

Введение

SHA iButton® фирмы Dallas (DS1963S) — это интеллектуальный маркер, который обладает множеством функций обеспечения безопасности и поддерживает множество услуг. В настоящем документе представлен обзор имеющихся приложений цифровой идентификации и транзакций (пересылок данных) с помощью приборов SHA iButton. Для подробной иллюстрации этапов, требуемых для установки данных обслуживания и проведения электронных транзакций, используется пример реализации обслуживания. В документе рассматривается использование основных вызовов API (Application Program Interface — программного интерфейса приложения), а также приводится аннотированный листинг команд и потоков данных для реализации каждого API.

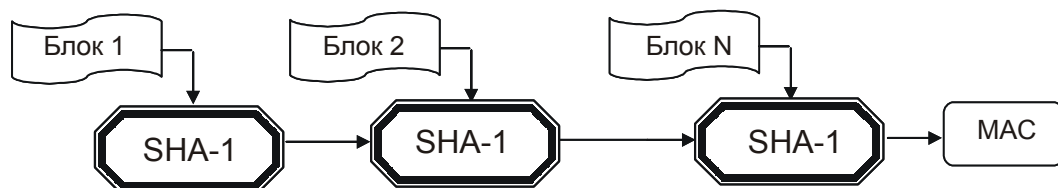
SHA iButtons подходят для применения в различных областях:

- транспортировка
- электронные дверные замки
- контроль доступа в здания
- управление сетевым доступом и доступом к компьютерам
- телефоны-автоматы
- счетчики парковки
- счетчики коммунальных услуг с предоплатой
- торговые автоматы
- авторизация программного обеспечения

Что такое SHA?

Алгоритм SHA (Secured Hash Algorithm) за многие годы разработки и совершенствования специалистами по шифрованию был признан одним из самых безопасных и широко применяемых алгоритмов хэширования. В приборе DS1963S фирмы Dallas Semiconductor (SHA iButton) реализован алгоритм SHA-1, который соответствует требованиям федерального стандарта по обработке информации FIPS 180-1 (Federal Information Processing Standard). В упрощенном представлении функция хэширования (хэш-функция) представляет собой процесс преобразования входной строки (называемой прообразом, или сообщением) в выходную более короткую строку фиксированной длины (называемую значением хэш-функции, профилем или дайджестом сообщения, или кодом аутентификации сообщения).

Алгоритм SHA очень эффективен для обнаружения ошибок или изменений во входной строке, поскольку проверка производится на более коротком сжатом профиле сообщения. Односторонняя функция хэширования обладает характеристиками, которые позволяют легко генерировать профиль сообщения, но при этом по заданному профилю очень трудно вычислить (восстановить) входное сообщение. Безопасность односторонней функции хэширования заключается в мощной однонаправленной операции генерирования профиля. Секретность может обеспечиваться встраиванием секретного ключа (секрета) во входную строку так, чтобы никто не смог, не зная секрета, сгенерировать соответствующий профиль. Алгоритм SHA берет входные потоки блоками по 512 бит (64 байт) и создает выходной код длиной 20 бит, называемый кодом аутентификации сообщения (Message Authentication Code — MAC), или профилем сообщения. Если входная строка имеет длину, отличающуюся от 512 бит, то алгоритм дополняет ее до ближайшего числа, кратного 512 бит. На рисунке, приведенном ниже, показано формирование цепочки из множества входных блоков (по 512 бит) для генерации выходного кода MAC.

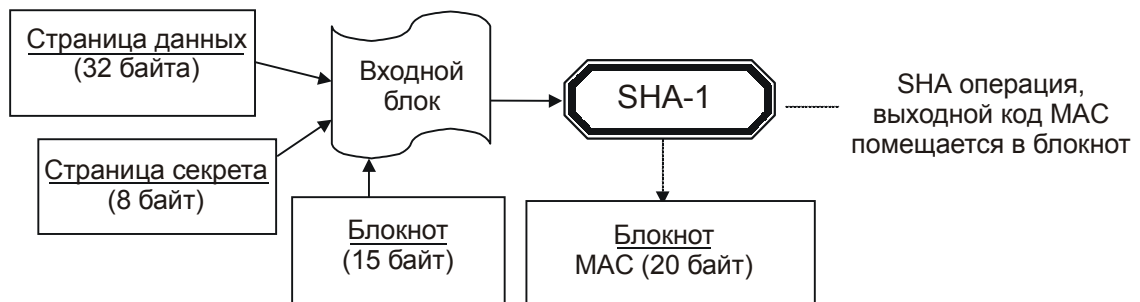


iButton является зарегистрированной торговой маркой Dallas Semiconductor.

Применение SHA в SHA iBUTTON фирмы Dallas (DS1963S)

Прибор SHA iButton (DS1963S) фирмы Dallas реализует алгоритм SHA-1 в специальной быстродействующей схеме, которая выполняет одно вычисление SHA-1 менее чем за 1 мс. Входные данные для вычислительного устройства (процессора) SHA-1 ограничиваются одним блоком. По структуре блок входных данных для процессора SHA-1 состоит из данных пользователя, хранящихся на странице данных для чтения/записи (32 байт), в блокнотной памяти (15 байт) и секретного ключа, или секрета (8 байт), который был предустановлен поставщиком услуг (провайдером) на странице секретов, не имеющей доступа для чтения (см. Рис. 1). Результат вычисления (MAC) размещается в блокнотной памяти (Scratchpad) для проведения последующих операций. В зависимости от конкретной вызванной SHA-операции, этот выходной код MAC может быть, а может и не быть спрятан в блокнотной памяти от внешнего доступа.

Рис. 1. Генерация кода MAC в SHA iBUTTON



Характеристики SHA iBUTTON

SHA iButton — это прочный 4-Кбитный носитель данных с возможностью чтения/записи, доступ к которым может быть легко осуществлен с помощью минимального аппаратного обеспечения. Встроенный 512-битный процессор SHA-1 может быть активизирован для вычисления 160-битных кодов аутентификации сообщения (MAC) на основе информации, хранящейся в приборе. Данные передаются последовательно при помощи протокола 1-Wire[®], для работы которого требуется лишь один провод данных (сигнальный) и общий провод (земля). Один прибор SHA iButton может обслуживать до восьми независимых приложений, каждое со своим собственным секретом и счетчиком страниц.

SHA iButton может функционировать и как сопроцессор, который хранит секретные ключи системы и помогает местному хосту транзакции (пересылки данных) вычислять коды MAC, необходимые для аутентификации маркера пользователя и проверки достоверности данных приложения. SHA iButton, подобно другим приборам семейства iButton, имеет дополнительную область памяти, называемую блокнотной памятью (или блокнотом), которая действует как буфер при записи данных в память прибора. Блокнот SHA iButton используется также для передачи сегментов данных в процессор SHA-1 или для приема/сравнения кодов аутентификации сообщений. При записи в iButton данные сначала записываются в блокнот, откуда они могут быть считаны и проверены на наличие ошибок передачи данных. После того, как данные были проверены, команда копирования блокнота передает данные в память по заданному адресу. Это процесс гарантирует целостность данных в среде, где не обеспечивается надежного электрического контакта.

Прибор SHA iButton DS1963S имеет следующие основные характеристики:

- 4096-битная энергонезависимая память с возможностью чтения/записи, организованная в виде 16 страниц по 256 бит каждая
- Восемь из 16 страниц памяти имеют индивидуальные 64-битные секреты и предназначенные только для чтения 32-битные не переполняющиеся счетчики циклов записи страницы
- Встроенный 512-битный процессор SHA-1
- В качестве маркера пользователя, прибор может поддерживать до восьми независимых услуг
- Может работать как сопроцессор для хранения секретов системы и вычисления кодов MAC, необходимых для аутентификации маркеров пользователя и проверки достоверности данных обслуживания

1-Wire является зарегистрированной торговой маркой Dallas Semiconductor.

Организация памяти SHA jBUTTON

Память данных с обычным доступом чтения/записи (16 страниц, 32 байт на страницу)

Номер страницы	Диапазон адресов (ТА1...ТА2)	Номер секрета	Номер счетчика	Инкремент счетчика
0	0000h...001Fh	0	0	Нет
1	0020h...003Fh	1	1	Нет
2	0040h...005Fh	2	2	Нет
3	0060h...007Fh	3	3	Нет
4	0080h...009Fh	4	4	Нет
5	00A0h...00BFh	5	5	Нет
6	00C0h...00DFh	6	6	Нет
7	00E0h...00FFh	7	7	Нет
8	0100h...011Fh	0	0	С записью
9	0120h...013Fh	1	1	С записью
10	0140h...015Fh	2	2	С записью
11	0160h...017Fh	3	3	С записью
12	0180h...019Fh	4	4	С записью
13	01A0h...01BFh	5	5	С записью
14	01C0h...01DFh	6	6	С записью
15	01E0h...01FFh	7	7	С записью

Память секретов без доступа для чтения (восемь 64-битных секретов)

Номер страницы	Диапазон адресов (ТА1...ТА2)	Описание
16	0200h...0207h	Секрет 0
	0208h...020Fh	Секрет 1
	0210h...0217h	Секрет 2
	0218h...021Fh	Секрет 3
17	0220h...0227h	Секрет 4
	0228h...022Fh	Секрет 5
	0230h...0237h	Секрет 6
	0238h...023Fh	Секрет 7

Заметим, что только запись на страницы с 8 по 15 будет увеличивать значение счетчиков страниц. При выполнении определенных операций каждый счетчик и секрет совместно используются соответствующей парой страниц данных. Например, страницы 0 и 8 совместно используют счетчик с номером 0 и секрет с номером 0. Запись на страницу 8 увеличит на 1 содержимое счетчика с номером 0, но запись на страницу 0 не приведет к какому-либо воздействию на счетчик с номером 0. Выполнение команды чтения аутентифицированной страницы (Read Authenticated Page) для страниц 0 или 8 приведет к возвращению счетчика 0 и использованию секрета 0 для вычисления MAC-кода.

