

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность систем и
технологий»

Специальность 100503 – Информационная безопасность
автоматизированных систем

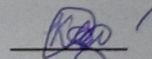
Специализация – Защищенные автоматизированные системы
управления

ДИПЛОМНЫЙ ПРОЕКТ

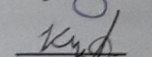
на тему

СПОСОБ ОБЕСПЕЧЕНИЯ КРИПТОГРАФИЧЕСКОЙ ПРОВЕРКИ ЦЕЛОСТНОСТИ
СПЕЦИАЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МИКРОКОНТРОЛЛЕРОВ

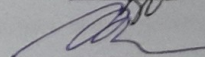
Студент

 / Кашубина А. П.

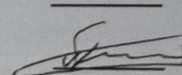
Руководитель

 Кузьмин Д. С.

Нормоконтролёр

 / Иванов А. П.

Рецензент

 Печилейский С. В.

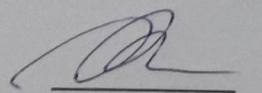
Работа допущена к защите (протокол заседания кафедры от 20.06.2016 № 14)

Заведующий кафедрой

 С.Л. Зефирин

Работа защищена с отметкой отлично (протокол заседания ГЭК
от 24.06.16 № 13)

Секретарь ГЭК

 А.П. Иванов

Пенза 2016

РЕФЕРАТ

Отчет содержит 75 с., 4 рис., 6 табл., 31 источник, 3 прил.

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ, ЦЕЛОСТНОСТЬ ИНФОРМАЦИИ,
КРИПТОГРАФИЯ, СПЕЦИАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
МИКРОКОНТРОЛЛЕРОВ, ИМИТОВСТАВКА.

Объектом исследований в дипломном проекте является способ обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров, предназначенный для использования в перспективной аппаратуре криптографической защиты информации.

Цель дипломного проекта — разработка способа обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров.

В процессе дипломного проектирования был проведен анализ способа криптографической проверки целостности специального программного обеспечения микроконтроллеров, применяемого в настоящее время, с целью поиска в нем уязвимостей, была разработана модель угроз и нарушителя компрометации специального программного обеспечения микроконтроллеров, был проведен анализ и выбор криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности специального программного обеспечения микроконтроллеров, был разработан способ криптографической проверки целостности специального программного обеспечения микроконтроллеров, защищенный от обнаруженных уязвимостей.

В результате дипломного проектирования был программно реализован разработанный способ обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров.

СОДЕРЖАНИЕ

Введение.....	7
1 Модель угроз и нарушителя компрометации СПО микроконтроллеров.....	9
2 Анализ способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, с целью поиска в нем уязвимостей.....	15
2.1 Описание способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время.....	15
2.2 Реализация способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, на языке программирования С для платформы Arduino Uno.....	18
2.2.1 Общие сведения.....	18
2.2.2 Описание программой реализации.....	19
2.2.3 Вывод результатов.....	21
2.3 Описание уязвимостей.....	21
2.4 Компрометация способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время.....	25
2.4.1 Описание компрометации.....	25
2.4.2 Реализация компрометации способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, на языке программирования С для платформы Arduino Uno.....	26
3 Анализ и выбор криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности СПО микроконтроллеров.....	29
3.1 Описание доступных криптографических примитивов.....	29

3.1.1 Блочные шифры	29
3.1.2 Поточные шифры	33
3.1.3 Асимметричные шифры	36
3.1.4 Хэш-функции	37
3.2 Выбор криптографических примитивов, позволяющих осуществить криптографическую проверку целостности	39
4 Разработка способа проверки целостности СПО микроконтроллеров, защищенного от обнаруженных уязвимостей	42
5 Реализация разработанного способа на языке программирования С для платформы Arduino Uno	49
5.1 Описание программной реализации	49
5.2 Вывод результатов	51
Заключение	54
Список использованных источников	55
Приложение А. Исходный текст программной реализации способа обеспечения криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, для платформы Arduino Uno	59
Приложение Б. Исходный текст компрометации способа обеспечения криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, для платформы Arduino Uno	66
Приложение В. Акт о внедрении	74

ОПРЕДЕЛЕНИЯ

В настоящем отчете используются следующие термины с соответствующими определениями.

Бутлоадер — (англ. bootloader — загрузчик) небольшая программа, записанная обычно в конце энергонезависимой памяти микроконтроллера и запускаемая сразу же после старта микроконтроллера, которой доступна вся внутренняя память.

Имитовставка — строка бит фиксированной длины, полученная применением симметричного криптографического метода к сообщению, добавляемая к сообщению для обеспечения его целостности и аутентификации источника данных. [1]

Ключ — изменяемый параметр в виде последовательности символов, определяющий криптографическое преобразование. [1]

Микроконтроллер — самостоятельная микропроцессорная система, которая содержит центральный микропроцессор, вспомогательные схемы (буферные регистры, контроллер) и устройства ввода/вывода, размещенные в общем корпусе (одном чипе).

Последовательность псевдослучайных чисел — последовательность чисел, полученная в результате выполнения некоторого арифметического (вычислительного) процесса, используемая в конкретном случае вместо последовательности случайных чисел.

Последовательность случайных чисел — последовательность чисел, каждое из которых не может быть предсказано (вычислено) только на основе знания предшествующих ему чисел данной последовательности.

Целостность — способность данных не подвергаться изменению или аннулированию в результате несанкционированного доступа.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ГПСП — генератор псевдослучайной последовательности;

ИТ — информационные технологии;

МК — микроконтроллер;

НСД — несанкционированный доступ;

ПЛИС — программируемая логическая интегральная схема;

ПЭВМ – персональная электронная вычислительная машина;

СПО — специальное программное обеспечение;

УВК — устройство ввода ключей;

ЭНП — энергонезависимая память;

GPIO — англ. general-purpose input/output (интерфейс ввода/вывода общего назначения).

ВВЕДЕНИЕ

Одним из самых надежных способов защиты информации является криптографическая защита. Именно в этом направлении активнее всего ведется развитие специализированных ИТ-компаний в России. В настоящее время много говорится о защите информации путем ее шифрования. Но уделяется недостаточно внимания безопасности самих устройств, реализующих криптографическое преобразование.

Сегодня для шифрования данных наиболее широко применяют три вида шифраторов: аппаратные, программно-аппаратные и программные. Их основное различие заключается не только в способе реализации шифрования и степени надёжности защиты данных, но и в цене. Несмотря на то, что цена аппаратных шифраторов существенно выше программных, разница в цене не сравнима с значительным повышением качества защиты информации.

Основным модулем аппаратного шифратора является блок управления, управляющий работой всего шифратора. Обычно блок управления реализуется на базе микроконтроллера.

В связи с этим, актуальной является проблема защиты программного обеспечения микроконтроллера, используемого в аппаратуре криптографической защиты информации.

На момент написания дипломного проекта в открытой литературе были найдены сведения о способах защиты программного обеспечения микроконтроллеров. Однако описанные методики имеют ограничения применения в изделиях военного и специального назначения, связанные с использованием асимметричной криптографии и криптографически нестойких методов, отсутствия возможности периодической смены ключа и др.

Целью дипломного проекта является разработка и реализация способа обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров.

В результате дипломного проектирования были проведены:

- анализ способа криптографической проверки целостности специального программного обеспечения микроконтроллеров, применяемого в настоящее время, с целью поиска в нем уязвимостей;
- разработка модели угроз и нарушителя компрометации специального программного обеспечения микроконтроллеров;
- анализ и выбор криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности специального программного обеспечения микроконтроллеров;
- разработка способа криптографической проверки целостности специального программного обеспечения микроконтроллеров, защищенного от обнаруженных уязвимостей.

В результате дипломного проекта был программно реализован разработанный способ обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров.

1 Модель угроз и нарушителя компрометации СПО микроконтроллеров

В работе рассмотрена несанкционированная модификация СПО микроконтроллеров.

Угрозы безопасности СПО микроконтроллеров можно классифицировать в соответствии со следующими признаками:

- по видам возможных источников угроз — внешние и внутренние (угрозы природных и антропогенных катастроф в рамках данной работы не рассматриваются):
внутренний нарушитель — разработчик СПО, имеет полный доступ к исходным кодам и средствам проектирования, специалист в данной области, следовательно от него не существует способов защиты средствами микроконтроллера;
- по типу информации, на которую направлена реализация угроз — программное обеспечение микроконтроллеров;
- по виду нарушаемого свойства информации — нарушение целостности;
- по способам реализации угроз — модификация СПО микроконтроллера;
- по используемой уязвимости;
- по объекту воздействия — энергонезависимая память микроконтроллера, в которой хранится СПО.

В таблице 1 приведены угрозы нарушения целостности СПО микроконтроллеров.

Таблица 1 — Угрозы нарушения целостности СПО микроконтроллеров

№ элемента	Угроза
1	Подмена микроконтроллера
2	Модификация СПО
3	Изменение схемы платы (перестановка перемычек, замыкание выходов и т. д.)

В данном случае под модификацией СПО понимается любое его изменение, в том числе возврат к предыдущей версии.

Потенциал нарушителя может быть высоким, средним или низким. Более высокий уровень включает все возможности более низких. Для каждого из вариантов задан свой набор возможностей:

- атакующий с высоким потенциалом имеет физический доступ к системе, способен заменить микроконтроллер на собственный внутри платы, произвести установку новых элементов конструкции и даже отдельных плат;

- атакующий со средним потенциалом имеет физический доступ к системе; может воздействовать на информационные входы контроллера через стандартные интерфейсы; способен загружать любое СПО, изменять настройки, загружать свой код вместо оригинального на программаторе, переставлять перемычки, замыкать выводы микросхем и т. п.;

- атакующий с низким потенциалом не имеет физического доступа к системе, может воздействовать на нее удаленно по каналам, имеющим выход за пределы контролируемой зоны, собирать информацию по побочным каналам.

Учитывая область применения данной модели, персонал информационных и телекоммуникационных систем специального назначения является доверенным и не может выступать в качестве источника угроз. При этом должна быть реализована защита чувствительной информации при работе персонала под принуждением.

Таблица 2 содержит список источников угроз нарушения целостности СПО микроконтроллеров.

Таблица 2 — Источники угроз нарушения целостности СПО микроконтроллеров

№ элемента	Источники угроз
1	Разведывательные службы иностранных государств
2	Иностранные политические, экономические и военные структуры
3	Международные террористические организации
4	Персонал информационных и телекоммуникационных систем специального назначения, работающий под принуждением
5	Криминальные структуры и организованная преступность
6	Непредвиденные сбои в программах и аппаратуре
7	Диверсанты и диверсионная деятельность [2]

В таблице 3 представлены уязвимости ПО микроконтроллеров.

Таблица 3 — Уязвимости ПО микроконтроллеров

№ элемента	Уязвимости
1	Ошибки персонала информационных и телекоммуникационных систем специального назначения
2	Ошибки при проектировании и реализации СПО
3	Ненадежное функционирование информационных и телекоммуникационных систем специального назначения: отказ технических средств и сбои программного обеспечения в информационных и телекоммуникационных системах специального назначения
4	Закладки в СПО, наличие недокументированных возможностей
5	Неполнота или отсутствие организационных мер и недостаточность физической защиты на объекте эксплуатации

В таблице 4 приведены соответствующие методы нападений.

Таблица 4 — Методы нападений

№ элемента	Нападения
1	Информационно-технические воздействия (в том числе радиоэлектронная борьба, проникновение в компьютерные сети) со стороны вероятных противников
2	Действия персонала информационных и телекоммуникационных систем специального назначения, работающего под принуждением
3	Внедрение закладок в СПО, недокументированных возможностей
4	Физическое воздействие на элементы информационной системы

Далее было составлено полное описание каждой идентифицированной угрозы по модели ГОСТ Р ИСО/МЭК 15408, согласно которой описание каждой угрозы должно включать:

- нападения пригодные для реализации угрозы (возможность, методы, уязвимости);
- объекты нападений (объекты защиты);
- тип потери;
- масштаб ущерба;
- источники угрозы. Для источников угроз – людей модель нарушителя должна включать:

- указание их опыта;
- указание знаний;
- указание доступных ресурсов, необходимых для реализации угрозы, возможную мотивацию их действий [3].

Для всех угроз объектом защиты является программное обеспечение микроконтроллеров. Угрозам присущи следующие типы потерь:

- риск денежной потери — низкий;
- риск потери производительности - средний;
- риск затруднения деятельности - высокий.

Масштаб ущерба характеризуется как высокий.

Источниками угроз являются:

- разведывательные службы иностранных государств (большой опыт деятельности, значительные знания в сфере, объемные ресурсы для реализации угрозы, мотивация – политические и экономические интересы, высокий потенциал нарушителя);
- иностранные политические, экономические и военные структуры (большой опыт деятельности, значительные знания в сфере, объемные ресурсы для реализации

угрозы, мотивация – политические, военные и экономические интересы, высокий потенциал нарушителя);

— международные террористические организации (значительный опыт деятельности, средние знания в сфере, средние ресурсы для реализации угрозы, мотивация – политические, военные и экономические интересы, средний потенциал нарушителя);

— криминальные структуры и организованная преступность (средний опыт деятельности, средние знания в сфере, достаточные ресурсы для реализации угрозы, мотивация – политические и коммерческие интересы, низкий потенциал нарушителя);

— диверсанты и диверсионная деятельность (средний опыт деятельности, высокие знания в сфере, достаточные ресурсы для реализации угрозы, мотивация – личные и коммерческие интересы, высокий потенциал нарушителя).

Первая угроза - подмена микроконтроллера.

Методами нападения, которыми осуществляется данная угроза, являются информационно-технические воздействия со стороны вероятных противников и физические воздействия на элементы информационной системы.

Перечисленные методы нападения используют уязвимость «Неполнота или отсутствие организационных мер и недостаточность физической защиты на объекте эксплуатации».

Вторая угроза - модификация СПО.

Методами нападения, которыми осуществляется данная угроза, являются информационно-технические воздействия (в том числе радиоэлектронная борьба, проникновение в компьютерные сети) со стороны вероятных противников; действия персонала информационных и телекоммуникационных систем специального назначения, работающего под принуждением; внедрение закладок в СПО, недокументированных возможностей; физические воздействия на элементы информационной системы.

Перечисленные методы нападения используют следующие уязвимости: ошибки при проектировании и реализации СПО, закладки в СПО, наличие недокументированных возможностей, ошибки персонала информационных и телекоммуникационных систем специального назначения; ненадежное функционирование информационных и телекоммуникационных систем специального назначения; отказ технических средств и сбои программного обеспечения в информационных и телекоммуникационных системах специального назначения; неполнота или отсутствие организационных мер и недостаточность физической защиты на объекте эксплуатации.

Последняя угроза - изменение схемы платы (перестановка перемычек, замыкание выходов и т. д.).

