам

Атака по побочным каналам (side-channel attack) представляет собой класс атак на криптосистему, который использует о физических процессах в приборе и направлен на уязвимости криптосистемы в ее практической реализации [19]. Как следствие, криптографические реализации должны оцениваться на предмет их сопротивления таким атакам, и необходимо учитывать различные контрмеры.

Как отмечают многие эксперты по информационной безопасности, сильный криптографический алгоритм не может автоматически гарантировать безопасность системы. К тому же, факт того, что каждый компонент системы защищен, не обязательно означает, что вся система безопасна.

С конца 1990-х годов известно, что реализация криптографических алгоритмов невозможна без утечки информации из разных боковых каналов. Первая статья, которая демонстрирующая эксплуатация бокового канала была опубликована П. Кохером [20] в 1996 году. Он подчеркнул, что аппаратные реализации криптосистем могут предоставлять характеристики синхронизации, которые дают утечку информации о закрытых ключах.

Таким образом, при реальной реализации криптографической системы необходимо учитывать возможность атак по сторонним каналам криптографического устройства. Информация, из которой можно извлечь полезные данные, продемонстрирована на рисунке 3.1.



Рисунок 3.1 – Сторонние каналы криптографического устройства

В целом атаки по побочным каналам можно классифицировать по факту вмешательства (пассивные, активные), по характеру воздействия (разрушающие, полуразрушающие, неразрушающие) и по способу анализа полученных данных. В зависимости от методов, используемых в процессе анализа выборочных данных, все атаки можно разделить на простые атаки по сторонним каналам (SSCA - simple side channel attack) и атаки по дифференциальному побочному каналу (DSCA - differential side channel attack).

В SSCA атака использует выходной сигнал побочного канала в зависимости от выполненных операций. Секретный ключ в данном случае может быть непосредственно рассчитан из трассировки данных бокового канала. При этом сигнал побочного канала должен быть более ярко выражен, чем шум [21].

С другой стороны, когда простая атака по сторонним каналам неосуществима из-за слишком большого шума в измерениях, используется атака по дифференциальному побочному каналу с использованием статистических методов. Дифференциальные атаки по побочным каналам используют корреляцию между данными и выходом побочных каналов криптографического устройства. Поскольку это соотношение обычно очень мало, для его

эффективного использования необходимо использовать статистические методы. При дифференциальной атаке по боковому каналу злоумышленник использует гипотетическую модель атакующего устройства, которая используется для прогнозирования выходного сигнала побочного канала устройства, и качество этой модели зависит от возможностей атакующего.

4 Виды атак по сторонним каналам

На данный момент было исследовано достаточно большое количество атак по побочным каналам [22]. Так, выделяют следующие типы атак по сторонним каналам:

1. Атаки **по времени** (time attacks).

Одним из наиболее изученных типов атак остаются атаки по времени. Данный вид атак основан на статистическом анализе измерений времени, которое требуется для обработки данных для того или иного устройства [23]. Хотя обычно атаки по времени используются для атаки на слабые вычислительные устройства, такие как смарт-карты, данный вид атак также применим к более сложным программным системам. В частности, была разработана атака по времени против OpenSSL, позволяющая извлечь секретные ключи из веб-сервера на основе OpenSSL, работающего на компьютере в локальной сети.

2. Атаки **по электромагнитному излучению** (electromagnetic analysis attacks).

Данный тип атаки по стороннему каналу заключается в оценке электромагнитного излучения, которое электронные шифрующие устройства производят во время своей работы. Процесс атаки и дешифровки данных происходит на основе спектральных компонентов излучения. Атаки по электромагнитному излучению оказываются достаточно эффективными против криптографических реализаций, выполняющих работу на основе данных, которые обрабатываются в текущий момент, в том числе, и алгоритм быстрого возведения в степень, использующийся в алгоритме RSA. Атаки по электромагнитному излучению достаточно часто производятся не отдельно, а в сочетании с прочими атаками по сторонним каналам, например, в комбинации с атакой по энергопотреблению.

При таком виде атак требуется физический доступ к криптографическому устройству, на рисунке 4.1 демонстрируется устройство [24], разработанное

исследователями Тель-Авивского университета, способное на небольших расстояниях распознавать операции, выполняемые процессором и тем самым извлечь ключи шифрования.

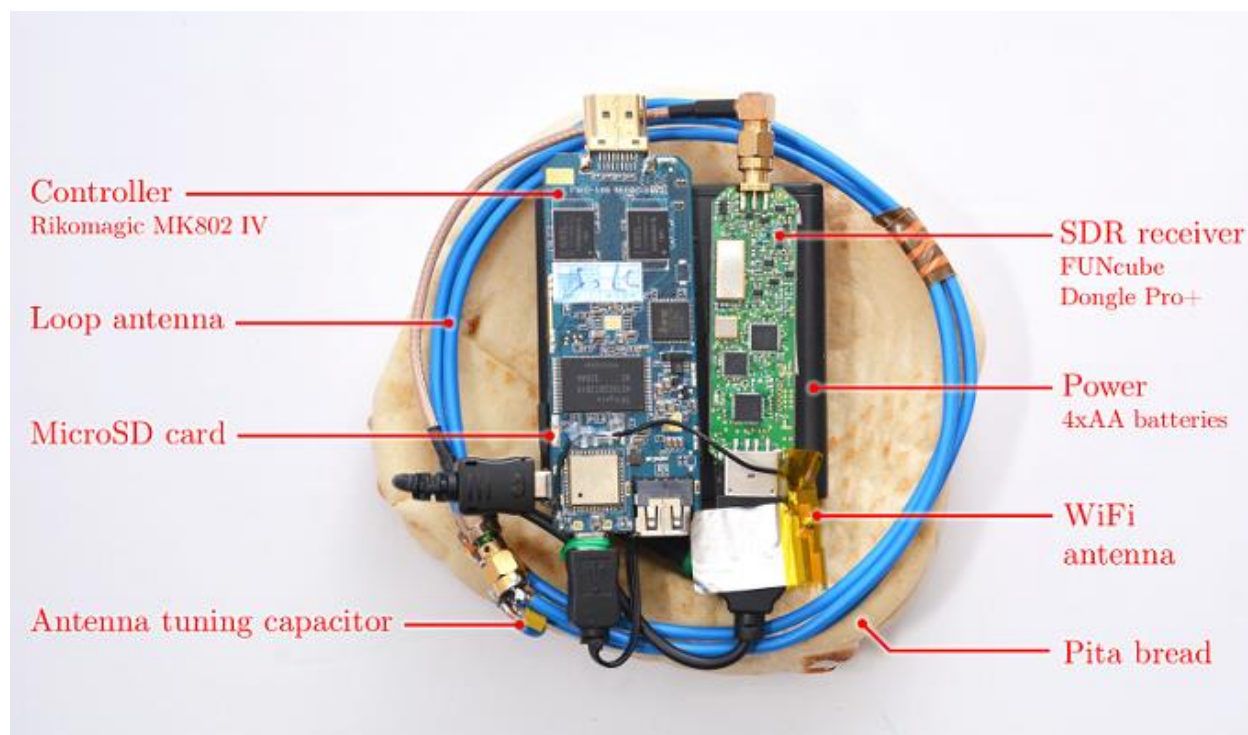


Рисунок 4.1 – портативное устройство для захвата сигнала РІТА

3. Атаки **по энергопотреблению** (power analysis attack).

Данный тип атаки представляет собой пассивную атаку, которая заключается в оценке и анализе изменений в энергопотреблении устройства, тем самым делая доступным получение информации об операциях, выполняемых во время шифрования. При этом, в 1998 году были выделены два основных типа атак по энергопотреблению [20]: простая и дифференциальная. Простая атака по энергопотреблению (SPA) заключается в визуальном сравнении и поиске корреляций на графиках электрической активности исследуемого устройства с течением времени. В то же время, дифференциальная атака (DPA) представляет собой более продвинутую и эффективную разновидность атаки по энергопотреблению [25]. С ее помощью становится возможным вычисление промежуточных значений, которые используются в используемом алгоритме. Это происходит посредством статистического анализа собранных в процессе вычисления нескольких операций данных.

На рисунке 4.2 продемонстрировано наблюдение за битами ключа RSA[26], первый пик соответствует итерации алгоритма быстрого возведения в степень без умножения, второй пик – с умножением. Для того чтобы анализ был более тривиальным между шагами алгоритма искусственно вносилась задержка, которой соответствует падение потребляемой энергии между первым и вторым пиками

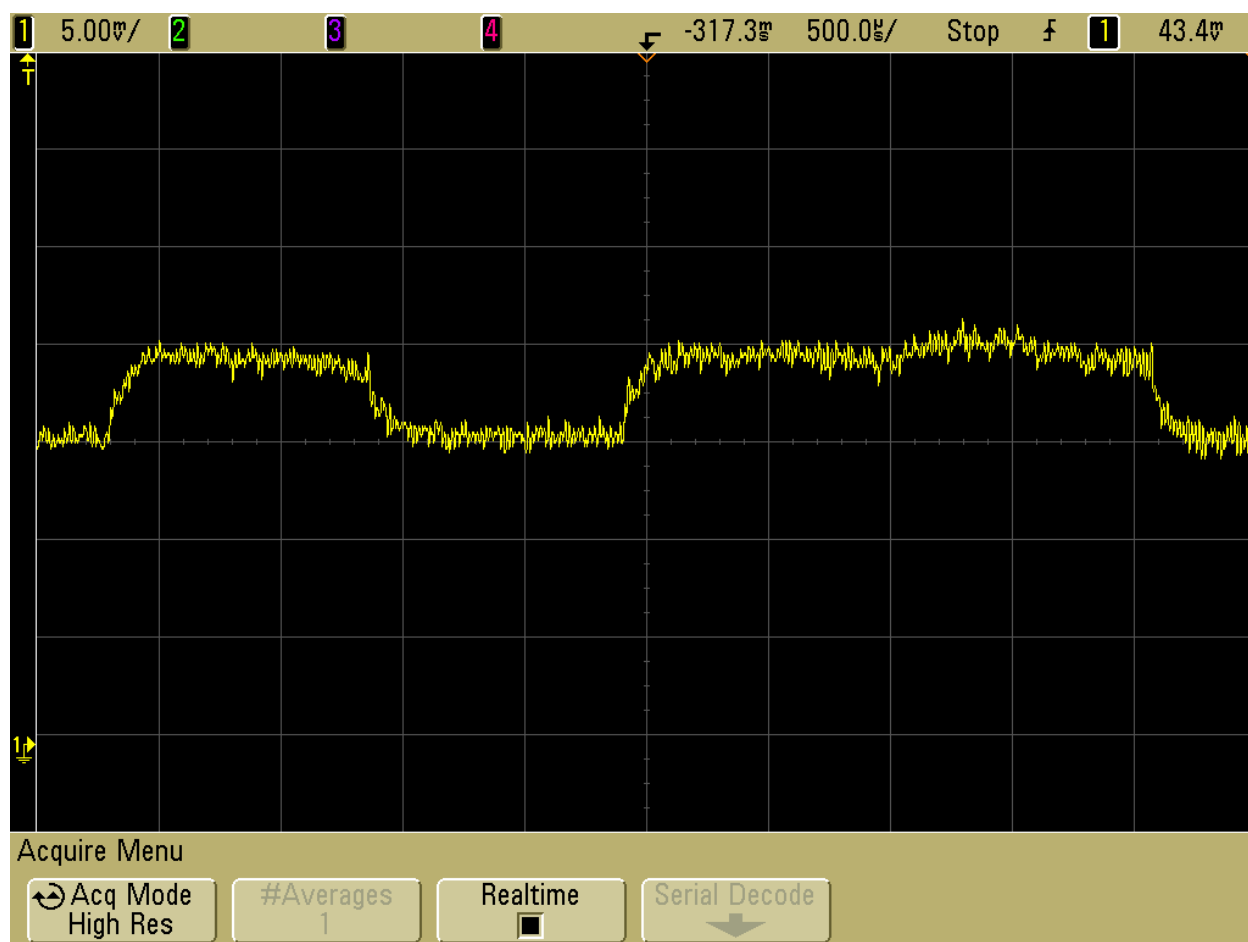


Рисунок 4.2 – атака по энергопотреблению на алгоритм RSA

4. Атаки по видимому излучению (visible light attack).

Данный вид атак относится к пассивному типу атак. Он основан на фиксации и оценке интенсивности света, который был рассеян от монитора или шифраторов со световыми индикаторами. Интенсивность света, излучаемого экраном, как функция времени, соответствует видеосигналу, свернутому с импульсной характеристикой люминофоров. Эксперименты с нетипичным цветным монитором персонального компьютера показывают, что в излучаемом

свете остается достаточно высокочастотного контента, чтобы можно было восстановить читаемый текст путем деконволюции, то есть обратной свертке сигнала, полученного с помощью быстрого фотодатчика.

5. Акустические атаки (acoustic attack).

Акустический метод атак использует акустический криптоанализ. Данный способ основан на пассивном анализе звуков, производимых устройством во время работы над задачей [27]. Однако данный способ зависит от конкретного компьютерного оборудования. Авторы протестировали множество ноутбуков и несколько рабочих компьютеров. Обнаружилось, что практически на всех машинах можно отличить неработающий процессор от выполняющего вычисления ЦП. Таким образом при помощи установки показанной на рисунке 4.3 исследователям удалось извлечь ключи RSA используя низкочастотный акустический криптоанализ[28].

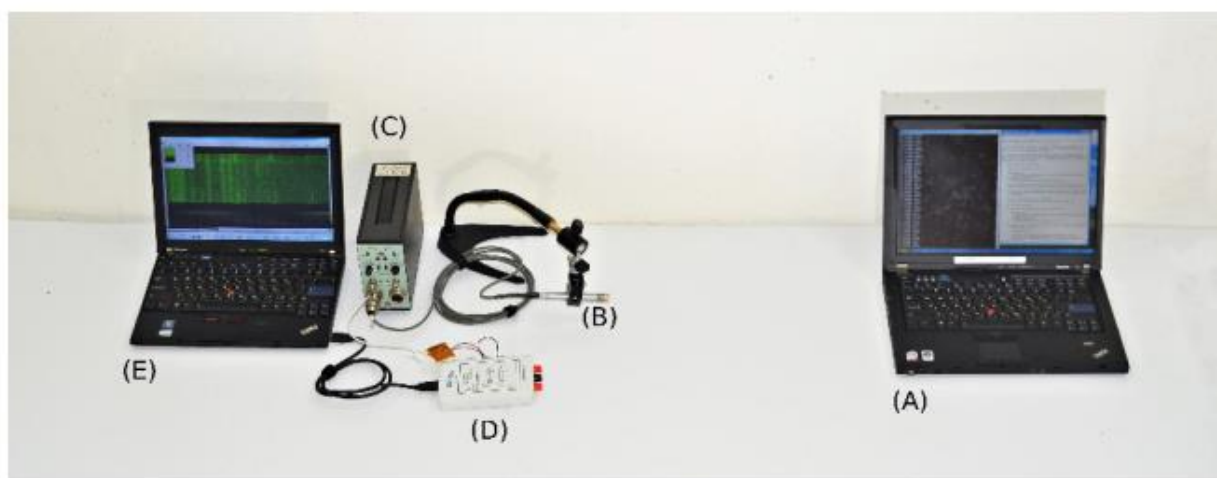


Рисунок 4.3 –Технология низкочастотного акустического криптоанализа

На данном рисунке А – Lenovo ThinkPad T61, выполняющий криптографические операции, В – микрофон, установленный на предварительном усилителе, С – блок питания и усилитель для микрофона, D – устройство MyDAQ с RC фильтром нижних частот 10 кГц, Е – атакующее устройство. При помощи данной установки удалось извлечь ключи с расстояния 1 метр.

6. Атаки по ошибкам вычислений (attacks on calculation errors).

Активный тип побочных атак, основанный на эффекте воздействия на информацию, ее активного искажения и анализа результатов на разных этапах работы. Данный тип побочных атак может быть осуществлен различными методами, такими как изменение энергопотребления криптосистемы, конструкции шифратора, напряжения питания системы, тактовой частоты устройства, повышение температуры некоторой части шифратора.

Для атак по ошибкам вычислений может быть выделена классификация по типу полученной ошибки: они могут быть постоянные или переменные, ошибки локальные и в произвольном месте, те, что требуют применения в определенное время (например, тактовая частота), и независимые от момента воздействия.

Успешная атака на криптографические модули по ошибке вычислений подразумевает два этапа: внедрение ошибки и обработка ошибок. Первый шаг состоит в том, чтобы ввести ошибку в необходимое время в течение процесса. Внедрение неисправности сильно зависит от аппаратного обеспечения устройства, а также от изменения напряжения, часов, температуры, излучения, света и т.д. Второй шаг состоит в обработке результата ошибки и анализе поведения прибора. Работа с неисправностью зависит от дизайна и реализации программного обеспечения, от спецификации алгоритма.

В целом, можно утверждать, что в независимости от типа атак, непосредственный процесс атаки зависит, в основном, не от самого алгоритма, а от конкретной реализации данного криптографического протокола. Следует заметить, что для достижения наибольшей эффективности атаки различные типы атак комбинируются и выполняются в сочетании. Это позволяет достичь более качественных результатов.

Так как темой данной магистерской диссертации являются атаки по температуре некоторых частей устройства, то данный тип атак будет более подробно рассмотрен далее.

5 Атаки по данным со стороны температурного канала

Утечка информации через атаки по температурным данным, хотя и упоминается в современной литературе, но рассмотрена недостаточно подробно.

В некоторых исследованиях упоминается, что охлаждающий вентилятор может передавать информацию об обработанных данных косвенно через изменение температуры процессора [29]. Также были проведены исследования, где продемонстрировали способ извлечения битов из возможного ключа RSA. В данных исследованиях предполагалась низкочастотная утечка битов, то есть утечку бит за три минуты. Кроме того, известно, что IP-ядра, встроенные в FPGA, могут передавать информацию другим IP-ядрам в системе через показатели температуры.

Также исследуются активные температурные атаки, которые влияют непосредственно на температуру устройства (охлаждение или нагрев). Большинство из них демонстрируют эффективность низкотемпературных атак, например, как показали С. Скоробогатов [30] и D. Samyde [31] в 2002 году. Они сообщили, что, посредством охлаждения устройства SRAM до -50°C , представляется возможным заморозить данные и затем восстановить память устройства даже через несколько секунд после отключения питания (используя свойство хранения данных ячейки SRAM). Т. Мюллер использовал ту же идею, которая предоставила инструмент под названием FROST [32].

Наиболее подробная атака с использованием температурного канала описана в исследованиях Hutter и Schmidt [33]. В исследовании 2014 года авторы представляют практические результаты утечки данных CMOS-устройств через сторонний канал температуры. В данном исследовании измерение температур относится к пассивным типам атак, одним из самых трудных для обнаружения. Пассивные атаки – это такие атаки, при которых Eve, пассивный злоумышленник [34], не воздействует на криптосистему, а только читает данные из нее. При такой атаке криптосистема продолжает свою работу без изменений.

Нагревая процессор устройства до $154,4^{\circ}\text{C}$, им удалось добиться наиболее вероятного появления ошибок и извлечь информацию о ключах, менее чем за 30 минут (Рисунок 5.1). Результаты показали, что модель утечки информации по температурным данным линейно коррелирует с моделью утечки информации по мощности, но ограничена физическими свойствами теплопроводности и емкости. Кроме того, сообщается о неисправностях нагрева при эксплуатации устройства за пределами указанных температурных показателей. Эффективность такого рода атак демонстрируется практической атакой на реализацию RSA.

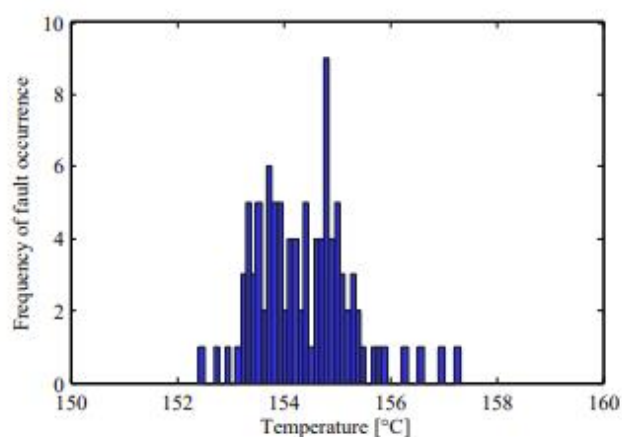
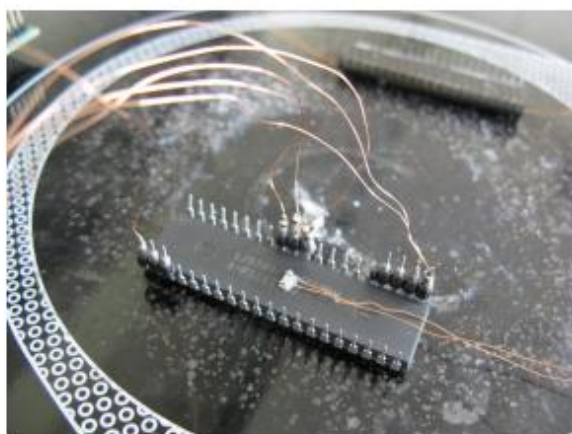


Рисунок 5.1 – Ошибки вычисления в зависимости от температуры процессора

Температурная атака коррелирует с атакой по энергопотреблению, но это не делает этот тип атаки простым. Температурную атаку довольно сложно проанализировать, потому что изменение температуры является инерционным процессом, который не может измениться мгновенно. Это означает, что при таком типе атаки важно учитывать большое количество факторов. Этот тип атаки требует дальнейшего более глубокого изучения, что и было проделано в данной работе. Желательно осуществить атаку на криптосистему через температурный канал вместе с другой, дополнительной сторонней информацией.

Таким образом, атаки на криптографические модули по сторонним каналам используют характерную информацию, извлеченную из реализации криптографических примитивов и протоколов. Эта характеристическая

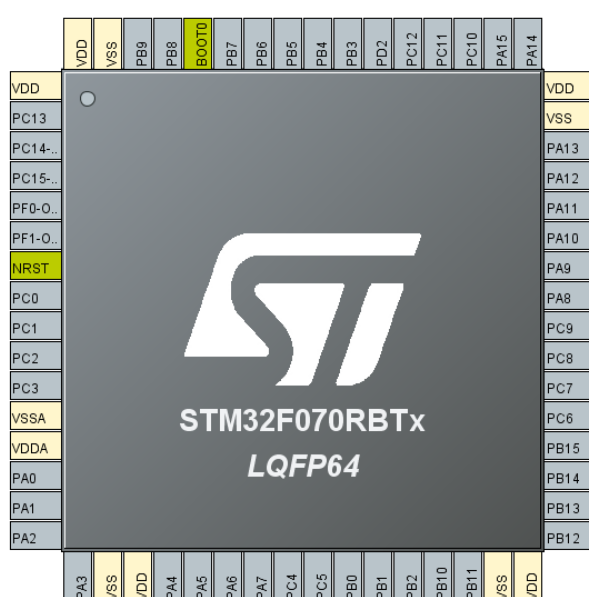
информация может быть извлечена из характеристик времени, энергопотребления или электромагнитного излучения. Другие формы информации о побочных каналах могут быть результатом аппаратных или программных ошибок, вычислительных ошибок и изменений частоты или температуры.

Так, можно заключить, что данные атаки представляют собой серьезную угрозу безопасности криптографических модулей, так как небольшая разница в данных конкретных реализаций криптографической задачи может иметь большое значение для безопасности всей криптосистемы в целом. Таким образом, возможности атак по сторонним каналам, в том числе по температурным данным, необходимо учитывать при разработке криптографических алгоритмов.