Память счетчиков с доступом только для чтения (девять 32-битных счетчиков)

Номер страницы	Диапазон адресов (ТА1...ТА2)	Описание
19	0260h...0263h	Счетчик циклов записи страницы 8
	0264h...0267h	Счетчик циклов записи страницы 9
	0268h...026Bh	Счетчик циклов записи страницы 10
	026Ch...026Fh	Счетчик циклов записи страницы 11
	0270h...0273h	Счетчик циклов записи страницы 12
	0274h...0277h	Счетчик циклов записи страницы 13
	0278h...027Bh	Счетчик циклов записи страницы 14
	027Ch...027Fh	Счетчик циклов записи страницы 15
20	0280h...0283h	Счетчик циклов записи секрета 0
	0284h...0287h	Счетчик циклов записи секрета 1
	0288h...028Bh	Счетчик циклов записи секрета 2
	028Ch...028Fh	Счетчик циклов записи секрета 3
	0290h...0293h	Счетчик циклов записи секрета 4
	0294h...0297h	Счетчик циклов записи секрета 5
	0298h...029Bh	Счетчик циклов записи секрета 6
	029Ch...029Fh	Счетчик циклов записи секрета 7
21	02A0h...02A3h	Счетчик PRNG (исполнительный счетчик процессора SHA)

Применение SHA iBUTTON для электронных платежей

Прибор SHA iButton предназначен для работы в качестве маркера пользователя, а также сопроцессора. Сопроцессор должен выполнять все вычисления кода MAC, требуемые для аутентификации прибора и проверки достоверности данных обслуживания во время сеанса транзакции, исключая таким образом необходимость вычисления кода SHA и значительно сокращая цикл обработки. Одним из дополнительных преимуществ использования SHA iButton в качестве сопроцессора является то, что секреты системы хранятся внутри таблетки iButton, при этом секреты не могут быть считаны извне. В настоящем документе мы предполагаем, что сопроцессор SHA iButton используется в местном хосте для выполнения необходимой аутентификации прибора и проверки достоверности данных обслуживания. Ниже приводятся определения некоторых терминов, используемых в этом документе:

местный хост — аппаратный блок, состоящий из необходимых компонентов, способный выполнять электронные транзакции с пользовательскими SHA семейства iButton (или другими маркерами электронных кошельков). Функционально местный хост может иметь три основных компонента: блок управления транзакцией (обычно микропроцессор); интерфейс пользователя, например, дисплей и считывающее устройство для маркера; и сопроцессор.

номер адреса (AN), идентификационное ПЗУ, регистрационный номер, серийный номер — используются по очереди, относятся к 64-битному номеру, записанному при помощи лазера в процессе производства, что гарантирует абсолютную уникальность DS1963S или любых других приборов шины 1-Wire.

блок управления транзакцией (TCU) — компонент (обычно микропроцессор), который обеспечивает обмен данными между сопроцессором и маркерами пользователя для выполнения аутентификации прибора и проверки достоверности данных обслуживания.

секрет аутентификации системы, секрет аутентификации мастера — используются по очереди, относятся к секретному ключу, установленному в сопроцессор с целью аутентификации приборов пользователя.

секрет аутентификации прибора — секретный ключ, установленный в iButton пользователя так, чтобы местный хост мог проверить, принадлежит ли прибор данной системе. Секрет аутентификации прибора сделан уникальным для конкретного прибора пользователя посредством привязки секрета аутентификации системы к номеру адреса прибора пользователя.

секрет подписи системы, секрет подписи мастера — используются по очереди, относятся к секретному ключу, установленному в сопроцессор с целью подписи и проверки достоверности данных обслуживания.

сигнатура подписи — код аутентификации сообщения (MAC), вычисленный из данных обслуживания, секрета подписи системы и других технических данных прибора. Сигнатура подписи обычно встраивается в страницу данных обслуживания так, чтобы местный хост мог быстро проверить эти данные.

данные обслуживания, данные пользователя, данные приложения, данные счета, данные транзакции — данные, которые полностью представляют услугу, например, для управления доступом или при электронных платежах.

прибор пользователя, маркер пользователя — цифровой носитель данных обслуживания для цифровой аутентификации или электронных платежей. В настоящем документе всегда, когда упоминается прибор пользователя или маркер пользователя, подразумевается SHA iButton DS1963S.

сoproцессор — в контексте настоящего документа, сопроцессор — это вычислительное устройство, способное выполнять все необходимые вычисления кода MAC для проведения транзакции.

Установка обслуживания

Чтобы использовать SHA iButton для аутентификации прибора и проверки достоверности данных, соответствующие данные и секреты должны быть установлены и в сопроцессоре, и в iButton пользователя. Этот процесс часто называют инициализацией обслуживания. В сопроцессор устанавливается два секрета: секрет аутентификации системы и секрет подписи системы. Секрет аутентификации системы используется, чтобы создать для каждого прибора пользователя уникальный секрет аутентификации и проверить, принадлежит ли прибор пользователя данной системе. Секрет подписи системы используется для создания (генерации) сигнатуры подписи и проверки, являются ли данные обслуживания достоверными. В прибор iButton пользователя необходимо установить только один секрет — уникальный секрет аутентификации прибора. Важно отметить, что секрет аутентификации прибора пользователя (SHA iButton) делается уникальным и отличным от секрета аутентификации системы за счет использования номера адреса прибора пользователя в качестве составной части входных данных для вычисления секрета прибора.

Существует два типа данных обслуживания: статические и динамические. Статические данные обслуживания никогда не изменяются во время транзакции, тогда как динамические данные всегда обновляются при каждой транзакции. Например, данные обслуживания для приложений управления доступом в офис или гостиницу, содержащие информацию о разрешенном времени доступа, нуждаются в проверке их достоверности, но не изменяются процессором замка (местным хостом). С другой стороны, торговый автомат или счетчик времени парковки требует списания (дебетования) соответствующей суммы со счета и новой цифровой подписи данных уменьшившегося баланса таким образом, чтобы их достоверность снова можно было проверить другими местными хостами системы. Защита данных обслуживания может и не потребоваться, если системе необходимо только знать, принадлежит ли прибор пользователя к данной системе (зарегистрированный пользователь), и если она не принимает решения о транзакции, основываясь на содержимом данных обслуживания.

В общем случае, ввод в действие системы электронного платежа включает в себя следующие этапы.

Для установки обслуживания:

- (1) Установка секрета аутентификации системы в сопроцессор SHA iButton.
- (2) Установка секрета подписи системы в сопроцессор SHA iButton (если необходима защита данных).

Для каждого прибора пользователя:

- (3) Установка секрета аутентификации прибора пользователя в SHA iButton пользователя.
- (4) Установка подписанных данных обслуживания в SHA iButton пользователя (если необходима защита данных).

Для проведения транзакции:

- (5) Выполнение аутентификации прибора пользователя с помощью сопроцессора SHA iButton.
- (6) Выполнение проверки достоверности данных пользователя с помощью сопроцессора SHA iButton (если необходима защита данных).

Аутентификация прибора

Для аутентификации SHA iButton пользователя, блок управления транзакцией (TCU) должен «попросить» сопроцессор вычислить произвольный (случайный) запрос¹ и послать его на iButton пользователя, чтобы тот вычислил ответный код MAC с помощью команды чтения аутентифицированной страницы (Read Authenticated Page). Для вычисления ответного MAC-кода прибор iButton пользователя использует запрос, данные обслуживания, свой собственный номер адреса и секрет аутентификации. Затем ответный код MAC считывается местным хостом для сравнения с результатом его собственного вычисления, сделанного позже. Для проверки ответного MAC-кода, TCU должен запросить сопроцессор сначала вновь вычислить уникальный секрет аутентификации прибора iButton пользователя при помощи номера адреса прибора пользователя и секрета аутентификации системы. Затем TCU должен запросить сопроцессор вычислить код MAC, используя вновь вычисленный секрет аутентификации прибора и код запроса, который был послан на iButton пользователя.