Методами нападения, которыми осуществляется данная угроза, являются информационно-технические воздействия со стороны вероятных противников; действия персонала информационных и телекоммуникационных систем специального назначения, работающего под принуждением; физическое воздействие на элементы информационной системы.

Перечисленные методы нападения используют следующие уязвимости: ошибки персонала информационных и телекоммуникационных систем специального назначения; неполнота или отсутствие организационных мер и недостаточность физической защиты на объекте эксплуатации.

2 Анализ способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, с целью поиска в нем уязвимостей

2.1 Описание способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время

Процедура криптографической проверки целостности СПО была предоставлена филиалом «Аргус» АО «ПНИЭИ» как материал для ознакомления на преддипломную практику. Данная процедура использовалась в опытно-конструкторских работах филиала «Аргус» АО «ПНИЭИ» по согласованию со специализированной организацией ФГУП ПФ НТЦ «Атлас».

Механизм обеспечения криптографической проверки целостности СПО микроконтроллеров, рекомендуемый специализированными организациями, состоит в следующей процедуре:

- перед загрузкой производится заполнение оставшегося свободным объема flash-памяти случайным шумом;
- затем выполняется алгоритм криптографической проверки целостности, приведенный на рисунке 1.

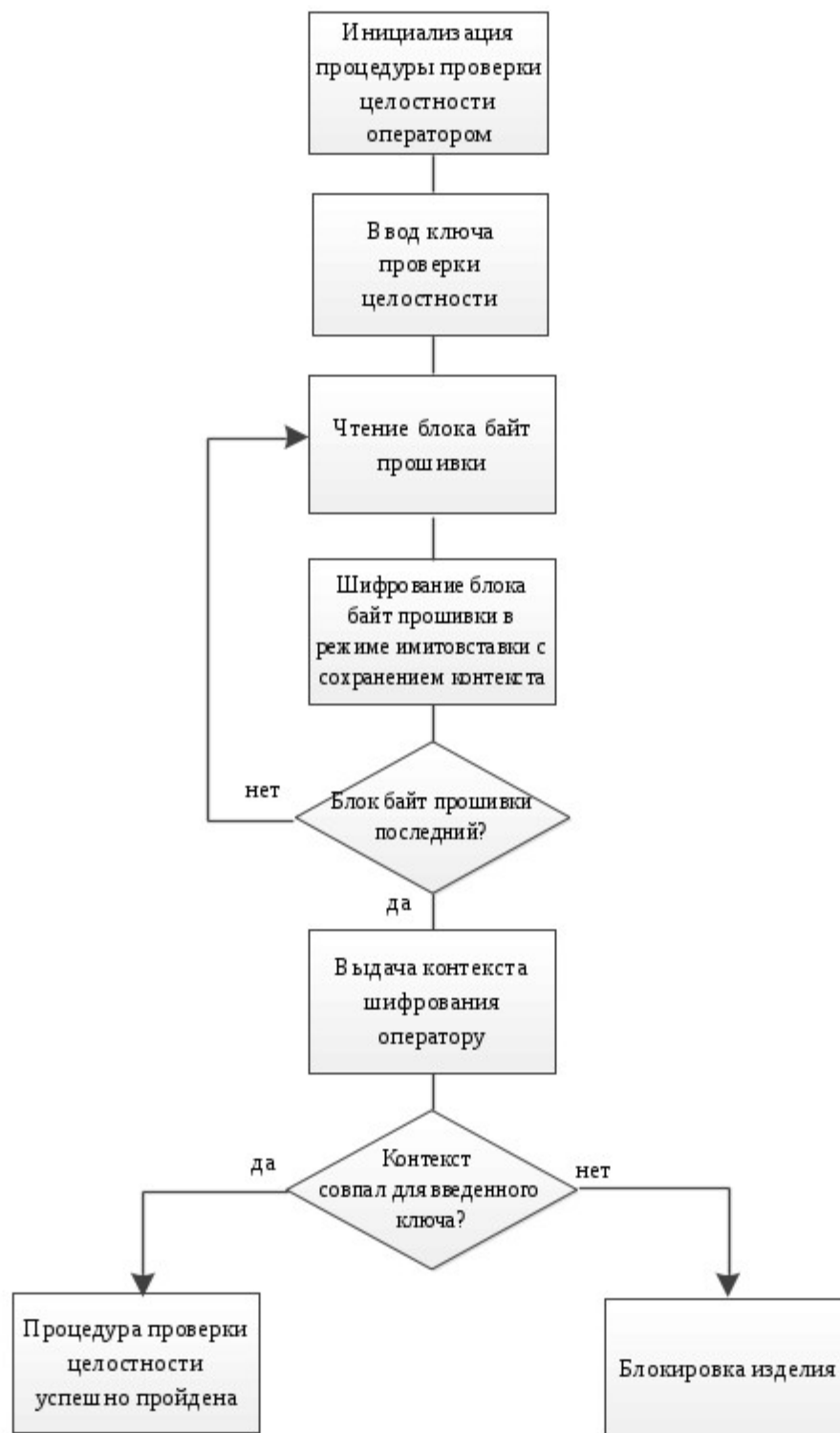


Рисунок 1 - Механизм обеспечения криптографической проверки целостности СПО МК, рекомендуемый специализированными организациями

2.2 Реализация способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, на языке программирования С для платформы Arduino Uno

2.2.1 Общие сведения

В качестве аппаратной платформы была выбрана плата Arduino Uno, на которой установлен контроллер AVR ATmega328. Платформа имеет 14 цифровых вход/выходов (6 из которых могут использоваться как выходы ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, разъем USB, силовой разъем, разъем ICSP и кнопку перезагрузки. Технические характеристики контроллера приведены в таблице 5. Внешний вид платы приведен на рисунке 2. [4]

Таблица 5 - Технические характеристики контроллера Arduino Uno

Микроконтроллер	ATmega328
Рабочее напряжение	5 В
Цифровые Входы/Выходы	14
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	32 Кб (ATmega328) из которых 0,5 Кб для загрузчика
ОЗУ	2 Кб (ATmega328)
EEPROM	1 Кб (ATmega328)
Тактовая частота	16 МГц



Рисунок 2 - Внешний вид платы Arduino Uno

2.2.2 Описание программой реализации

Была написана реализация описанного в пункте 2.2.1 способа обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров, рекомендуемых специализированными организациями, для платформы Arduino Uno на языке программирования C. Разработка производилась с использованием среды разработки Arduino IDE, в состав которой включены стандартные библиотеки по работе с flash-памятью (`avr/pgmspace.h`) и COM-портом (`SoftwareSerial.h`). Исходный код реализации приведен в приложении А.

Первоначальное заполнение оставшегося свободным объема flash-памяти

производилось путем расчета свободного места и сохранения массива со случайными значениями рассчитанной длины (noise.h). Массив генерировался скриптом на языке Python, приведенным ниже, где 25046 — количество свободных байт во флеш-памяти микроконтроллера, с использованием стандартной библиотеки языка random:

```
import random
array = [ ]
for i in range(0, 25046):
    array.append((random.randint(0,255)))
print(array)
```

Running: /home/chelsea/Py/diplo/noise.py (Wed Mar 30 10:07:29 2016)

[14, 231, 35, 108, 77, 75, 22, 230, 140, 203, 200, 190, 252, ...
171, 146]

Для целей работы достаточно использования генератора псевдослучайной последовательности с хорошими статистическими характеристиками. Криптографическая атака на используемый генератор шума в данной работе не рассматривается, т. к. в оригинальной процедуре проверки подразумевается, что используется криптостойкий генератор случайных последовательностей. Библиотека использует вихрь Мерсенна в качестве основного генератора, который имеет период $2^{19937} - 1$. [5]

После добавления в проект заголовочного файла с массивом случайных чисел, среда разработки выдала следующее сообщение, приведенное на рисунке 3.

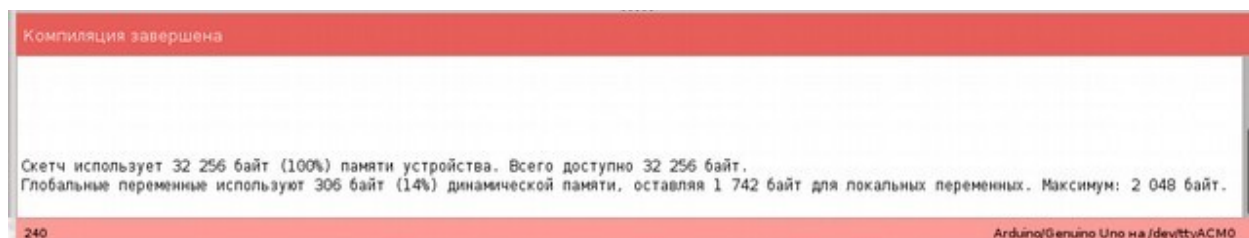


Рисунок 3 - Сообщение среды разработки о количестве свободного места во flash-памяти

2.2.3 Вывод результатов

Ввод ключа алгоритма шифрования и получение рассчитанного значения имитовставки осуществляется через СОМ-интерфейс. Пример работы приведен на рисунке 4.

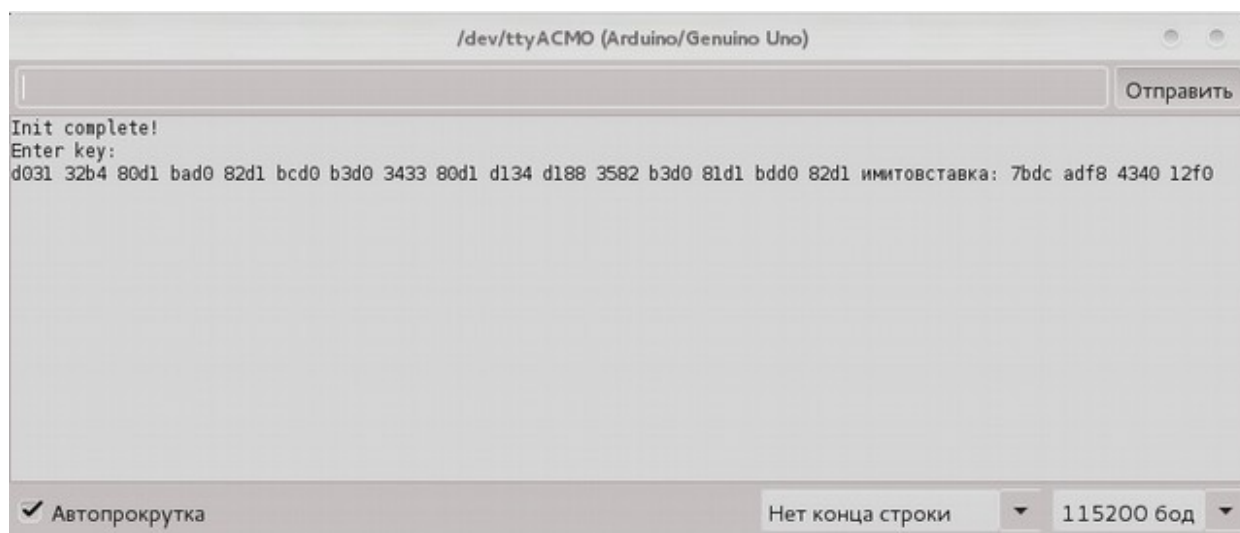


Рисунок 4 — Работа с контроллером

2.3 Описание уязвимостей

В разделе рассматриваются возможные уязвимости рекомендованных механизмов обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров.

Не будут рассмотрены угрозы, исходящие от внутреннего нарушителя — разработчика СПО, который имеет полный доступ к исходным кодам и средствам проектирования, является специалистом в данной области, следовательно от него не существует способов защиты средствами микроконтроллера. Также не рассматривается несанкционированная модификация кода бутлоадера — программы, которая запускается сразу же после старта микроконтроллера. Считаем, что его целостность реализована внешними средствами, а после сохранения кода в память микроконтроллера она защищается от модификации аппаратными средствами.

Для расчета имитовставки в рассмотренной процедуре криптографической

проверки целостности используется алгоритм шифрования согласно «ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». [6]

ГОСТ — это блочный шифр со сбалансированной структурой Файстеля, с размером блока 64 бита, 256-битным ключом и 32 раундами. Каждый раунд содержит сложение с ключом по модулю 2^{32} , набор из восьми 4-битных блоков подстановки (S-блоков) и простой циклический сдвиг на 11 битов. Особенностью ГОСТа является возможность хранения S-блоков в секрете, что можно представить как второй ключ, увеличивающий эффективный ключевой материал до 610 битов. Один набор S-блоков был опубликован в 1994 году в рамках спецификации хэш-функции ГОСТ-Р 34.11-94 и, как писал Шнайер, использовался Центральным Банком Российской Федерации. Если где-то применяются секретные S-блоки, то они могут быть извлечены из программных реализаций и из микросхем, по факту чего были опубликованы соответствующие работы.

В теории, 256-битные ключи останутся безопасными по крайней мере 200 лет. ГОСТ был широко исследован ведущими экспертами в области криптографии, известными в области анализа блочных шифров, такими как Шнайер, Бирюков, Данкельман, Вагнер, множеством австралийских, японских и российских учёных, экспертами по криптографии от ISO.

Согласно Бирюкову и Вагнеру, структура ГОСТ, включающая обратный порядок подключей в последних раундах, делает его стойким против атак скольжения (т. н. «слайд-атаки»). Однако из-за наличия большой величины самоподобия в шифре, это позволяет проводить атаки инверсии ключей на комбинации неподвижных точек и свойства «отражения» (т. н. «рефлексивные атаки») для определённых слабых ключей. Сложность этой атаки 2^{192} и 2^{32} подобранных открытых текстов. Новые атаки также используют отражение и фактически взломали ГОСТ, что и было представлено на конференции FSE 2011. Эти атаки также были открыты независимо автором данной работы. Атака требует

2^{132} байтов памяти, что фактически хуже, чем более медленные атаки с меньшим требованием к памяти.

Множество новых атак на основе самоподобия работают против всех ключей ГОСТа и позволяют взламывать полнораундовый ГОСТ с 256-битным ключом, а не только для слабых ключей, как было ранее. Все эти атаки требуют значительно меньше памяти и они значительно быстрее. Эти новые атаки могут рассматриваться как примеры новой общей парадигмы криптоанализа блочных шифров, называемой «редукция алгебраической сложности», с обобщением этих атак на множество частных случаев атак с известными неподвижными точками, скольжением, инволюциями и циклами. Важно, что среди семейства всех этих атак есть такие, которые позволяют проводить криптоанализ ГОСТ без всяких отражений и без каких-либо симметричных точек, которые проявляются в ходе вычислений.

Практический результат пока скромный: 2^{64} известных открытых текста и 2^{64} памяти для хранения пар «открытый текст/шифротекст» позволяют взломать ГОСТ в 2^8 быстрее, чем простой перебор. Но в плане криптоанализа это делает полностью справедливым утверждение о том, что «ГОСТ взломан». [7]

В ГОСТ 28147 в режиме выработки имитовставки в случае, если длина самого сообщения, либо последнего блока, меньше длины блока, используется дополнение. При этом используется простое дополнение нулевыми битами. Такой вариант приводит к атакам Оракула дополнения (*padding oracle attack*).