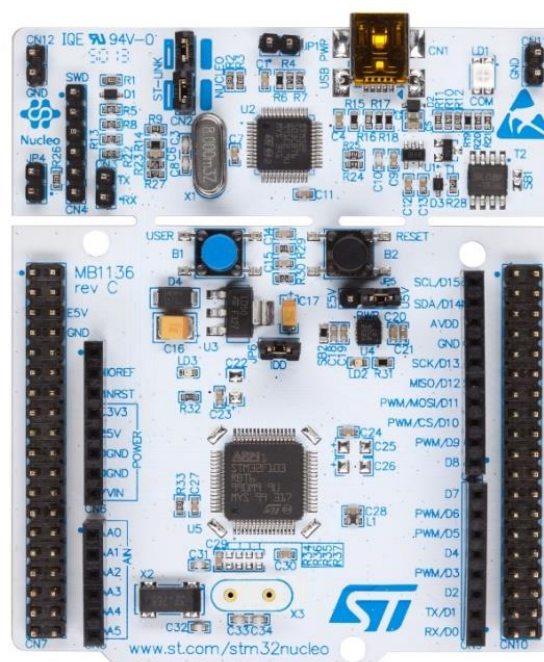
Для того, чтобы противодействовать атакам по побочным каналам, могут быть использованы множество различных методов, такие как экранирование, добавление шумов в побочном канале, балансировка времени выполнения операций, уравнивание энергопотребления системы, маскирование и произведение вычислений вслепую [35].

6 Получение экспериментальных данных

В данной работе производится проверка уязвимостей в безопасности криптосистемы RSA, реализованной на микроконтроллере STM32F070RB (Рисунок 6.1а) на плате разработки STM32 Nucleo-64 (Рисунок 6.1б) и проверяется гипотеза извлечения ключей шифрования по показаниям температуры процессора не нарушая его целостность используя встроенный температурный датчик.



а) микроконтроллер



б) плата разработки

Рисунок 6.1 – Используемый микроконтроллер в исследованиях

Для чистоты эксперимента температура процессора измеряется с помощью контроллера DMA (контроллера прямого доступа к памяти). Прямой доступ к памяти (DMA) используется для обеспечения высокоскоростной передачи данных между периферийными устройствами и памятью, а также памятью в память. Данные могут быть быстро перемещены DMA без каких-либо действий процессора. Это сохраняет ресурсы ЦП свободными для других операций [36].

Для получения экспериментальных данных были реализованы следующие задачи:

1. Был рассчитан публичный и секретный ключ. Полученные значения публичного ключа ($e;n$) были записаны в код программы выполнявшей шифрование сообщения на ПК. Значения секретного ключа ($d;n$) – в код программы микроконтроллера, выполнявшего дешифровку сообщения (Приложение А).
2. При помощи программы (Приложение Б) было получено зашифрованное сообщение. По рассчитанному зашифрованному сообщению выполнялась дешифровка на микроконтроллере stm32.
3. Дешифрование одного и того же сообщения производилось последовательно много раз, чтобы получить достаточное для анализа количество измерений АЦП с внутреннего температурного сенсора микроконтроллера при помощи прямого доступа к памяти (ПДП) (Приложение В).
4. Значения температуры при помощи usb-кабеля по UART соединению были переданы виртуальному COM порту компьютера для последующей обработки и нахождения корреляции между дешифруемым сообщением и температурой процессора stm32. Вместо usb-кабеля также может быть Bluetooth, Wi-Fi или любой другой модуль, подключенный к соответствующим выводам микроконтроллера

6.1 Вычисление ключей шифрования на ПК.

Код программы предварительных расчетов на ПК представлен в приложении А.

1. Ожидается ввод чисел p и q и производится их проверка на простоту.

Были выбраны $p = 10007$ и $q = 399989$, такие, чтобы их произведение n не превышало $\sqrt{\text{unsigned long long}}$ ($n < 4294967296$). Если выбрать n близкое к 18446744073709551615 (unsigned long long), то при шифровании и дешифровании может возникнуть переполнения переменной и это приведет к неверному результату работы алгоритма.

2. Вычисляется n , а также функция Эйлера $\varphi(n)$.
3. Выбирается число e удовлетворяющее условиям:
 - $e < \varphi(n)$;
 - e взаимно простое с $\varphi(n)$;
 - $e \in (1; \varphi(n))$.
4. Выполняется поиск числа d , по расширенному алгоритму Евклида [37] такого что:

$$(d * e) \bmod (\varphi(n)) = 1$$

Результат выполнения этого алгоритма продемонстрирован на рисунке 6.2.

```

"C:\Users\b gore\Desktop\Master's thesis\calculate_key\bin\Debug\calculate_key.exe"
Calculation of the private key and public key for the RSA algorithm
//p = 10007
//q = 399989
//n = 4002689923
//phin = 4002279928
//e = 3
//d = 2668186619
Process returned 0 (0x0)   execution time : 6.803 s
Press any key to continue.
  
```

Рисунок 6.2 – Нахождение ключей шифрования

При реализации без использования библиотек длинной арифметики следует учесть максимальный размер ключей p и q таким, чтобы n не превышало 64bit, что в десятичном представлении соответствует 18446744073709551615

(0xFFFFFFFFFFFFFFFF в шестнадцатеричном). Это ограничение вызвано размерами целочисленных переменных. Если продолжать работать с переменными такого типа (int), важно учесть, что при генерировании закрытого ключа, шифрования и дешифрования находится остаток от деления на n , т.е. размер частного также не должен превышать «unsigned long long».

Выбор правильного размера ключа для реальных криптосистем – важная задача. Для этого необходимо знать оценки трудоёмкости разложения простых чисел различной длины, сделанные Шроппелем [38].

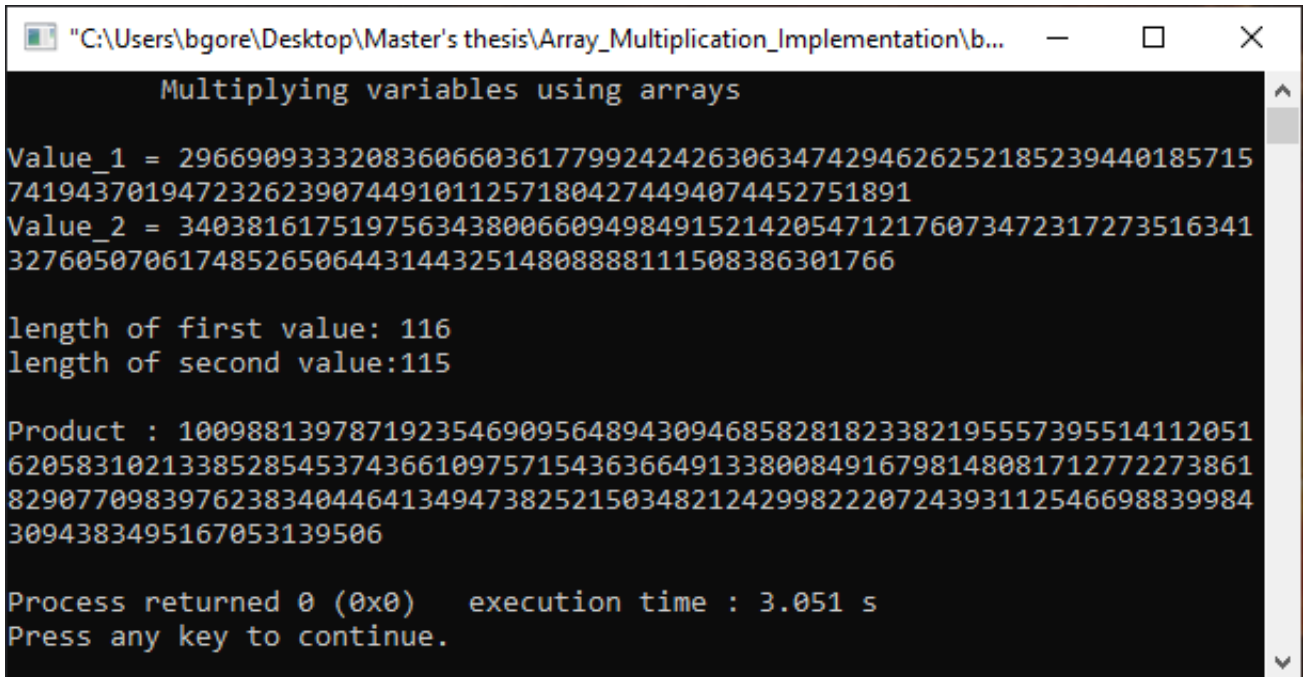
Таблица 6.1 – Оценки трудоёмкости разложения простых чисел

lg n	Число операций	Примечания
50	$1.4 \cdot 10^{10}$	Раскрываем на суперкомпьютерах
100	$2.3 \cdot 10^{15}$	На пределе современных технологий
200	$1.2 \cdot 10^{23}$	За пределами современных технологий
400	$2.7 \cdot 10^{34}$	Требует существенных изменений в технологии
800	$1.3 \cdot 10^{51}$	Не раскрываем

Согласно рекомендациям авторов RSA алгоритма [38], размеры модуля n выбираются по следующим критериям:

- 768 бит - для частных лиц;
- 1024 бит - для коммерческой информации;
- 2048 бит - для секретной информации.

Для примера, воспользуемся полученными от 17 февраля 2020г. p и q , при разложении полупростого числа n – RSA-232 (768 bit)[14]. Для работы с величинами таких размеров целочисленный тип переменных не подходит. Решение которое может помочь справиться с данной задачей можно реализовать при помощи массивов. На рисунке 6.3 представлен результат выполнения программы (Приложение Г), где реализовано умножение двух чисел. Этот код работает по принципу умножения в столбик. Если дальше продолжать работать с массивами, то нужно реализовать остальные необходимые алгоритмы для математических операций что по своей сути является трудоёмкой задачей и похожа на написание собственной библиотеки длинной арифметики.



```

"C:\Users\b gore\Desktop\Master's thesis\Array_Multiplication_Implementation\b...
Multiplying variables using arrays

Value_1 = 296690933320836066036177992424263063474294626252185239440185715
74194370194723262390744910112571804274494074452751891
Value_2 = 340381617519756343800660949849152142054712176073472317273516341
3276050706174852650644314432514808888111508386301766

length of first value: 116
length of second value:115

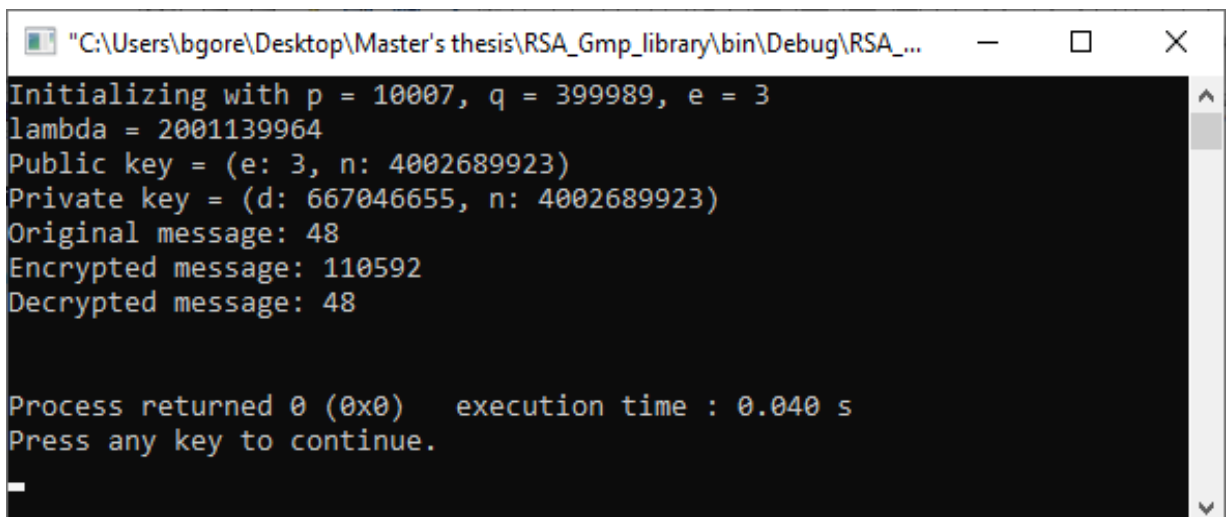
Product : 100988139787192354690956489430946858281823382195557395514112051
6205831021338528545374366109757154363664913380084916798148081712772273861
8290770983976238340446413494738252150348212429982220724393112546698839984
3094383495167053139506

Process returned 0 (0x0)   execution time : 3.051 s
Press any key to continue.

```

Рисунок 6.3 – Умножение чисел при помощи массивов

Также есть и альтернативные решения этой задачи. Один из вариантов – это использование готовых библиотек, таких как GMP[39], MPIR, или Mini-GMP. Реализация RSA алгоритма при помощи библиотеки длинной арифметики GMP для ключей из рисунка 6.2 представлена на рисунке 6.4. Еще одно из решений – это использование модульной арифметики собственного типа данных с фиксированным размером (int1024_t).



```

"C:\Users\b gore\Desktop\Master's thesis\RSA_Gmp_library\bin\Debug\RSA_...
Initializing with p = 10007, q = 399989, e = 3
lambda = 2001139964
Public key = (e: 3, n: 4002689923)
Private key = (d: 667046655, n: 4002689923)
Original message: 48
Encrypted message: 110592
Decrypted message: 48

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.

```

Рисунок 6.4 – Реализация RSA с GMP библиотекой для $n=4002689923$

При помощи данной библиотеки можно зашифровывать и дешифровывать сообщения большей длины, в отличие от реализации, использующей

целочисленные переменные. Для примера рассчитаем ключи и зашифруем символ «0» используя p и q полученными при разложении числа n RSA – 232[14] (Рисунок 6.5).

```

"C:\Users\bgore\Desktop\Master's thesis\RSA_Gmp_library\bin\Debug\RSA_Gmp_library.exe"
Initializing with p = 2966009333208360660361779924242630634742946262521852394401857157419437019472326239074491011257180427449407445
2751891, q = 34038161751975634380066094984915214205471217607347231727351634132760507061748526506443144325148088881115083863017669,
e = 65537
lambda = 50494069893596177345478244715473429140911691097778697757056025810291551066926427268718305487857718183245669004245850073133
3308732746155352688026446430204004562371937392519915879851896338284238441937093010286928644574894591145196260
Public key = (e: 65537, n: 10098813978719235469095648943094685828182338219555739551411205162058310213385285453743661097571543636649
13380084917065169921701524733294389270280234380960909804976440540711201965410747553824948672771374075011577182305398340606162079)
Private key = (d: 14026344544500334293214999848104029441541378708745612290266642505399357419067024862703766596067103461037084459500
674437056812007689768769085685218208434722226497720680391611892407399985328357429169244522213002364686909013714364373, n: 10098813
97871923546909564894309468582818233821955573955141120516205831021338528545374366109757154363664913380084917065169921701524733294389
270280234380960909804976440540711201965410747553824948672771374075011577182305398340606162079)
Original message: 48
Encrypted message: 3183766375361424916978125645280336912977753963467025915232200996978168325533915641436861936209438406545393359978
10037707649283472816902731600136521161141818329850734793389994849439255159919782148575794953302188887128654601237829552
Decrypted message: 48

Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.

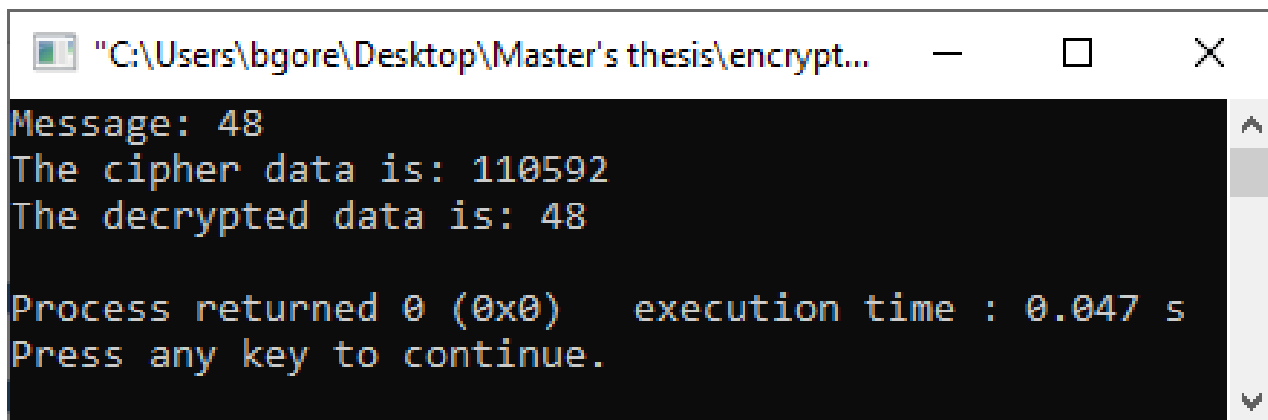
```

Рисунок 6.5 – Шифрование ключами RSA - 232

Отметим немаловажный вычислительный аспект реализации RSA. Для корректной работы приходится использовать аппарат длинной арифметики. Если используется ключ длиной k бит, то для операций по открытому ключу требуется $O(k^2)$ операций, по закрытому ключу - $O(k^3)$ операций, а для генерации новых ключей требуется $O(k^4)$ операций. По сравнению с тем же алгоритмом DES, RSA требует в тысячи и десятки тысяч раз большее времени [38].

6.2 Получение зашифрованного сообщения.

Для шифрования сообщений используют 8 битную кодировку ASCII символов. На рисунке 6.6 показан результат выполнения шифрования без использования библиотеки длинной арифметики по ключам из рисунка 6.2 символа «0», который согласно 8 битной кодировке по ASCII таблице соответствует в десятичном виде числу 48, или в шестнадцатеричной числу 30.

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\bgore\Desktop\Master's thesis\encrypt...". The command prompt displays the following text: "Message: 48", "The cipher data is: 110592", "The decrypted data is: 48", "Process returned 0 (0x0) execution time : 0.047 s", and "Press any key to continue.".

```
"C:\Users\bgore\Desktop\Master's thesis\encrypt...  
Message: 48  
The cipher data is: 110592  
The decrypted data is: 48  
  
Process returned 0 (0x0) execution time : 0.047 s  
Press any key to continue.
```

Рисунок 6.6 – Результат шифрования

Для того чтобы с большей долей вероятности увидеть различия температурных зависимостей, кроме шифрования десятичного представления ASCII символов было решено дешифровывать числовые сообщения с различной величиной. В данной таблице для анализа результатов были выбраны значения «1024» и «133964360», зашифрованные значения которых имеют расстояние Хэмминга равное 30 (в двоичной системе счисления отличаются на 30 единичных бит). В таблице 6.2 представлены некоторые исходные сообщения и соответствующие им зашифрованные значения.

Таблица 6.2 – Шифрование данных с разным расстоянием Хэмминга

№	Data	Encrypted
1	2	8
2	3	27
3	100	1000000
4	1500	375000000
5	4002689921	4002689915
6	4002689920	4002689896
7	133964360	2147483647
8	1024	1073741824

6.3 Получение данных температуры в процессе дешифровки

Для получения достаточного количества анализируемых температурных измерений, одни и те же данные дешифровывались последовательно (2500 раз). Т.к. считывание температуры по DMA выполняется медленнее дешифрования.

Температурные показания по DMA на протяжении выполнения дешифрования сохранялись в массив, как только последние данные были дешифрованы, считывание температуры прекращалось и все записанные значения встроенного температурного датчика с АЦП отправлялись по UART на Com порт компьютера (Рисунок 6.7).

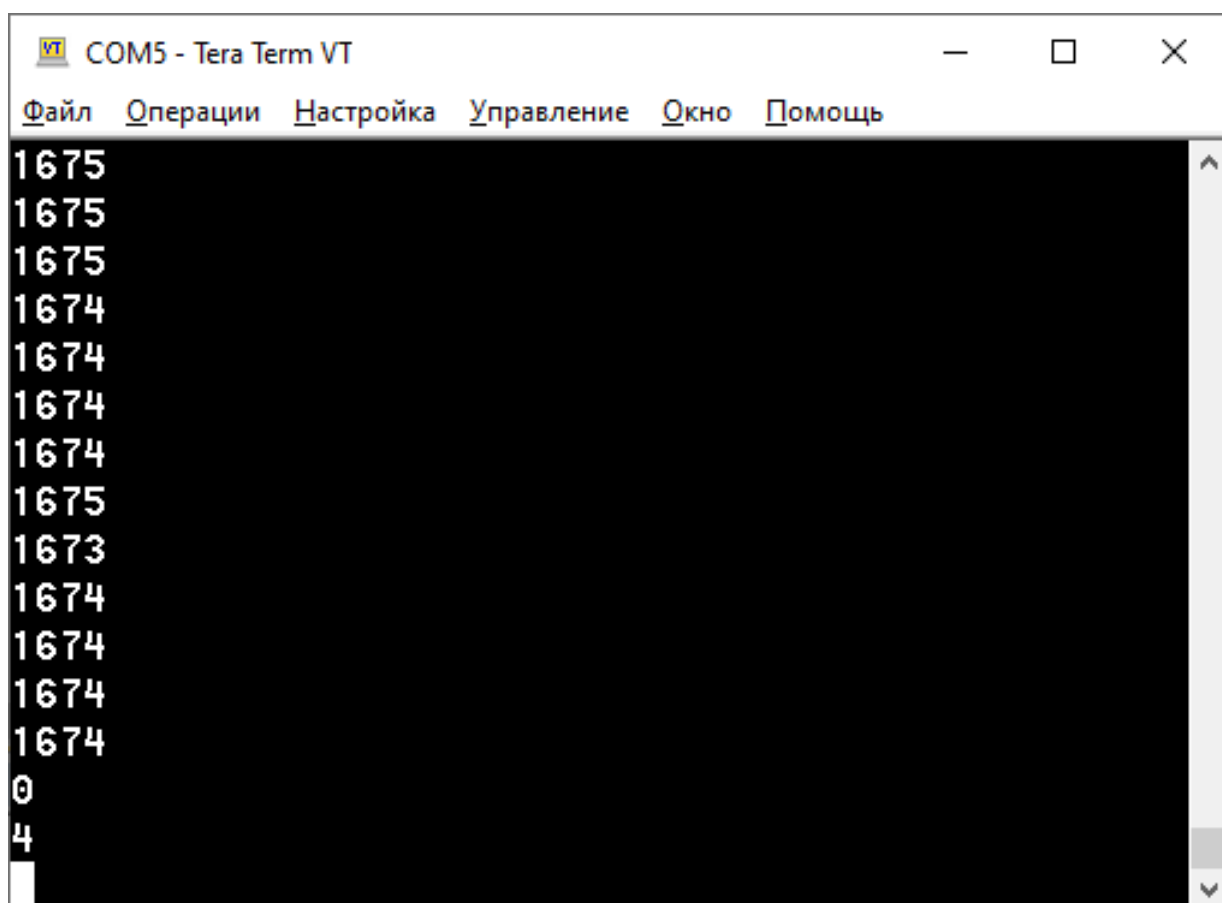


Рисунок 6.7 –Значения АЦП температурного датчика STM32

Полученные данные согласно формулам из справочного руководства[40] (Рисунок 6.8) не трудно перевести в градусы Цельсия:

$$\text{Temperature (in } ^\circ\text{C)} = \frac{V_{30} - V_{\text{SENSE}}}{\text{Avg_Slope}} + 30 ^\circ\text{C}$$
$$V_{\text{SENSE}} = \frac{\text{TS_DATA}}{4095} \times V_{\text{DD}}$$

Рисунок 6.8 – Формула конвертации значения АЦП в градусы Цельсия

7 Анализ полученных данных.

Утечка температуры линейно связана с моделью утечки мощности, но ограничена физическим свойством теплопроводности и емкости. Также стоит отметить, что дешифрование различных данных затрачивает разное время на выполнение алгоритма. Таким образом по количеству полученных измерений по DMA можно определить время работы алгоритма. В таблице 7.1 представлены символы от «0» до «9», их ASCII значения, результат их шифрования, а также количество измерений, полученных в процессе дешифровки. Каждый символ дешифровывался последовательно 2500 раз. За время выполнения дешифрования были получены значения с АЦП температурного датчика для каждого символа. Так как для каждого символа требуется разное время на дешифрование, количество полученных измерений отличается друг от друга.