Затем этот вычисленный сопроцессором код MAC сравнивается с ответным кодом MAC, считанным из iButton пользователя, для определения, действителен ли прибор пользователя. Этот двухступенчатый процесс необходим по той причине, что местный хост не может считать секрет прибора пользователя, а секрет аутентификации прибора пользователя отличается от секрета аутентификации системы. Заметим, что для аутентификации прибора требуется только, чтобы прибор пользователя имел верный секрет аутентификации, содержимое данных обслуживания при этом значения не имеет.

Защита данных обслуживания

Защита данных обслуживания обеспечивается встраиванием сгенерированного системой кода MAC (сигнатуры подписи) в данные обслуживания. Сигнатура подписи генерируется при помощи секрета подписи системы, данных обслуживания, номера страницы данных обслуживания и значения счетчика страниц, а также номера адреса прибора пользователя. Используя сигнатуру подписи, можно обнаружить любые несанкционированные изменения и не допустить воспроизведения данных обслуживания. Для проверки достоверности данных обслуживания, TCU должен запросить сопроцессор вновь вычислить код MAC (сигнатуру подписи) из секрета подписи системы и всех элементов данных обслуживания и данных прибора пользователя, полученных во время обслуживания. Затем этот код MAC сравнивается со встроенной сигнатурой для определения, являются ли данные достоверными. Для обслуживания динамических данных, после обновления данных обслуживания вычисляется новая сигнатура подписи, чтобы отразить изменение данных и новое значение счетчика страниц. Затем эта новая сигнатура встраивается в данные обслуживания и сохраняется в приборе пользователя.

¹ Этот запрос является случайным в том смысле, что он генерируется при помощи счетчика процессора SHA, который увеличивает свое значение каждый раз, когда используется процессор SHA, и очень маловероятно, что iButton пользователя получит один и тот же запрос дважды. Более подробную информацию по генерированию запросов и секретов см. в документе AN152.

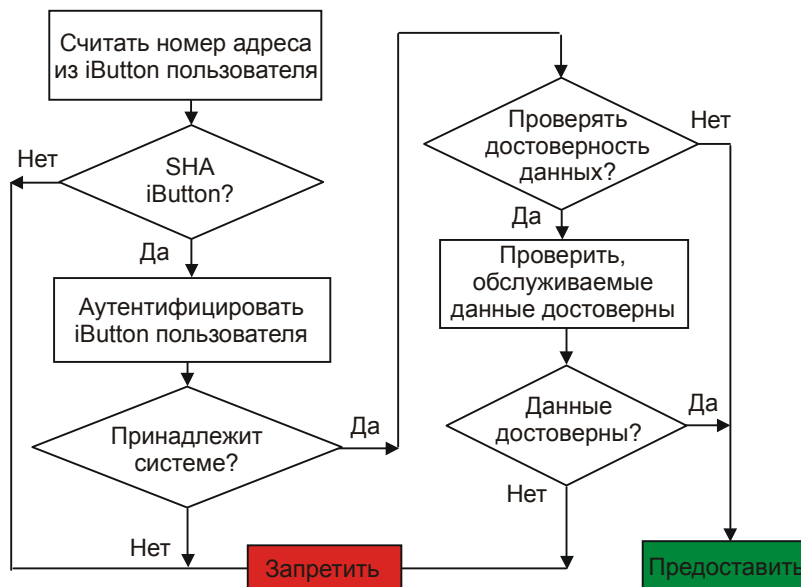
Обслуживание статических данных

При обслуживании статических данных требуется только аутентификация прибора и проверка достоверности данных обслуживания для того, чтобы принять решение об обслуживании. Аутентификация прибора представляет собой процесс, при котором проверяется, принадлежит ли прибор пользователю данной системе. Аутентификация прибора может быть единственной процедурой, которая требуется в определенных приложениях контроля доступа. Есть различные способы выполнения аутентификации прибора: (а) чтение номера адреса (AN) прибора и его поиск в базе данных, чтобы узнать, принадлежит ли прибор данной системе; (б) выполнение процедуры запроса и ответа (отклика) для проверки, содержит ли прибор достоверный секретный код; (с) другие методы. В SHA iButtons для аутентификации прибора используется второй метод. При этом методе запроса и ответа местный хост требует, чтобы прибор пользователя вычислил ответный код MAC, основываясь на скрытом секрете аутентификации прибора, данных памяти, связанных с этим секретом, и коде запроса, который передает местный хост. Прибор пользователя никогда не раскрывает свой секрет внешнему миру. Этот механизм аутентификации повышает безопасность системы, что особенно выигрышно, когда канал связи между местным хостом и прибором пользователя не является защищенным. Более того, местный хост будет выдавать различный запрос каждый раз, когда прибор пользователя будет посылать запрос на обслуживание, так чтобы ответный код MAC каждый раз отличался, делая, таким образом, перехват передаваемых данных бесполезным.

Типичные этапы для выполнения обслуживания статических данных (см. Рис. 2):

- (1) Считать номер адреса (AN) из прибора iButton пользователя.
- (2) Если это не SHA iButton, то перейти к другому процессу принятия решения.
- (3) Аутентифицировать iButton пользователя с помощью последовательности запрос—ответ; выйти, если iButton пользователя не принадлежит данной системе.
- (4) Нужно ли проверять достоверность данных обслуживания? Если нет, то предоставить обслуживание.
- (5) Проверить, являются ли данные обслуживания достоверными; если нет, то выйти.
- (6) Предоставить обслуживание.

Рис. 2. Дерево принятия решения об обслуживании статических данных

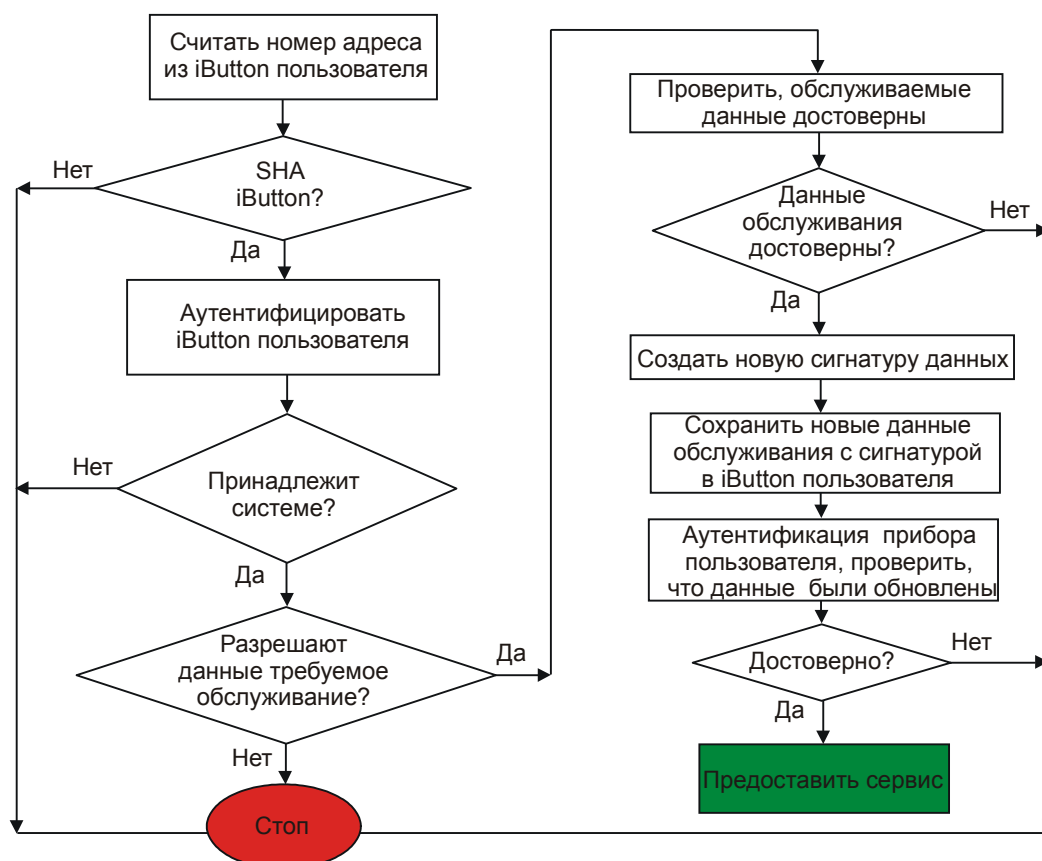


Обслуживание динамических данных

В дополнение к процессу аутентификации и проверки достоверности данных, которые выполняются для проведения обслуживания статических данных, обслуживание с использованием динамических данных требует дополнительных этапов для изменения данных обслуживания, создания новой сигнатуры и сохранения вновь подписанных данных в iButton пользователя. Ниже приводятся типичные этапы:

- (1) Считать номер адреса (AN) из прибора iButton пользователя.
- (2) Если это не SHA iButton, то перейти к другому процессу принятия решения.
- (3) Аутентифицировать SHA iButton пользователя с помощью последовательности запрос—ответ; выйти, если SHA iButton пользователя не принадлежит данной системе.
- (4) Разрешают ли данные обслуживания требуемое обслуживание; если нет, то выйти.
- (5) Выполнить проверку достоверности данных обслуживания; выйти, если они недостоверны.
- (6) Изменить данные обслуживания и создать новую сигнатуру.
- (7) Сохранить новые данные обслуживания с сигнатурой в iButton пользователя.
- (8) Снова выполнить аутентификацию прибора пользователя, чтобы убедиться, что данные обслуживания в приборе пользователя были обновлены.
- (9) Предоставить обслуживание.