Атака Оракула дополнения - это атака, которая выполняется с помощью дополнения криптографического сообщения. В криптографии блоки переменной длины открытого текста сообщения часто должны быть дополнены (расширены), чтобы быть совместимыми с базовым размером блока криптографического примитива. Атака полагается на наличие «оракула дополнения», который свободно отвечает на запросы о том, правильно ли выполнено дополнение сообщения. Такие данные могут позволить

злоумышленникам расшифровать (а иногда и зашифровать) сообщения через оракула, используя ключ оракула, не зная ключа шифрования. Атака Оракула дополнения в основном связана с режимом шифрования CBC блочных шифров. [8]

Для включения в новую версию национального стандарта ГОСТ Р 34.13—2015 «Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров» [1] была выбрана наиболее изученная к настоящему моменту конструкция OMAC1 (CMAC) . Описываемый алгоритм расчета имитовставки использует дополнение блока «padding method 2» ISO/IEC 9797-1, который предлагает добавлять в конец сообщения единичный бит, а оставшиеся заполнять нулями. В этом случае была доказана стойкость к атакам «оракула дополнения», о чем сообщается в работе [9].

Описанная выше уязвимость алгоритма ГОСТ 28147-89 является теоретической и в данной реализации применяться не может. Атака Оракула дополнения также не может быть произведена, поскольку размер СПО, равный объему ЭНП микроконтроллера, является кратным размеру блока.

При анализе процедуры обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров была обнаружена практическая уязвимость, заключающаяся в следующем. Существует возможность сжать СПО до таких размеров, чтобы на освободившееся место записать алгоритм декомпрессии, расчета имитовставки и выполнения злоумышленных действий. При этом случайные значения, которыми flash-память дополняется плохо подвержены сжатию. Однако, скомпилированный код в большинстве случаев удается сжать до 30-50% исходного объема, что позволяет дополнить исполняемый код необходимыми для компрометации функциями.

Атаку на СПО с использованием данной уязвимости может произвести злоумышленник со средним и высоким потенциалом. Это может быть физический НСД к устройству, либо внедрение закладки на этапе производства.

Как уже говорилось в разделе 1, персонал информационных и телекоммуникационных систем специального назначения является доверенным и не может выступать в качестве источника угроз. Также нельзя защититься от разработчика СПО средствами микроконтроллера. Поэтому внедрение закладки на этапе производства как и подмена СПО при процедуре штатного обновления не рассматривается в данном проекте.

2.4 Компрометация способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время

2.4.1 Описание компрометации

В качестве алгоритма архивации в данной работе использовался алгоритм Лемпеля-Зива-Сторера-Цимански (LZSS), который был представлен в 1982 году. Алгоритм является улучшенной версией LZ77, которая просчитывает, не увеличит ли размер результата замена исходных данных кодированными. LZSS до сих пор используется в популярных архиваторах, например RAR.

Идея этого алгоритма основана на том предположении, что входящие в сжимаемый текст слова неоднократно повторяются. Сжатие достигается заменой обнаруженных повторов ссылками на предшествующие вхождения этих же слов. Т. е. повторное вхождение слова заменяется парой чисел <Длина слова, Расстояние до предыдущего вхождения слова>. Под словом понимается не слово естественного языка, а произвольная последовательность символов.

У алгоритма LZ77 возникают проблемы с самим сжатием. Они появляются, когда кодер не может найти совпадающую подстроку в словаре и выдает стандартный 3-компонентный код, пытаясь закодировать один символ. Такое кодирование существенно понижает производительность алгоритма.

Метка LZSS состоит только из смещения и длины. Проблема отсутствия

совпадений в словаре в алгоритме LZSS решается путем введения дополнительного служебного бита (со значением «0» для незакодированных символов и «1» для кодовых комбинаций), значение которого определяет, является ли следующая за ним кодовая комбинация кодовой парой или она представляет собой незакодированный символ в его исходном представлении. Такая техника позволяет записывать символы в явном виде, когда соответствующий им код имеет большую длину, а также позволяет обрабатывать ни разу не встреченные до текущего момента символы. [10]

Алгоритм LZSS, вообще говоря, является очень асимметричным. Если процедура сжатия достаточно сложна и совершает большой объем работы при обработке каждого символа сжимаемого сообщения, то декодер LZSS тривиально прост и может работать со скоростью, приближающейся к скорости процедуры обычного копирования информации.

Также LZSS нуждается в небольшом количестве памяти для хранения последних 2^N байт данных (где N настраивается), а декомпрессия выполняется довольно просто. Кроме того, можно подстроить размер «скользящего окна», что увеличит эффективность. В данной реализации размер окна равен 2^8 .

Таким образом, алгоритм сжатия данных без потерь LZSS является оптимальным выбором для реализации на микроконтроллере, так как он не требователен к аппаратным ресурсам и позволяет обеспечить хорошую скорость декодирования. При компиляции для микроконтроллера Arduino, декодер (heatshrink_decoder.h) занимает около 1 Кбайт ROM памяти и использует менее 300 байт ОЗУ.

2.4.2 Реализация компрометации способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, на языке программирования C для платформы Arduino Uno

C использованием стандартных библиотек, поставляемых в составе среды разработки Arduino IDE, и библиотеки, реализующей декодер LZSS (heatshrink_decoder.h),

была написана реализация описанного выше алгоритма компрометации специального программного обеспечения микроконтроллеров для платформы Arduino Uno. Исходный код приведен в приложении Б.

Кодирование проводилось путем сжатия первоначального СПО и сохранения массива с полученными значениями (compressed.h).

Сжатие СПО проводилось в несколько этапов:

- получение бинарного файла СПО myfile.bin:

```
chelsea@roxy:~/Arduino$      avr-objcopy      -O      binary
sketch_mar21a.ino.elf myfile.bin
```

- сжатие бинарного файла СПО выполняется следующей командой, где 8 — степень 2 при задании размера «скользящего окна», 4 — степень 2 при задании «глубины просмотра»:

```
chelsea@roxy:~/Arduino$ ./heatshrink -e -w 8 -l 4 myfile.bin
compressed.bin
```

- создание заголовочного файла с массивом, содержащим сжатые данные.

Массив генерировался скриптом на языке Python, приведенным ниже:

```
binfile = open('/home/chelsea/Arduino/compressed.bin', 'rb')
array = []
data = binfile.read()
for byte in data:
    array.append(byte)
print(array)
```

```
Running: /home/chelsea/Py/diplo/bin.py (Wed Mar 30 12:24:45
2016)
```

```
[134, 101, 41, 147, 88, 102, 82, 232, 0, 109, ... 155, 128, 31,
```

В результате было получено СПО, отличающееся от оригинального, но выдающее при криптографической проверке целостности по процедуре, описанной в подразделе 2.1, то же самое значение имитовставки.

3 Анализ и выбор криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности СПО микроконтроллеров

3.1 Описание доступных криптографических примитивов

3.1.1 Блочные шифры

«ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» [6] (Магма) — советский и российский стандарт симметричного шифрования, введённый в 1990 году. Данный алгоритм удовлетворяет военным критериям.

Алгоритм криптографического преобразования в ГОСТ 28147-89 представляет собой блочный шифр, построенный на сети Фейстеля, с размером блока 64 бита, 256-битным ключом и 32 раундами.

В дополнение к размеру ключа, ГОСТ 28147–89 имеет значительно более низкую стоимость исполнения по сравнению с AES и какими-либо ещё сходными системами шифрования. В действительности, он стоит намного меньше AES, которому требуется в четыре раза больше аппаратных логических вентилей ради значительно меньшего заявленного уровня безопасности.

Как уже рассматривалось в подразделе 2.3, ГОСТ 28147–89 не является теоретически стойким, так как показаны атаки, уменьшающие трудоемкость поиска ключа по сравнению с полным перебором. Кроме того имеет уязвимость алгоритм расчета имитовставки, связанный с процедурой дополнения блока.

ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» [11] включает помимо алгоритма ГОСТ 28147–89 под названием «Магма» криптографический алгоритм «Кузнечик» с длиной входного блока, равной 128 битам, и 256-битовым ключом.

Так же, как и AES, алгоритм «Кузнечик» основан на использовании SP-сети, однако при его разработке удалось реализовать ряд синтезных решений, которые позволяют исключить выявленные за последние годы недостатки алгоритма AES.

В «Кузнечике» при синтезе MDS-преобразования использовался подход, предложенный отечественным ученым-алгебраистом Александром Нечаевым и заключающийся в генерации матрицы с помощью линейного регистра сдвига. Такой подход позволяет в некоторых случаях существенно экономить объем требуемой для реализации алгоритма памяти и получить матрицу для всего 128-битного блока, обрабатываемого алгоритмом, в отличие от алгоритма AES, для которого такие матрицы действуют только на 32-битных подблоках. Подход, используемый в «Кузнечике», предпочтительнее, поскольку позволил в 1,5 раза сократить число итераций по сравнению с AES и обеспечить большую защиту от атак по побочным каналам. Например, как показано в статье Дениса Фомина, за счет блочности линейного преобразования AES при его реализации на платформе CUDA возможно практическое восстановление ключа с использованием Timing-атаки. Для алгоритма «Кузнечик» такая атака не применима.

В качестве нелинейного преобразования была выбрана подстановка, использованная в хэш-функции «Стрибог». Как показали 3 года интенсивных исследований отечественными и зарубежными специалистами, ее характеристики позволяют эффективно обеспечивать защиту от современных методов криптоанализа.

Алгоритм развертки ключа основан на схеме Фейстеля и использует в качестве функции усложнения итерационное преобразование алгоритма «Кузнечик». При таком подходе итерационные ключи шифрования имеют сложные нелинейные связи, что затрудняет применение атак со связанными ключами и метода биклик.

Стоит сказать, что алгоритм «Кузнечик» хотя и не имеет такой богатой истории, как алгоритм ГОСТ 28147–89, но его криптографические свойства уже подтверждены рядом научно-исследовательских работ зарубежных специалистов в области

криптографии.

Если говорить о быстродействии программной реализации алгоритма «Кузнечик», то результаты работ позволили получить скорость порядка 135 Мбайт/с на одном ядре процессора 3,3 ГГц и порядка 558 Мбайт/с на графическом процессоре. Однако аппаратная реализация алгоритма «Кузнечик» отличается от ГОСТ 28147-89 меньшей эффективностью и производительностью из-за вычислений в конечном поле. [12]

Advanced Encryption Standard (AES) — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником DES.

В июне 2003 года Агентство национальной безопасности США постановило, что шифр AES является достаточно надёжным, чтобы использовать его для защиты сведений, составляющих государственную тайну. Вплоть до уровня SECRET было разрешено использовать ключи длиной 128 бит, для уровня TOP SECRET требовались ключи длиной 192 и 256 бит. [13]

Алгоритм RC6 был разработан в 1998 г. рядом специалистов научного подразделения известнейшей фирмы RSA Data Security специально для участия в конкурсе AES. Алгоритм во многом унаследовал черты предыдущего алгоритма Рональда Ривеста — 64-битного блочного шифра RC5, разработанного в 1997 г. [14]

К достоинствам RC6 можно отнести:

- простая структура алгоритма облегчает его анализ. Кроме того, алгоритм унаследовал часть преобразований от своего предшественника — алгоритма RC5, тщательно проанализированного до конкурса AES;
- самый быстрый из алгоритмов-финалистов на 32-битных платформах;
- Зашифрование и расшифрование в алгоритме RC6 практически идентичны.

Недостатки:

- скорость шифрования при программной реализации сильно зависит от того, поддерживает ли платформа 32-битное умножение и вращение на переменное число бит;
- RC6 сложно реализуем аппаратно и в условиях ограниченных ресурсов;
- достаточно сложно защищается от атак по времени выполнения и потребляемой мощности;
- недостаточно полно поддерживает расширение ключей «на лету»;
- распараллеливание вычислений при шифровании алгоритмом RC6 реализуемо с ограничениями;
- запас криптостойкости несколько ниже, чем у остальных алгоритмов-финалистов.

Camellia — алгоритм симметричного блочного шифрования (размер блока 128 бит, ключ 128, 192, 256 бит) разработан компанией Mitsubishi Electric совместно с компанией Nippon Telegraph and Telephone Corporation (NTT). Первичный анализ алгоритма Camellia был выполнен его разработчиками. Показана стойкость алгоритма к линейному и дифференциальному криптоанализу, а также к использованию усеченных и невозможных дифференциалов, методу бумеранга, методу интерполяции, сдвиговым атакам и ряду других атак. [15]

В оценке ресурсоемкости и быстродействия алгоритма Camellia отмечается, что Camellia существенно проигрывает в скорости алгоритму Rijndael; предъявляет достаточно высокие требования к оперативной и энергонезависимой памяти.

Camellia — один из алгоритмов — победителей конкурса NESSIE. В рамках исследований, проведенных во время конкурса NESSIE, не было выявлено каких-либо проблем с криптостойкостью данного алгоритма. В настоящее время также не найдено каких-либо серьезных уязвимостей в алгоритме Camellia, однако видно, что активный анализ этого алгоритма будет продолжаться.

3.1.2 Поточные шифры

При работе ГОСТ 28147-89 [6] в режиме гаммирования (Режим гаммирования с обратной связью по выходу (Output Feedback, OFB)) особым образом формируется криптографическая гамма, которая затем побитно складывается по модулю 2 с исходным открытым текстом для получения шифротекста. Шифрование в режиме гаммирования лишено недостатков, присущих режиму простой замены. Так, даже идентичные блоки исходного текста дают разный шифротекст, а для текстов с длиной, не кратной 64 бит, «лишние» биты гаммы отбрасываются. Кроме того, гамма может быть выработана заранее, что соответствует работе шифра в поточном режиме. Выработка гаммы происходит на основе ключа и так называемой синхропосылки, которая задает начальное состояние генератора.

Особенность работы ГОСТ 28147-89 в режиме гаммирования заключается в том, что при изменении одного бита шифротекста изменяется только один бит расшифрованного текста. С одной стороны, это может оказывать положительное влияние на помехозащищённость; с другой - злоумышленник может внести некоторые изменения в текст, даже не расшифровывая его.

Алгоритм шифрования с обратной связью по шифротексту (Cipher Feedback, CFB) похож на режим гаммирования, однако гамма формируется на основе предыдущего блока зашифрованных данных, так что результат шифрования текущего блока зависит также и от предыдущих блоков. По этой причине данный режим работы также называют гаммированием с зацеплением блоков. При изменении одного бита шифротекста, полученного с использованием алгоритма гаммирования с обратной связью, в соответствующем блоке расшифрованного текста меняется только один бит, так же затрагивается последующий блок открытого текста. При этом все остальные блоки остаются неизменными.