Таблица 7.1 – Количество полученных измерений для различных символов ASCII символов

Symbol	ASCII	Encrypted	Measurements
0	48	110592	2211
1	49	117649	2198
2	50	125000	2222
3	51	132651	2213
4	52	140608	2207
5	53	148877	2202
6	54	157464	2204
7	55	166375	2229
8	56	175616	2219
9	57	185193	2223

По количеству измерений можно получить данные о том сколько раз дешифровывалось сообщение. При дешифровании 1250 раз сообщения с символом «1» и 1250 раз сообщения с символом «2» общее количество измерений составило 2210. К примеру, если выполнять шифрование 2500 раз символа «1» на микроконтроллере мы можем получить только 168 измерений, т.к. для шифрования используется открытая экспонента ($e = 3$), в отличии от

дешифрования ($d = 2668186619$). Если шифруемое сообщение приблизить к $n = 4002689923$, то в процессе дешифровки алгоритм будет выполнять инструкции деления по модулю и зашифровывая 2500 раз сообщение $MSG = 4002689920$ удалось получить 175 измерений.

Для получения температурных измерений на протяжении выполнения дешифрования – зашифрованные данные заранее записывались в код исполняемой программы на stm32. Каждые данные дешифровывались отдельно и последовательно 2500 раз. Это было необходимо для того чтобы получить около 2000 температурных измерений, т.к. дешифрование одного сообщения выполняется быстрее чем ПДП. Общая схема получения экспериментальных данных показана на рисунке 7.1.

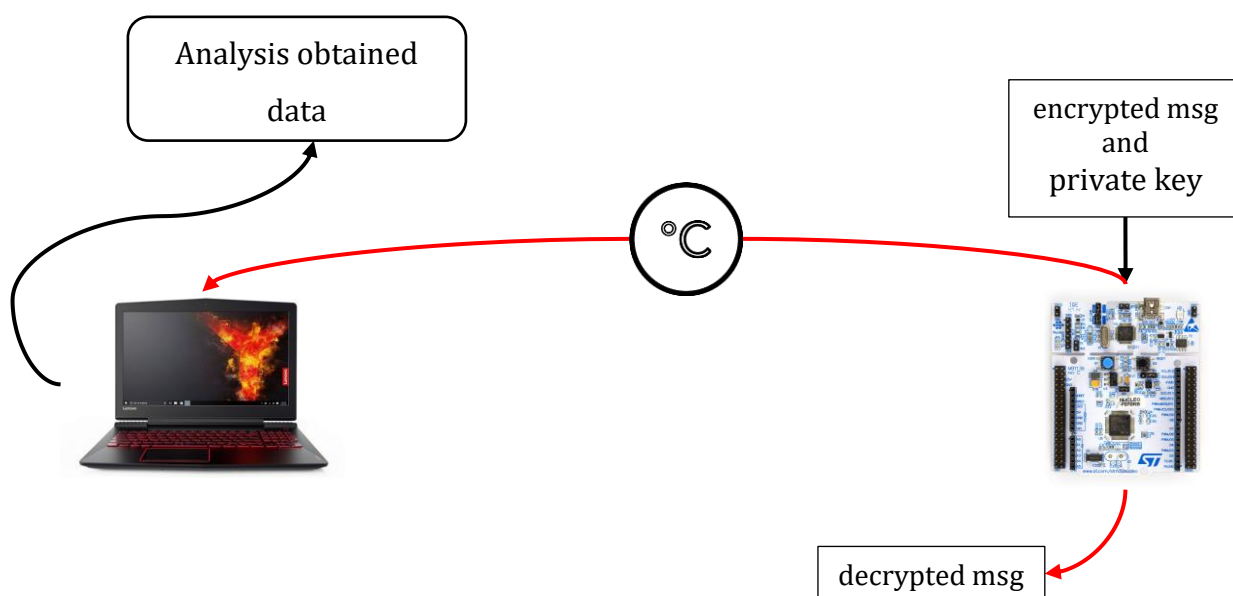


Рисунок 7.1 – Схема получения экспериментальных данных

Для того чтобы увидеть неискаженные данные, показания снимались при закрытой программе «Keil uVision», так как при включенном отладчике микроконтроллер потребляет большой ток, а это непосредственно влияет на температуру процессора. Дешифрованное сообщение оставалось в памяти Nucleo - STM32f070Rb. После завершения последнего дешифрования – по UART

отправлялись сохраненные данные температуры при помощи ПДП на протяжении выполнения всего алгоритма. После этого полученные данные при помощи вспомогательных программ подвергались анализу.

Эксперименты выполнялись в комнатных условиях при температуре 24.3 °C (Рисунок 7.2). Значения внешних условий были получены при помощи датчика DHT11[41] подключенного к микроконтроллеру ATmega328P-AU установленному на плате разработчика UNO R3 [Atmega 328P-AU+CH340G][42] свободному COM порту компьютера.

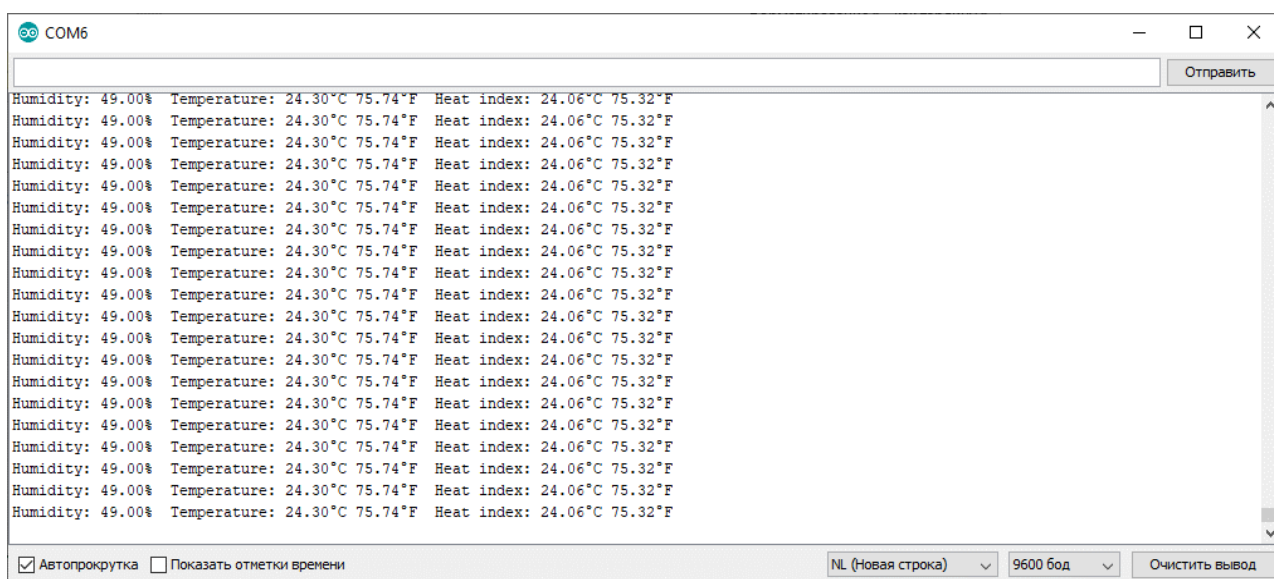


Рисунок 7.2 – Температура в комнате

Аналогичные эксперименты проводились для дешифрования данных из таблицы 6.2. На рисунке 7.3 представлены графики температурных изменений для сообщений 1-4 из таблицы 6.2 а также их усредненное значение на рисунке 7.4. Для каждого сообщения измерения проводились 4 раза и на основе полученных результатов строилось среднее значение температуры. Начальная температура для всех экспериментов искусственно была уставлена как 44,38 °C. Основное отличие данных графиков в количестве полученных измерений. Для сообщения «1500» потребовалось больше времени на дешифровку, а следовательно, получено большее количество измерений температуры (2216).

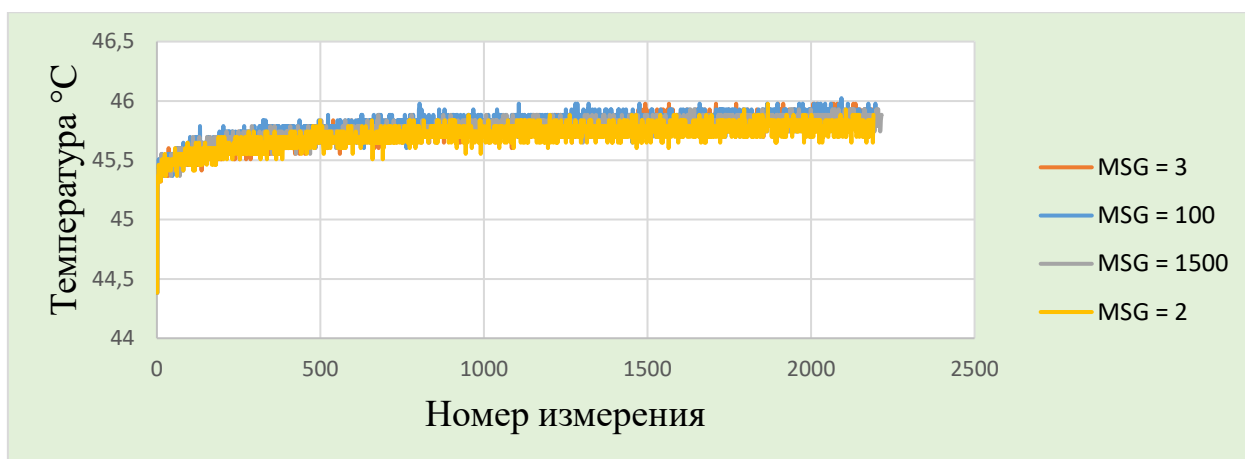


Рисунок 7.3 – Значения температуры для сообщений 1-4

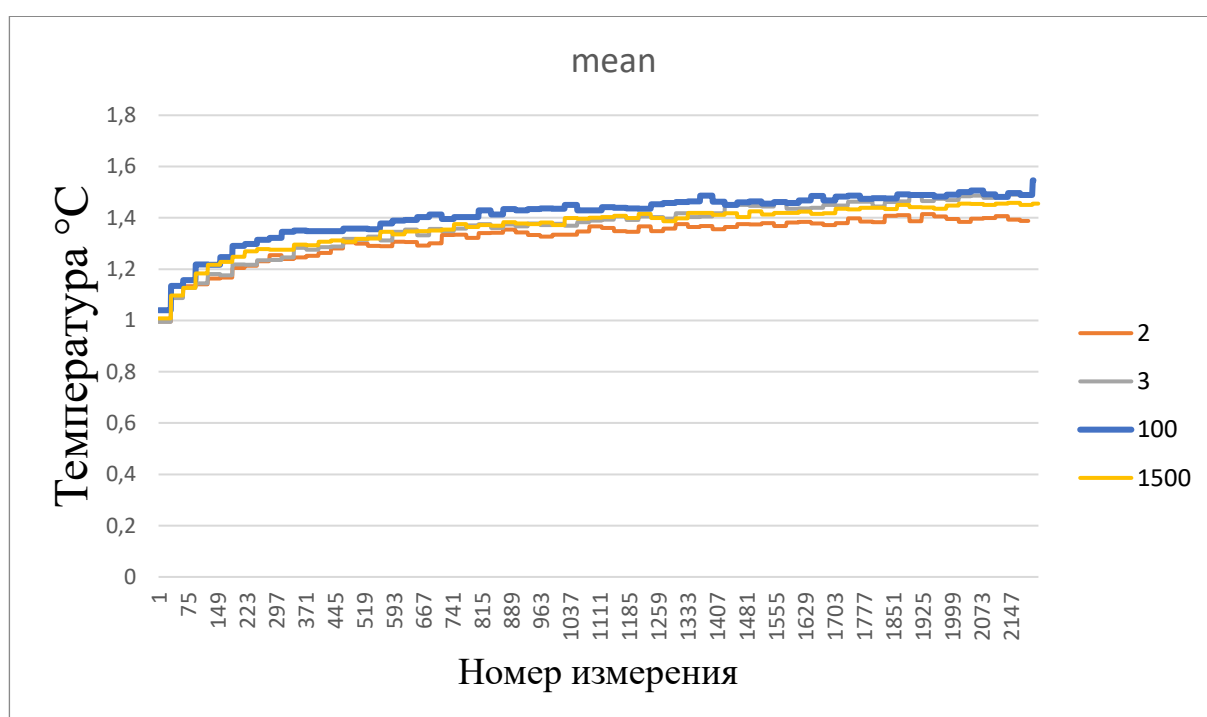


Рисунок 7.4 – Усредненные значения температуры для сообщений 1-4

Температура инициализации для данных измерений составляла от 44,5 до 44,3°C. Данный эксперимент также выполнялся и для значений 5 - 8 таблицы 6.2 (Рисунок 7.5).

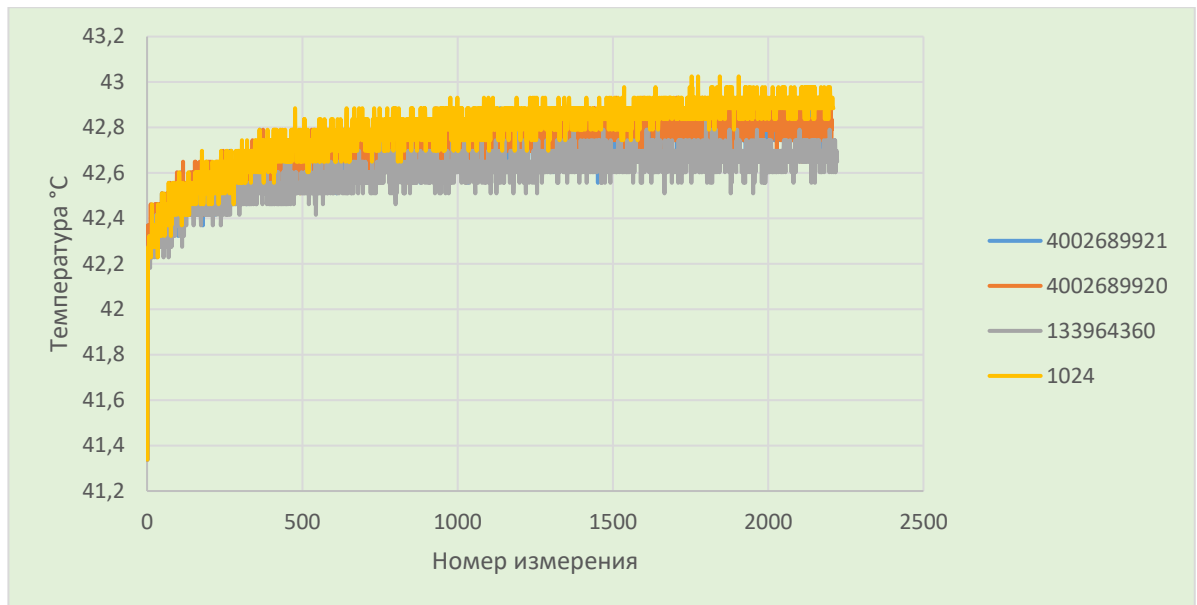


Рисунок 7.5 – Значения температуры процессора для сообщений 5-8

Из рисунков медленного нагрева процессора видно, что температура увеличивается очень медленно, на 1°C за первые 2 секунды, после этого по $0,05^{\circ}\text{C}$ за каждую секунду. Исходя из полученных данных можно сделать вывод: атака, использующая температурный боковой канал, возможна в случае утечки данных в течение нескольких миллисекунд или секунд.

Расположив температурные кривые последовательно, можно увидеть некоторые отличия в нарастаниях температуры для сообщений из таблицы 6.2 (Рисунок 7.6).

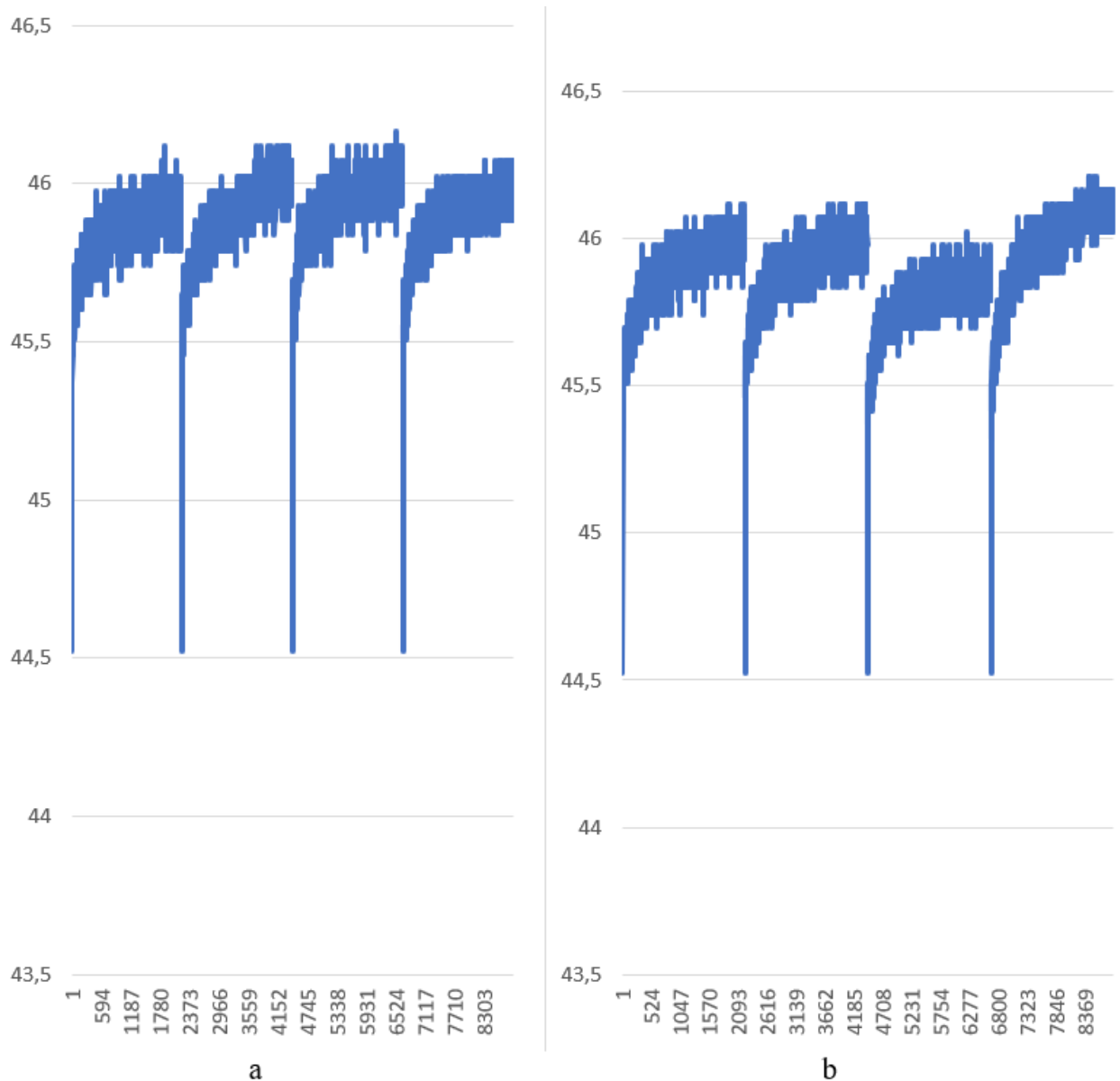


Рисунок 7.6 – Графики нарастания температуры сообщений таблицы 4.2

Также данный эксперимент проводился для всех символов таблицы 5.1, но при различных ключах шифрования, полученные зашифрованные данные и ключи при которых они были рассчитаны приведены в таблице 5.2. Каждое сообщение было дешифровано 2500 раз, при этом из-за разных размеров ключей, удалось получить различное количество температурных измерений, так как при дешифровании количество итераций в функции возведения в степень по модулю[43] отличается (Рисунок 7.7).

```

/* USER CODE BEGIN PFP */
int powMod(unsigned long long a, unsigned long long b, unsigned long long n)
{
    long long x = 1, y = a;
    while (b > 0) {
        if (b % 2 == 1)
            x = (x%n * y) % n;
        y = (y%n * y) % n; // (a*b) mod n = (a*(b mod n) mod n)
        b /= 2;
    }
    return x % n;
}
/*< MSG = powMod(Encrypted, d, n); */
}

/* USER CODE END PFP */

```

Рисунок 7.7 – Функция возведения в степень по модулю

Данная функция является основной при шифровании и дешифровании данных алгоритмом RSA и процедура выполнения алгоритма очень напоминает принцип умножения по методу русских крестьян [44].

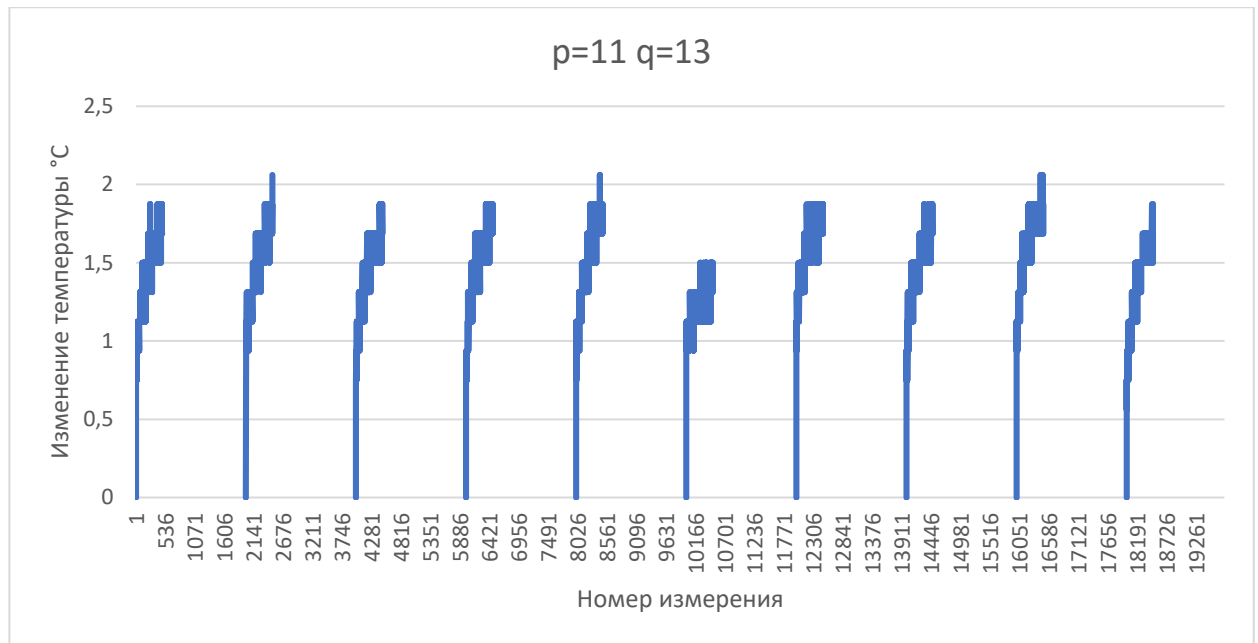
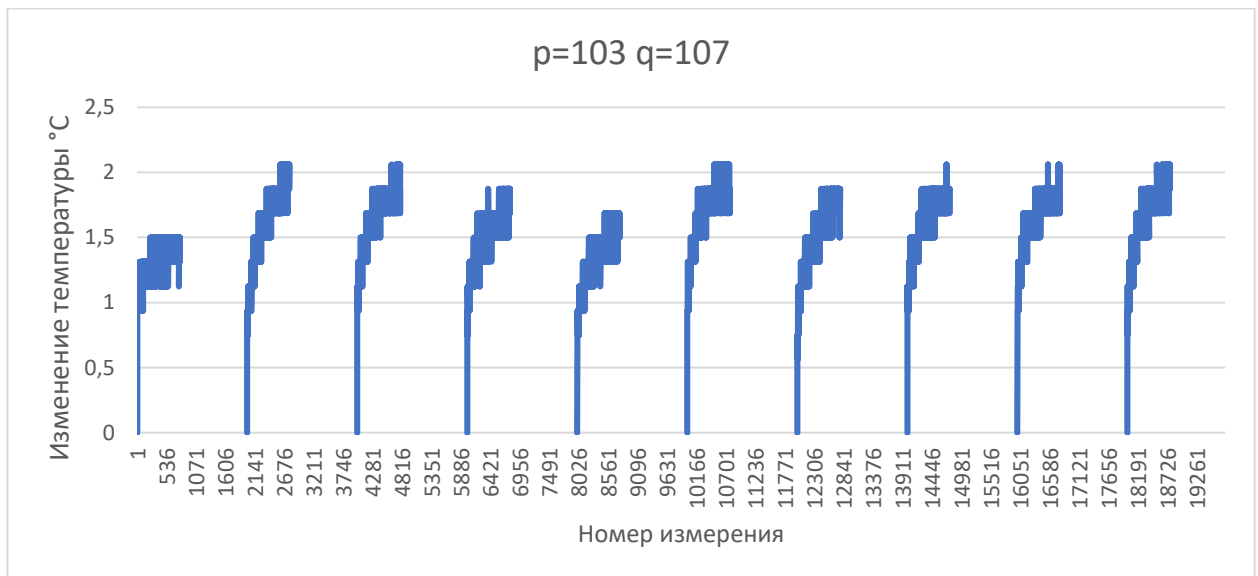
От того как реализован алгоритм возведения в степень по модулю в криптосистеме RSA зависит не только скорость выполнения шифрования/дешифрования, но и данные, которые возникают в побочном канале. Выделяют несколько основных методов возведения в степень по модулю[45]:

- Метод с использованием Китайской теоремы об остатках[46];
- Алгоритм Монтгомери[47];
- Метод повторяющихся возведения в квадрат и умножения.