Рис. 3. Дерево принятия решения об обслуживании динамических данных



Хостинг множественного обслуживания

Файловая структура 1-Wire (OWFS²) может использоваться для упрощения сосуществования множества услуг в одном приборе iButton пользователя, при этом данные для каждой услуги хранятся в ее собственном файле обслуживания. Для эффективного доступа множество файлов организовано в таблицу элементов директории. Местный хост считывает элемент файла для обслуживания в таблице директории, чтобы запросить фактическое расположение страницы и количество страниц, используемых для этой услуги. Фактические данные обслуживания считываются из файла обслуживания.

В сопроцессоре и других приборах iButton могут также храниться специальные файлы для идентификации этих приборов в качестве специальных. Например, мы создаем файл под названием COPR.0 в SHA iButton, чтобы идентифицировать данный прибор как сопроцессор, и хранить в этом файле все данные, необходимые TCU для выполнения транзакции. В приборе iButton можно создать еще один файл под названием COLL.102 для того, чтобы идентифицировать этот прибор как мобильное устройство сбора данных (коллектор), имеющее право извлекать из местного хоста данные, которые потом сбрасываются в центральную базу данных. При запуске блока управления транзакцией (TCU), он считывает информацию из сопроцессора iButton, чтобы подготовиться к проведению транзакции. Когда TCU обнаруживает таблетку iButton сбора данных, он выполняет необходимую аутентификацию и выдает запрашиваемые данные.

Файл сопроцессора

Прибор SHA iButton DS1963S может быть сконфигурирован для работы в качестве сопроцессора путем установки в него соответствующих секретов и данных системы. Для обслуживания статических данных без проверки достоверности данных в сопроцессор устанавливается только секрет аутентификации системы. Для обслуживания динамических или статических данных, требующих проверки достоверности, в сопроцессор должны быть установлены как секрет аутентификации системы, так и секрет подписи системы. Сопроцессор выполняет также все необходимые вычисления кода MAC для аутентификации iButton пользователя и проверки достоверности данных обслуживания. В нашем примере реализации сопроцессора, в SHA iButton создается файл (COPR.0) для идентификации данного прибора в качестве сопроцессора. Файл COPR.0 содержит все данные, необходимые блоку управления транзакцией для выполнения транзакции.

Таблица 1. Структура данных файла сопроцессора³

Структура файла сопроцессора COPR.0			
	Число байтов	Данные примера	Примечания
Имя файла данных обслуживания	5	DLSM.102	Сохраняется только базовое имя (DLSM) и расширение, а не разделитель (.). Байт расширения сохраняется шестнадцатиричном формате (66h = 102 d)
Номер страницы подписи	1	8	Разрешенные варианты выбора — страницы 0 или 8. Страница 0 используется для таблицы элементов файла
Номер аутентифицированной страницы	1	7	Начинается с достаточно большого номера страницы, чтобы предоставить достаточно количество страниц для хранения этого файла
Номер страницы рабочей области	1	9	Необходим сопроцессору для извлечения уникального секрета аутентификации прибора пользователя во время транзакции

² OWFS (1-Wire File Structure) — файловая структура 1-Wire, которая определяет структуру директории для данных, постоянно находящихся в приборах 1-Wire, включая приборы iButton. Она позволяет иметь такой же произвольный доступ к файлам, имеющим имена, как и в других файловых системах. Определения и правила OWFS являются достаточными для хранения множества файлов во вложенных директориях, используя приборы емкостью до 16 Мбайт. Эти приборы могут быть организованы как 2...65535 страницы по 32...256 байт. Более подробную информацию о структуре и поддержке реализации OWFS см. в документе «Application Note 114» фирмы Dallas Semiconductor/Maxim.

³ В примере реализации обслуживания все байты данных записываются, начиная с младших байтов.

Структура файла сопроцессора COPR.0			
Номер версии	1	1	Номер версии используется поставщиком услуг для отслеживания изменений конфигурации обслуживания
Код данных установки	4	04 0E 00 63 h	Код установки хранится в формате M D YY. YY = числу лет с 1900 года. Например, 4/14/1999 = 04 0E 00 63 h
Данные привязки секрета аутентификации прибора	39	39 байт FFh	Используется поставщиком услуг для привязки секрета аутентификации к SHA iButton пользователя. Первые 32 байт этого блока записываются на страницу данных, а оставшиеся 7 байт записываются в блокнотную память при вычислении секрета аутентификации прибора. Фактически, блокнотная память содержит 15 байт, которые являются входными данными для процессора SHA. Другие 8 байт — это номер страницы (1 байт) и номер адреса прибора без байта CRC
Код подписи услуги	3	3 байт 00h	Эти данные являются частью 15-ти байт, записанных в блокнотную память для создания сигнатуры данных обслуживания. Другие 12 байт — это номер страницы (1 байт), счетчик страниц (4 байт) и номер адреса прибора без CRC (7 байт)
<i>Длина имени провайдера (svcNameLen)</i>	1	20	Длина имени поставщика услуг (провайдера), регулируется
<i>Длина сигнатуры (signLen)</i>	1	20	Длина блока сигнатуры на странице счета пользователя. Это значение может быть уменьшено, чтобы освободить место для других (дополнительных) данных
<i>Длина дополнительных данных (auxDataLen)</i>	1	0	Длина блока дополнительных данных, регулируется. В случае нулевой длины дополнительные данные не предоставляются
Имя провайдера	20	Dallas Semiconductor	Имя поставщика услуг или описание. Длина ограничивается элементом <i>Длина имени провайдера</i>
Начальное значение сигнатуры	20	20 байт 00h	Длина начального блока должна соответствовать вышеупомянутому элементу <i>Длина сигнатуры</i> . Эти байты устанавливаются на место блока сигнатуры на странице данных пользователя для создания сигнатуры данных
Дополнительные данные	0		Длина дополнительных данных должна совпадать с вышеупомянутым элементом <i>Длина дополнительных данных</i>
Код алгоритма шифрования	1	0	Флаг, показывающий тип метода шифрования или кодирования, использованного для защиты содержимого этого файла
Флаг DS1961S	1	0	Булев флаг, указывающий, что установка секрета для этого сопроцессора была выполнена с соответствующим дополнением для использования с прибором DS1961S
Общее число байтов	100		

Заметим, что длина COPR.0 зависит от значений трех параметров длины: длины имени провайдера, длины сигнатуры и длины дополнительных данных. Эти параметры предназначены для того, чтобы учесть дополнительные потребительские характеристики, которые поддерживаются программным обеспечением, но их необходимо активизировать с помощью данных, содержащихся в файле COPR.0.

Файл данных обслуживания

В приведенном примере реализации мы создали файл обслуживания DLISM.102 в каждом приборе SHA iButton пользователя, чтобы идентифицировать его в качестве iButton пользователя. Файл DLISM.102 содержит следующие данные.

Таблица 2. Структура данных файла обслуживания

Структура файла обслуживания DLISM.102			
	Число байтов	Данные примера	Примечания
Код типа данных	1	0	Это поле может использоваться для последующей индикации типа данных транзакции. Например, 0 — для динамических и 1 — для статических данных, и т.д.
Сигнатура данных обслуживания	20	[вычислено]	Вычисляется местным хостом
Множитель/коэффициент преобразования	2	8B 48h	Эти данные могут использоваться для преобразования одной величины в другую. В примере обслуживания мы объединяем международный код валюты для USD и коэффициент преобразования 0.01 для конвертации из центов в доллары. Конкретный формат обычно зависит от услуги
Баланс счета	3	01 86 A0 h	100 000 центов (\$1000)
Идентификатор транзакции	2	12 34h	Идентификатор транзакции может использоваться для привязки к транзакции дополнительных справочных данных
Общее число байтов	28		

Заметим, что содержимое этого файла составляет 28 байт, но на странице хранится 32 байт в следующем формате (подробнее см. в AN114):

Длина файла (1 байт)	Содержимое файла (28 байт)	Указатель продолжения файла (1 байт)	CRC-16 (2 байт)
-------------------------	-------------------------------	---	--------------------

Когда местный хост обнаруживает SHA iButton пользователя, он считывает из таблицы элементов директории в приборе SHA iButton пользователя номер страницы (номер 13 в нашем примере обслуживания) для файла DLISM.102 и выполняет аутентификацию прибора, используя секрет, соответствующий странице 13 (номер секрета — 5).