При использовании данного режима следует иметь в виду, что синхропосылку

нельзя использовать повторно (например, при шифровании логически отдельных блоков информации - сетевых пакетов, секторов жёсткого диска и т. п.). Это обусловлено тем, что первый блок шифр-текста получен всего лишь сложением по модулю два с зашифрованной синхропосылкой; таким образом, знание всего лишь 8 первых байт исходного и шифрованного текста позволяют читать первые 8 байт любого другого шифротекста после повторного использования синхропосылки.

RC4 — потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях (например, в протоколах SSL и TLS, алгоритмах обеспечения безопасности беспроводных сетей WEP и WPA). Алгоритм RC4 строится на основе генератора псевдослучайных битов. На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Был признан устаревшим многими ИТ-компаниями. [16]

Основные преимущества шифра:

- высокая скорость работы;
- переменный размер ключа.

RC4 довольно уязвим, если:

- используются не случайные или связанные ключи;
- один ключевой поток используется дважды.

Rabbit — высокоскоростной поточный шифр впервые представленный в феврале 2003 года. Rabbit используют 128-битный ключ и 64-битный инициализирующий вектор. Шифр был разработан с целью использования в программном обеспечении, как обладающий высокой скоростью шифрования. Тем не менее, шифр также оказался быстрым и компактным при реализации в аппаратном обеспечении.

Основным компонентом шифра является генератор битового потока, который шифрует 128 битов сообщения за итерацию. Достоинство шифра в тщательном перемешивании его внутренних состояний между двумя последовательными итерациями.

Функция перемешивания полностью основана на арифметических операциях, доступных на современных процессорах, то есть S-блоки подстановок и поисковые таблицы не нужны для реализации шифра.

Rabbit является одной из старейших конструкций проекта eSTREAM. Данный поточный шифр не был подвержен каким-либо модификациям или дополнениям. Его спецификация оставалась неизменной с 2003 года и по данный момент. Шифр пережил все три этапа проекта и не на одном не был подвержен криптоаналитическим атакам. Кроме всего прочего данный алгоритм очень хорошо реализуется на новых процессорах семейства Intel. Как недостаток можно заметить тот факт, что шифр Rabbit обеспечивает уровень безопасности только в 128 бит. Rabbit предоставляет 128-битную защиту против атакующих, чья цель — один уникальный ключ. Если же атака происходит на несколько ключей за раз, и всё равно, который из них взломают, то защищённость снижается до 96 бит. [17]

Гаммирование — метод симметричного шифрования, заключающийся в «наложении» последовательности, состоящей из случайных чисел, на открытый текст. Клод Шеннон доказал, что при определённых свойствах гаммы этот метод шифрования является абсолютно стойким (то есть, не поддающимся взлому).

Для формирования гаммы (последовательности псевдослучайных чисел) нужно использовать аппаратные генераторы случайных чисел, основанные на физических процессах. Если гамма не будет случайной, для получения открытого текста потребуется подобрать только начальное состояние генератора псевдослучайных чисел.

Длина гаммы должна быть не меньше длины защищаемого сообщения (открытого текста). В противном случае для получения открытого текста потребуется подобрать длину гаммы, проанализировать блоки шифротекста угаданной длины, подобрать биты гаммы.

К недостаткам данного метода следует отнести большой размер гаммы равный

размеру шифруемых данных и сложность получения истинно случайной гаммы. Однако, данный метод обеспечивает наилучшую криптостойкость и наивысшую скорость шифрования.

3.1.3 Асимметричные шифры

ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи» [18] — российский стандарт, описывающий алгоритмы формирования и проверки электронной цифровой подписи. Принят и введён в действие Приказом Федерального агентства по техническому регулированию и метрологии вместо ГОСТ Р 34.10-2001 [19].

ГОСТ Р 34.10-2012 основан на эллиптических кривых. Стойкость основывается на сложности вычисления дискретного логарифма в группе точек эллиптической кривой, а также на стойкости хэш-функции. Для ГОСТ Р 34.10-2012 используется хэш-функция по ГОСТ Р 34.11-2012. Новый и старый ГОСТы цифровой подписи очень похожи друг на друга. Основное отличие — в старом стандарте часть операций проводится над полем, а в новом — над группой точек эллиптической кривой, поэтому требования налагаемые на простое число в старом стандарте более жёсткие, чем в новом. Использование математического аппарата группы точек эллиптической кривой, позволяет существенно сократить порядок модуля, без потери криптостойкости.

RSA — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других. [20]

Как при шифровании и расшифровании, так и при создании и проверке подписи алгоритм RSA по существу состоит из возведения в степень, которое выполняется как ряд умножений. В практических приложениях для открытого ключа обычно выбирается

относительно небольшой показатель, а зачастую группы пользователей используют один и тот же открытый показатель, но каждый с различным модулем.

Методы «быстрого умножения» – например, методы основанные на Быстром Преобразовании Фурье (FFT – Fast Fourier Transform) – выполняются меньшим количеством шагов; тем не менее они не получили широкого распространения из-за сложности программного обеспечения, а также потому, что с типичными размерами ключей они фактически работают медленнее. Алгоритм RSA намного медленнее, чем AES и другие алгоритмы, использующие симметричные блочные шифры.

Также к недостаткам стоит отнести большую длину ключа и медленную его генерацию.

Американский стандарт цифровой подписи DSS состоит из трех частей: алгоритма хэширования SHA (Secure Hash Algorithm), алгоритма порождения параметров p , q и алгоритма подписи DSA (Digital Signature Algorithm).

Алгоритм DSA может применяться только для цифровой подписи. Имеет недостатки, связанные со сложностью генерации псевдопростых чисел для использования в алгоритме и большим размером ключей. Для сравнения, при уровне безопасности в 80 бит, размер открытого ключа DSA равен, по крайней мере, 1024 бит. [21]

3.1.4 Хэш-функции

SHA-2 — семейство криптографических алгоритмов — однонаправленных хэш-функций, включающее в себя алгоритмы *SHA-224*, *SHA-256*, *SHA-384*, *SHA-512*, *SHA-512/256* и *SHA-512/224*. Хэш-функции *SHA-2* разработаны Агентством национальной безопасности США. Хэш-функции семейства *SHA-2* построены на основе структуры Меркла — Дамгарда. Ввиду алгоритмической схожести *SHA-2* с *SHA-1* и наличия у последней потенциальных уязвимостей принято решение, что *SHA-3* будет базироваться на совершенно ином алгоритме. [22]

SHA-3 (Keccak) — алгоритм хэширования переменной разрядности. стал

победителем конкурса криптографических алгоритмов, проводимым Национальным институтом стандартов и технологий США. Он стал самым производительным на аппаратной реализации среди финалистов, а также в нём был использован нераспространённый метод шифрования — функция губки. Таким образом, атаки, рассчитанные на SHA-2, не будут работать. Ещё одним существенным преимуществом SHA-3 является возможность его реализации на миниатюрных встраиваемых устройствах. Считается одним из наиболее современных стандартов в мире хэш-функций. [23]

ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования» [24] — действующий российский криптографический стандарт, определяющий алгоритм и процедуру вычисления хэш-функции.

Исходя из конструкции новой хэш-функции, можно сделать некоторые выводы о её криптостойкости и предположить, что её криптостойкость будет выше, чем у ГОСТ Р 34.11-1994 [25].

В качестве недостатка можно указать, что значения S-блока шифра Кузнечик и хэш-функции Стрибог не являются (псевдо)случайными числами, а сгенерированы на основе скрытого алгоритма, о чем было сообщено в докладе на конференции CRYPTO 2015.

Хэш-функция ГОСТ Р 34.11-2012 может быть в два раза быстрее хэш-функции ГОСТ Р 34.11-94 на современных процессорах, но медленнее на графических картах и системах с ограниченными ресурсами. Такие результаты производительности можно объяснить тем, что при вычислении новой хэш-функции используются только сложения по модулю 2 и инструкции пересылки данных. Старая хэш-функция содержит много инструкций перемешивания, которые не лучшим образом отображаются на набор команд ЦП. Но увеличенный размер состояний и таблиц подстановки хэш-функции ГОСТ Р 34.11 — 2012 делает её медленней на высокопараллельных вычислительных средствах, таких

как графические процессоры.

3.2 Выбор криптографических примитивов, позволяющих осуществить криптографическую проверку целостности

Описанные в [26] методики имеют ограничения применения в изделиях военного и специального назначения, связанные с использованием асимметричной криптографии и криптографически нестойких методов, отсутствия возможности периодической смены ключа и др.

Для военного использования можно применять сертифицированные блочные шифры: ГОСТ 28147-89, ГОСТ Р 34.12-2015; поточные шифры: ГОСТ 28147-89, ГОСТ Р 34.12-2015 в режимах OFB/CFB, шифрование методом гаммирования; хэш-функции: ГОСТ Р 34.11-2012. Для реализации цифровой подписи может использоваться ГОСТ Р 34.10-2012.

Современные системы, например лежащие в основе SSL, HTTPS, VPN и т. п., обычно шифруют все данные с помощью секретного ключа и симметричного алгоритма. Ключ одинаков у отправителя и получателя, устанавливается в начале сессии при помощи второй, асимметричной криптосистемы. Асимметричный алгоритм используется только для передачи секретного ключа в силу своей вычислительной сложности. Безопасность асимметричной криптосистемы основана на математической сложности решения той или иной задачи, например разложения на множители достаточно больших целых чисел (алгоритм RSA). То есть речь идет о том, что злоумышленник может перехватить сообщения, но на расшифровку уйдет неразумно много времени (от десятков до миллионов лет в зависимости от длины ключа). Как выясняется, квантовые компьютеры при помощи алгоритма Шора приходят к необходимому состоянию, соответствующему найденному решению математической задачи, существенно быстрее — почти так же быстро, как обычный компьютер проводит шифрование. Таким образом, несмотря на необходимость нескольких запусков и поддержку со стороны классических компьютеров,

квантовый компьютер может за очень короткое время подобрать числа для асимметричного алгоритма, что поможет атакующему извлечь секретный ключ и дальше спокойно расшифровывать основной обмен сообщениями. Качественные симметричные алгоритмы, например AES, кстати, не имеют особенностей, которые хотя бы теоретически приводили к таким опасным последствиям. В частности, квантовый компьютер, по имеющимся оценкам, может ускорить перебор ключей AES, но не радикально. Поиск 256-битного ключа AES на квантовом компьютере эквивалентен поиску 128-битного ключа на обычном — вполне терпимо, запас прочности остается большим. [27]

Квантовый компьютер может разрушить большинство, если не все традиционные криптосистемы, развернутые на практике, а именно, все системы на основе задачи факторизации целых чисел (например RSA) или задачи дискретного логарифмирования (как традиционных, так и на эллиптических кривых Диффи-Хеллмана и DSA; а также всю криптографию, основанную на спариваниях (pairing-based cryptography)).

Некоторые классические криптосхемы, основанные на вычислительно сложных задачах, сильно отличаются от указанных выше и их гораздо сложнее решить, они остаются независимыми от квантовых вычислений, и, следовательно, были названы квантово-устойчивыми, или «постквантовыми» криптосистемами. Они состоят из криптосистем, основанных на решетках (lattice-based cryptosystems) и криптосистем, основанных на синдромах (syndrome-based cryptosystems), например McEliece и Neiderreiter.

Такие криптосистемы даже имеют преимущество по скорости по сравнению с традиционными схемами. Например, для McEliece и Neiderreiter трудность зашифрования сообщения длины n равно $O(n^2)$, тогда как Диффи-Хеллман/DSA с n -битным ключом имеет временную сложность $O(n^3)$. С другой стороны, они страдают от больших ключей по сравнению со стандартными криптосистемами. [28]

В настоящее время практических реализаций пост-квантовых конструкций крайне мало. Наиболее известными из них являются McEliece, созданная в 1978 году и Niederreiter, разработанная в 1986 году Harald Niederreiter.

В связи с этим, цифровая подпись по ГОСТ Р 34.10-2012 не может использоваться, т. к. она основана на эллиптической криптографии, которая является уязвимой к квантовым компьютерам. Алгоритмы McEliece, Niederreiter, а также алгоритмы основанные на решетках не будут рассмотрены, т. к. они слабо изучены на данный момент.

4 Разработка способа проверки целостности СПО микроконтроллеров, защищенного от обнаруженных уязвимостей

В данном разделе рассматриваются наиболее общие принципы построения систем проверки целостности СПО микроконтроллеров, так как более подробное описание может быть разработано только с учетом специфики применения контроллера.

Первым делом должна быть реализована физическая защита устройства, затрудняющая доступ к технологическим разъемам на плате. Физические средства защиты информации предотвращают доступ посторонних лиц к защищаемой информации.

Организационные меры должны обеспечивать невозможность изменения аппаратной конфигурации незаметно для обслуживающего персонала (опломбирование корпуса). Для уменьшения угрозы диверсионной деятельности и ошибок пользователей, к работе с защищаемой информацией должны привлекаться только надежные, мотивированные, квалифицированные сотрудники.

Технические меры защиты должны предусматривать возможность удаления криптографически опасной информации при вскрытии корпуса, а также по команде оператора устройства.

Также с указанной временной периодичностью должен проводиться внешний контроль, при котором проверка целостности реализуется доверенным устройством проверки целостности с доверенным хранилищем ключей проверки в контролируемой зоне (регламентный контроль). В этом случае допускается распломбирование корпуса. Например, можно проводить контроль следующим образом: вскрывается корпус, в служебному разъему микроконтроллера подключается доверенная ПЭВМ, которая выполняет считывание СПО. Расчет имитовставки производится на ПЭВМ.

Далее должна быть реализована защита кода бутлоадера — программы, которая запускается сразу же после старта микроконтроллера. Логично реализовать проверку

целостности также внешними средствами, а после сохранения кода в память микроконтроллера сделать ее аппаратными средствами доступной только для чтения и выполнения.

Для обеспечения целостности СПО контроллера могут использоваться вычисление имитовставки и шифрование. Для исключения возможности компрометации ключа при использовании имитовставки он не должен храниться в СПО. Ключ должен быть одноразовым и вводиться оператором при каждой проверке с устройства ввода ключей. При сбое процедуры проверки нужно будет перезагрузить устройство (произвести аппаратный сброс) и выполнить проверку на другом ключе. Здесь неочевидным минусом может быть разработка устройства ввода ключей для криптографической проверки целостности. Результат расчета имитовставки должен выводиться на экран оператору, который принимает решение об успешности проверки, сравнивая значение с эталонным, полученным вместе с ключом проверки.

Далее будут описаны предлагаемые способы защиты от несанкционированной модификации СПО микроконтроллеров.

Первый способ защиты — удаление не только криптографически опасной информации при НСД к устройству, но и СПО, что ведет к трудностям эксплуатации, т. к. каждый раз в устройство требуется заново сохранить СПО и произвести инициализацию. При этом также необходимо реализовать механизм гарантированного удаления СПО из flash-памяти микроконтроллера. Однако это единственный способ, за исключением внешнего контроля, который позволяет защититься от злоумышленника с высоким потенциалом.