Таблица 7.2 – Шифрование данных таблицы 7.1 разными ключами

Experiment		1	2	3	4	5
p		11	103	4993	10007	10007
q		13	107	4999	10009	399989
n		143	11021	24960007	100160063	4002689923
$\varphi(n)$		120	10812	24950016	100140048	4002279928
e		7	5	5	5	3
d		103	4325	19960013	60084029	2668186619
Symbol	ASCII	Encrypted data				
0	48	126	9469	5203898	54483842	110592
1	49	36	7019	7915172	82155123	117649
2	50	41	10566	12979916	12019811	125000
3	51	116	1825	20545160	44545062	132651
4	52	13	1574	5803927	79723843	140608
5	53	92	3648	18835381	17555241	148877
6	54	76	8122	9884898	58524772	157464
7	55	55	10410	4084235	2484060	166375
8	56	56	1385	1611622	49931461	175616
9	57	73	562	2651889	731679	185193

В процессе дешифрования каждого сообщения были получены температурные значения, которые отображены на рисунках 7.8-7.12. Для того чтобы отобразить все графики на одном уровне – первое измерение было принято за 0.

Рисунок 7.8 – Графики роста температуры при $p=11 \quad q=13$ Рисунок 7.9 – Графики роста температуры при $p=103 \quad q=107$

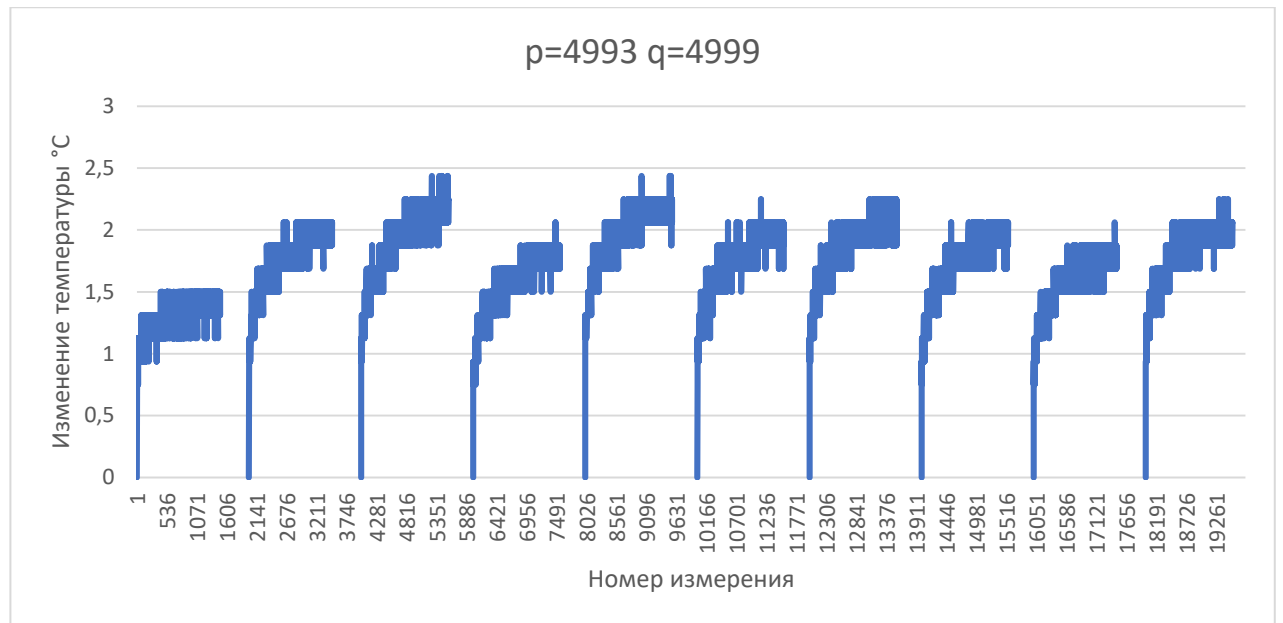


Рисунок 7.10 – Графики роста температуры при $p=4993 \quad q=4999$

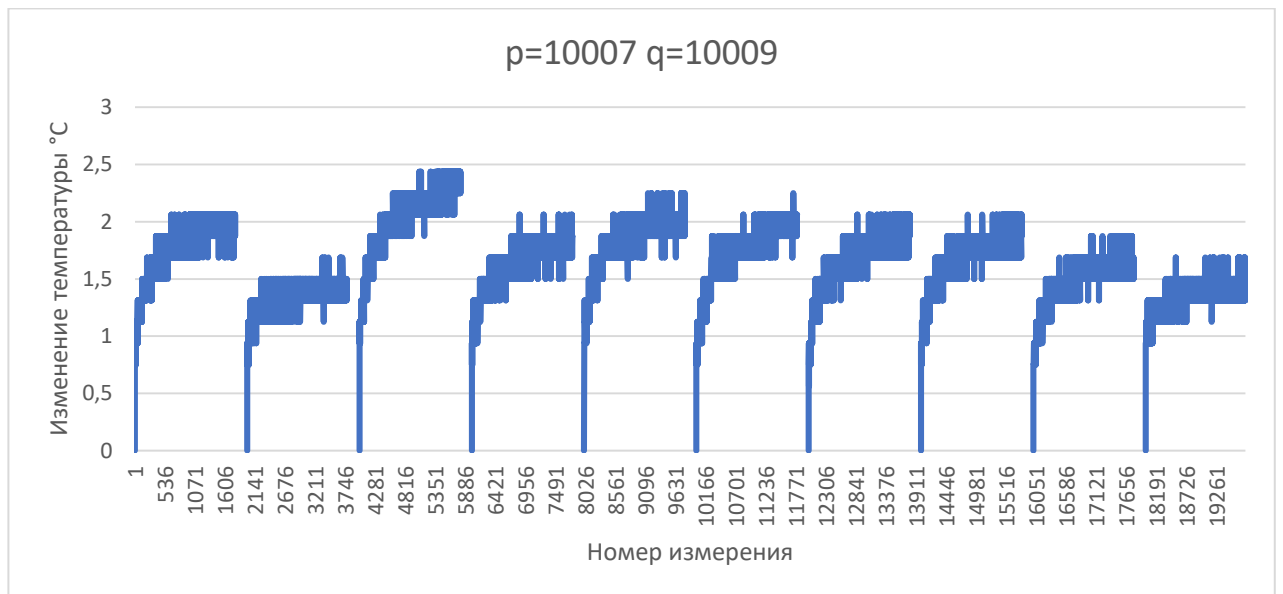


Рисунок 7.11 – Графики роста температуры при $p=10007 \quad q=10009$

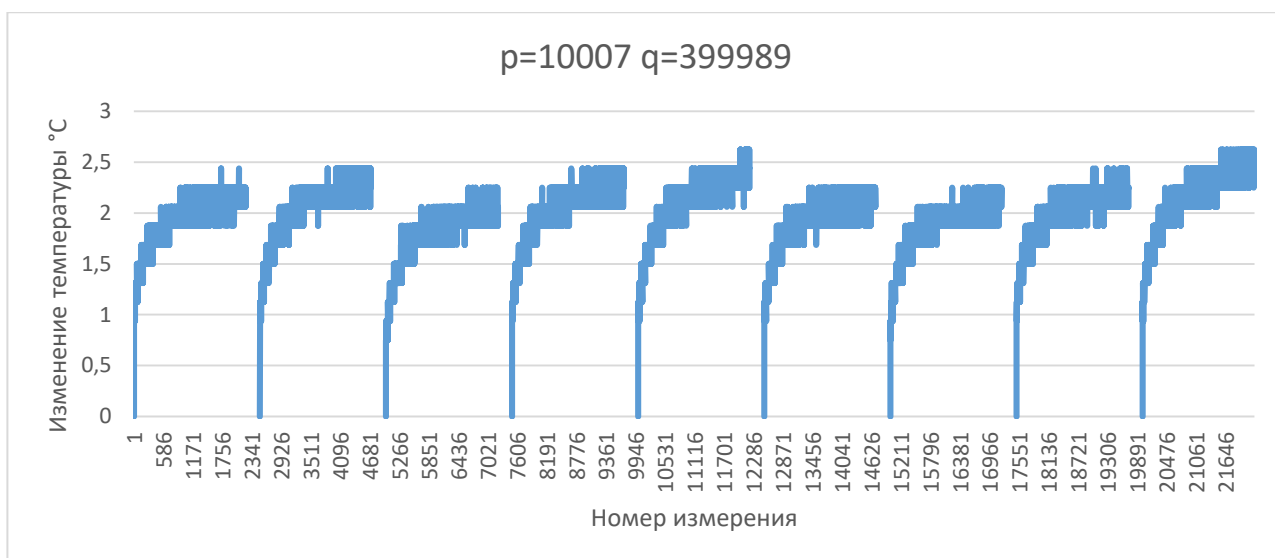


Рисунок 7.12 – Графики роста температуры при $p=10007$ $q=399989$

Количество измерений для каждого символа при различных ключах отличается и чем больше размер ключа, тем большее количество измерений удаётся получить. На рисунках 7.13 – 7.16 изображены графики, показывающие количество полученных измерений, при шифровании ключами разных размеров.

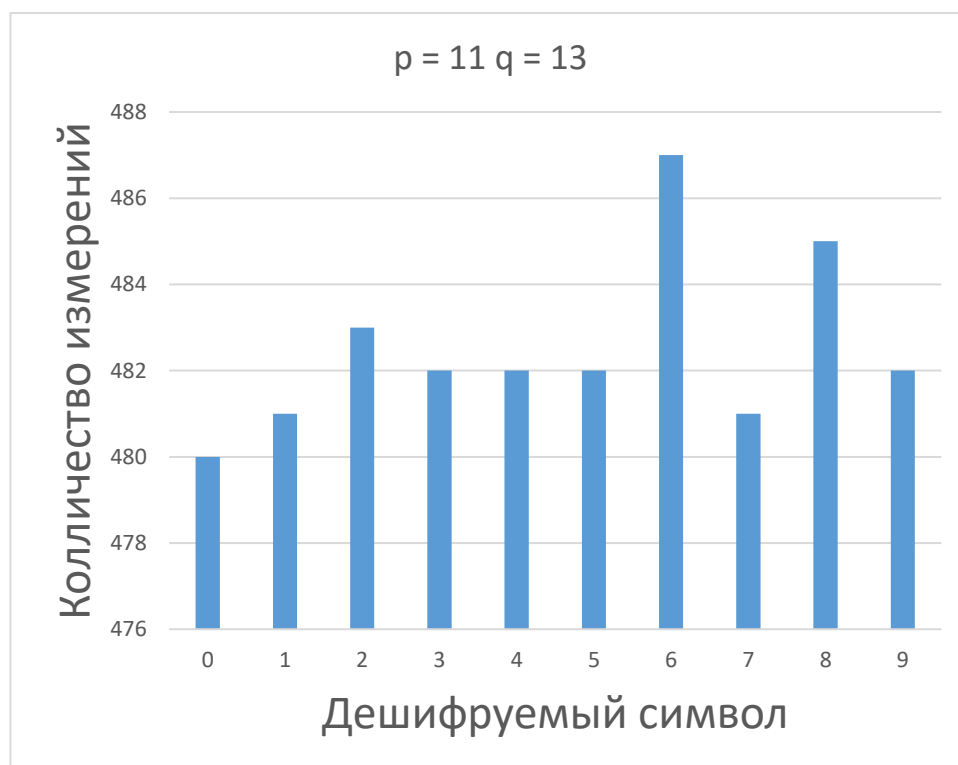


Рисунок 7.13 – Количество измерений при $n = 143$

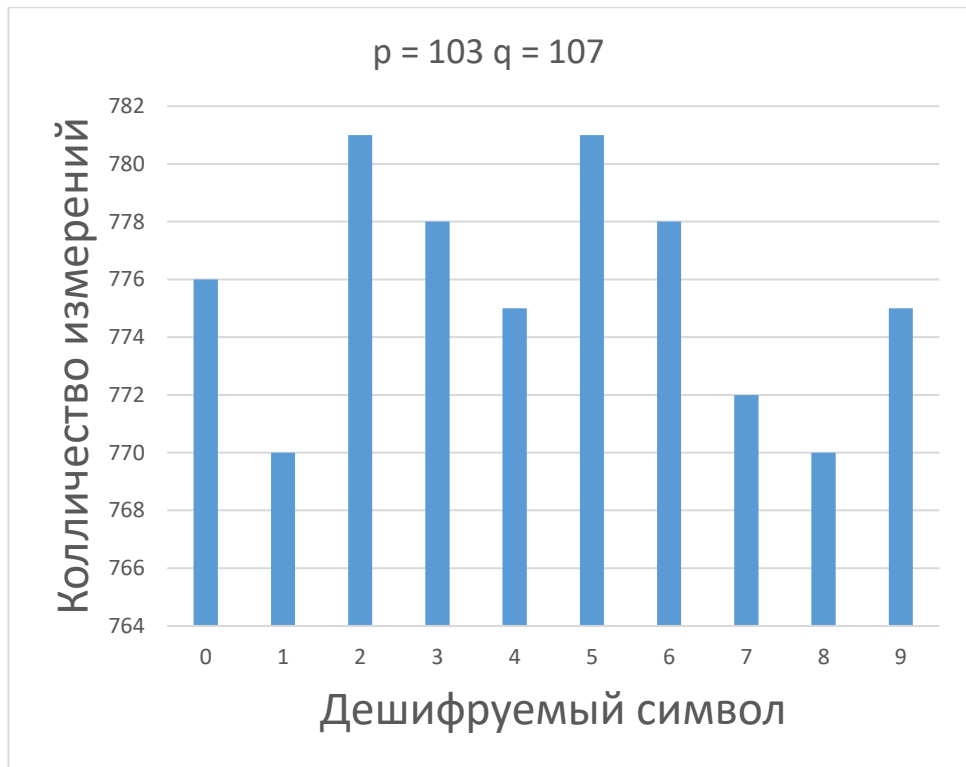


Рисунок 7.14 – Количество измерений при $n = 11021$

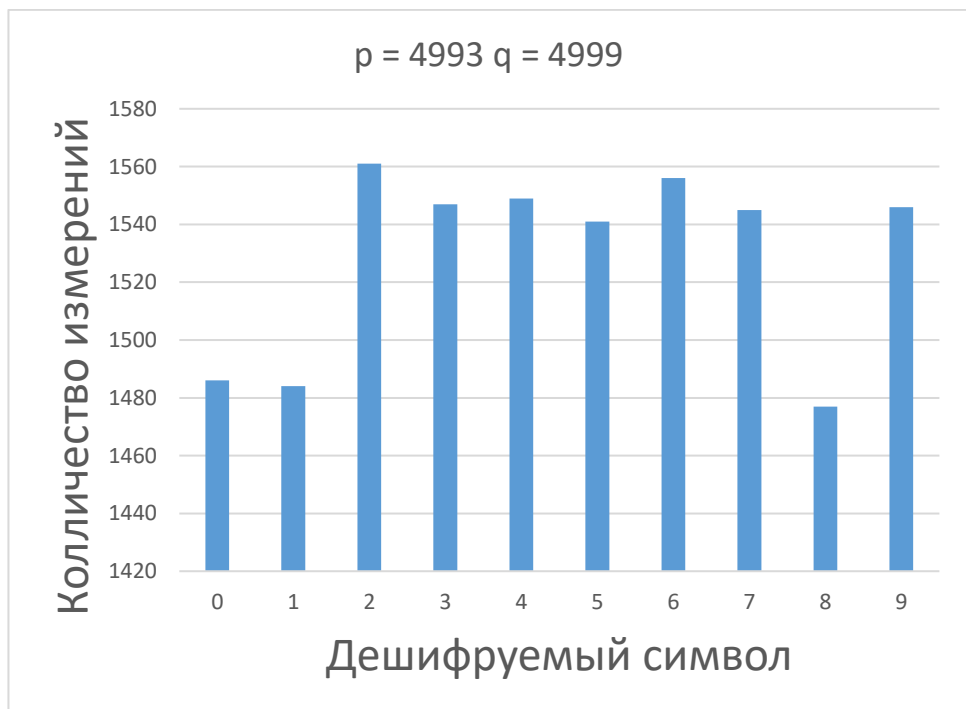


Рисунок 7.15 – Количество измерений при $n = 24960007$

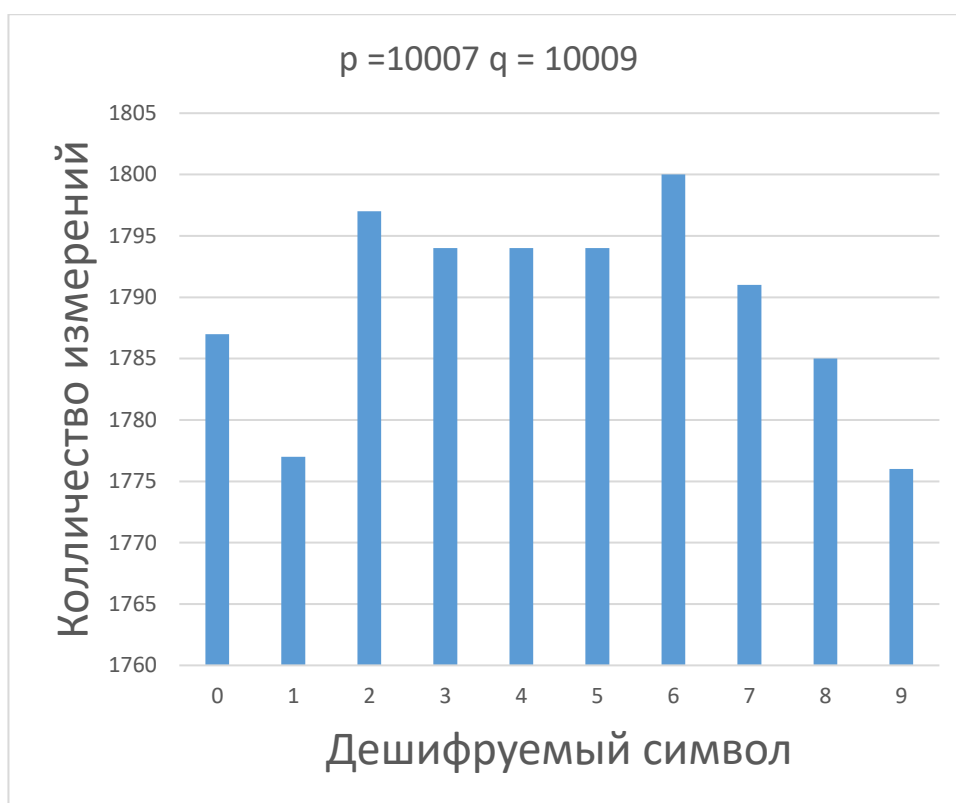


Рисунок 7.16 – Количество измерений при $n = 100160063$

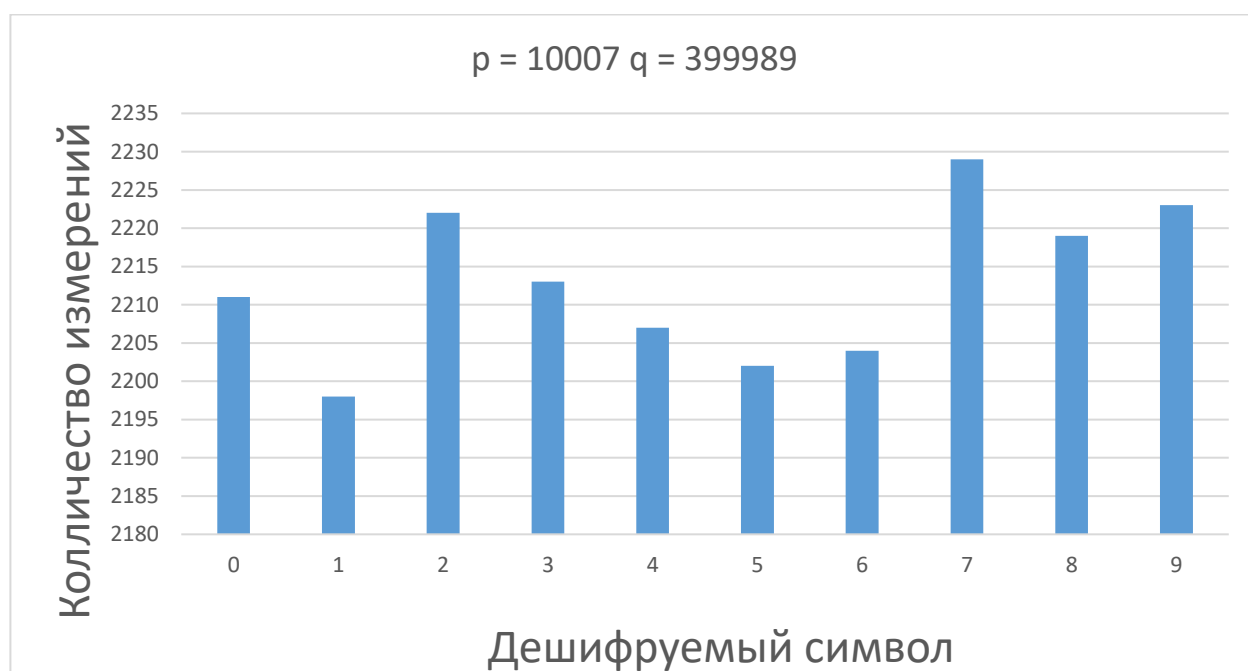


Рисунок 7.17 – Количество измерений при $n = 4002689923$

Поскольку зашифрованное сообщение было одинаковым для всех экспериментов, мы можем сделать вывод, что температура процессора зависит не только от дешифрованного сообщения, но и от используемых ключей. Это

связано с тем, что для сообщений, в которых размер ключа был больше, требовалось больше времени для выполнения всего алгоритма.

Так как температура процессора напрямую зависит от потребляемого микросхемой тока, было решено измерить потребляемый ток процессором. В режиме ожидания микросхема потребляет меньший ток (Рисунок 7.18), в отличие от потребления во время выполнения кода программы.