Установка данных обслуживания

Перед проведением любой транзакции в сопроцессор и *iButton* пользователя должны быть установлены соответствующие данные обслуживания и секреты, — процесс, называемый также инициализацией прибора. Аутентификация прибора необходима для обслуживания как статических, так и динамических данных. Если при обслуживании используются динамические данные (например, в приложениях электронных платежей), то данные обслуживания нуждаются в проверке и обновлении во время каждой транзакции. В этом случае сопроцессор должен содержать два секрета системы: один для аутентификации и другой для проверки и новой подписи данных обслуживания. Этапы для установки двух секретов системы идентичны, за исключением того, что секрет подписи системы должен устанавливаться в секрет с номером 0, и обычно вычисляется при помощи различных входных данных (неполных фраз, или *partial phrases*). На стороне прибора пользователя защита данных обслуживания обеспечивается встраиванием сигнатуры, вычисленной из данных обслуживания и секрета подписи системы, в страницу данных обслуживания. Основные этапы для установки обслуживания в сопроцессор и SHA *iButton* пользователя состоят в следующем (подробное описание представлено в последующих разделах этого документа).

Для обслуживания, не требующего проверки достоверности данных:

- (1) Установить секрет аутентификации системы в сопроцессор.
- (2) Установить уникальный секрет аутентификации прибора в *iButton* пользователя посредством привязки секрета аутентификации системы к номеру адреса *iButton* пользователя и номеру страницы данных обслуживания.
- (3) Записать данные обслуживания в *iButton* пользователя (если требуется).

Для обслуживания, требующего проверки достоверности данных:

- (1) Установить секрет аутентификации системы в сопроцессор.
- (2) Установить секрет подписи системы в сопроцессор.
- (3) Установить уникальный секрет аутентификации прибора в *iButton* пользователя посредством привязки секрета аутентификации системы к номеру адреса *iButton* пользователя и номеру страницы данных обслуживания.
- (4) Вычислить сигнатуру подписи из секрета подписи системы, данных обслуживания, номера страницы и значения счетчика циклов записи страницы данных обслуживания, а также номера адреса прибора пользователя.
- (5) Встроить сигнатуру подписи в данные обслуживания и записать их в *iButton* пользователя.

Генерация секрета в SHA *iBUTTON*

Установку секрета в SHA *iButton* следует производить путем записи данных (неполных фраз, подробнее см. в документе AN152) в предназначенную для этого страницу и в блокнотную память, вычисляя код MAC из данных и копируя выбранные байты MAC в указанный номер секрета (см. Рис. 4). Следует избегать прямой записи в секрет. Генерация секрета с помощью встроенного процессора SHA также позволяет участвовать в процессе установки секрета сразу множеству приборов (что называется совместным использованием секрета), после чего ни один из них не может воспроизвести секрет системы без взаимодействия с другими приборами. Этот процесс значительно снижает риск раскрытия секрета. Последовательность операций для генерации секрета системы (аутентификации или подписи) из N неполных фраз ($partials[k]$, где $k = 0 \dots N - 1$) представлена на Рис. 5. При каждой итерации для вычисления кода MAC используются входные данные двух источников: 47 байт входных данных (32 байт, записанные на странице данных, и 15 байт, записанные в блокнот) и текущее содержимое секрета (8 байт). Секрет обновляется с помощью нового кода MAC, который был вычислен перед самым началом следующей итерации. Предполагается, что в начале цикла ($k = 0$) входные данные, поступающие из секрета, имеют нулевое значение. Поскольку на каждом этапе создается новый секрет, который становится входными данными для следующего вычисления, итоговый секрет системы является функцией всех предшествующих входных неполных фраз.

Вышеупомянутая последовательность равным образом применима к вычислению секрета аутентификации системы и секрета подписи системы, за исключением того, что секрет подписи системы должен быть установлен в секрет с номером 0 ($cSignSecretNum = 0$) сопроцессора. Для установки в SHA *iButton* секрета системы из множества неполных фраз, используется функция общего назначения: *installSystemSecret*, подробности использования см. в Приложении А.

Рис. 4. Генерация секрета в SHA iBUTTON

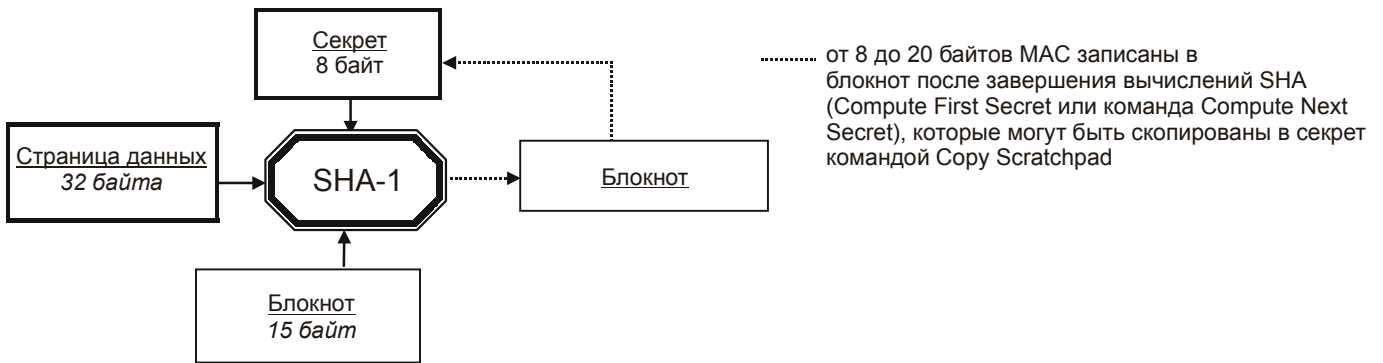
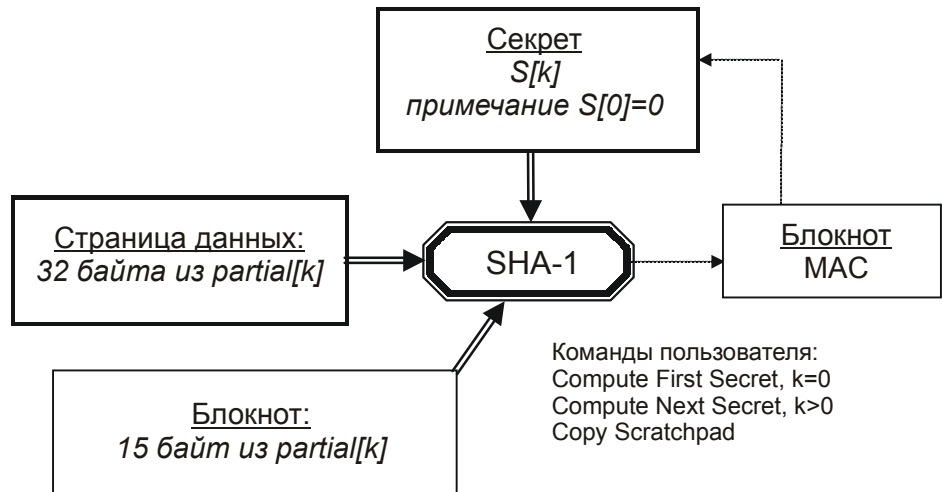


Рис. 5. Генерация секрета системы при совместном использовании секрета

Цикл: k=0 до N-1



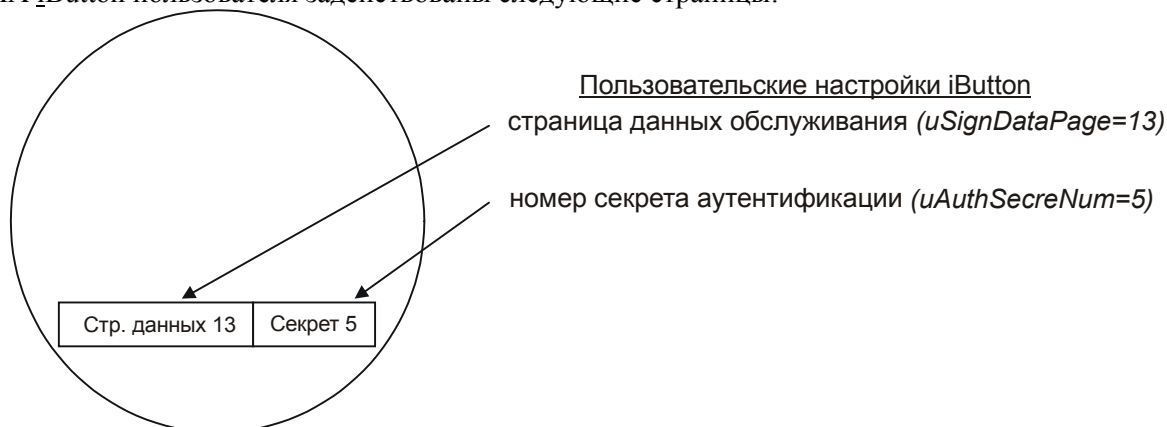
Прежде, чем начнется следующая итерация, секрет скорректирован промежуточным результатом MAC, записанным в блокнот после вычисления SHA

Конец цикла

Пример обслуживания

В качестве примера обслуживания рассматривается услуга электронного платежа с дебетованием баланса счета и подписанием его вновь при каждой транзакции новой сигнатурой подписи. Заметим, что аутентификация прибора не проверяет содержимого страницы данных обслуживания, а только использует секрет аутентификации прибора, связанный с этой страницей, в iButton пользователя. Для подписи данных обслуживания секрет подписи системы сохраняется в сопроцессорах, а не в iButton пользователя. Поэтому мы можем использовать только одну страницу данных и один секрет для хостинга приложения электронного платежа: страницу данных для данных обслуживания и соответствующий ей секрет для аутентификации прибора. Для справки ниже приводятся обозначения и переменные, использованные в нашем примере обслуживания. Заметим, что в приборе хостинга множественного обслуживания номер страницы данных обслуживания (*uSignDataPage*) и номер секрета аутентификации (*uAuthSecreNum*) могут отличаться в каждом iButton пользователя.