Второй способ защиты - реализация внешнего доверенного устройства проверки целостности с доверенным хранилищем ключей проверки также позволяет защититься от злоумышленника с высоким потенциалом. Данный метод имеет следующие недостатки — сложность создания и эксплуатации малогабаритного доверенного устройства проверки

целостности СПО микроконтроллеров.

Третий вариант защиты связан с хранением в ЭНП СПО в зашифрованном виде. СПО зашифровывается и сохраняется в ЭНП микроконтроллера. Затем при старте устройства производится ввод ключа и расшифрование СПО в оперативную память. Целостность проверяется расчетом имитовставки от зашифрованного СПО. В данном случае появляются технические сложности, такие как модификация бутлоадера, сравнительно малый объем ОЗУ по сравнению с flash-памятью. Также возникает необходимость периодической смены ключа и перешифрования СПО. К случаю НСД должен быть реализован механизм гарантированной очистки оперативной памяти. Несомненный плюс данного метода — это невозможность сжатия СПО, так как в зашифрованном виде битовое распределение близко к равномерному.

Для защиты от уязвимости со сжатием СПО можно использовать аппаратный сторожевой таймер на время подсчета имитовставки. Идея метода состоит в следующем: с распаковкой имитовставка будет считаться дольше. Нужно будет предусмотреть возможные отклонения, связанные, например, с температурным режимом, когда время расчета будет отличаться.

Главное условие данного метода заключается в том, что устройство с таймером должно быть независимым (независимый старт и вывод результата оператору), иначе МК может не запрашивать значение у таймера и всегда выводить корректное значение.

В пользу использования таймера можно сказать, что помимо вычитывания всей flash-памяти микроконтроллера и подсчета имитовставки нужно выполнить распаковку. Причем архивация должна быть достаточно эффективной в плане сжатия (т. к. код СПО добивается несжимаемым шумом), то есть алгоритм распаковки достаточно сложный и относительно медленный (составление словарей и т. п.). Также исходим из того, что подсчет имитовставки выполняется быстро, то есть алгоритм оптимизирован, чтобы нельзя было его дополнительно ускорить на время, необходимое для распаковки.

Таким образом, минусы данного подхода:

- ко времени запаса помимо температурных отклонений также должна добавляться задержка таймера и дрожание кнопки. Это время может использоваться злоумышленным подмененным программным обеспечением для разархивирования;
- существует вероятность, что злоумышленник сможет написать более быструю реализацию имитовставки, используя математические, алгоритмические и программные низкоуровневые оптимизации, и также использовать выигрыш во времени для разархивирования;
- злоумышленный код может пропускать процедуру проверки целостности введенного ключа проверки, и также использовать выигрыш во времени для разархивирования;
- в свободную оперативную память можно помещать заранее распакованную часть СПО, это обычно десятая часть всей ЭНП.

Следует отметить, что в реализации на Arduino Uno в рамках данной работы выигрыша времени от описанных выше недочетов все равно не хватит на распаковку, т. к. была выявлена зависимость разницы во времени расчета примерно в 1,4 раза между оригинальной процедурой и расчетом имитовставки с распаковкой (алгоритм расчета имитовставки использовался идентичный):

- время расчета по оригинальной процедуре 1117200 мкс;
- время расчета с распаковкой 1570640 мкс.

Для защиты от этого можно также использовать промежуточное значение последнего раунда проверки целостности ключа как дополнительный блок при расчете имитовставки СПО.

Стоит отметить, что при использовании некоторых алгоритмов шифрования время расчета недетерминировано и зависит от ключа. С одной стороны использование таких алгоритмов позволяет нивелировать преимущества злоумышленника по более

скоростной реализации. В другой стороны становится невозможным использование внутреннего таймера, а для внешнего требуется с каждым ключом проверки вводить также время расчета.

Ниже описывается четвертый способ защиты.

Таймер может быть отдельной микросхемой на той же плате, где и МК. В этом случае процедура проверки целостности состоит с следующим: оператор нажимает на кнопку "проверить целостность" - отсылается аппаратная команда МК и взводится таймер (правильное время вычисления сохранено и его подменить нельзя (нельзя перепрошить либо нет прошивки как таковой)). Через заданное время таймер отправляет сигнал. И МК как завершил проверку отправляет сигнал. Сигналы поступают на аппаратную схему голосования. Если сигнал от МК приходит позже, чем от таймера, производить блокировку устройства. Время таймера должно быть рассчитано с запасом, с учетом температурных и других условий, с учетом рекомендаций от спец. организаций.

Минус данного подхода заключается в том, что для таймера и индикации должна использоваться какая-то программируемая схема, чтобы не менять элемент таймера для каждого устройства. При этом, если таймера есть программное обеспечение, его также можно скомпрометировать. Здесь можно использовать в качестве таймера ПЛИС, в которой после сохранения конфигурации аппаратно запрещается ее изменение (запаять служебные лапки, не оставляя служебного разъема). При смене СПО значение таймера не изменится, т. к. процедура расчета зафиксирована, и объем ЭНП неизменен.

Данный способ не только защищает от обнаруженной уязвимости со сжатием, но и позволяет защититься от подмены бутлоадера и всех типов нарушителя, кроме высокого.

Пятый способ защиты - сторожевой таймер, реализованный в виде отдельного доверенного устройства. В этом случае при вводе ключа с устройства ввода ключей нужно взводить таймер одновременно с началом ввода ключа. На ключевом носителе

должна быть сигнализация о чтении ключа и блокировка информационного обмена с МК, чтобы МК не мог раньше взвода таймера вычитать ключ и начать рассчитывать имитовставку. Таймер и ввод ключа в этом случае начинаются одновременно по кнопке, а до этого момента УВК отключено (вставлен в разъем, но заблокирован).

Этот механизм защищает от злоумышленника с высоким потенциалом.

Однако существует сложность создания доверенного устройства и его интеграция с проверяемым устройством, реализуемым МК, и УВК. Логичным кажется вариант подключать данное устройство в разрез между ключевым носителем и контролируемым микроконтроллером.

Все разработанные способы были сравнены. Результаты сравнения приведены в сводной таблице 6.

Таблица 6 — Сравнение разработанных способов обеспечения целостности СПО МК

№ способа	Суть способа	Потенциал злоумышленника	Недостатки
1	Удаление СПО	высокий	— реализация гарантированного удаления СПО; — трудности эксплуатации.
2	Внешнее доверенное устройство проверки	высокий	— сложность создания; — трудности эксплуатации.
3	Шифрование СПО	средний	— гарантированная очистка ОЗУ; — модификация бутлоадера; — недостаточный объем ОЗУ.
4	Сторожевой таймер на плате с МК	средний	— время запаса, которое может использовать злоумышленник; — подмена конфигурации таймера.

Продолжение таблицы 6

№ способа	Суть способа	Потенциал злоумышленника	Недостатки
5	Доверенное устройство со сторожевым таймером	высокий	<ul style="list-style-type: none"> — сложность создания доверенного устройства; — время запаса, которое может использовать злоумышленник; — сложность интеграции.

Таким образом, способы, позволяющие защититься от злоумышленника с высоким потенциалом сложны в разработке и эксплуатации. Предлагается использовать комплекс организационных и технических мер. Для защиты от злоумышленника со средним потенциалом использовать микросхему со сторожевым таймером, расположенным на той же плате, что и МК. Для защиты от злоумышленника с высоким потенциалом использовать доверенное устройство со сторожевым таймером.

5 Реализация разработанного способа на языке программирования С для платформы Arduino Uno

5.1 Описание программной реализации

Следующим этапом проекта стало макетирование пятого способа защиты - сторожевого таймера, реализованного в виде отдельного доверенного устройства. В качестве доверенного устройства, реализующего УВК и сторожевой таймер, использовался ноутбук, к которому через последовательный интерфейс подключено контролируемое устройство — платформа Arduino Uno с микроконтроллером AVR.

Для реализации функций доверенного устройства была создана программная модель - был написан скрипт на языке Python с использованием библиотеки pySerial. [29]

Фрагмент кода, реализующий инициализацию проверяемого устройства приведен ниже:

```
import serial
from datetime import datetime

# инициализация последовательного интерфейса
ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)

# взаимодействие с микроконтроллером
line = ser.readline()
print(line)

line = ser.readline()
print(line)

line = ser.readline()
print(line)
```

Сторожевой таймер сконфигурирован на время вычисления имитовставки — 1228920 мкс, где 1117200 мкс — время расчета имитовставки по оригинальной процедуре в данной реализации, полученное в предыдущем опыте. Время запаса составляет 10%.

Фрагмент кода, реализующий функции сторожевого таймера и ввод ключа приведен ниже:

```
# ввод ключа
key = {0x88, 0x99, 0xaa, 0xbb,
       0xcc, 0xdd, 0xee, 0xff,
       0x00, 0x11, 0x22, 0x33,
       0x44, 0x55, 0x66, 0x77,
       0xfe, 0xdc, 0xba, 0x98,
       0x76, 0x54, 0x32, 0x10,
       0x01, 0x23, 0x45, 0x67,
       0x89, 0xab, 0xcd, 0xef}

keystroke = str(key).encode('ascii')
ser.write(keystroke)

# инициализация сторожевого таймера
good_time = 1228920
dt = datetime.now()
timer = dt

ser.write('\n'.encode('utf-8'))
line = ser.readline()
print('key: ' + str(line))

line = ser.readline()

# получение времени проверки
dt = datetime.now()
timer = dt - timer

print(line.decode('utf-8'))
```

```
answer = line.decode('utf-8')
var = answer.split(':')
```

Далее программная модель доверенного устройства производит проверку полученного значения имитовставки и сравнивает время вычисления с эталонным. Для удобства принятия решения о правильности проверки в рамках моделирования, оператору выдаются сообщения об успешности проверок. Фрагмент кода, реализующий функции проверки, приведен ниже:

```
print('Время расчета от доверенного устройства: ', timer)
times = int(timer.seconds) * 1000000 + int(timer.microseconds)

if (times > good_time):
    print('Проверка провалена. Время вычисления превышено')
else:
    print('Проверка времени вычисления пройдена')

if ('c125 fff1 b9e8 c6a8' != var[1]):
    print('Проверка провалена. Имитовставка не совпадает')
else:
    print('Проверка имитовставки пройдена')
ser.close() # close port
```

5.2 Вывод результатов

В этом подразделе представлены примеры результатов работы программной модели сторожевого таймера, реализованного в виде отдельного доверенного устройства, описанной в подразделе 5.1.

В качестве СПО микроконтроллера и скомпрометированного методом сжатия СПО использовались программы из разделов 2.2 и 2.4 соответственно.

Вывод результатов работы модели доверенного устройства при проверке оригинального СПО приведен ниже:

Running: /home/chelsea/Py/diplo/ser.py (Thu May 26 13:30:40 2016)

b'Init complete!\r\n'
b'Enter key: \r\n'
key: b'307b 202c 2c31 3120 3633 202c 3331 2c37 3120 2c36 3120 2c37 3120 3235 202c 3531 '
имитовставка: c125 fff1 b9e8 c6a8 Time: 1117108

Время расчета от доверенного устройства: 0:00:01.128678

Проверка времени вычисления пройдена

Проверка имитовставки пройдена

Время вычисления имитовставки, подсчитанное на микроконтроллере, слегка отличается от времени, полученного моделью доверенного устройства (1117108 мкс против 1128678 мкс), что является следствием задержки в линии связи и задержки при выводе результатов на экран. Однако проверка была пройдена, т. к. время задержки укладывается в 10% запас.

Далее в микроконтроллер было сохранено скомпрометированное СПО. Как можно увидеть на выводе результата работы модели при проверке целостности скомпрометированного СПО, приведенного ниже, хотя значение имитовставки получено идентичное, проверка целостности не считается успешно пройденной.

Вывод результатов работы модели доверенного устройства при проверке скомпрометированного СПО приведен ниже:

Running: /home/chelsea/Py/diplo/ser.py (Thu May 26 13:35:53 2016)

b'Init complete!\r\n'
b'Enter key: \r\n'

*key: b'307b 202c 6c31 f120 3633 102c a331 5c37 3120 4c36 7120
2c37 0120 3235 812c 3531 '*

имитовставка: c125 fff1 b9e8 c6a8 Time: 1570900

Время расчета от доверенного устройства: 0:00:01.574062

Проверка провалена. Время вычисления превышено

Проверка имитовставки пройдена

Таким образом, моделирование показало, что использование доверенного устройства со сторожевым таймером позволяет защититься от компрометации СПО микроконтроллера с использованием сжатия. Как было показано в разделе 4, данный метод позволит защититься от злоумышленника с высоким потенциалом.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта были решены следующие задачи:

- произведён анализ способа криптографической проверки целостности специального программного обеспечения микроконтроллеров, применяемого в настоящее время, с целью поиска в нем уязвимостей;
- разработана модель угроз и нарушителя компрометации специального программного обеспечения микроконтроллеров;
- проведен обзор способов защиты программного обеспечения микроконтроллеров, описанных в открытой литературе. Результаты были опубликованы в [26];
- был выполнен анализ и выбор криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности специального программного обеспечения микроконтроллеров. Результаты анализа были опубликованы в [30];
- проведена компрометация способа криптографической проверки целостности, используемой в настоящее время. Результаты анализа приняты к публикации в [31].

В результате дипломного проектирования был разработан способ обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров, защищенный от обнаруженных уязвимостей. Результаты выполнения работы используются в рабочем процессе АО «ПНИЭИ», что подтверждено актом о внедрении, представленном в приложении В.

Таким образом, все пункты технического задания выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 ГОСТ Р 34.13—2015 «Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров» // Официальное издание. – М.: Стандартиформ, 2015.
- 2 Доктрина информационной безопасности Российской Федерации (утв. Президентом РФ от 9 сентября 2000 г. N Пр-1895) // Российская газета – №187, 2000.
- 3 ГОСТ Р ИСО/МЭК 15408-3-2008. Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий. Часть 3. Требования доверия к безопасности // Официальное издание. – М.: Стандартиформ, 2009.
- 4 Arduino Uno [Электронный ресурс] : Русскоязычный сайт проекта Ардуино. – Режим доступа: <http://arduino.ru/Hardware/ArduinoBoardUno>, свободный. – Загл. с экрана.
- 5 Python 3.5.1 documentation [Электронный ресурс] : Python 3.5.1 documentation. – 2016. – Режим доступа: <https://docs.python.org/3/>, свободный. – Загл. с экрана.
- 6 ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» // Официальное издание - М.: ИПК Издательство стандартов, 1996.
- 7 Взломан блочный шифр ГОСТ 28147-89 [Электронный ресурс]: Информационный портал Positive Technologies. – Режим доступа: <http://www.securitylab.ru/news/405678.php>, свободный. – Загл. с экрана.
- 8 Padding oracle attack [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: https://en.wikipedia.org/wiki/Padding_oracle_attack, свободный. – Загл. с экрана.
- 9 «Immunising CBC Mode Against Padding Oracle Attacks: A Formal Security Treatment». Security and Cryptography for Networks. // SCN 2008, Lecture Notes in Computer

Science (Springer Verlag), 2008, с. 340–357.