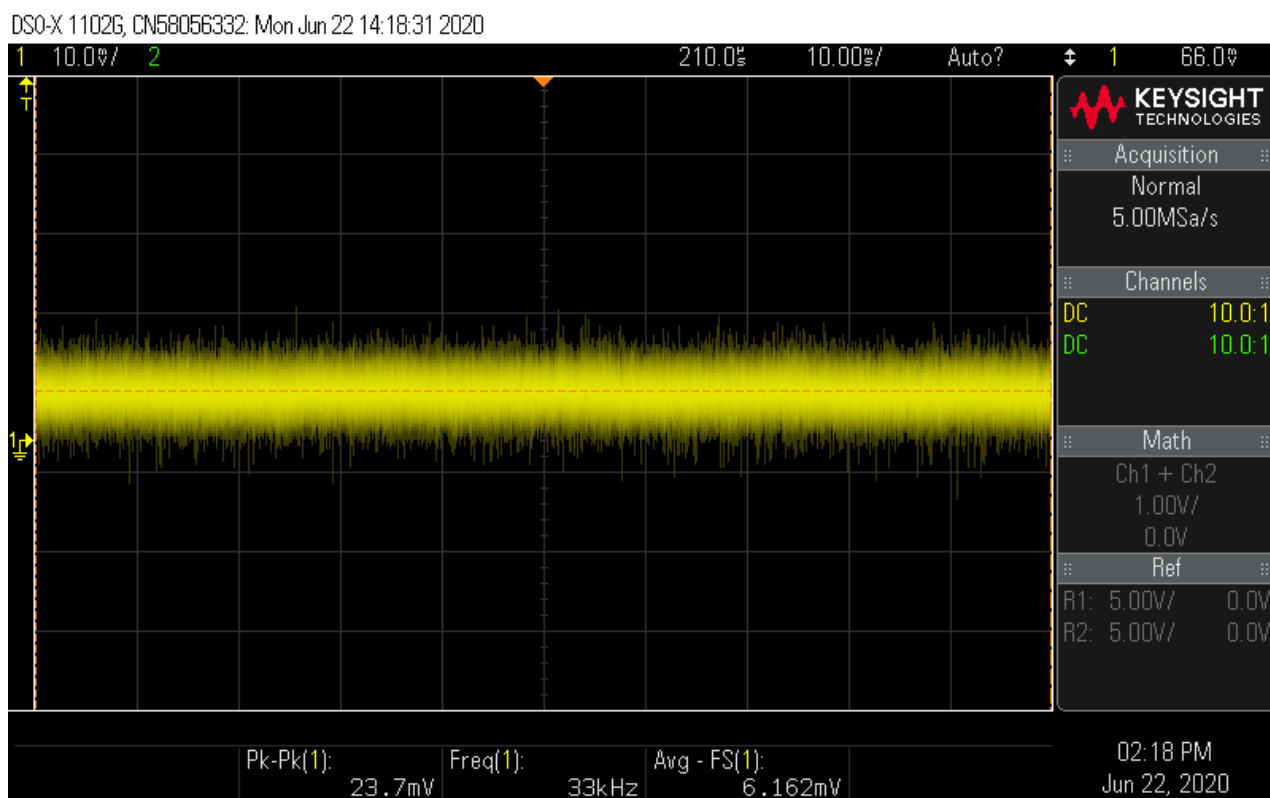


Рисунок 7.18 – Потребляемый ток процессором в режиме ожидания

Измерения потребляемого тока снимались с JP6 при помощи осциллографа keysight через резистор номиналом 1Ом, по схеме представленной на рисунке 7.19.

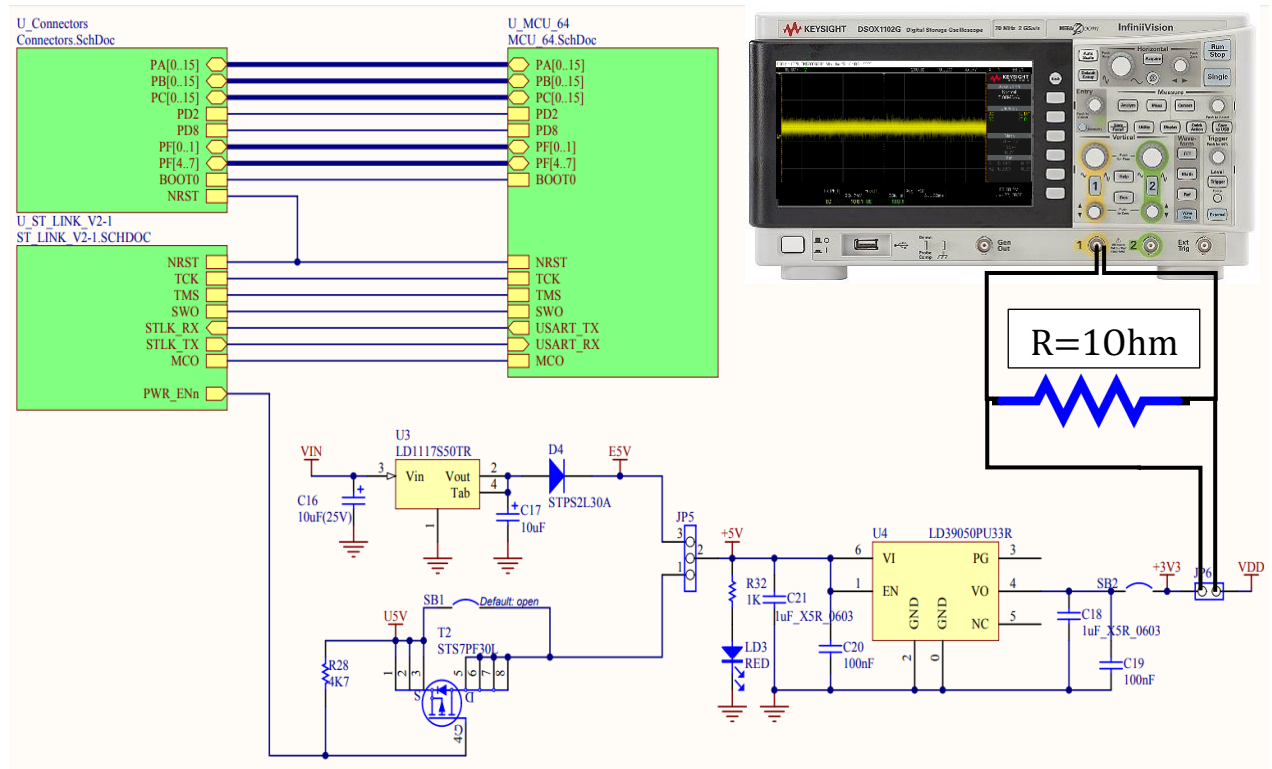


Рисунок 7.19 – Измерение потребляемого тока процессором

Для того чтобы убедиться в работоспособности теории был реализован код, который в бесконечном цикле дешифровывал одно сообщение, а при нажатии кнопки – начиналась дешифровка другого. Полученные результаты отличались на 700мкА, но после анализа полученных результатов была определена причина такого различия. Причиной повышения тока была кнопка «B1». Данная кнопка подключена к подтягивающему резистору R30 (Рисунок 7.20). Номинал резистора R30 составляет 4,7кОм. Разделив по закону ома $4,7/3,3$ получим искомые 700мкА.

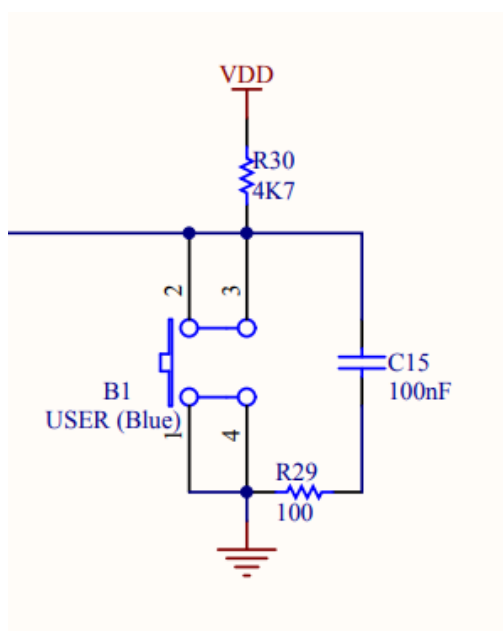


Рисунок 7.20 – Схема подключения кнопки B1 на Nucleo STM32F070Rb

После внесения правок было выяснено что в процессе выполнения дешифрования потребляемый ток процессором составляет в районе 20 мкА.

Изменения потребляемого тока небольшие в зависимости от дешифруемого сообщения не большое, но при достаточном количестве данных можно извлечь из совокупности некоторую информацию. Ниже на рисунках 7.21 – 7.36 приведены графики потребляемого тока и графики роста температуры процессора в течении дешифрования, для данных из таблицы 6.2, при ключах рассчитанных по $p = 10007$ и $q = 39998$ соответственно. Для каждого эксперимента было проведено несколько измерений, по которым строились средние значения нарастания температуры процессора.

DSO-X 1102G, CN58056332, Mon Jun 22 14:05:26 2020

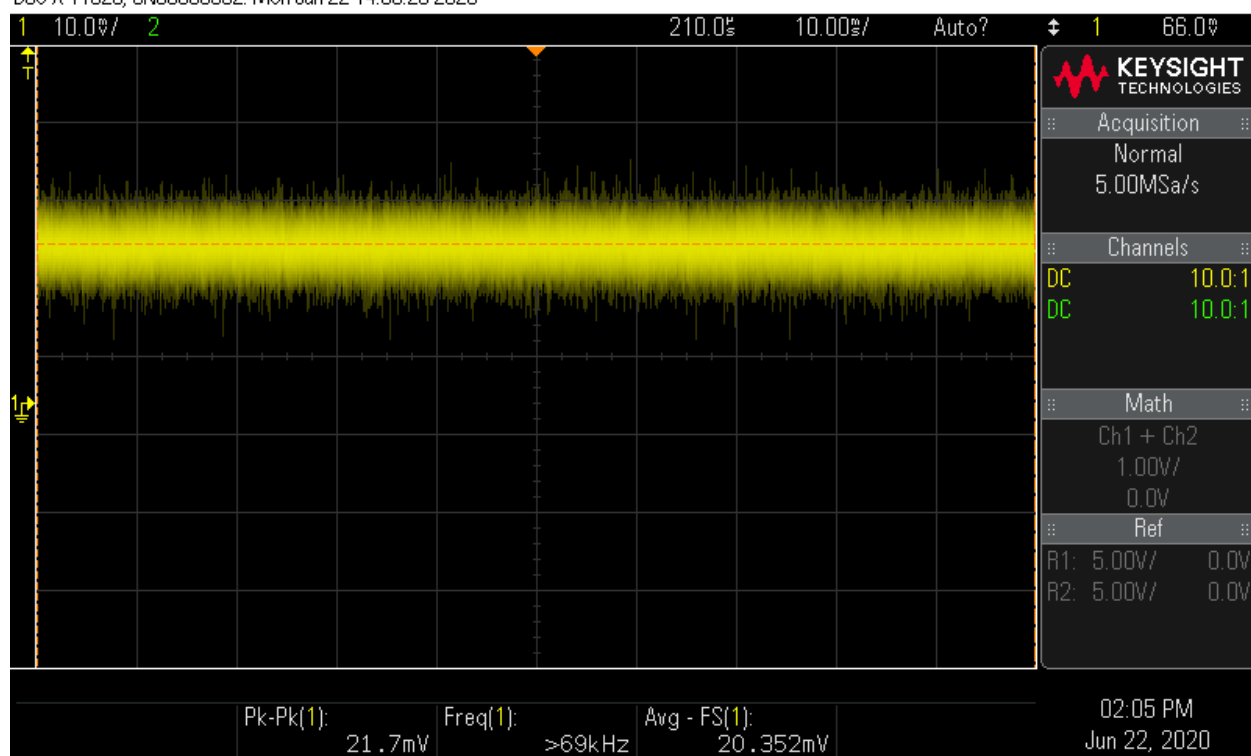


Рисунок 7.21 – Потребляемый ток при дешифровании числа 8

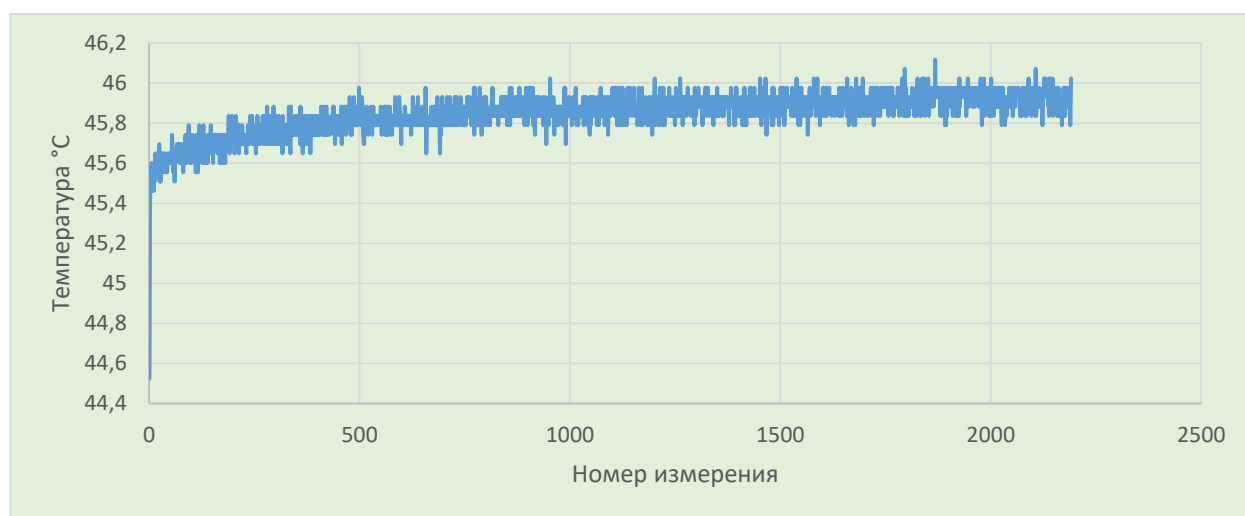


Рисунок 7.22 – Средний рост температуры процессора при дешифровании числа 8

Измерения начинались при температуре процессора равной 44,52423545 °C, за 2500 дешифрований процессор нагрелся до температуры 46,02351137 °C. При этом количество полученных измерений составило 2191

DSO-X 1102G, CN58056332: Mon Jun 22 14:06:09 2020

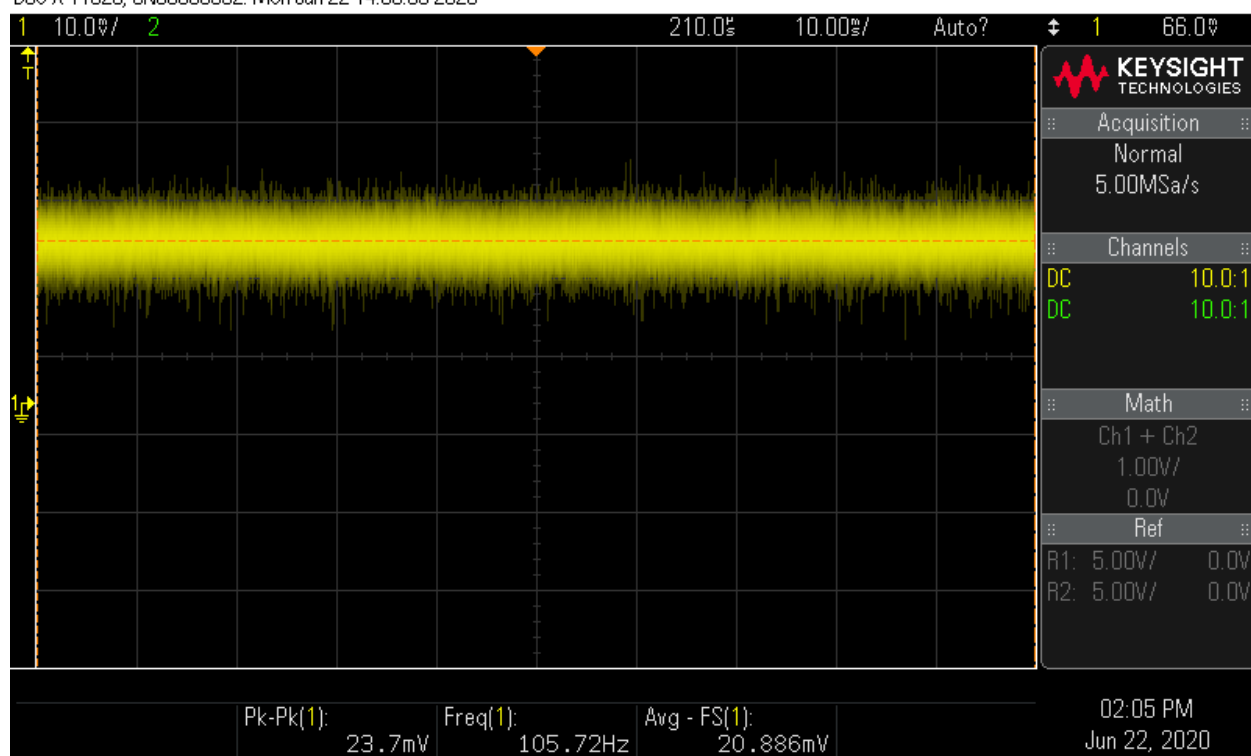


Рисунок 7.23 – Потребляемый ток при дешифровании числа 27

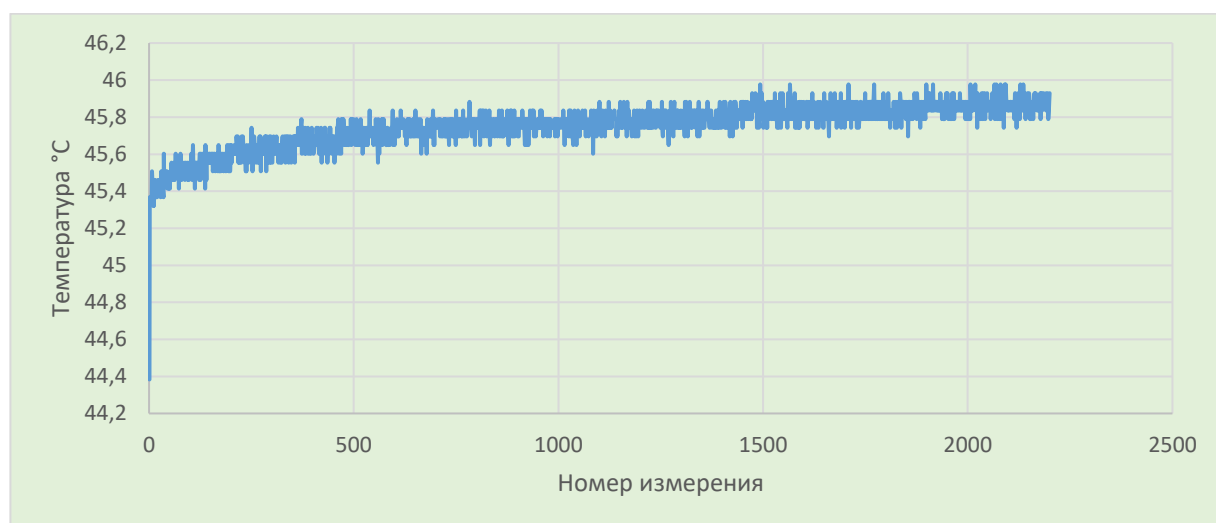


Рисунок 7.24 – Средний рост температуры процессора при дешифровании числа 27

Первое измерение температуры составило 44,38367834 °C, в результате процессор нагрелся до 45,92980663 °C. Количество измерений 2200.

DSO-X 1102G, CN58056332, Mon Jun 22 14:06:41 2020

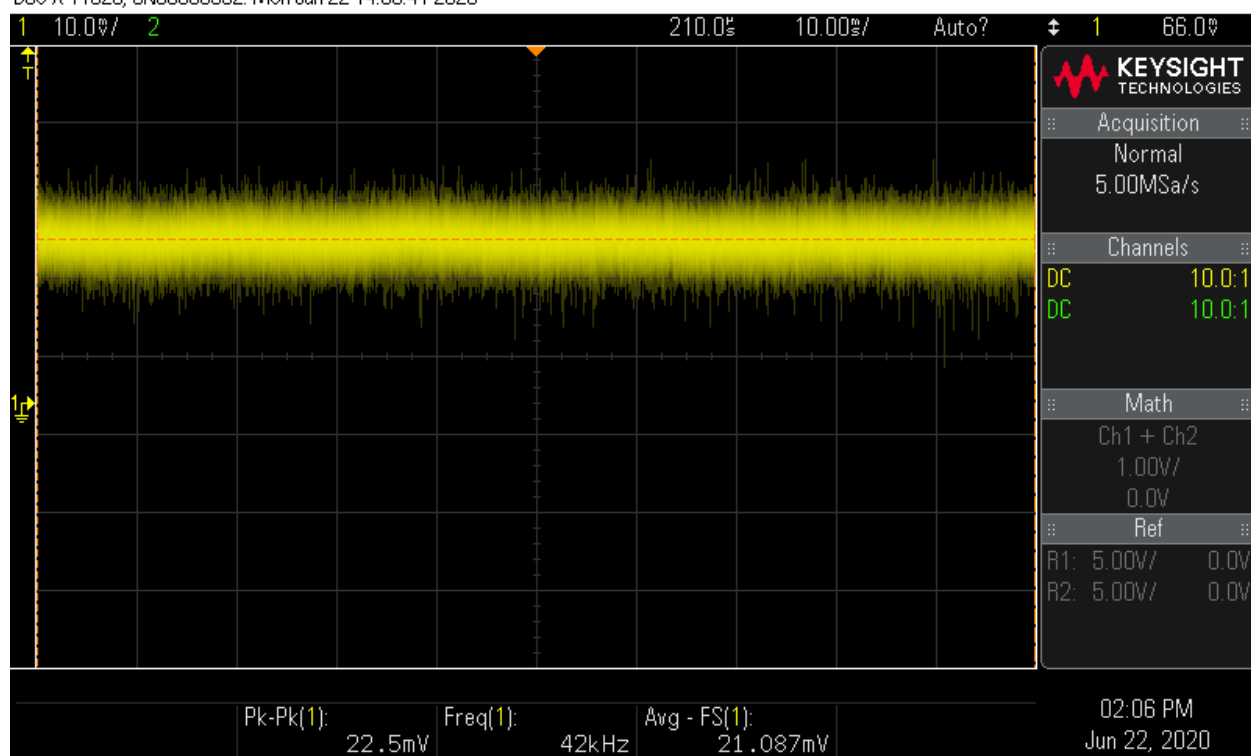


Рисунок 7.25 – Потребляемый ток при дешифровании числа 1000000

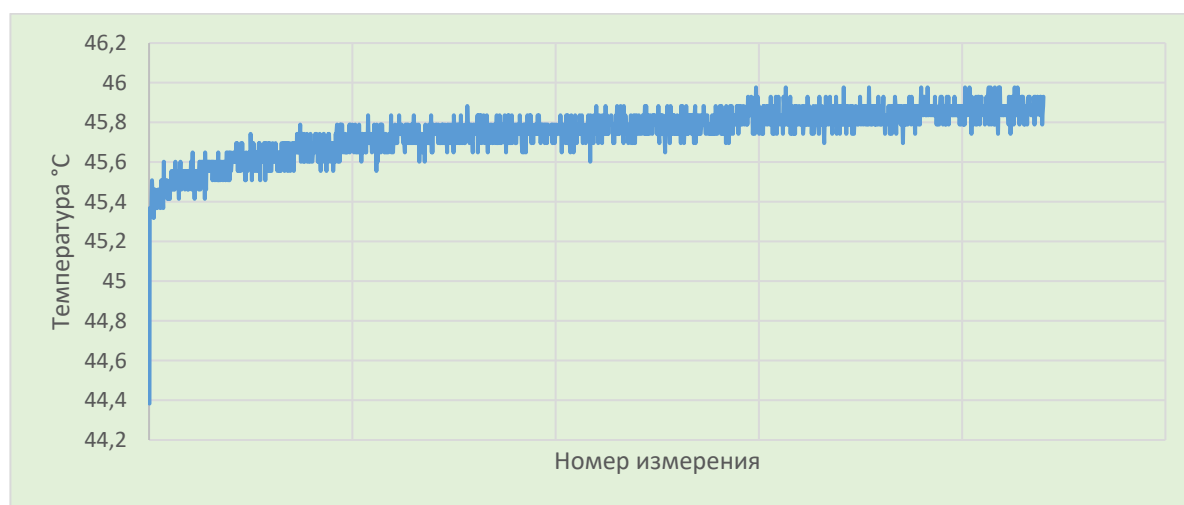


Рисунок 7.26 – Средний рост температуры процессора при дешифровании числа 1000000

Начальная температура процессора для данного эксперимента была 44,38367834 °C, при дешифровании числа 1000000 2500 раз процессор нагрелся до температуры 45,92980663 °C, за 2204 измерения.