В SHA *iButton* пользователя задействованы следующие страницы:



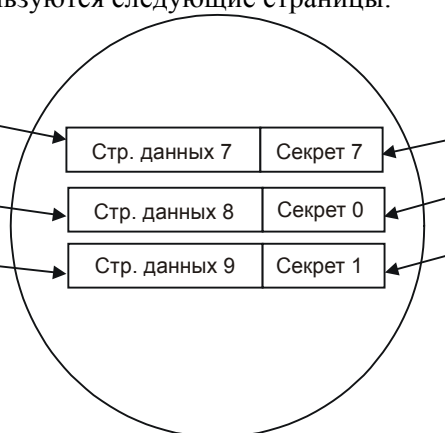
В сопроцессоре SHA *iButton* используются следующие страницы:

Настройки сопроцессора iButton

Страница данных аутентификации системы (*cAuthDataPage=7*)

Страница данных подписи системы (*cSignDataPage=8*)

Страница данных рабочей области (*wkDataPage=9*)



Настройки сопроцессора iButton

Секрет аутентификации системы (*cAuthSecretNum=7*)

Секрет подписи системы (*cSignSecretNum=0*)

Секрет рабочей области (*wkSecretNum=1*)

Таблица 3. Перечень переменных и значений

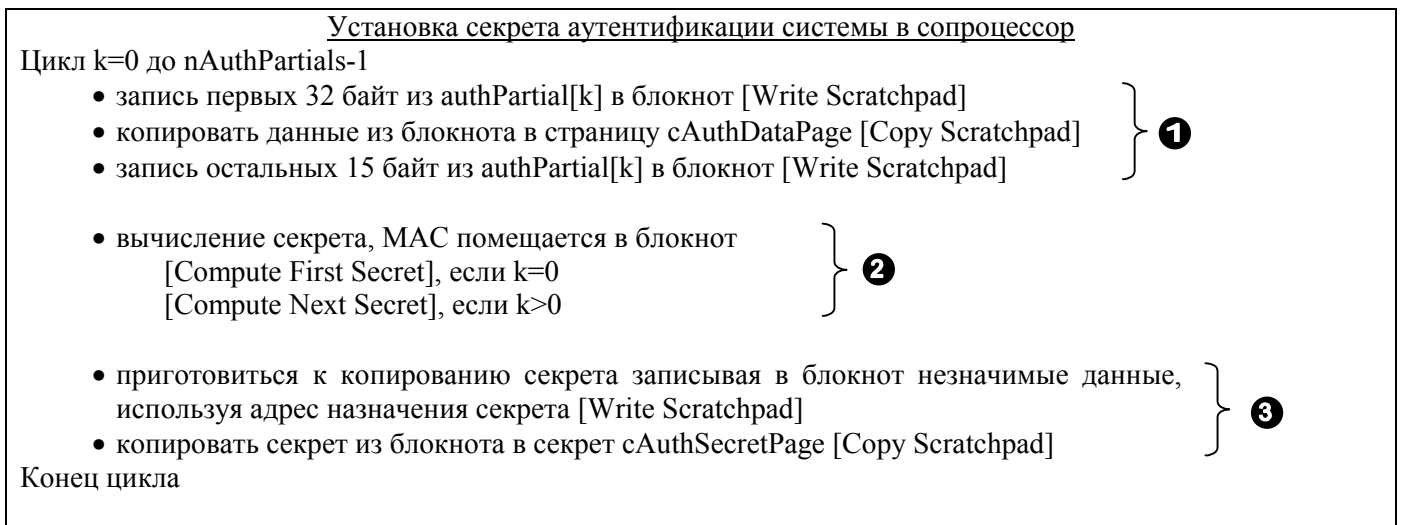
Перечень переменных		
Переменная	Значение в примере обслуживания	Описание
<i>cAN</i>		номер адреса сопроцессора <i>iButton</i>
<i>uAN</i>		номер адреса <i>iButton</i> пользователя
<i>cAuthDataPage</i>	7	страница данных аутентификации системы, в сопроцессоре
<i>cAuthSecretNum</i>	7	номер секрета аутентификации системы, в сопроцессоре
<i>cSignDataPage</i>	8	страница данных подписи системы, в сопроцессоре
<i>cSignSecretNum</i>	0	номер секрета подписи системы, в сопроцессоре
<i>uAuthDataPage</i>	13	страница данных аутентификации прибора пользователя
<i>uAuthSecretNum</i>	5	номер секрета аутентификации прибора пользователя
<i>uSignDataPage</i>	13	номер страницы данных обслуживания в <i>iButton</i> пользователя

Перечень переменных		
Переменная	Значение в примере обслуживания	Описание
<i>svcName</i>	Dallas Semiconductor	имя поставщика услуг, дополненное справа байтами 00h, если необходимо
<i>nAuthPartials</i>	1	число неполных фраз для вычисления секрета аутентификации системы
<i>nSignPartials</i>	1	число неполных фраз для вычисления секрета подписи системы
<i>authPartial[j]</i> <i>j = 0 to nAuthPartials-1</i>	47 байт FFh	неполные фразы для вычисления секрета аутентификации системы — 47 байт в каждом элементе матрицы: 32 байт, записанных на страницу данных, и 15 байт — в блокнотную память
<i>signPartial[j]</i> <i>j = 0 to nSignPartials-1</i>	47 байт FFh	неполные фразы для вычисления секрета подписи системы — 47 байт в каждом элементе матрицы: 32 байт, записанных на страницу данных, и 15 байт — в блокнотную память
<i>authBind</i>	39 байт 00h	данные привязки секрета аутентификации прибора — 39-байтный блок, используемый для привязки секрета аутентификации системы к прибору пользователя. Первые 32 байт записываются на страницу данных, а остальные 7 байт записываются в блокнотную память для вычисления секрета аутентификации прибора
<i>signCode</i>	3 байт 00h	3-байтный код, используемый для создания сигнатуры подписи данных обслуживания
<i>signInitial</i>	20 байт 00h	начальное значение для вычисления сигнатуры подписи данных обслуживания
<i>uData</i>		содержимое страницы данных обслуживания
<i>TA1, TA2, ES</i>		адресные регистры прибора и регистр состояния
<i>uSignDataPageWCC</i>		счетчик циклов записи страницы данных обслуживания, в <i>iButton</i> пользователя ($=uAuthDataPageWCC$)
<i>cFileName</i>	COPR.0	имя файла сопроцессора, хранящееся в сопроцессоре
<i>uFileName</i>	DLSM.102	имя файла обслуживания, хранящееся в <i>iButton</i> пользователя
<i>wkDataPage</i>	9	номер страницы данных рабочей области в сопроцессоре
<i>wkSecretNum</i>	1	номер секрета рабочей области, в сопроцессоре

Установка секрета аутентификации системы в сопроцессор

Ниже приводится последовательность команд и поток данных для установки секрета аутентификации системы в SHA *iButton*. Подробности реализации функции API (*installSystemSecret*) представлены в Приложении А.

Рис. 6. Установка секрета аутентификации системы в сопроцессор



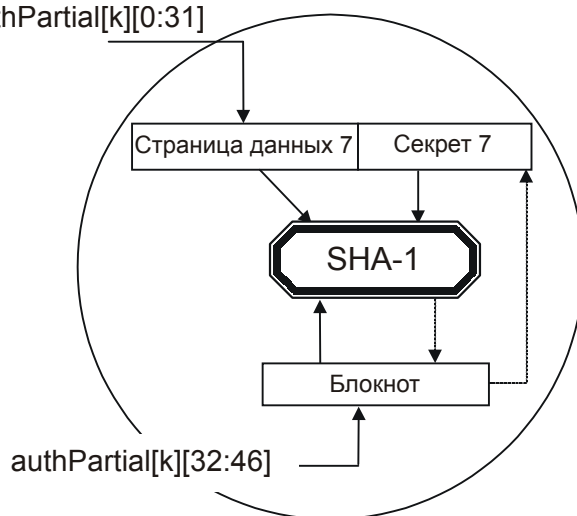
Чтобы установить секрет аутентификации системы в сопроцессор, используя API:

$s = installSystemSecret(cAN, cAuthDataPage, cAuthSecretNum, nAuthPartials, authPartial)$

$s = eraseDataPage(cAN, cAuthDataPage)$

Вызов функции *eraseDataPage* (описанной в Приложении А) стирает последнюю неполную фразу, записанную в сопроцессор, для того чтобы не допустить ее раскрытия.