10 Lempel–Ziv–Storer–Szymanski [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: <https://en.wikipedia.org/wiki/Lempel-Ziv-Storer-Szymanski>, свободный. – Загл. с экрана;

11 ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» // Официальное издание. – М.: Стандартинформ, 2015.

12 ГОСТ Р 34.12–2015: чего ожидать от нового стандарта? [Электронный ресурс] : научный журнал «Information Security / Информационная безопасность». – Электрон. журн. – №4, 2015. – Режим доступа к журн.: <http://www.itsec.ru/articles2/crypto/gost-r-chego-ozhidat-ot-novogo-standarta/>, свободный. – Загл. с экрана.

13 Advanced Encryption Standard [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard, свободный. – Загл. с экрана.

14 RC6 [Электронный ресурс] : Википедия, свободная энциклопедия, – 2015. – Режим доступа: <https://ru.wikipedia.org/wiki/RC6>, свободный. – Загл. с экрана.

15 Camellia (алгоритм) [Электронный ресурс] : Википедия, свободная энциклопедия, – 2015. – Режим доступа: [https://ru.wikipedia.org/wiki/Camellia_\(алгоритм\)](https://ru.wikipedia.org/wiki/Camellia_(алгоритм)), свободный. – Загл. с экрана.

16 RC4 [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: <https://ru.wikipedia.org/wiki/RC4>, свободный. – Загл. с экрана.

17 Rabbit [Электронный ресурс] : Википедия, свободная энциклопедия, – 2015. – Режим доступа: <https://ru.wikipedia.org/wiki/Rabbit>, свободный. – Загл. с экрана.

18 ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой

подписи» // Официальное издание. – М.: Стандартинформ, 2012.

19 ГОСТ Р 34.10-2001 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи» // Официальное издание. – М.: Госстандарт России, 2001.

20 RSA [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: <https://ru.wikipedia.org/wiki/RSA>, свободный. – Загл. с экрана.

21 Digital Signature Standard [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: https://ru.wikipedia.org/wiki/Digital_Signature_Standard, свободный. – Загл. с экрана.

22 SHA-2 [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: <https://ru.wikipedia.org/wiki/SHA-2>, свободный. – Загл. с экрана.

23 SHA-3 [Электронный ресурс] : Википедия, свободная энциклопедия, – 2016. – Режим доступа: <https://ru.wikipedia.org/wiki/SHA-3>, свободный. – Загл. с экрана.

24 ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования» // Официальное издание. – М.: Стандартинформ, 2012.

25 ГОСТ Р 34.11-94 «Информационная технология. Криптографическая защита информации. Функция хэширования» // Официальное издание. – М.: Госстандарт России, 2001.

26 Кашубина А. П. Обзор методов обеспечения криптографической целостности в различных программно-аппаратных средах // Актуальные направления научных исследований XXI века: теория и практика - Сборник работ международной научно-практической конференции «Молодежный форум: технические и математические науки» / под ред. Бугакова В. М. - Воронеж: Изд-во ФГБОУ ВО «ВГЛТУ», 2015. – с. 284-288.

27 Квантовые компьютеры и конец безопасности [Электронный ресурс] : блог

АО «Лаборатория Касперского», – 2013. – Режим доступа: <https://blog.kaspersky.ru/kvantovye-kompyutery-i-konec-bezopasnosti/1989/>, свободный. – Загл. с экрана.

28 Misoczki1Rafael, Barreto Paulo S. L. M. Compact McEliece Keys from Goppa Codes Post-quantum cryptography. // 16th Annual International Workshop, SAC 2009 / под ред. Michael J. Jacobson Jr.. - Calgary, Alberta, Canada, 2009. – с. 376-392.

29 PySerial 3.0 documentation [Электронный ресурс] : PySerial 3.0 documentation, – 2015. – Режим доступа: <https://pythonhosted.org/pyserial/index.html>, свободный. – Загл. с экрана.

30 Кашубина А. П. Выбор криптографических примитивов, позволяющих осуществить криптографическую проверку целостности программного обеспечения микроконтроллеров // Сборник статей III ежегодной межвузовской научно-практической конференции «Информационные технологии в науке и образовании. Проблемы и перспективы» / под ред. Фионовой Л. Р., Дурина А. В. - Пенза : Изд-во ПГУ, 2016. – с. 445-447.

31 Кашубина А. П. Компрометация механизма обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров // научный журнал "Инжиниринг и технологии". - №1 2016 / под ред. Кревчика В. Д., Печерской Е. А. - Пенза : Изд-во ПГУ, 2016 (находится в процессе публикации).

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный текст программной реализации способа обеспечения
криптографической проверки целостности СПО микроконтроллеров, используемого в
настоящее время, для платформы Arduino Uno

```

/**
  * Диплом
  * @author Chelsea @date 21.03.2016
  */

#include <SoftwareSerial.h>
#include <avr/pgmspace.h>
#include <noise.h>

#define PRINT(a) (Serial.print(a))
#define PRINT_HEX(a) (PRINT(String(a, HEX)))

#define BLOCK_SIZE (4) // in words
#define KEY_SIZE (16) // in words

const unsigned short PERMUTATION_TABLE_1[256] PROGMEM = {182, 253, 35,
104, 221, 79, 174, 163, 14, 29, 160, 121, 38, 92, 233, \
131, 52, 6,
108, 247, 114, 159, 27, 2, 106, 110, 138, 212, 214, 153, \
97, 244, 202,
211, 41, 215, 51, 59, 54, 130, 147, 200, 102, 224, 26, \
252, 69, 194,
113, 185, 99, 89, 149, 115, 226, 58, 148, 7, 186, 180, \
133, 181,
188, 10, 152, 63, 232, 122, 157, 98, 109, 196, 125, 207, \
83, 11, 42,
241, 120, 219, 167, 151, 76, 82, 75, 240, 67, 238, 230, \
17, 127, 184,
246, 245, 172, 145, 142, 39, 65, 198, 154, 32, 150, 49, \
96, 176, 23,
139, 86, 170, 210, 80, 66, 216, 228, 134, 21, 161, 93, 56, \
242, 22, 40,
62, 13, 18, 3, 100, 60, 197, 175, 64, 162, 126, 78, 254, \
146, 123,
155, 91, 37, 165, 250, 19, 187, 229, 73, 117, 57, 136, 124, \
55, 0, 68,
189, 169, 140, 191, 48, 16, 45, 208, 190, 25, 71, 20, 218, \
28, 251, 9,
239, 50, 199, 137, 61, 81, 43, 118, 237, 235, 135, 31, 158, \
8, 72, 236,
227, 132, 85, 5, 101, 111, 204, 141, 36, 166, 15, 116, 119, \
1, 183, 222,
203, 24, 87, 195, 129, 156, 143, 177, 112, 171, 223, 178, \
88, 107, 201,
173, 206, 94, 44, 74, 95, 231, 164, 220, 34, 47, 225, 30, \
243, 144,
103, 192, 168, 84, 213, 12, 77, 46, 70, 179, 4, 53, 90, 249, \
105, 128,
248, 33, 193, 209, 205, 255, 234, 217
};

const unsigned short PERMUTATION_TABLE_2[256] PROGMEM = {72, 51, 100,
144, 31, 192, 43, 12, 124, 66, 57, 13, 201, 95, 94, 254, 128, 114, \
120, 75, 54,
197, 131, 16, 18, 134, 203, 44, 2, 215, 160, 194, 0, 170, 64, 224, \
87, 82, 130,
145, 136, 155, 140, 19, 234, 46, 103, 151, 117, 195, 109, 132, 219, \
171, 123,
191, 179, 65, 249, 126, 106, 122, 119, 169, 84, 255, 190, 129, 231, 168, \
138, 147,
188, 80, 142, 243, 230, 216, 96, 60, 1, 229, 92, 210, 222, 199, 42, \

```

```

35, 153, 212,
39, 164, 220, 91, 211, 81, 70, 248, 30, 118, 90, 158, 161, 237, \
226, 9, 93, 111, 213, 149, 78, 85, 74, 185, 127, 20, 146, 25, 141, \
253, 156, 246, 198, 247, 56, 108, 165, 238, 63, 17, 11, 228, 29, 174, \
86, 113, 178, 50, 184, 166, 15, 48, 225, 76, 49, 71, 101, 189, 61, \
244, 135, 209, 105, 152, 208, 251, 121, 88, 139, 159, 36, 58, 23, \
8, 7, 45, 26, 177, 40, 217, 148, 182, 98, 27, 83, 245, 21, 233, \
172, 110, 69, 125, 187, 232, 204, 34, 206, 157, 176, 236, 205, 104, \
89, 59, 73, 196, 79, 175, 5, 41, 183, 24, 28, 116, 218, 10, 133, 115, \
221, 77, 180, 181, 250, 112, 240, 173, 14, 3, 137, 32, 47, 102, 239, \
68, 143, 207, 162, 200, 241, 167

```

```

};
const unsigned short PERMUTATION_TABLE_3[256] PROGMEM = {165, 90, 122,
59, 6, 143, 37, 214, 88, 123, 42, 159, 92, 250, 35, 232, 163, 207, \
240, 216, 89, 190, 104, 192, 191, 91, 121, 217, 113, 115, 145, 70, \
227, 223, 189, 114, 148, 105, 231, 208, 39, 212, 32, 63, 247, 137, 127, \
220, 174, 116, 30, 120, 124, 183, 101, 54, 204, 179, 244, 58, 48, \
22, 111, 138, 193, 177, 96, 151, 225, 107, 233, 221, 33, 150, 16, \
21, 61, 194, 29, 26, 49, 51, 235, 126, 149, 44, 164, 206, 99, 147, \
31, 55, 13, 43, 10, 248, 153, 185, 34, 198, 36, 222, 129, 4, 155, \
210, 215, 81, 76, 142, 236, 152, 228, 224, 226, 182, 24, 252, 95, \
254, 162, 14, 136, 97, 196, 53, 156, 8, 9, 172, 154, 199, 100, 186, 201, \
167, 134, 195, 67, 176, 246, 77, 74, 238, 230, 181, 65, 109, 103, \
237, 139, 106, 140, 60, 125, 253, 112, 128, 251, 68, 46, 130, 241, \
211, 5, 239, 173, 71, 1, 180, 85, 178, 98, 40, 135, 19, 94, 144, 79, \
245, 75, 50, 11, 158, 3, 200, 69, 169, 72, 218, 243, 18, 82, 47, \
146, 62, 242, 131, 213, 80, 12, 20, 202, 187, 64, 119, 84, 133, 160, \
249, 7, 2, 229, 27, 57, 117, 132, 219

```

```

};
const unsigned short PERMUTATION_TABLE_4[256] PROGMEM = {133, 225,
105, 85, 122, 78, 19, 76, 104, 94, 137, 200, 157, 103, 33, 7, 64, 79, \
251, 60, 72, 208, 119, 47, 5, 125, 117, 240, 3, 255, 205, 115, 156, \
179, 93, 213, 180, 176, 91, 245, 8, 235, 129, 127, 244, 199, 215, 13, \
82, 71, 218, 210, 191, 23, 206, 181, 153, 29, 234, 22, 102, 254, 49, \

```

```

230, 18, 43,
98, 21, 197, 20, 0, 144, 151, 143, 138, 154, 6, 184, 2, 145, 41, \
63, 141, 53, 171, 126, 173, 211, 174, 135, 70, 222, 248, 214, 111, \
224, 158, 45,
188, 150, 55, 130, 31, 149, 108, 80, 96, 190, 229, 95, 132, 109, \
128, 107, 168, 231, 221, 249, 121, 172, 75, 207, 131, 15, 193, 247, \
165, 10, 238,
120, 252, 202, 16, 90, 113, 112, 212, 81, 134, 35, 160, 194, 89, \
42, 86, 101,
196, 146, 142, 147, 30, 32, 223, 236, 250, 12, 99, 9, 39, 155, 66, \
36, 167, 110,
175, 139, 178, 182, 88, 185, 186, 148, 209, 24, 219, 140, 114, 198, \
50, 203, 48,
253, 162, 187, 106, 74, 54, 11, 152, 233, 124, 136, 195, 38, 92, 216, \
183, 189,
217, 226, 26, 169, 68, 87, 57, 246, 61, 34, 83, 46, 52, 77, 170, 228, \
116, 204,
1, 4, 242, 27, 56, 166, 37, 73, 164, 84, 118, 192, 44, 69, 159, \
177, 65, 100,
25, 220, 241, 237, 51, 97, 59, 28,
201, 163, 14,

```

```
};
```

```
unsigned short permutation = 0;
```

```

void processBlock(unsigned short *pData, unsigned short *pResult,
unsigned short *pKey)
{
    unsigned short tmp1 = 0, tmp2 = 0, r1, r2;

    int i, k;

    for (k = 0; k < BLOCK_SIZE; k++)
    {
        pResult[k] ^= pData[k];
    }

    for (i = 0; i < 16; i++)
    {
        r1 = pResult[0];
        r2 = pResult[1];

        for (k = 0; k < 2; k++)
        {
            pResult[k] += pKey[(i * 2 + k) % 16];
        }

        // tmp1 = PERMUTATION_TABLE_1[pResult[0] >> 8];
        permutation = pgm_read_word(&(PERMUTATION_TABLE_1[(pResult[0] >>
8)]));
        // PRINT(permutation);
        tmp1 = permutation;

        // pResult[0] = (tmp1 << 8) | (PERMUTATION_TABLE_2[(pResult[0]) &
0xFF]);
        permutation = pgm_read_word(&(PERMUTATION_TABLE_2[(pResult[0]) &
0xFF]));
        pResult[0] = (tmp1 << 8) | permutation;
    }
}

```

```

        // tmp2 = PERMUTATION_TABLE_3[pResult[1] >> 8];
        permutation = pgm_read_word(&(PERMUTATION_TABLE_3[pResult[1] >>
8]]));
        tmp2 = permutation;

        // pResult[1] = (tmp2 << 8) | (PERMUTATION_TABLE_4[(pResult[1])
& 0xFF]);
        permutation = pgm_read_word(&(PERMUTATION_TABLE_4[(pResult[1]) &
0xFF]]));
        pResult[1] = (tmp2 << 8) | permutation;

        tmp1 = ((pResult[0]) & 0xFFE0) >> 5;
        pResult[0] <<= 11;
        tmp2 = ((pResult[1]) & 0xFFE0) >> 5;
        pResult[1] <<= 11;
        pResult[0] ^= tmp2;
        pResult[1] ^= tmp1;

        pResult[0] ^= pResult[2];
        pResult[1] ^= pResult[3];
        pResult[2] = r1;
        pResult[3] = r2;
    }
    return;
}

void ToggleLed()
{
    static bool bValue = HIGH;
    digitalWrite(13, bValue);
    if (bValue == HIGH)
    {
        bValue = LOW;
    }
    else
    {
        bValue = HIGH;
    }
    return;
}

void Led3Flashes()
{
    byte i;
    static bool bValue = LOW;
    digitalWrite(13, bValue);
    for (i = 0; i < 6; i++)
    {
        delay(300);
        ToggleLed();
    }
    delay(300);
    return;
}

void LedDeath()
{
    while (1)
    {
        Led3Flashes();
    }
}