DSO-X 1102G, CN58056332, Mon Jun 22 14:08:02 2020

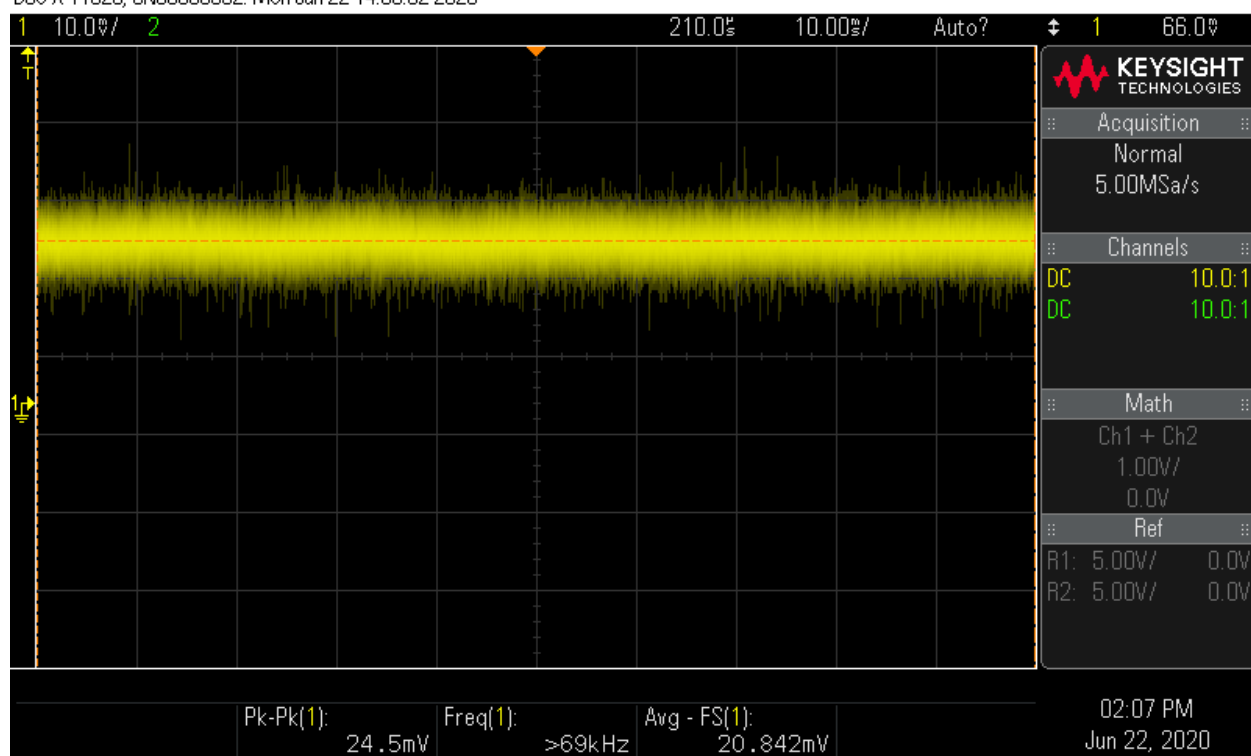


Рисунок 7.27 – Потребляемый ток при дешифровании числа 3375000000

Рисунок 7.28 – Средний рост температуры процессора при дешифровании
числа 3375000000

Начальная температура была 44,43053071 °C, конечная 45,92980663 °C, всего удалось получить 2216 температурных показаний.

DSO-X 1102G, CN58056332: Mon Jun 22 14:09:00 2020

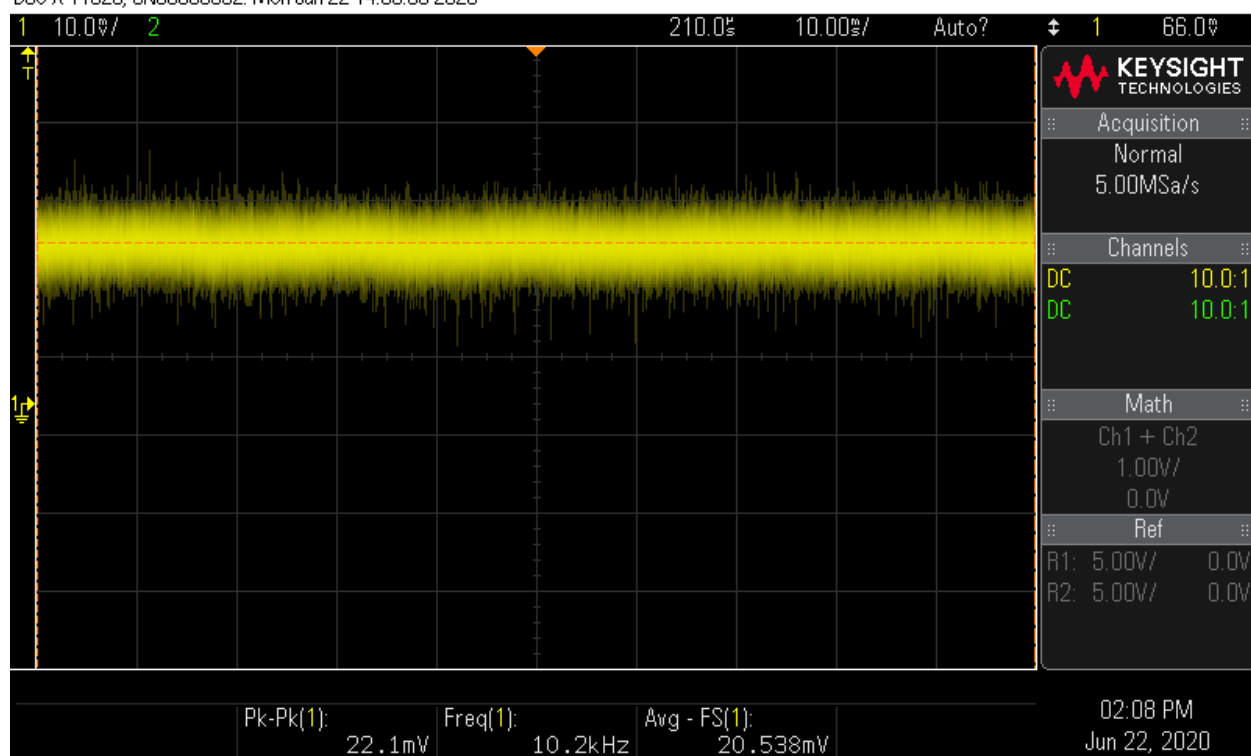


Рисунок 7.29– Потребляемый ток при дешифровании числа 4002689915

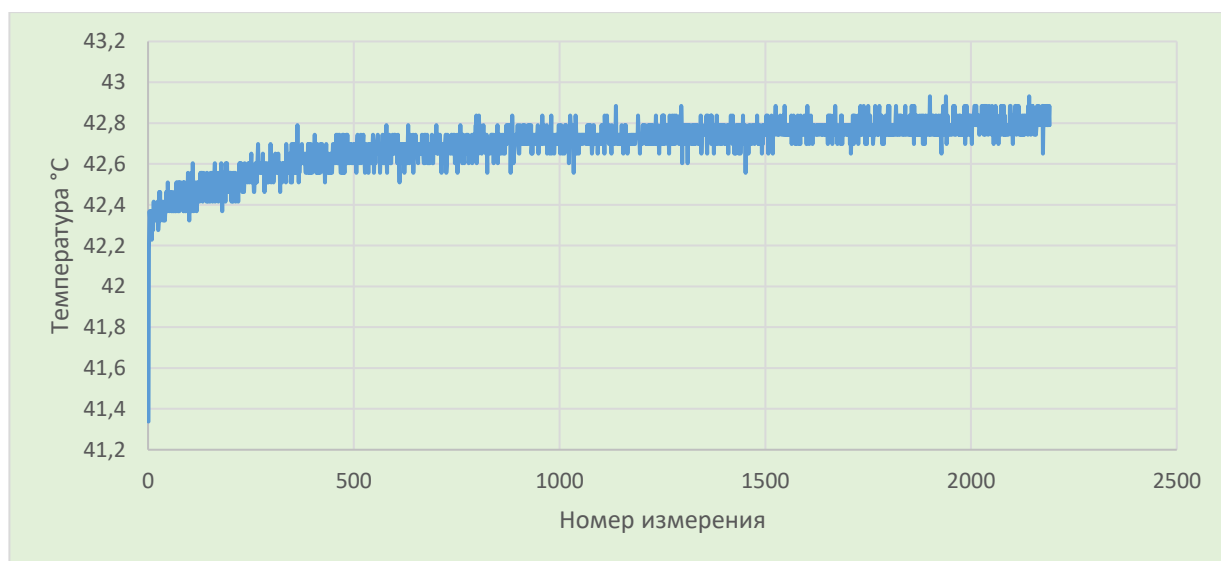


Рисунок 7.30 – Средний рост температуры процессора при дешифровании числа 4002689915

Измерения начались при температуре 41,33827413 °C, после окончания дешифрования температура процессора стала 42,93125479 °C, всего удалось получить 2208 температурных измерений.

DSO-X 1102G, CN58056332, Mon Jun 22 14:09:48 2020

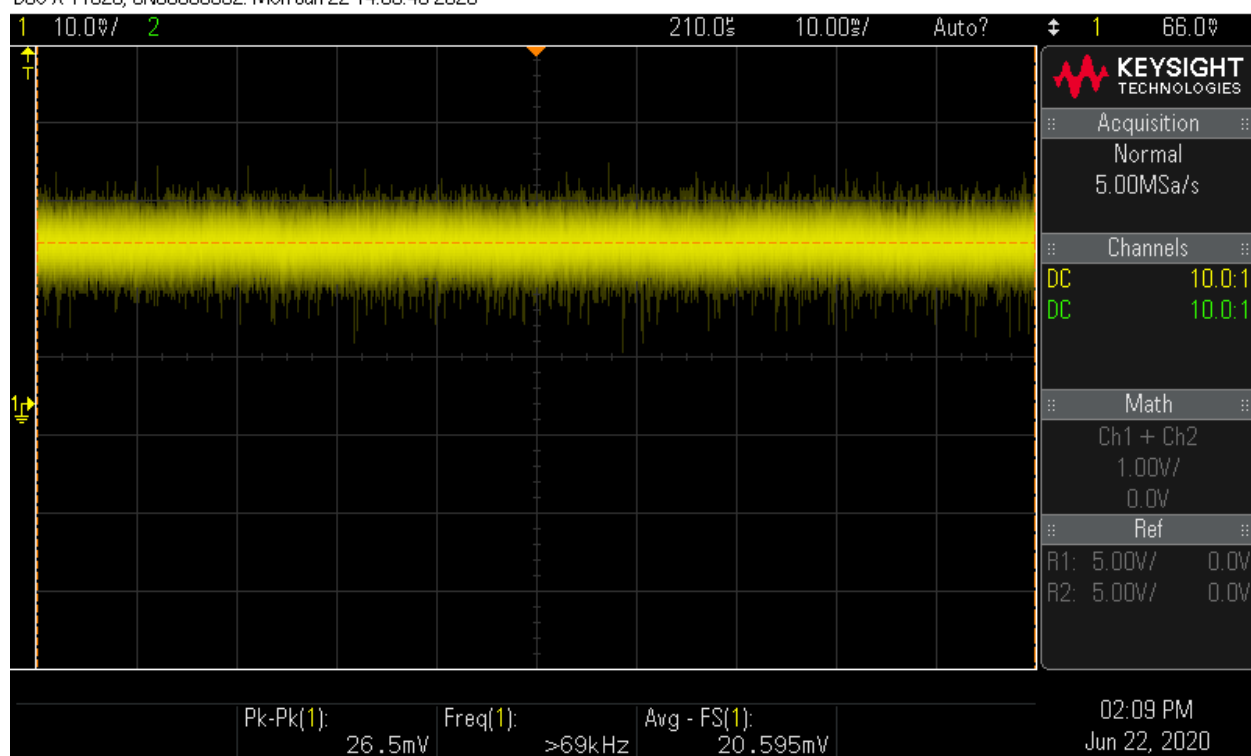


Рисунок 7.31 – Потребляемый ток при дешифровании числа 4002689896

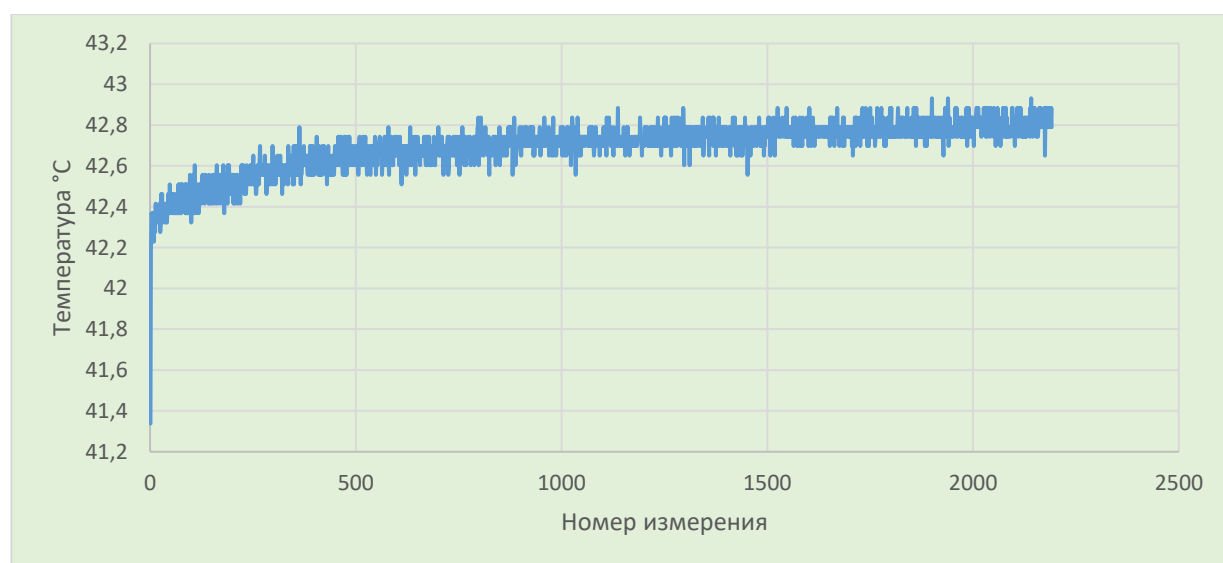


Рисунок 7.32 – Средний рост температуры процессора при дешифровании числа 4002689896

Первое полученное измерение температуры составило 42,18161683 °C, после окончания дешифрования температура процессора стала 43,63404038 °C, за время выполнения дешифрования было получено 2205 измерений

DSO-X 1102G, CN58056332, Mon Jun 22 14:10:52 2020

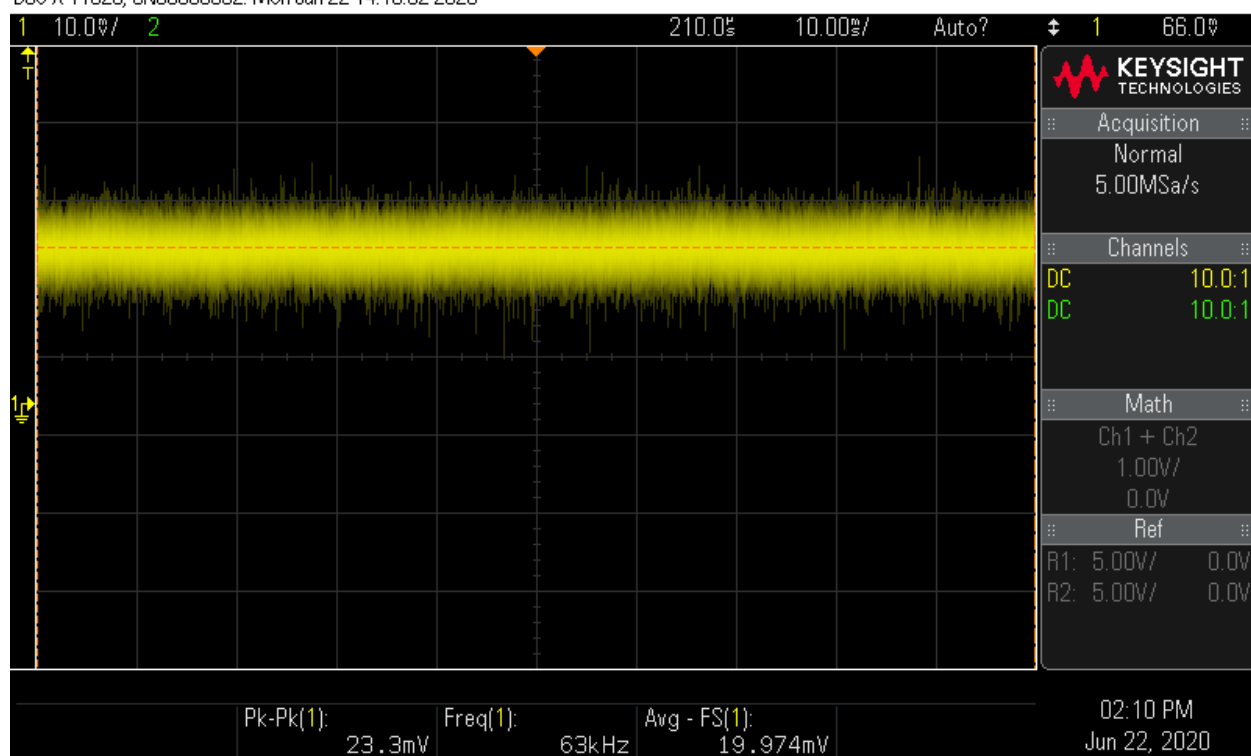


Рисунок 7.33 – Потребляемый ток при дешифровании числа 2147483648

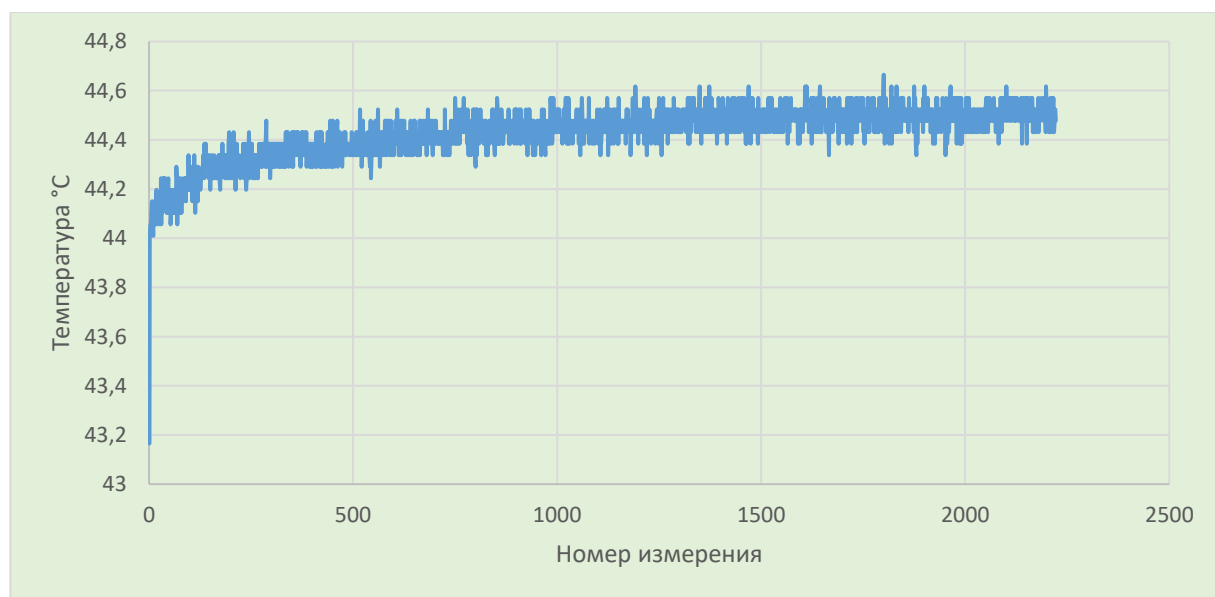


Рисунок 7.34 – Средний рост температуры процессора при дешифровании числа 2147483648

Начальная температура равнялась 43,16551665 °C градусам, после завершения дешифрования температура процессора достигла 44,47738308 °C, за время выполнения дешифрования было получено 2222 измерений

DSO-X 1102G, CN58056332, Mon Jun 22 14:16:49 2020

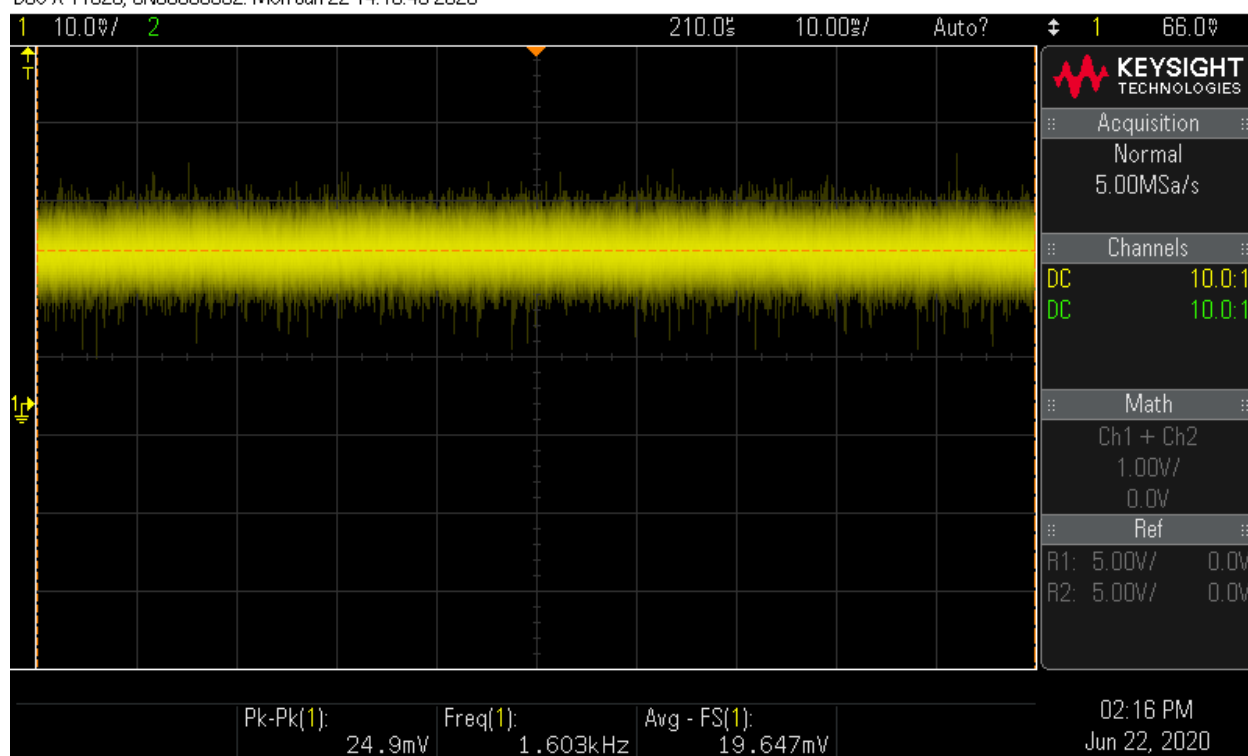


Рисунок 7.35 – Потребляемый ток при дешифровании числа 1073741824

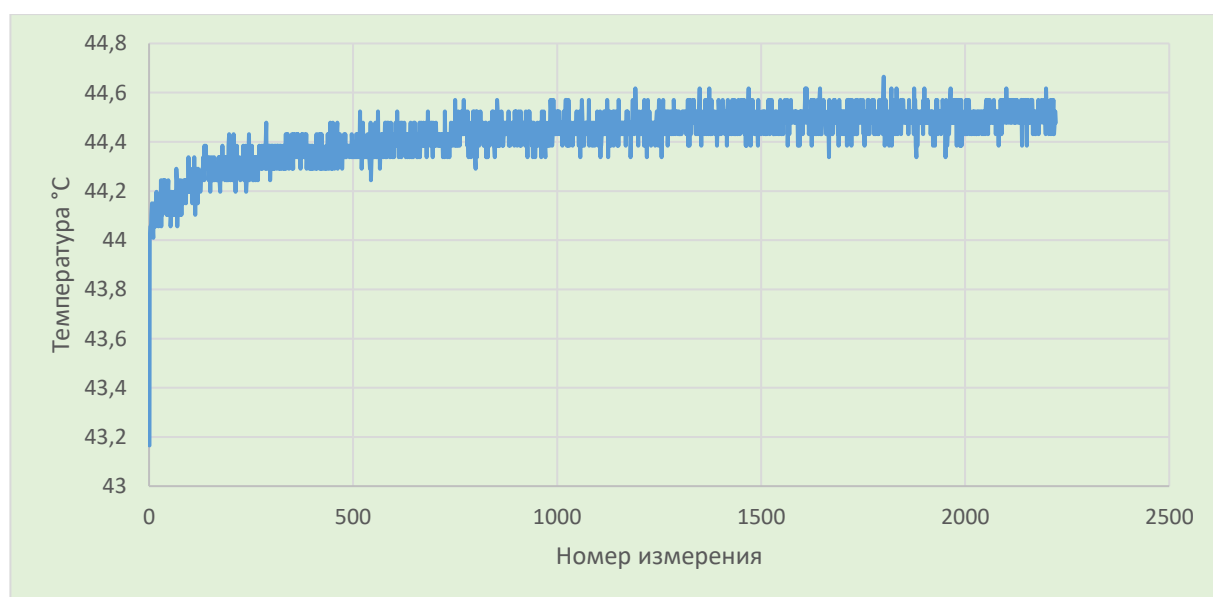


Рисунок 7.36 – Средний рост температуры процессора при дешифровании числа 1073741824

Последний эксперимент проводился при начальной температуре 47,61649204 °C, после завершения дешифрования температура процессора выросла до 49,16262033 градусов Цельсия, за время выполнения дешифрования было получено 2210 измерений.