$authPartial[k][0:31]$



Сопроцессор

Секрет аутентификации системы установлен в секрет $cAuthSecretNum(=7)$ сопроцессора. Часть фразы записана в страницу $cAuthDataPage(=7)$ и блокнот для вычисления секрета.

Следует иметь в виду, что в этом документе, для краткости, в листинге вызовов API игнорируется любая проверка ошибок и повторные итерации. В практической реализации каждый возврат функции следует сопровождать проверкой на ошибки и создавать вокруг вызова функции соответствующие циклы итерации. Например, фрагмент программы для установки секрета аутентификации системы в сопроцессор с соответствующей проверкой на ошибки и циклами повторения, может выглядеть следующим образом:

```

// ограничение счета итераций
loopLimit = 5
s = 1
loop = 0
do while (loop < loopLimit and s <> 0)
// функция возвращает 0, если нет ошибок
s = installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)
loop = loop + 1
end loop

// выйти, если мы продолжаем иметь ошибки после предустановленного числа ветвлений
if (s <> 0) then
    exit
end if

s = 1
loop = 0
do while (loop < loopLimit and s <> 0)
// функция возвращает 0, если нет ошибок
s = eraseDataPage(cAN,cAuthDataPage)
loop = loop + 1
end loop

// выйти, если мы продолжаем иметь ошибки после предустановленного числа итераций
if (s <> 0) then
    exit
end if
...continue...

```

Установка секрета подписи системы в сопроцессор

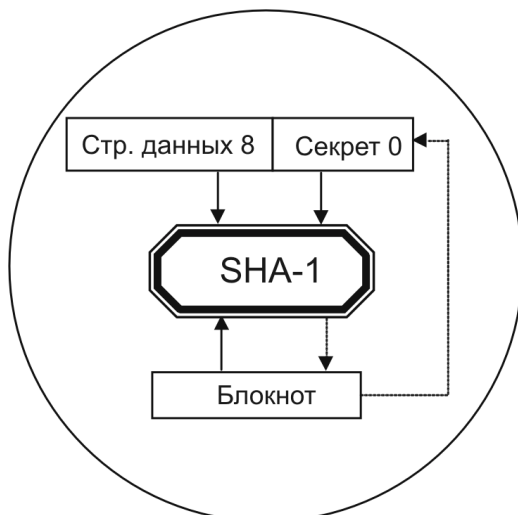
Последовательность команд и поток данных для установки секрета подписи системы в SHA iButton идентичны последовательности команд и потоку данных для установки секрета аутентификации системы, за исключением неполных фраз, а также страницы данных и секрета, в которые производится установка (секрет подписи системы должен быть установлен в секрет 0).

Чтобы установить секрет подписи системы в сопроцессор, используя API:

```

s = installSystemSecret(cAN,cSignDataPage, cSignSecretNum, nSignPartials,signPartial)
s = eraseDataPage(cAN,cSignDataPage)

```



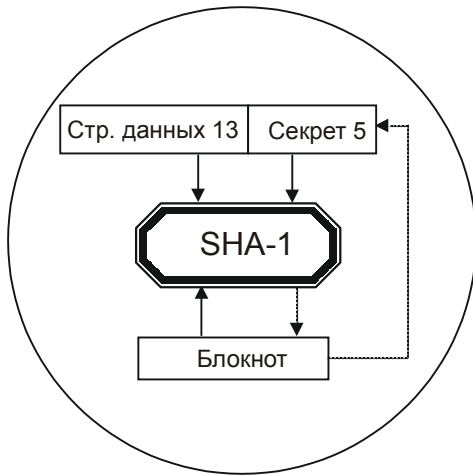
Сопроцессор

Секрет подписи системы установлен в секрет `cSignSecretNum (=0)` сопроцессора. Часть фразы записана в страницу `cAuthDataPage (=8)` и блокнот для вычисления секрета.

Заметим, что секрет подписи системы должен быть установлен в секрет 0 ($cSignSecretNum = 0$), а соответствующая ему страница данных $cSignDataPage$ может иметь номер 0 или 8. В нашем примере обслуживания для директории OWFS используется страница 0, поэтому мы используем $cSignDataPage = 8$.

Установка секрета аутентификации прибора в **iBUTTON** пользователя

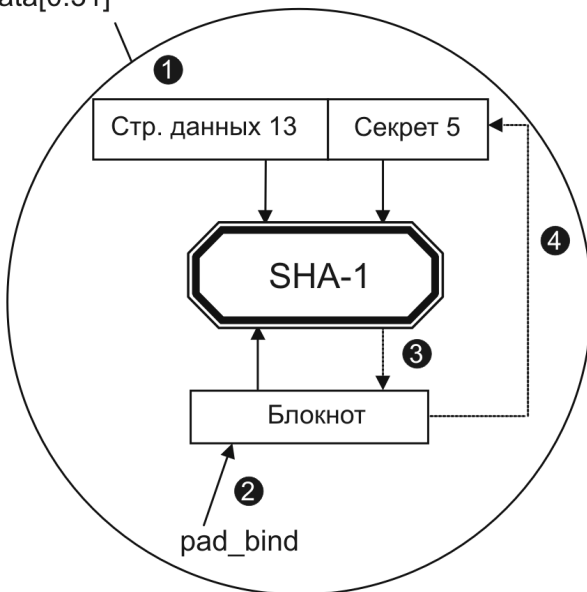
Для установки секрета аутентификации прибора в **iButton** пользователя, сначала в **iButton** пользователя устанавливается секрет аутентификации системы ($secretAuthSystem$), затем уникальный секрет прибора вычисляется из данных привязки системы ($bindData$), секрета аутентификации системы ($secretAuthSystem$), номера страницы данных обслуживания ($uSignDataPage$) и номера адреса (uAN) **iButton** пользователя. Именно номер адреса **iButton** пользователя делает секрет аутентификации прибора уникальным для **iButton**. Ниже показан двухступенчатый процесс, на втором этапе которого реализуется API ($bindSecretToiButton$) (подробности см. в Приложении А).



Секрета аутентификации **iButton** пользователя (шаг 1)
 Секрет аутентификации системы ($secretAuthSystem$) установлен в секрет $uAuthSecretNum (=5)$ **iButton** пользователя.
 Часть фразы записана в страницу $uAuthDataPage (=13)$ и блокнот для вычисления секрета.

Привязка данных прибора и данных обслуживания для создания секрета аутентификации прибора

$bindData[0:31]$



Секрета аутентификации **iButton** пользователя (шаг 2)
 Секрет аутентификации прибора ($secretAuthDevice$) установлен в секрет $uAuthSecretNum (=5)$ **iButton** пользователя, объединяющий секрета аутентификации системы ($secretAuthSystem$) с данными обслуживания и данными **iButton** пользователя $iButtonspecific (uAuthDataPage, uAN)$.

Дополнение данных привязки, данных обслуживания и данных прибора пользователя в блокнотной памяти для вычисления уникального секрета аутентификации прибора (*pad_bind*):

Смещение	0:7	8:11	12:12	13:19	20:22	23:31
Число байтов	8	4	1	7	3	9
Данные	00h	bindData[32:35]	uAuthDataPage	uAN[0:6]	bindData[36:38]	00h

Установка секрета аутентификации системы iButton пользователя

Цикл k=0 до nAuthPartials-1

- запись первых 32 байт из authPartial[k] в блокнот [Write Scratchpad]
- копировать данные из блокнота в страницу sAuthDataPage [Copy Scratchpad]
- запись остальных 15 байт из authPartial[k] в блокнот [Write Scratchpad]

- вычисление секрета, MAC помещается в блокнот
если k=0 [Compute First Secret]
если k>0 [Compute Next Secret]

- подготовиться к копированию секрета записывая в блокнот незначимые данные, используя адрес назначения секрета [Write Scratchpad]
- копировать секрет из блокнота в секрет sAuthSecretPage [Copy Scratchpad]

Конец цикла



Привязка данных обслуживания и данных прибора пользователя для вычисления уникального секрета аутентификации прибора

- запись первых 32 байт bindData в блокнот [Write Scratchpad] } **1**
- копировать данные из блокнота в страницу uAuthDataPage [Copy Scratchpad] }

- запись блока данных заполненной связи (*pad_bind*) и сформированных остальных 7 байт bindData, номера страницы аутентификации данных (uAuthDataPage), номера адреса пользователя iButton (uAn без CRC) в блокнот [Write Scratchpad] } **2**

- вычисление секрета [Compute Next Secret], результат MAC помещается в блокнот } **3**