```

```

    }
}

void ReadMemoryBlock(unsigned short blockNumber, unsigned short*
pBuffer)
{
    uint16_t i;
    for (i = blockNumber * BLOCK_SIZE; i < ((blockNumber + 1) *
BLOCK_SIZE); i++)
    {
        pBuffer[i - blockNumber * BLOCK_SIZE] = pgm_read_word((void *)i);
    }
    return;
}

void setup() {
    // put your setup code here, to run once:
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
    // Open serial communications and wait for port to open:
    Serial.begin(115200);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port
only
    }
    Serial.println(F("Init complete!"));

    unsigned short Buffer[BLOCK_SIZE];
    unsigned short i;
    unsigned short key[KEY_SIZE] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
    unsigned short result[BLOCK_SIZE] = {0, 0, 0, 0};

    Serial.println(F("Enter key: "));
    Serial.setTimeout(0xffffffff);
    Serial.readBytesUntil(0, (byte*)key, sizeof(key));

    for (i = 0; i < KEY_SIZE; i++)
    {
        PRINT_HEX(key[i]);
        PRINT(" ");
    }

    for (i = 0; i < 512; i++)
    {
        ReadMemoryBlock(i, Buffer);

        /* for (i = 0; i < 4; i++)
        {
            PRINT_HEX(Buffer[i]);
            PRINT("\n ");
        }
        */
        processBlock(Buffer, result, key);
    }
    PRINT("ИМИТОВСТАВКА: ");
    for (i = 0; i < BLOCK_SIZE; i++)
    {

```

```

    PRINT_HEX(result[i]);
    PRINT(" ");
}
}

void loop() {
    // put your main code here, to run repeatedly:
    //PRINT_HEX(45);
    delay(1000);
    byte i;
    for (i = 0; i < 25046; i++)
    {
        i = pgm_read_byte(&(noize[i]));
    }
    // if (Serial.available()) {
    //     Serial.write(Serial.read());
    // }
}

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Исходный текст компрометации способа обеспечения криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, для платформы Arduino Uno

```

/**
  * Диплом
  * @author Chelsea @date 21.03.2016
  */

#include <SoftwareSerial.h>
#include <avr/pgmspace.h>
#include <heatshrink_decoder.h>
#include <compressed.h>

#define PRINT(a) (Serial.print(a))
#define PRINT_HEX(a) (PRINT(String(a, HEX)))

#define BLOCK_SIZE (4) // in words
#define KEY_SIZE (16) // in words
#define DECOMPRESSED_BUFFER_SIZE (32) // in bytes

const unsigned short PERMUTATION_TABLE_1[256] PROGMEM = {182, 253, 35,
104, 221, 79, 174, 163, 14, 29, 160, 121, 38, 92, 233, \
131, 52, 6,
108, 247, 114, 159, 27, 2, 106, 110, 138, 212, 214, 153, \
97, 244, 202,
211, 41, 215, 51, 59, 54, 130, 147, 200, 102, 224, 26, \
252, 69, 194,
113, 185, 99, 89, 149, 115, 226, 58, 148, 7, 186, 180, \
133, 181,
188, 10, 152, 63, 232, 122, 157, 98, 109, 196, 125, 207, \
83, 11, 42,
241, 120, 219, 167, 151, 76, 82, 75, 240, 67, 238, 230, \
17, 127, 184,
246, 245, 172, 145, 142, 39, 65, 198, 154, 32, 150, 49, \
96, 176, 23,
139, 86, 170, 210, 80, 66, 216, 228, 134, 21, 161, 93, 56, \
242, 22, 40,
62, 13, 18, 3, 100, 60, 197, 175, 64, 162, 126, 78, 254, \
146, 123,
155, 91, 37, 165, 250, 19, 187, 229, 73, 117, 57, 136, 124, \
55, 0, 68,
189, 169, 140, 191, 48, 16, 45, 208, 190, 25, 71, 20, 218, \
28, 251, 9,
239, 50, 199, 137, 61, 81, 43, 118, 237, 235, 135, 31, 158, \
8, 72, 236,
227, 132, 85, 5, 101, 111, 204, 141, 36, 166, 15, 116, 119, \
1, 183, 222,
203, 24, 87, 195, 129, 156, 143, 177, 112, 171, 223, 178, \
88, 107, 201,
173, 206, 94, 44, 74, 95, 231, 164, 220, 34, 47, 225, 30, \
243, 144,
103, 192, 168, 84, 213, 12, 77, 46, 70, 179, 4, 53, 90, 249, \
105, 128,
248, 33, 193, 209, 205, 255, 234, 217

};

const unsigned short PERMUTATION_TABLE_2[256] PROGMEM = {72, 51, 100,
144, 31, 192, 43, 12, 124, 66, 57, 13, 201, 95, 94, 254, 128, 114, \
120, 75, 54,
197, 131, 16, 18, 134, 203, 44, 2, 215, 160, 194, 0, 170, 64, 224, \
87, 82, 130,
145, 136, 155, 140, 19, 234, 46, 103, 151, 117, 195, 109, 132, 219, \
171, 123,

```

```

191, 179, 65, 249, 126, 106, 122, 119, 169, 84, 255, 190, 129, 231, 168, \
138, 147,
188, 80, 142, 243, 230, 216, 96, 60, 1, 229, 92, 210, 222, 199, 42, \
35, 153, 212,
39, 164, 220, 91, 211, 81, 70, 248, 30, 118, 90, 158, 161, 237, \
37, 97, 235,
226, 9, 93, 111, 213, 149, 78, 85, 74, 185, 127, 20, 146, 25, 141, \
67, 214, 55,
253, 156, 246, 198, 247, 56, 108, 165, 238, 63, 17, 11, 228, 29, 174, \
252, 163, 4,
86, 113, 178, 50, 184, 166, 15, 48, 225, 76, 49, 71, 101, 189, 61, \
52, 150, 186,
244, 135, 209, 105, 152, 208, 251, 121, 88, 139, 159, 36, 58, 23, \
6, 223, 202,
8, 7, 45, 26, 177, 40, 217, 148, 182, 98, 27, 83, 245, 21, 233, \
154, 38, 22,
172, 110, 69, 125, 187, 232, 204, 34, 206, 157, 176, 236, 205, 104, \
62, 53, 227,
89, 59, 73, 196, 79, 175, 5, 41, 183, 24, 28, 116, 218, 10, 133, 115, \
242, 107,
221, 77, 180, 181, 250, 112, 240, 173, 14, 3, 137, 32, 47, 102, 239, \
33, 193, 99,
68, 143, 207, 162, 200, 241, 167

```

```

};
const unsigned short PERMUTATION_TABLE_3[256] PROGMEM = {165, 90, 122,
59, 6, 143, 37, 214, 88, 123, 42, 159, 92, 250, 35, 232, 163, 207, \
73, 28, 56,
240, 216, 89, 190, 104, 192, 191, 91, 121, 217, 113, 115, 145, 70, \
175, 108,
227, 223, 189, 114, 148, 105, 231, 208, 39, 212, 32, 63, 247, 137, 127, \
45, 38, 184,
220, 174, 116, 30, 120, 124, 183, 101, 54, 204, 179, 244, 58, 48, \
188, 66, 118,
22, 111, 138, 193, 177, 96, 151, 225, 107, 233, 221, 33, 150, 16, \
102, 83, 209,
21, 61, 194, 29, 26, 49, 51, 235, 126, 149, 44, 164, 206, 99, 147, \
15, 234, 41,
31, 55, 13, 43, 10, 248, 153, 185, 34, 198, 36, 222, 129, 4, 155, \
17, 93, 161,
210, 215, 81, 76, 142, 236, 152, 228, 224, 226, 182, 24, 252, 95, \
203, 110,
254, 162, 14, 136, 97, 196, 53, 156, 8, 9, 172, 154, 199, 100, 186, 201, \
25, 171, 87,
167, 134, 195, 67, 176, 246, 77, 74, 238, 230, 181, 65, 109, 103, \
78, 86, 168,
237, 139, 106, 140, 60, 125, 253, 112, 128, 251, 68, 46, 130, 241, \
0, 170, 52,
211, 5, 239, 173, 71, 1, 180, 85, 178, 98, 40, 135, 19, 94, 144, 79, \
157, 141, 23,
245, 75, 50, 11, 158, 3, 200, 69, 169, 72, 218, 243, 18, 82, 47, \
255, 166,
146, 62, 242, 131, 213, 80, 12, 20, 202, 187, 64, 119, 84, 133, 160, \
197, 205,
249, 7, 2, 229, 27, 57, 117, 132, 219
};

```

```

const unsigned short PERMUTATION_TABLE_4[256] PROGMEM = {133, 225,
105, 85, 122, 78, 19, 76, 104, 94, 137, 200, 157, 103, 33, 7, 64, 79, \
67, 58, 62,
251, 60, 72, 208, 119, 47, 5, 125, 117, 240, 3, 255, 205, 115, 156, \

```

```

179, 93, 213, 180, 176, 91, 245, 8, 235, 129, 127, 244, 199, 215, 13, 239, 227, \
82, 71, 218, 210, 191, 23, 206, 181, 153, 29, 234, 22, 102, 254, 49, 243, 17, 232, \
98, 21, 197, 20, 0, 144, 151, 143, 138, 154, 6, 184, 2, 145, 41, 230, 18, 43, \
63, 141, 53, 171, 126, 173, 211, 174, 135, 70, 222, 248, 214, 111, 161, 40, 123, \
188, 150, 55, 130, 31, 149, 108, 80, 96, 190, 229, 95, 132, 109, 224, 158, 45, \
128, 107, 168, 231, 221, 249, 121, 172, 75, 207, 131, 15, 193, 247, 165, 10, 238, \
120, 252, 202, 16, 90, 113, 112, 212, 81, 134, 35, 160, 194, 89, 42, 86, 101, \
196, 146, 142, 147, 30, 32, 223, 236, 250, 12, 99, 9, 39, 155, 66, 36, 167, 110, \
175, 139, 178, 182, 88, 185, 186, 148, 209, 24, 219, 140, 114, 198, 50, 203, 48, \
253, 162, 187, 106, 74, 54, 11, 152, 233, 124, 136, 195, 38, 92, 216, 183, 189, \
217, 226, 26, 169, 68, 87, 57, 246, 61, 34, 83, 46, 52, 77, 170, 228, 116, 204, \
1, 4, 242, 27, 56, 166, 37, 73, 164, 84, 118, 192, 44, 69, 159, 177, 65, 100, \
25, 220, 241, 237, 51, 97, 59, 28, 201, 163, 14,
};

```

```

    unsigned short permutation = 0;

    void processBlock(unsigned short *pData, unsigned short *pResult,
unsigned short *pKey)
    {
        unsigned short tmp1 = 0, tmp2 = 0, r1, r2;

        int i, k;

        for (k = 0; k < BLOCK_SIZE; k++)
        {
            pResult[k] ^= pData[k];
        }

        for (i = 0; i < 16; i++)
        {
            r1 = pResult[0];
            r2 = pResult[1];

            for (k = 0; k < 2; k++)
            {
                pResult[k] += pKey[(i * 2 + k) % 16];
            }

            // tmp1 = PERMUTATION_TABLE_1[pResult[0] >> 8];
            permutation = pgm_read_word(&(PERMUTATION_TABLE_1[(pResult[0] >>
8)]));
            // PRINT(permutation);
            tmp1 = permutation;

            // pResult[0] = (tmp1 << 8) | (PERMUTATION_TABLE_2[(pResult[0]) &
0xFF]);

```

```

        permutation = pgm_read_word(&(PERMUTATION_TABLE_2[(pResult[0]) &
0xFF]));
        pResult[0] = (tmp1 << 8) | permutation;

        //      tmp2 = PERMUTATION_TABLE_3[pResult[1] >> 8];
        permutation = pgm_read_word(&(PERMUTATION_TABLE_3[pResult[1] >>
8]));
        tmp2 = permutation;

        //      pResult[1] = (tmp2 << 8) | (PERMUTATION_TABLE_4[(pResult[1])
& 0xFF]);
        permutation = pgm_read_word(&(PERMUTATION_TABLE_4[(pResult[1]) &
0xFF]));
        pResult[1] = (tmp2 << 8) | permutation;

        tmp1 = ((pResult[0]) & 0xFFE0) >> 5;
        pResult[0] <<= 11;
        tmp2 = ((pResult[1]) & 0xFFE0) >> 5;
        pResult[1] <<= 11;
        pResult[0] ^= tmp2;
        pResult[1] ^= tmp1;

        pResult[0] ^= pResult[2];
        pResult[1] ^= pResult[3];
        pResult[2] = r1;
        pResult[3] = r2;
    }
    return;
}

```

```

void Decode(byte* CompressedStart, unsigned int CompressedSize,
unsigned short *pKey, unsigned short *imitoresult)
{
    heatshrink_decoder hsd;

    heatshrink_decoder_reset(&hsd);
    for (int i = 0; i < 8; i++)
    {
        imitoresult[i] = 0;
    }

    byte CompressedByte;

    byte DecompressedBuffer[DECOMPRESSED_BUFFER_SIZE];

    unsigned int offset = 0;
    unsigned int readed = 0;
    unsigned int input_size = 0;
    byte Status;
    unsigned int decompressed_size = 0;

    while (readed < CompressedSize)
    {
        input_size = 1;

        //Читаем по байту и кладем в декодер по байту. Медленно, но
удобно.
        while ((readed < CompressedSize) && (input_size != 0))
        {
            CompressedByte = pgm_read_word((void*)CompressedStart);

```

```

        if (heatshrink_decoder_sink(&hsd, &CompressedByte, 1,
&input_size) == HSDR_SINK_ERROR_NULL)
        {
            PRINT("\nError sink");
        }
        readed += input_size;
        CompressedStart++;
    }
    CompressedStart--; //последняя итерация лишняя, т.к. input_size =
0;

    // достаем из декодера сколько получится
    if (heatshrink_decoder_poll(&hsd, DecompressedBuffer + offset,
DECOMPRESSED_BUFFER_SIZE - offset, &input_size) == HSDR_POLL_ERROR_UNKNOWN)
    {
        PRINT("\nError poll");
    }
    decompressed_size += input_size;

    int full_blocks = (input_size + offset) / 8;
    // Рассчитываем имитовставку
    for (int i = 0; i < full_blocks; i++)
    {
        processBlock((unsigned short*)(DecompressedBuffer + (i * 8)),
imitoresult, pKey);
    }