В таблице 7.3 приведены значения потребляемого тока, в течении выполнения дешифрования сообщений, а количество полученных измерений, а также разница между начальной температурой процессора и конечной. Коэффициент корреляции между потребляемым током и изменением температуры составил 0,732864. Не учитывая измерений потребляемого тока для четвертого эксперимента, можно утверждать, что температура процессора зависит от того, какое сообщение дешифруется, т.к. наблюдается линейная зависимость температуры процессора от потребляемого тока (Рисунок 7.37).

Таблица 7.3 – Изменение температуры и потребляемы ток при дешифровании данных

№	Шифруемые значения	Дешифруемые значения	Изменение температуры	Потребляемый ток	Количество полученных измерений
1	2	8	1,59	20,352	2191
2	3	27	1,59	20,886	2200
3	100	1000000	1,64	21,087	2204
4	1500	375000000	1,54	20,842	2216
5	4002689921	4002689915	1,59	20,538	2208
6	4002689920	4002689896	1,59	20,595	2205
7	133964360	2147483647	1,59	20,341	2222
8	1024	1073741824	1,49	19,647	2210

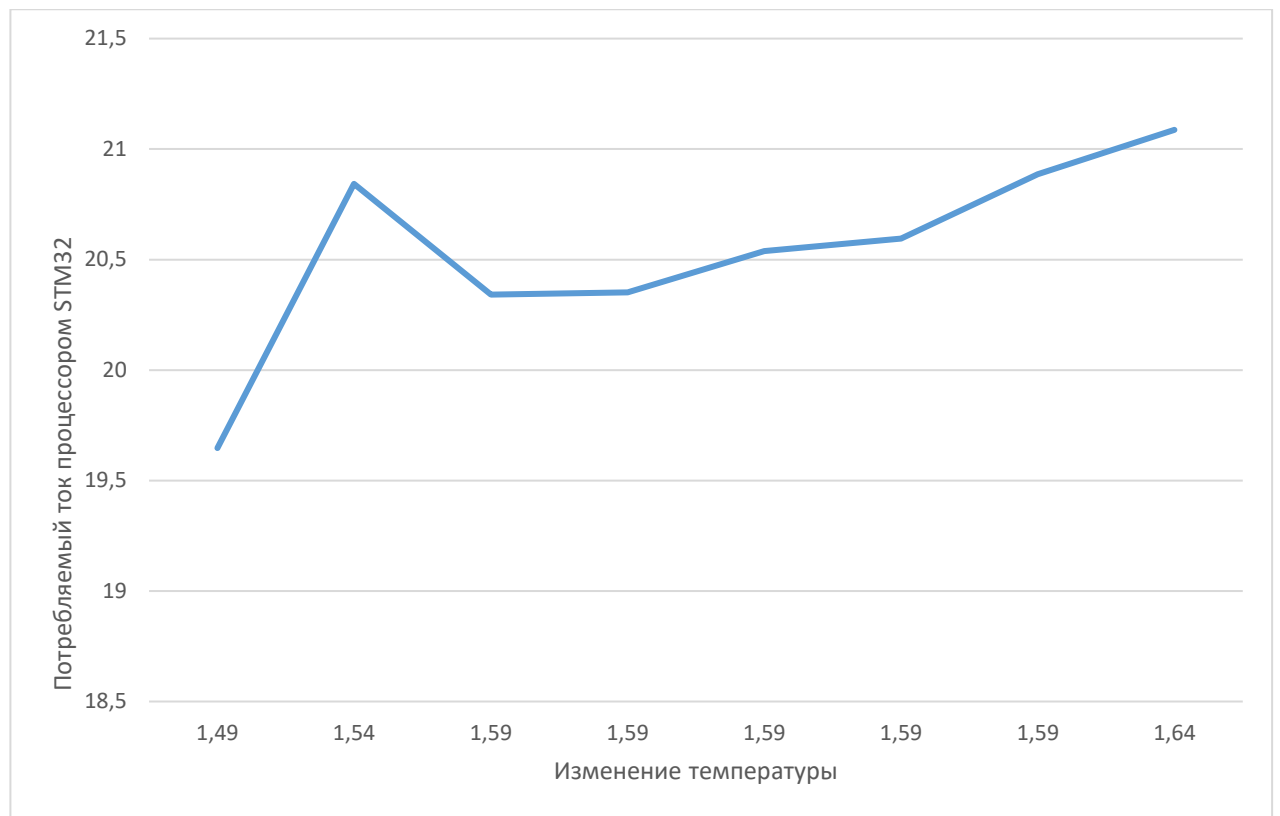


Рисунок 7.37 – Зависимость температуры от потребляемого тока процессором

Заключение

В ходе работы получены новые экспериментальные данные для различных ключей шифрования. При пассивных атаках использование температурных измерений вычислителя с целью получения ключей является возможным, т.к. при аппаратной реализации любого криптографического устройства, даже с самым математически сложным криптоалгоритмом, невозможно избежать утечки информации по побочным каналам. Однако данную информацию можно сделать неразборчивой и предотвратить несанкционированный доступ. Так, встроенный температурный датчик можно использовать не только в корыстных целях, но и для предотвращения несанкционированного доступа. К криптографическому устройству можно добавить защитный алгоритм, который срабатывает при изменении температуры и перезаписывает секретный ключ, таким образом, чтобы при температурах с высокой вероятностью ошибки вычислений в энергозависимой памяти не находилась секретная информация.

Уязвимость RSA, на данный момент, обусловлена не алгоритмом, а его практической реализацией. При использовании больших ключей с математической точки зрения требуется большое количество усилий и ресурсов для факторизации полупростого числа. Однако использование сторонних данных, возникающих в процессе выполнения алгоритма на устройстве, могут позволить злоумышленнику извлечь полезную информацию и получить ключи шифрования даже самого сложного алгоритма.

При пассивной атаке на температурный канал криптографического устройства извлечь ключи шифрования не предоставляется возможным, однако температурный анализ устройства в совокупности с другими сторонними каналами может помочь извлечь ключи шифрования. Так, например, при атаке по времени вычислений в совокупности с температурным каналом можно выяснить, вносилась ли специальная задержка вычислений для защиты от атак по времени или нет.

Активные атаки по температурному каналу уже были исследованы [29]. При активных атаках удалось успешно извлечь ключи шифрования RSA на основе ошибочных вычислений. В исследованиях Michael Hutter и Jorn-Marc Schmidt в 2014 году был использован внешний температурный датчик, который нарушал целостность процессора. Использование встроенных датчиков температуры для подобного рода атак может быть более перспективным, т.к. визуально целостность устройства не будет нарушена.

В целом, необходимо подчеркнуть, что решения по безопасности для Internet of Things еще не полностью изучены. По мере развития технологии и ее роли в современном мире, необходимо продолжать научные исследования и предлагать современные решения, способные улучшить ситуацию в сфере безопасности интернета вещей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of things: a survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutorials. 2015. Vol. 17(4). P. 2347–2376.
2. Lu U., Da Xu L. Internet of things (IoT) Cybersecurity Research: A review of current research topics // IEEE Internet of Things Journal. 2018. P. 1. DOI:10.1109/JIOT.2018.2869847
3. Kalkan K., Zeadally S. Securing internet of things with software defined networking // IEEE Communications Magazine. 2017. Vol. 56(9). P. 186-192. DOI:10.1109/MCOM.2017.1700714
4. Ferguson N., Schroepel R., Whiting D. (2001). A simple algebraic representation of Rijndael. Selected Areas in Cryptography, Proc. SAC, Lecture Notes in Computer, 2259, 103-111.
5. Advanced Encryption Standard [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard (дата обращения 27.02.2020)
6. Сетии Фейстеля. [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Feistel_cipher (дата обращения 01.03.2020).
7. ГОСТ 28147-89 [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ГОСТ_28147-89 (дата обращения 01.03.2020).
8. Sukhoparov M.E., Lebedev I.S. Realization symmetric encryption on algorithm gost 28147-89 PER GPU // Information Security Problems. Computer Systems. 2016. N 4. P. 140-145. [Электронный ресурс] – Режим доступа: <http://jisp.ru/en/volume/aspekty-informacionnoj-bezopasnosti-4/> (дата обращения 20.03.2020).
9. Rivest R. L., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems // Commun ACM. 1978. P. 120–126. DOI:10.1145/359340.359342
10. Dime W., Hellman M. Multiuser cryptographic techniques. In Proceedings of AFIPS 1976 NCC. AFIPS Press, Montvale, New Jersey, 1976. P. 109-112.

11. Dime W., Hellman M.E. New directions in cryptography. IEEE Trans. Info. Theory, IT. 1976. Vol. 22(6). P. 644-654. [Электронный ресурс] – Режим доступа: <https://ee.stanford.edu/~hellman/publications/24.pdf> (дата обращения 17.03.2020).
12. Коутинхо С. Введение в теорию чисел. Алгоритм RSA. С. Коутинхо; пер. с англ. М.: Постмаркет, 2001. 323 с.
13. Menezes A. J., Oorschot P. V., Vanstone S. A. Handbook of Applied Cryptography. CRC Press, 1996. 816 p.
14. Разложение числа. Институт вычислительной математики им. Г.И. Марчука Российской академии наук [Электронный ресурс]. – Режим доступа https://www.inm.ras.ru/math_center/в-ивм-ран-получено-разложение-числа-rsa-232/ (дата обращения: 24.03.2020).
15. Число Ферма [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Число_Ферма/ (дата обращения 26.02.2020).
16. Алгоритмы быстрого возведения в степень [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Алгоритмы_быстрого_возведения_в_степень /](https://ru.wikipedia.org/wiki/Алгоритмы_быстрого_возведения_в_степень/) (дата обращения 27.03.2020).
17. ASCII [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/ASCII> (дата обращения 27.03.2020).
18. Режим шифрования [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Режим_шифрования (дата обращения 28.03.2020).
19. Атаки на криптосистемы по побочным каналам [Электронный ресурс]. – Режим доступа: http://cryptowiki.net/index.php?title=Атаки_на_криптосистемы_по_побочным_каналам (дата обращения 01.04.2020).
20. Kocher P.C. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, Proceedings, 1109, p. 104–113.

- 21.Blake I.F., Serroussi G., Smart N.P. (2005). Advances in elliptic curve cryptography. Cambridge University Press.
- 22.Zhou Y., DengGuo Fenguo. (2006). Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. Information Security Seminar WS, 0607. [Электронный ресурс]. – Режим доступа: <https://eprint.iacr.org/2005/388> (дата обращения 27.04.2020).
- 23.Brumley D., Boneh D. (2003). Remote timing attacks are practical. CRYPTO'99, LNCS, 1666, 388-397.
- 24.Genkin D., Pachmanov L., Pipman I., Tromer E. (2015) Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation. In: Güneysu T., Handschuh H. (eds) Cryptographic Hardware and Embedded Systems. Lecture Notes in Computer Science, vol 9293. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-662-48324-4_11
- 25.Kocher P., Jaffe J., Jun B. (2011). Introduction to differential power analysis. J Cryptogr Eng, 1, 5–27. DOI:10.1007/s13389-011-0006-y
26. Power analysis [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Power_analysis (дата обращения 27.04.2020).
- 27.Shamir A., Trommer E. (2011). Acoustic cryptanalysis: on noisy people and noisy cars. A preliminary description of the concept. [Электронный ресурс]. – Режим доступа: <https://www.tau.ac.il/~tromer/acoustic/> (дата обращения 27.04.2020).
- 28.Genkin D., Shamir A., Tromer E. (2013). [Электронный ресурс]. – Режим доступа: <http://www.cs.tau.ac.il/~tromer/papers/acoustic-20131218.pdf> (дата обращения 27.04.2020).
- 29.Brouchier J., Dabbous N., Kean T., Marsh C., Naccache D. (2009). Thermocommunication. Temperature attacks. IEEE Secur. Priv., 7(2), 79–82.
- 30.Skorobogatov S. (2002). Low temperature data remanence in static RAM. Technical report, University of Cambridge Computer Laboratory.

31. Samyde D., Skorobogatov S.P., Anderson R.J., Quisquater J.-J. (2002). On a new way to read data from memory. In: IEEE Security in Storage Workshop (SISW02), 65–69.
32. Muller T., Spreitzenbarth M. (2013). FROST. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, 7954, 373-388. Springer, Heidelberg.
33. Hutter M., Schmidt M. (2014). The Temperature Side Channel and Heating Fault Attacks. Cryptography ePrint Archive, 190.
34. Алиса и Боб [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Алиса_и_Боб (дата обращения 27.04.2020).
35. Boneh D., DeMillo R.A., Lipton R.J. (2001). On the Importance of Eliminating Errors in Cryptographic Computations. Journal of Cryptology, 14(2), 101-119.
36. STM32 Manual. (2017). Life Augmented. [Электронный ресурс]. – Режим доступа: <https://www.st.com/resource/en/datasheet/stm32f070c6.pdf> (дата обращения 18.03.2020).
37. Расширенный Алгоритм Евклида. [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Euclidean_algorithm (дата обращения 08.05.2020).
38. Баричев С. (2002). Криптография без секретов, 1, 25 с. [Электронный ресурс]. – Режим доступа: http://www.bnti.ru/dbtexts/ipks/old/analmat/1_2002/crypto4.pdf (дата обращения 08.05.2020).
39. Springer, Heidelberg (2012) Torbjörn Granlund and the GMP development team. GNU MP: The GNU Multiple Precision Arithmetic Library, 5.1.3 edition. [Электронный ресурс]. – Режим доступа: <https://gmplib.org/gmp-man-6.2.0.pdf> (дата обращения 01.06.2020).
40. RM0360. Reference manual for STM32F030x4/6/8/C and STM32F070x6/B advanced ARM. [Электронный ресурс]. – Режим доступа: <https://clck.ru/PTtwW> (дата обращения 11.04.2020).

41. Technical Data Sheet for DHT11 Humidity & Temperature Sensor [Электронный ресурс]. – Режим доступа: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf> (дата обращения 19.05.2020).
42. Data Sheet для Uno R3 [Atmega 328P-AU+CH340G] (HKSHAN). WEB-портал для разработчиков электроники Терра Электроника [Электронный ресурс]. – Режим доступа: https://barnaul.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FU%252FUNO_R3%2528CH340G%2529.pdf (дата обращения 19.05.2020).
43. Modular exponentiation [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Modular_exponentiation (дата обращения 18.05.2020).
44. Умножение по методу русских крестьян [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/273525/> (дата обращения 08.05.2020).
45. Алгоритмы быстрого возведения в степень по модулю [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/> (дата обращения 08.05.2020).
46. Молдовян Н.А. (2010). Теоретический минимум и алгоритмы цифровой подписи. СПб.: БВХ-Петербург: Книжный Дом «ЛИБРОКОМ», 304 с.
47. Алгоритм Монтгомери [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_Монтгомери (дата обращения 08.05.2020).

Приложение А

(обязательное)

Листинг кода расчета ключей RSA.

```

1  #include <stdio.h>
2  int checkPrime(unsigned long long n) {
3      int i;
4      unsigned long long m = n/2;
5      for (i = 2; i <= m; i++) {
6          if (n % i == 0) {
7              return 0; // Not Prime
8          }
9      }
10     return 1;          // Prime
11 }
12 int findGCD(unsigned long long n1, unsigned long long n2) {
13     unsigned long long i, gcd;
14     for(i = 1; i <= n1 && i <= n2; ++i) {
15         if(n1 % i == 0 && n2 % i == 0)
16             gcd = i;
17     }
18     return gcd;
19 }
20 unsigned long p, q;
21 unsigned long long n;
22 unsigned long long phin;
23 unsigned long e, d;
24
25
26 int main() {
27     p = 11;
28     q = 13;
29     printf("Calculation of the private key and public key for the RSA algorithm\n");
30     while (1) {
31
32         printf("//p = %u\n", p); //scanf("%u", &p);
33         printf("//q = %u\n", q); //scanf("%u", &q);
34
35         if (!(checkPrime(p) && checkPrime(q)))
36             printf("Both numbers are not prime. Please enter prime numbers only...\n");
37         else if (!checkPrime(p))
38             printf("The first prime number you entered is not prime, please try again...\n");
39         else if (!checkPrime(q))
40             printf("The second prime number you entered is not prime, please try
again...\n");
41         else
42             break;
43     }
44     n = p * q;
45     phin = (p - 1) * (q - 1);

```

```

46     printf("//n = %u\n",n);
47     printf("//phin = %u\n",phin);
48 //=====
49 //===== Public key =====
50 //=====
51     int i;
52     i = 0;
53     for (e = 3; e < phin; i++) {
54         if (findGCD(phin, e) == 1)
55             break;
56         if (e<phin)
57             e = pow(2,(pow(2,i)))+1;
58         if (findGCD(phin, e) == 1)
59             break;
60     else
61         for (e = 3; e<phin; e++){
62             if (findGCD(phin, e) == 1)
63                 break;
64     }
65     }
66     printf("//e = %u\n",e);
67 //=====
68 //===== Private key =====
69 //=====
70 //https://en.wikipedia.org/wiki/Euclidean_algorithm
71     unsigned long long a, b, p=1, q=0, r=0, s=1, x, y;
72     a = phin;
73     b = e;
74     while (a && b) {
75         if (a>=b) {
76             a = a - b;
77             p = p - r;
78             q = q - s;
79         } else
80         {
81             b = b - a;
82             r = r - p;
83             s = s - q;
84         }
85     }
86     if (a) {
87         x = p;
88         y = q;
89     } else
90     {
91         x = r;
92         y = s;
93     }
94     d = a*x+b*y;
95 /*
96     for (d = e + 1; d < n; d++) {
97         if ( (((d % phin)* e) % phin) == 1)

```

```
98         break;
99     }*/
100     printf("//d = %lu\n",d);
101     if (((d % phin)* e) % phin) != 1){
102         printf("d is wrong!");
103         return 1;}
104
105     return 0;
106 }
```

Приложение Б

(обязательное)

Листинг кода получения зашифрованных данных.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  int powMod(unsigned long long a, unsigned long long b, unsigned long long n) {
6      long long x = 1, y = a;
7
8      while (b > 0) {
9          if (b % 2 == 1)
10             x = (x%n * y) % n;
11             y = (y%n * y) % n; // (a*b) mod n = (a*(b mod n) mod n)
12             b /= 2;
13     }
14
15     return x % n;
16 }
17 int main()
18 {
19     unsigned long long cipher;
20     unsigned long long decrypt;
21     unsigned long long e;
22     unsigned long long d;
23     unsigned long long n;
24
25     unsigned long data;
26
27     //p = 10007
28     //q = 399989
29     n = 4002689923;
30     //phin = 4002279928
31     e = 3;
32     d = 2668186619;
33
34     //data = 48;
35     // printf("data: %u\n", data);
36
37     /**< encrypt */
38     /* for(int i=1;i<=e;i++)
39         { cipher=(data*(cipher))%n; } */
40     cipher = powMod(data, e, n);
41     printf("The cipher data is: %u\n", cipher);
42
43     /**< decrypt */
44     /*
45
46     for (int i = 1; i <= d; i++)

```

```
47     {  
48         decrypt = (decrypt*cipher)%n;  
49     }  
50     */// cipher = 110592;  
51     decrypt = powMod(cipher, d, n);  
52     printf("The decrypted data is: %u\n", decrypt);  
53  
54     return 0;  
55 }
```

Приложение В

(обязательное)

Листинг кода получение данных побочного канала на STM32.