    // Оставшиеся перегоняем в начало буфера и запоминаем offset
    offset = (input_size + offset) % 8;
    for (int i = 0; i < offset; i++)
    {
        DecompressedBuffer[i] = DecompressedBuffer[i + full_blocks * 8];
    }
}

// завершаем работу декодера (входные данные закончились)

// крутим пока еще есть данные на выходе
Status = heatshrink_decoder_finish(&hsd);

while (Status == HSDR_FINISH_MORE)
{
    // достаем из декодера сколько получится
    if (heatshrink_decoder_poll(&hsd, DecompressedBuffer + offset,
DECOMPRESSED_BUFFER_SIZE - offset, &input_size) == HSDR_POLL_ERROR_UNKNOWN)
    {
        PRINT("\nError poll");
    }
    decompressed_size += input_size;

    int full_blocks = (input_size + offset) / 8;

    // Рассчитываем имитовставку
    for (int i = 0; i < full_blocks; i++)
    {
        processBlock((unsigned short*)(DecompressedBuffer[i * 8])),

```

```

imitoresult, pKey);
    }

    // Оставшиеся данные копируем в начало буфера и запоминаем offset
    offset = (input_size + offset) % 8;
    for (int i = 0; i < offset; i++)
    {
        DecompressedBuffer[i] = DecompressedBuffer[i + full_blocks * 8];
    }
    Status = heatshrink_decoder_finish(&hsd);
}

if (Status == HSDR_FINISH_DONE)
{
    if (heatshrink_decoder_poll(&hsd, DecompressedBuffer + offset,
        DECOMPRESSED_BUFFER_SIZE - offset, &input_size) == HSDR_POLL_ERROR_UNKNOWN)
    {
        PRINT("\nError poll");
    }
    decompressed_size += input_size;

    int full_blocks = (input_size + offset) / 8;

    // Рассчитываем имитовставку
    for (int i = 0; i < full_blocks; i++)
    {
        processBlock((unsigned short*)&(DecompressedBuffer[i * 8])),
        imitoresult, pKey);
    }
    offset = (input_size + offset) % 8;
}
else
{
    PRINT("\nError finish");
}

// Если осталось количество байт не кратное 8, то ошибка. Размер
прошивки выровнен.
if (offset != 0)
{
    PRINT("\nError offset = ");
}

return;
}

void ReadMemoryBlock(unsigned short blockNumber, unsigned short*
pBuffer)
{
    uint16_t i;
    for (i = blockNumber * BLOCK_SIZE; i < ((blockNumber + 1) *
BLOCK_SIZE); i++)
    {
        pBuffer[i - blockNumber * BLOCK_SIZE] = pgm_read_word((void *)i);
    }
    return;
}

```

```

void setup() {
    // put your setup code here, to run once:
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
    // Open serial communications and wait for port to open:
    Serial.begin(115200);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port
    }
    Serial.println(F("Init complete!"));

    unsigned short Buffer[BLOCK_SIZE];
    unsigned short i;
    unsigned short key[KEY_SIZE] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
    unsigned short result[BLOCK_SIZE] = {0, 0, 0, 0};

    Serial.println(F("Enter key: "));
    Serial.setTimeout(0xffffffff);
    Serial.readBytesUntil(0, (byte*)key, sizeof(key));

    for (i = 0; i < KEY_SIZE; i++)
    {
        PRINT_HEX(key[i]);
        PRINT(" ");
    }

    Decode((byte*)compressed, sizeof(compressed), key, result);

    PRINT("ИММОБСТАКА: ");
    for (i = 0; i < BLOCK_SIZE; i++)
    {
        PRINT_HEX(result[i]);
        PRINT(" ");
    }
}

void loop() {
    // put your main code here, to run repeatedly:
    //PRINT_HEX(45);
    delay(1000);
    // if (Serial.available()) {
    //     Serial.write(Serial.read());
    // }
}

```

ПРИЛОЖЕНИЕ В

(обязательное)

Акт о внедрении

“УТВЕРЖДАЮ”

Первый заместитель директора

Филиала «Аргус» АО «ПНИЭИ»



С.А. Мещеринов

“3” — 25 — 2016 г.

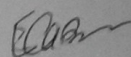
АКТ О ВНЕДРЕНИИ

результатов выпускной квалификационной работы

Кашубиной А.П.

Комиссия в составе **Саблина Е.А.** (председатель), **Кузьмина Д.С.** и **Легошина В.В.** установила, что представленные в выпускной квалификационной работе результаты исследований по доработке штатных алгоритмов криптографической проверки целостности использованы в научно-исследовательской работе по перспективам создания универсального средства защиты информации.

Председатель комиссии,
начальник НИЛ-2



Е.А. Саблин

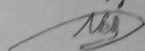
Члены комиссии:

инженер-программист 2 кат.

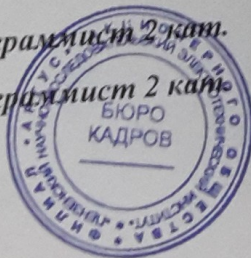


Д.С. Кузьмин

инженер-программист 2 кат.



В.В. Легошина



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

УДК 004.056.2

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Выпускающая кафедра:
Информационная безопасность систем
и технологий
Учебная группа 11ПИ1

УТВЕРЖДАЮ

Зав. кафедрой ИБСТ

к.т.н., доцент

С.Л. Зефиров

27 04 2016 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

студента группы 11ПИ1 Кашубиной Анны Павловны

Специальность 100503 – Информационная безопасность
автоматизированных систем

Тема дипломного проекта: Способ обеспечения криптографической
проверки целостности специального программного обеспечения
микроконтроллеров.

Руководитель ВКР

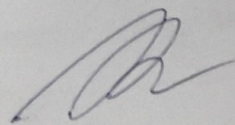
инженер-программист

филиала НИП «Аргус» АО «ПНИЭИ» Кузьмин Дмитрий Сергеевич

Иванов А.П.

Нормоконтролер

к.т.н., доцент



1 Цели и задачи ВКР

Цели работы:

- разработка способа обеспечения криптографической проверки целостности специального программного обеспечения (СПО) микроконтроллеров;
- программная реализация разработанного способа обеспечения криптографической проверки целостности СПО микроконтроллеров.

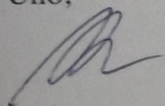
Разработанный способ предназначен для использования в перспективной аппаратуре криптографической защиты информации.

2 Тактико-технические требования к выполнению ВКР

Технические требования:

- должен быть проведен анализ способа криптографической проверки целостности СПО микроконтроллеров, применяемого в настоящее время, с целью поиска в нем уязвимостей;
- должен быть разработан способ криптографической проверки целостности, защищенный от обнаруженных уязвимостей;
- способ должен быть реализован программно;
- в разработанном способе могут применяться организационные и технические меры (в том числе с возможным привлечением оператора проверяемого изделия);
- программная реализация должна быть написана на языке C/C++;
- платформой для проверки программной реализации должна быть

Arduino Uno;



– используемые криптографические примитивы должны быть защищены по вычислениям от перспективных вычислительных возможностей злоумышленника.

В ВКР должны быть рассмотрены следующие вопросы:

– разработка модели угроз и нарушителя компрометации СПО микроконтроллеров;

– анализ способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время, с целью поиска в нем уязвимостей;

– анализ и выбор криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности СПО микроконтроллеров;

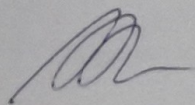
– разработка и программная реализация способа проверки целостности СПО микроконтроллеров, защищенного от обнаруженных уязвимостей.

3 Требования к разрабатываемой документации

Отчет по ВКР выполняется в соответствии с требованиями ГОСТ 7.32.

Перечень и содержание графической части ВКР:

– способ криптографической проверки целостности СПО микроконтроллеров, используемый в настоящее время - 1 л., ф. А1 (плакат);



– схема компрометации способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время - 1 л., ф. А1 (плакат);

– сравнительная таблица криптографических примитивов, которые могут быть использованы для реализации криптографической проверки целостности СПО микроконтроллеров - 1л., ф. А1 (плакат);

– описание выбранного криптографического примитива для реализации криптографической проверки целостности СПО микроконтроллеров - 1л., ф. А1 (плакат);

– модель угроз и нарушителя компрометации СПО
микроконтроллеров- 1 л., ф. А1 (плакат);

– способ криптографической проверки целостности СПО микроконтроллеров с устраненными уязвимостями – 1л., ф. А1 (плакат).

Электронная копия отчета и графической части ВКР должна быть представлена на магнитном или оптическом носителе.

4 Сроки выполнения ВКР

Тема утверждена приказом ректора ПГУ № 491а/о от «25» апреля
2016 г.

Дата выдачи задания 4 апреля 2016 г.

Время дипломного проектирования с 18 апреля 2016 г. по 10 июля

Задание к исполнению принял 15 апреля 2016 г.

Исполнитель ВКР _____ Кашубина А. П.



Ректору Пензенского
государственного университета
А. Д. Гулякову
студента 5 курса ФПИТЭ
очной формы обучения
Кашубиной Анны Павловны

заявление.

Я, Кашубина Анна Павловна, студент 5 курса очной формы обучения специальности 10.05.03 – «Информационная безопасность автоматизированных систем», прошу провести проверку с использованием системы «Антиплагиат» дипломного проекта на тему «Способ обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров», выполненной мной самостоятельно, на содержание элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

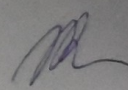
Я ознакомлена с действующим Стандартом университета «Выпускная квалификационная работа обучающихся по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры», согласно которому обнаружение плагиата является основанием для недопуска ВКР к защите и отчисления из университета.

 /А.П. Кашубина/

01.06.2016

ВКР представлена на проверку 01.06.2016.

Руководитель ВКР  /Д.С. Кузьмин/



Протокол

проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Способ обеспечения криптографической проверки целостности специального программного обеспечения микроконтроллеров» студента 5 курса А. П. Кашубиной специальности 10.05.03 – «Информационная безопасность автоматизированных систем».

Руководитель ВКР инженер-программист филиала НИП «Аргус» АО «ПНИЭИ» Кузьмин Дмитрий Сергеевич.

Отчет о проверке № 1

ФИО: Иванов Алексей Петрович
дата выгрузки: 02.06.2016 22:31:56
пользователь: ap_ivanov@mail.ru / ID: 1267386
отчет предоставлен сервисом «Антиплагиат»
на сайте <http://www.antiplagiat.ru>

Информация о документе

№ документа: 29
Имя исходного файла: ВКР_Кашубина_АП_11ПН1.pdf
Размер текста: 473 кБ
Тип документа: Не указано
Символов в тексте: 98497
Слов в тексте: 10420
Число предложений: 487

Информация об отчете

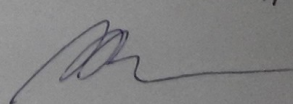
Дата: Отчет от 02.06.2016 22:31:59 - Последний готовый отчет
Комментарий: не указано
Оценка оригинальности: 83.85%
Заимствования: 16.15%
Цитирование: 0%



Оригинальность: 83.85%
Заимствования: 16.15%
Цитирование: 0%

08.06.2016 г.

Секретарь ГЭК

 Д. С. Кузьмин
 А. П. Иванов

Отзыв

на выпускную квалификационную работу Кашубиной А. П. по теме:
«Способ обеспечения криптографической проверки целостности специального
программного обеспечения микроконтроллеров»

В дипломном проекте рассматривается актуальная проблема информационной безопасности: защита целостности СПО микроконтроллеров, применяемых в перспективной аппаратуре криптографической защиты информации.

Содержание проекта полностью соответствует цели и поставленным задачам. Дипломный проект состоит из введения, пяти разделов, заключения, списка использованных источников и трех приложений.

Во введении содержится обоснование актуальности проекта, его цели и основные этапы.

В первом разделе описана модель угроз целостности и нарушителя компрометации СПО микроконтроллеров. Описаны основные угрозы нарушения целостности СПО микроконтроллеров и возможности атакующего.

Во втором разделе рассматривается способ криптографической проверки целостности СПО микроконтроллеров, используемый в настоящее время. Предложена реализация данного способа на языке программирования С для платформы Arduino Uno. По результатам анализа способа криптографической проверки целостности СПО микроконтроллеров были найдены уязвимости. Через одну из уязвимостей была произведена компрометация способа криптографической проверки целостности СПО микроконтроллеров, используемого в настоящее время.

В третьем разделе рассмотрены криптографические примитивы, которые могут быть использованы для реализации криптографической проверки целостности СПО микроконтроллеров. Сделан обоснованный выбор криптографических примитивов для достижения поставленной задачи.

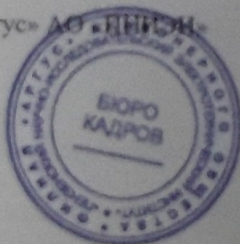
В четвертом разделе описаны общие принципы построения систем проверки целостности СПО микроконтроллеров. А также описаны пять предлагаемых способов обеспечения целостности СПО микроконтроллеров, защищающие от найденных уязвимостей. Отмечены их достоинства и недостатки.

В пятом разделе описываются результаты моделирования одного из предложенных способов защиты.

Проект оформлена в соответствии с ГОСТ 7.32-2001, содержит таблицы и рисунки, повышающие наглядность текстового материала. Дипломный проект соответствует предъявляемым требованиям, заслуживает высокой оценки и может быть допущен к защите.

инженер-программист

филиала НИИ «Аргус» АО «НИИЭИ»



Куз
Кузьмин Дмитрий Сергеевич

15821106 *DR* /*Кузмин Д.С.*/

Рецензия

на выпускную квалификационную работу Кашубиной А. П. по теме:
«Способ обеспечения криптографической проверки целостности специального
программного обеспечения микроконтроллеров»

Содержание дипломного проекта полностью соответствует выбранной теме. Все пункты технического задания выполнены в полном объеме. Дипломный проект может быть допущен к защите.

Проблема защиты целостности СПО микроконтроллеров является актуальной проблемой современной техники, а результаты проекта могут быть использованы при построении новых систем защиты информации для её решения.

Полученные результаты хорошо обоснованы, приведены примеры реализации заявленных идей. В частности, не только предложена идея компрометации СПО, но и предоставлен конкретный пример. Так же приведён пример реализации разработанного способа защиты.

Проект оформлен в соответствии с ГОСТ 7.32-2001, содержит таблицы и рисунки, повышающие наглядность текстового материала.

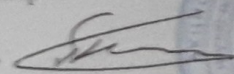
Необходимо отметить, что проведенная работа требует больших трудозатрат и хороших тематических знаний.

В качестве недочета стоит отметить, что предложенные механизмы защиты недостаточно описаны с практической стороны их реализации, однако данные вопросы выходят за рамки дипломного проекта.

Дипломный проект соответствует предъявляемым требованиям и заслуживает отметки «отлично».

Ведущий инженер

АО «ПНИЭИ»



Ведущий инженер Сергей Викторович

Э Белов / Мещеряков А. П.