```

1 /* USER CODE BEGIN Header */
2 /**
3
4  * @file      : main.c
5  * @brief     : Main program body
6
7  * @attention
8  *
9  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
10 * All rights reserved.</center></h2>
11 *
12 * This software component is licensed by ST under BSD 3-Clause license,
13 * the "License"; You may not use this file except in compliance with the
14 * License. You may obtain a copy of the License at:
15 *          opensource.org/licenses/BSD-3-Clause
16 *
17
18 */
19 /* USER CODE END Header */
20
21 /* Includes -----*/
22 #include "main.h"
23
24 /* Private includes -----*/
25 /* USER CODE BEGIN Includes */
26
27 /* USER CODE END Includes */
28
29 /* Private typedef -----*/
30 /* USER CODE BEGIN PTD */
31
32 /* USER CODE END PTD */
33
34 /* Private define -----*/
35 /* USER CODE BEGIN PD */
36
37 /* USER CODE END PD */
38
39 /* Private macro -----*/
40 /* USER CODE BEGIN PM */
41
42

```

```

43 /* USER CODE END PM */
44
45 /* Private variables -----*/
46 ADC_HandleTypeDef hadc;
47 DMA_HandleTypeDef hdma_adc;
48
49 TIM_HandleTypeDef htim3;
50
51 UART_HandleTypeDef huart2;
52 DMA_HandleTypeDef hdma_usart2_rx;
53
54 /* USER CODE BEGIN PV */
55 //=====
56 //=====  RSA Value  =====
57 //=====
58 //p = 10007;
59 //q = 399989;
60 //n=4002689923;
61 //e = 3;
62 //d=2668186619;
63
64 uint32_t tempDMA[2300] = {0};
65 uint64_t d = 2668186619;
66 //uint64_t e = 3;
67 uint64_t n = 4002689923;
68 unsigned long Encrypted = 110592;
69 //unsigned long Encrypted2 = 125000;
70 unsigned long MSG = 0;
71 unsigned short j = 1*2500;
72
73 /* USER CODE END PV */
74
75 /* Private function prototypes -----*/
76 void SystemClock_Config(void);
77 static void MX_GPIO_Init(void);
78 static void MX_DMA_Init(void);
79 static void MX_ADC_Init(void);
80 static void MX_TIM3_Init(void);
81 static void MX_USART2_UART_Init(void);
82 /* USER CODE BEGIN PFP */
83 int powMod(unsigned long long a, unsigned long long b, unsigned long long n) {
84     long long x = 1, y = a;
85
86     while (b > 0) {
87         if (b % 2 == 1)
88             x = (x%n * y) % n;
89         y = (y%n * y) % n; // (a*b) mod n = (a*(b mod n) mod n)
90         b /= 2;
91     }
92
93     return x % n;
94 }

```



```

95 /* USER CODE END PFP */
96
97 /* Private user code -----*/
98 /* USER CODE BEGIN 0 */
99
100 /* USER CODE END 0 */
101
102 /**
103  * @brief The application entry point.
104  * @retval int
105  */
106 int main(void)
107 {
108     /* USER CODE BEGIN 1 */
109
110     /* USER CODE END 1 */
111
112     /* MCU Configuration-----*/
113
114     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
115     HAL_Init();
116
117     /* USER CODE BEGIN Init */
118
119     /* USER CODE END Init */
120
121     /* Configure the system clock */
122     SystemClock_Config();
123
124     /* USER CODE BEGIN SysInit */
125
126     /* USER CODE END SysInit */
127
128     /* Initialize all configured peripherals */
129     MX_GPIO_Init();
130     MX_DMA_Init();
131     MX_ADC_Init();
132     MX_TIM3_Init();
133     MX_USART2_UART_Init();
134     /* USER CODE BEGIN 2 */
135     //start the timer
136     HAL_TIM_Base_Start(&htim3);
137     //start ADC as DMA
138     HAL_ADC_Start_DMA(&hadc, tempDMA, 2300);
139
140
141
142     /* USER CODE END 2 */
143
144     /* Infinite loop */
145     /* USER CODE BEGIN WHILE */
146     while (j)

```

```

147 {
148     j--;
149 //===== decrypt =====
150 //     if (j>1250)
151         MSG = powMod(Encrypted, d, n);
152 //     else
153 //         MSG = powMod(Encrypted2, d, n);
154
155     if (j==0)
156     {
157         HAL_ADC_Stop_DMA(&hadc);
158
159         uint8_t tempUART[5] = {0};
160
161
162 //sent temperature by UART
163         for (unsigned short z=0; z<2300; z++)
164         {
165             sprintf(tempUART, "%lu\n", tempDMA[z]);
166             HAL_UART_Transmit(&huart2,tempUART, 5,100);
167             if (tempDMA[z]==0)
168                 return 0;
169         }
170     }
171
172 //     HAL_ADC_Start(&hadc);
173 //     tempConvert(int z, tempDMAvar)
174 /*     for(int z = 0;z<100;z++){
175         HAL_Delay(500);
176         tempArray[z] = tempDMA[0];
177     }
178     if(HAL_ADC_PollForConversion(&hadc, 100) == HAL_OK)
179     {
180         TS_ADC_DATA = HAL_ADC_GetValue(&hadc);
181         Vsense = (float) TS_ADC_DATA/4095*VDDA;
182         Temperature = (V_30-Vsense)/Avg_Slope+30;
183     }
184 */
185 */
186
187 }
188 /* USER CODE END WHILE */
189
190 /* USER CODE BEGIN 3 */
191
192 /* USER CODE END 3 */
193 }
194
195 /**
196  * @brief System Clock Configuration
197  * @retval None
198  */

```

```

199 void SystemClock_Config(void)
200 {
201     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
202     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
203
204     /** Initializes the CPU, AHB and APB busses clocks
205     */
206     RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSI14;
207     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
208     RCC_OscInitStruct.HSI14State = RCC_HSI14_ON;
209     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
210     RCC_OscInitStruct.HSI14CalibrationValue = 16;
211     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
212     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
213     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
214     RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
215     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
216     {
217         Error_Handler();
218     }
219     /** Initializes the CPU, AHB and APB busses clocks
220     */
221     RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
222         |RCC_CLOCKTYPE_PCLK1;
223     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
224     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
225     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
226
227     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
228     {
229         Error_Handler();
230     }
231 }
232
233 /**
234  * @brief ADC Initialization Function
235  * @param None
236  * @retval None
237  */
238 static void MX_ADC_Init(void)
239 {
240
241     /* USER CODE BEGIN ADC_Init 0 */
242
243     /* USER CODE END ADC_Init 0 */
244
245     ADC_ChannelConfTypeDef sConfig = {0};
246
247     /* USER CODE BEGIN ADC_Init 1 */
248

```

```

249  /* USER CODE END ADC_Init 1 */
250  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
    number of conversion)
251  */
252  hadc.Instance = ADC1;
253  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
254  hadc.Init.Resolution = ADC_RESOLUTION_12B;
255  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
256  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
257  hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
258  hadc.Init.LowPowerAutoWait = DISABLE;
259  hadc.Init.LowPowerAutoPowerOff = DISABLE;
260  hadc.Init.ContinuousConvMode = DISABLE;
261  hadc.Init.DiscontinuousConvMode = DISABLE;
262  hadc.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
263  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
264  hadc.Init.DMAContinuousRequests = ENABLE;
265  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
266  if (HAL_ADC_Init(&hadc) != HAL_OK)
267  {
268      Error_Handler();
269  }
270  /** Configure for the selected ADC regular channel to be converted.
271  */
272  sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
273  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
274  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
275  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
276  {
277      Error_Handler();
278  }
279  /* USER CODE BEGIN ADC_Init 2 */
280
281  /* USER CODE END ADC_Init 2 */
282
283  }
284
285  /**
286   * @brief TIM3 Initialization Function
287   * @param None
288   * @retval None
289   */
290  static void MX_TIM3_Init(void)
291  {
292
293      /* USER CODE BEGIN TIM3_Init 0 */
294
295      /* USER CODE END TIM3_Init 0 */
296
297      TIM_ClockConfigTypeDef sClockSourceConfig = {0};
298      TIM_MasterConfigTypeDef sMasterConfig = {0};
299

```

```

300  /* USER CODE BEGIN TIM3_Init 1 */
301
302  /* USER CODE END TIM3_Init 1 */
303  htim3.Instance = TIM3;
304  htim3.Init.Prescaler = 4800;
305  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
306  htim3.Init.Period = 100;
307  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
308  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
309  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
310  {
311      Error_Handler();
312  }
313  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
314  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
315  {
316      Error_Handler();
317  }
318  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
319  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
320  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
321  {
322      Error_Handler();
323  }
324  /* USER CODE BEGIN TIM3_Init 2 */
325
326  /* USER CODE END TIM3_Init 2 */
327
328  }
329
330  /**
331   * @brief USART2 Initialization Function
332   * @param None
333   * @retval None
334   */
335  static void MX_USART2_UART_Init(void)
336  {
337
338      /* USER CODE BEGIN USART2_Init 0 */
339
340      /* USER CODE END USART2_Init 0 */
341
342      /* USER CODE BEGIN USART2_Init 1 */
343
344      /* USER CODE END USART2_Init 1 */
345      huart2.Instance = USART2;
346      huart2.Init.BaudRate = 115200;
347      huart2.Init.WordLength = UART_WORDLENGTH_8B;
348      huart2.Init.StopBits = UART_STOPBITS_1;
349      huart2.Init.Parity = UART_PARITY_NONE;
350      huart2.Init.Mode = UART_MODE_TX_RX;
351      huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;

```

```

352 huart2.Init.OverSampling = UART_OVERSAMPLING_16;
353 huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
354 huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
355 if (HAL_UART_Init(&huart2) != HAL_OK)
356 {
357     Error_Handler();
358 }
359 /* USER CODE BEGIN USART2_Init 2 */
360
361 /* USER CODE END USART2_Init 2 */
362
363 }
364
365 /**
366  * Enable DMA controller clock
367  */
368 static void MX_DMA_Init(void)
369 {
370
371     /* DMA controller clock enable */
372     __HAL_RCC_DMA1_CLK_ENABLE();
373
374     /* DMA interrupt init */
375     /* DMA1_Channel1_IRQn interrupt configuration */
376     HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
377     HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
378     /* DMA1_Channel4_5_IRQn interrupt configuration */
379     HAL_NVIC_SetPriority(DMA1_Channel4_5_IRQn, 0, 0);
380     HAL_NVIC_EnableIRQ(DMA1_Channel4_5_IRQn);
381
382 }
383
384 /**
385  * @brief GPIO Initialization Function
386  * @param None
387  * @retval None
388  */
389 static void MX_GPIO_Init(void)
390 {
391     GPIO_InitTypeDef GPIO_InitStruct = {0};
392
393     /* GPIO Ports Clock Enable */
394     __HAL_RCC_GPIOC_CLK_ENABLE();
395     __HAL_RCC_GPIOF_CLK_ENABLE();
396     __HAL_RCC_GPIOA_CLK_ENABLE();
397
398     /*Configure GPIO pin Output Level */
399     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
400
401     /*Configure GPIO pin : PC13 */
402     GPIO_InitStruct.Pin = GPIO_PIN_13;
403     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

```

```

404 GPIO_InitStruct.Pull = GPIO_NOPULL;
405 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
406
407 /*Configure GPIO pin : PA10 */
408 GPIO_InitStruct.Pin = GPIO_PIN_10;
409 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
410 GPIO_InitStruct.Pull = GPIO_NOPULL;
411 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
412 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
413
414 }
415
416 /* USER CODE BEGIN 4 */
417 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
418 {
419     /* Prevent unused argument(s) compilation warning */
420     UNUSED(hadc);
421
422     /* NOTE : This function should not be modified. When the callback is needed,
423             function HAL_ADC_ConvCpltCallback must be implemented in the user file.
424     */
425     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
426 }
427 /* USER CODE END 4 */
428
429 /**
430  * @brief This function is executed in case of error occurrence.
431  * @retval None
432  */
433 void Error_Handler(void)
434 {
435     /* USER CODE BEGIN Error_Handler_Debug */
436     /* User can add his own implementation to report the HAL error return state */
437
438     /* USER CODE END Error_Handler_Debug */
439 }
440
441 #ifndef USE_FULL_ASSERT
442 /**
443  * @brief Reports the name of the source file and the source line number
444  *        where the assert_param error has occurred.
445  * @param file: pointer to the source file name
446  * @param line: assert_param error line source number
447  * @retval None
448  */
449 void assert_failed(uint8_t *file, uint32_t line)
450 { /* USER CODE BEGIN Header */
451 /**
452
453  * @file      : main.c
454  * @brief     : Main program body

```

```

455
*****
456  * @attention
457  *
458  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
459  * All rights reserved.</center></h2>
460  *
461  * This software component is licensed by ST under BSD 3-Clause license,
462  * the "License"; You may not use this file except in compliance with the
463  * License. You may obtain a copy of the License at:
464  *      opensource.org/licenses/BSD-3-Clause
465  *
466
*****
467  */
468  /* USER CODE END Header */
469
470  /* Includes -----*/
471  #include "main.h"
472
473  /* Private includes -----*/
474  /* USER CODE BEGIN Includes */
475
476  /* USER CODE END Includes */
477
478  /* Private typedef -----*/
479  /* USER CODE BEGIN PTD */
480
481  /* USER CODE END PTD */
482
483  /* Private define -----*/
484  /* USER CODE BEGIN PD */
485
486  /* USER CODE END PD */
487
488  /* Private macro -----*/
489  /* USER CODE BEGIN PM */
490
491
492  /* USER CODE END PM */
493
494  /* Private variables -----*/
495  ADC_HandleTypeDef hadc;
496  DMA_HandleTypeDef hdma_adc;
497
498  TIM_HandleTypeDef htim3;
499
500  UART_HandleTypeDef huart2;
501  DMA_HandleTypeDef hdma_usart2_rx;
502
503  /* USER CODE BEGIN PV */
504  //=====

```



```

505 //===== RSA Value =====
506 //=====
507 //p = 10007;
508 //q = 399989;
509 //n=4002689923;
510 //e = 3;
511 //d=2668186619;
512
513 uint32_t tempDMA[2300] = {0};
514 uint64_t d = 2668186619;
515 //uint64_t e = 3;
516 uint64_t n = 4002689923;
517 unsigned long Encrypted = 110592;
518 //unsigned long Encrypted2 = 125000;
519 unsigned long MSG = 0;
520 unsigned short j = 1*2500;
521
522 /* USER CODE END PV */
523
524 /* Private function prototypes -----*/
525 void SystemClock_Config(void);
526 static void MX_GPIO_Init(void);
527 static void MX_DMA_Init(void);
528 static void MX_ADC_Init(void);
529 static void MX_TIM3_Init(void);
530 static void MX_USART2_UART_Init(void);
531 /* USER CODE BEGIN PFP */
532 int powMod(unsigned long long a, unsigned long long b, unsigned long long n) {
533     long long x = 1, y = a;
534
535     while (b > 0) {
536         if (b % 2 == 1)
537             x = (x%n * y) % n;
538         y = (y%n * y) % n; // (a*b) mod n = (a*(b mod n) mod n)
539         b /= 2;
540     }
541
542     return x % n;
543 }
544 /* USER CODE END PFP */
545
546 /* Private user code -----*/
547 /* USER CODE BEGIN 0 */
548
549 /* USER CODE END 0 */
550
551 /**
552  * @brief The application entry point.
553  * @retval int
554  */
555 int main(void)
556 {

```

```

557  /* USER CODE BEGIN 1 */
558
559  /* USER CODE END 1 */
560
561  /* MCU Configuration-----*/
562
563  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
564  HAL_Init();
565
566  /* USER CODE BEGIN Init */
567
568  /* USER CODE END Init */
569
570  /* Configure the system clock */
571  SystemClock_Config();
572
573  /* USER CODE BEGIN SysInit */
574
575  /* USER CODE END SysInit */
576
577  /* Initialize all configured peripherals */
578  MX_GPIO_Init();
579  MX_DMA_Init();
580  MX_ADC_Init();
581  MX_TIM3_Init();
582  MX_USART2_UART_Init();
583  /* USER CODE BEGIN 2 */
584  //start the timer
585  HAL_TIM_Base_Start(&htim3);
586  //start ADC as DMA
587  HAL_ADC_Start_DMA(&hadc, tempDMA, 2300);
588
589
590
591  /* USER CODE END 2 */
592
593  /* Infinite loop */
594  /* USER CODE BEGIN WHILE */
595  while (j)
596  {
597      j--;
598  //===== decrypt =====
599  //      if (j>1250)
600      MSG = powMod(Encrypted, d, n);
601  //      else
602  //      MSG = powMod(Encrypted2, d, n);
603
604  if (j==0)
605  {
606      HAL_ADC_Stop_DMA(&hadc);
607
608      uint8_t tempUART[5] = {0};

```

```

609
610
611 //sent temperature by UART
612     for (unsigned short z=0; z<2300; z++)
613     {
614         sprintf(tempUART, "%lu\n", tempDMA[z]);
615         HAL_UART_Transmit(&huart2,tempUART, 5,100);
616         if (tempDMA[z]==0)
617             return 0;
618     }
619 }
620
621 // HAL_ADC_Start(&hadc);
622 // tempConvert(int z, tempDMAvar)
623 /* for(int z = 0;z<100;z++){
624     HAL_Delay(500);
625     tempArray[z] = tempDMA[0];
626 }
627 if(HAL_ADC_PollForConversion(&hadc, 100) == HAL_OK)
628 {
629     TS_ADC_DATA = HAL_ADC_GetValue(&hadc);
630     Vsense = (float) TS_ADC_DATA/4095*VDDA;
631     Temperature = (V_30-Vsense)/Avg_Slope+30;
632 }
633 */
634 */
635
636 }
637 /* USER CODE END WHILE */
638
639 /* USER CODE BEGIN 3 */
640
641 /* USER CODE END 3 */
642 }
643
644 /**
645  * @brief System Clock Configuration
646  * @retval None
647  */
648 void SystemClock_Config(void)
649 {
650     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
651     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
652
653     /** Initializes the CPU, AHB and APB busses clocks
654     */
655     RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSI14;
656     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
657     RCC_OscInitStruct.HSI14State = RCC_HSI14_ON;
658     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
659     RCC_OscInitStruct.HSI14CalibrationValue = 16;

```

```

660 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
661 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
662 RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
663 RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
664 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
665 {
666     Error_Handler();
667 }
668 /** Initializes the CPU, AHB and APB busses clocks
669 */
670 RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
671         |RCC_CLOCKTYPE_PCLK1;
672 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
673 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
674 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
675
676 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
677 {
678     Error_Handler();
679 }
680 }
681
682 /**
683  * @brief ADC Initialization Function
684  * @param None
685  * @retval None
686 */
687 static void MX_ADC_Init(void)
688 {
689
690     /* USER CODE BEGIN ADC_Init 0 */
691
692     /* USER CODE END ADC_Init 0 */
693
694     ADC_ChannelConfTypeDef sConfig = {0};
695
696     /* USER CODE BEGIN ADC_Init 1 */
697
698     /* USER CODE END ADC_Init 1 */
699     /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
number of conversion)
700 */
701     hadc.Instance = ADC1;
702     hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
703     hadc.Init.Resolution = ADC_RESOLUTION_12B;
704     hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
705     hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
706     hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
707     hadc.Init.LowPowerAutoWait = DISABLE;
708     hadc.Init.LowPowerAutoPowerOff = DISABLE;
709     hadc.Init.ContinuousConvMode = DISABLE;

```

```

710  hadc.Init.DiscontinuousConvMode = DISABLE;
711  hadc.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
712  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
713  hadc.Init.DMAContinuousRequests = ENABLE;
714  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
715  if (HAL_ADC_Init(&hadc) != HAL_OK)
716  {
717      Error_Handler();
718  }
719  /** Configure for the selected ADC regular channel to be converted.
720  */
721  sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
722  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
723  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
724  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
725  {
726      Error_Handler();
727  }
728  /* USER CODE BEGIN ADC_Init 2 */
729
730  /* USER CODE END ADC_Init 2 */
731
732  }
733
734  /**
735   * @brief TIM3 Initialization Function
736   * @param None
737   * @retval None
738   */
739  static void MX_TIM3_Init(void)
740  {
741
742  /* USER CODE BEGIN TIM3_Init 0 */
743
744  /* USER CODE END TIM3_Init 0 */
745
746  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
747  TIM_MasterConfigTypeDef sMasterConfig = {0};
748
749  /* USER CODE BEGIN TIM3_Init 1 */
750
751  /* USER CODE END TIM3_Init 1 */
752  htim3.Instance = TIM3;
753  htim3.Init.Prescaler = 4800;
754  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
755  htim3.Init.Period = 100;
756  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
757  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
758  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
759  {
760      Error_Handler();
761  }

```

```

762 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
763 if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
764 {
765     Error_Handler();
766 }
767 sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
768 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
769 if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
770 {
771     Error_Handler();
772 }
773 /* USER CODE BEGIN TIM3_Init 2 */
774
775 /* USER CODE END TIM3_Init 2 */
776
777 }
778
779 /**
780  * @brief USART2 Initialization Function
781  * @param None
782  * @retval None
783  */
784 static void MX_USART2_UART_Init(void)
785 {
786
787     /* USER CODE BEGIN USART2_Init 0 */
788
789     /* USER CODE END USART2_Init 0 */
790
791     /* USER CODE BEGIN USART2_Init 1 */
792
793     /* USER CODE END USART2_Init 1 */
794     huart2.Instance = USART2;
795     huart2.Init.BaudRate = 115200;
796     huart2.Init.WordLength = UART_WORDLENGTH_8B;
797     huart2.Init.StopBits = UART_STOPBITS_1;
798     huart2.Init.Parity = UART_PARITY_NONE;
799     huart2.Init.Mode = UART_MODE_TX_RX;
800     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
801     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
802     huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
803     huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
804     if (HAL_UART_Init(&huart2) != HAL_OK)
805     {
806         Error_Handler();
807     }
808     /* USER CODE BEGIN USART2_Init 2 */
809
810     /* USER CODE END USART2_Init 2 */
811
812 }
813

```

```

814 /**
815  * Enable DMA controller clock
816  */
817 static void MX_DMA_Init(void)
818 {
819
820  /* DMA controller clock enable */
821  __HAL_RCC_DMA1_CLK_ENABLE();
822
823  /* DMA interrupt init */
824  /* DMA1_Channel1_IRQn interrupt configuration */
825  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
826  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
827  /* DMA1_Channel4_5_IRQn interrupt configuration */
828  HAL_NVIC_SetPriority(DMA1_Channel4_5_IRQn, 0, 0);
829  HAL_NVIC_EnableIRQ(DMA1_Channel4_5_IRQn);
830
831 }
832
833 /**
834  * @brief GPIO Initialization Function
835  * @param None
836  * @retval None
837  */
838 static void MX_GPIO_Init(void)
839 {
840  GPIO_InitTypeDef GPIO_InitStruct = {0};
841
842  /* GPIO Ports Clock Enable */
843  __HAL_RCC_GPIOC_CLK_ENABLE();
844  __HAL_RCC_GPIOF_CLK_ENABLE();
845  __HAL_RCC_GPIOA_CLK_ENABLE();
846
847  /*Configure GPIO pin Output Level */
848  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
849
850  /*Configure GPIO pin : PC13 */
851  GPIO_InitStruct.Pin = GPIO_PIN_13;
852  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
853  GPIO_InitStruct.Pull = GPIO_NOPULL;
854  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
855
856  /*Configure GPIO pin : PA10 */
857  GPIO_InitStruct.Pin = GPIO_PIN_10;
858  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
859  GPIO_InitStruct.Pull = GPIO_NOPULL;
860  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
861  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
862
863 }
864
865 /* USER CODE BEGIN 4 */

```

```

866 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
867 {
868     /* Prevent unused argument(s) compilation warning */
869     UNUSED(hadc);
870
871     /* NOTE : This function should not be modified. When the callback is needed,
872             function HAL_ADC_ConvCpltCallback must be implemented in the user file.
873     */
874     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
875 }
876 /* USER CODE END 4 */
877
878 /**
879  * @brief This function is executed in case of error occurrence.
880  * @retval None
881  */
882 void Error_Handler(void)
883 {
884     /* USER CODE BEGIN Error_Handler_Debug */
885     /* User can add his own implementation to report the HAL error return state */
886
887     /* USER CODE END Error_Handler_Debug */
888 }
889
890 #ifdef USE_FULL_ASSERT
891 /**
892  * @brief Reports the name of the source file and the source line number
893  *        where the assert_param error has occurred.
894  * @param file: pointer to the source file name
895  * @param line: assert_param error line source number
896  * @retval None
897  */
898 void assert_failed(uint8_t *file, uint32_t line)
899 {
900     /* USER CODE BEGIN 6 */
901     /* User can add his own implementation to report the file name and line number,
902        tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
903     /* USER CODE END 6 */
904 }
905 #endif /* USE_FULL_ASSERT */
906
907 /***** (C) COPYRIGHT STMicroelectronics *****/END OF
FILE****/
908
909 /* USER CODE BEGIN 6 */
910 /* User can add his own implementation to report the file name and line number,
911    tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
912 /* USER CODE END 6 */
913 }
914 #endif /* USE_FULL_ASSERT */
915

```


916 /***** (C) COPYRIGHT STMicroelectronics *****/
END OF
FILE****/

Приложение Г

(обязательное)

Листинг кода реализации умножения при помощи массивов.

```

1 #include <stdio.h>
2 #include <string.h>
3 //p =
29669093332083606603617799242426306347429462625218523944018571574194370194723262
390744910112571804274494074452751891
4 //q =
34038161751975634380066094984915214205471217607347231727351634132760507061748526
50644314432514808888111508386301766
5
6 int main()
7 {
8     int a[116],b[116]; //size in decimal of your p and q
9     int RSA_size = 232;
10    int ans[233]={0};
11    int i,j,tmp;
12    char s1[117],s2[117];
13    printf("      Multiplying variables using arrays\n\n");
14    printf("Value_1 = ");scanf(" %s",s1);
15    printf("Value_2 = ");scanf(" %s",s2);
16    int l1 = strlen(s1);
17    int l2 = strlen(s2);
18
19    printf("\nlength of first value: %d\nlength of second value:%d\n",l1,l2);
20
21    //preobrozovanie simvola v chislo i zapis zadom na pered
22    for(i = l1-1,j=0;i>=0;i--,j++)
23    {
24        a[j] = s1[i]-'0';
25    }
26    for(i = l2-1,j=0;i>=0;i--,j++)
27    {
28        b[j] = s2[i]-'0';
29    }
30    //umnozhenie v stolbik
31    for(i = 0;i < l2;i++)
32    {
33        for(j = 0;j < l1;j++)
34        {
35            ans[i+j] += b[i]*a[j];
36        }
37    }
38    for(i = 0;i < l1+l2;i++)
39    {
40        tmp = ans[i]/10;
41        ans[i] = ans[i]%10;

```

```
42     ans[i+1] = ans[i+1] + tmp;
43 }
44
45 //nahodim dlinu proizvedeniya
46 for(i = l1+l2; i>= 0;i--)
47 {
48     if(ans[i] > 0)
49         break;
50 }
51 printf("\nProduct : ");
52
53 //pechataem proizvedenie v privichnoi cheloveku forme
54 for(;i >= 0;i--)
55 {
56     printf("%d",ans[i]);
57 }
58 printf("\n");
59 return 0;
60 }
```