- подготовиться к копированию секрета записывая в блокнот незначимые данные, используя адрес назначения секрета [Write Scratchpad] } **4**
- копировать секрет из блокнота в секрет sAuthSecretPage [Copy Scratchpad] }

Чтобы установить секрет аутентификации прибора, используя API:

s = installSystemSecret(uAN, uAuthDataPage, uAuthSecretNum, nAuthPartials, authPartial)

s = bindSecretToiButton(uAN, uAuthDataPage, uAuthSecretNum, bindData, uAuthDataPage, uAN)

s = eraseDataPage(uAN, uAuthDataPage)

Создание сигнатуры данных обслуживания в сопроцессоре

Сигнатура данных обслуживания вычисляется из данных обслуживания и секрета подписи системы. Сигнатура может встраиваться⁴ в страницу данных обслуживания и проверяться местным хостом во время каждой транзакции. Например, в нашем примере обслуживания мы встроили сигнатуру на страницу данных обслуживания следующим образом (байт длины, указатель продолжения и байты CRC-16 необходимы для OWFS):

Встраивание сигнатуры данных в страницу данных обслуживания

Смещение	0:0	1:1	2:21	22:23	24:26	27:28	29:29	30:31
Число байтов	1	1	20	2	3	2	1	2
Данные	длина	тип данных	блок сигнатуры	коэффициент преобразования	баланс счета	идентификатор транзакции	указатель продолжения	CRC-16

Поскольку сигнатура занимает часть страницы данных обслуживания, блок сигнатуры должен быть установлен (инициализирован) в некоторое известное значение (*signInitial*)⁵ для вычисления сигнатуры. Заметим, что с целью предотвращения несанкционированного копирования и повторного использования данных обслуживания (делания «денег»), для вычисления сигнатуры в качестве входных данных следует брать такие технические данные прибора пользователя, как номер адреса прибора (*uAN*), номер страницы (*uSignDataPage*) и значение счетчика циклов записи (*uSignDataPageWCC*) страницы данных обслуживания. Важно также, что данные обслуживания устанавливаются на страницу, значение счетчика циклов записи которой возрастает при каждой операции записи, если копирование данных обслуживания не допускается. Процесс создания сигнатуры данных обслуживания реализуется в *createDataSignature* API, подробности реализации см. в разделе 0.

```
// создать сигнатуру; переданные данные uData_ini имеют блок сигнатуры, установленный в signInitial
sig = createDataSignature(cAN, cSignDataPage, cSignSecretNum, uData_ini, signCode,
    uAN, uSignDataPage, uSignDataPageWCC)
```

Дополненный блок данных подписи (pad_signCode):

Смещение	0:7	8:11	12:18	19:19	20:22	23:31
Число байтов	8 дополняющих	4	7	1	3	9 дополняющих
Данные	8 байт 00h	$uSignDataPageWCC + 1^6$	$uAN[0:7]$	<i>uSignDataPage</i>	<i>signCode</i>	9 байт 00h

⁴ Заметим, что сигнатуру подписи не обязательно встраивать на страницу данных обслуживания. Она могла бы храниться на отдельной странице и, таким образом, освободить все 32-байтное пространство для исходных (необработанных) данных обслуживания.

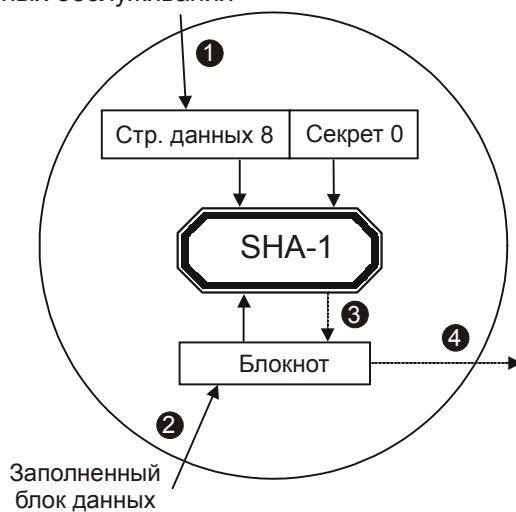
⁵ Если подпись не встраивается как часть 32-байтной страницы данных обслуживания, то инициализация не требуется:
 $sig = createDataSignature(cAN, cSignDataPage, cSignSecretNum, uData, signCode, uAN, uSignDataPage, uSignDataPageWCC)$

⁶ $WCC + 1$ становится новым счетчиком циклов записи страницы данных обслуживания после того, как на нее были записаны подписанные данные обслуживания, поскольку каждая запись на эту страницу увеличивает значение счетчика циклов записи на 1 (на страницах с 8 по 15).

Создание сигнатуры данных обслуживания в сопроцессоре

- запись `uData_ini` в блокнот [Write Scratchpad] } **1**
- копировать данные из блокнота в страницу `cSignDataPage` [Copy Scratchpad] } **1**
- запись данных подписи (*pad_signCode*) состоящих из: *signCode*, *uSignDataPageWCC*, *uAN* (without the CRC), и *uSignDataPage* в блокнот [Write Scratchpad] } **2**
- вычисление сигнатуры [Sign Data Page];
результат MAC помещается в блокнот } **3**
- чтение сигнатуры MAC из блокнота [Read Scratchpad] } **4**

Инициализация
данных обслуживания



Сопроцессор

Вычисление сигнатуры данных обслуживания из:

- инициализация данных обслуживания (*uData_ini*)
- секрет подписи системы (*cSignSecretNum=0*)
- Код подписи системы (*signCode*)
- Номер адреса iButton пользователя *uAN* (без CRC)
- Номер страницы данных обслуживания (*uSignDataPage*) и значение счетчика циклов записи (*uSignDataPageWCC*)

Установка обслуживания: обобщение

Ниже приводится краткое изложение основных вызовов API для установки обслуживания.

Обслуживание статических данных без проверки достоверности данных

Установка данных обслуживания в сопроцессор

```
// установить секрет аутентификации системы в сопроцессор
s = installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)
```

```
// стереть неполную фразу, чтобы не допустить ее случайного раскрытия
s = eraseDataPage(cAN,cAuthDataPage)
```

Установка данных обслуживания в iButton пользователя

```
// установить секрет аутентификации системы в iButton пользователя
s = installSystemSecret(uAN,uAuthDataPage,uAuthSecretNum,nAuthPartials,authPartial)
```

// привязать данные обслуживания, идентификатор прибора пользователя к секрету системы для создания уникального секрета аутентификации прибора

```
s = bindSecretToiButton(uAN,uAuthDataPage,uAuthSecretNum,bindData,uAuthDataPage,uAN)
```

```
s = eraseDataPage(uAN,uAuthDataPage) // стереть данные привязки со страницы данных
```

Обслуживание с проверкой достоверности данных

Установка данных обслуживания в сопроцессор

```
// установить секрет аутентификации системы в сопроцессор
s = installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)
```

```
// установить секрет подписи системы в сопроцессор
s = installSystemSecret(cAN,cSignDataPage, cSignSecretNum, nSignPartials,signPartial)
```

```
s = eraseDataPage(cAN,cSignDataPage) // стереть неполную фразу, чтобы не допустить ее случайного раскрытия
```

```
s = eraseDataPage(cAN,cAuthDataPage)
```

Установка данных обслуживания в iButton пользователя

```
// установить секрет аутентификации системы в iButton пользователя
s = installSystemSecret(uAN,uAuthDataPage,uAuthSecretNum,nAuthPartials,authPartial)
```

// привязать данные обслуживания, идентификатор прибора пользователя к секрету системы для создания уникального секрета аутентификации прибора

```
s = bindSecretToiButton(uAN,uAuthDataPage,uAuthSecretNum,bindData,uAuthDataPage,uAN)
```

```
// создать сигнатуру данных обслуживания в сопроцессоре
sig = createDataSignature(cAN,cSignDataPage,cSignSecretNum,uData_ini,signCode,
uAN,uSignDataPage,uSignDataPageWCC)
```

```
// встроить сигнатуру в страницу данных обслуживания
sData = ...
```

```
// записать данные обслуживания со встроенной сигнатурой в iButton пользователя
s = writeDataPage(uAN,uSignDataPage,sData)
```

Проведение транзакций

При проведении транзакций с SHA iButton может потребоваться два этапа: аутентификация местным хостом SHA iButton пользователя и обновление данных обслуживания. Местный хост может состоять из трех логических компонентов: сопроцессора, блока управления транзакцией (TCU) и интерфейса пользователя, например, дисплея, считывающего устройства для маркера и обслуживающего устройства. В качестве TCU может выступать персональный компьютер или микроконтроллер. Важно, чтобы сопроцессор и другие элементы, содержащие секрет и конфиденциальную информацию, имели физическую и электронную защиту в соответствии с уровнем безопасности.

Аутентификация iBUTTON пользователя

Процесс аутентификации SHA iButton пользователя, который описывается на Рис. 7, может быть представлен в виде вызовов API следующим образом:

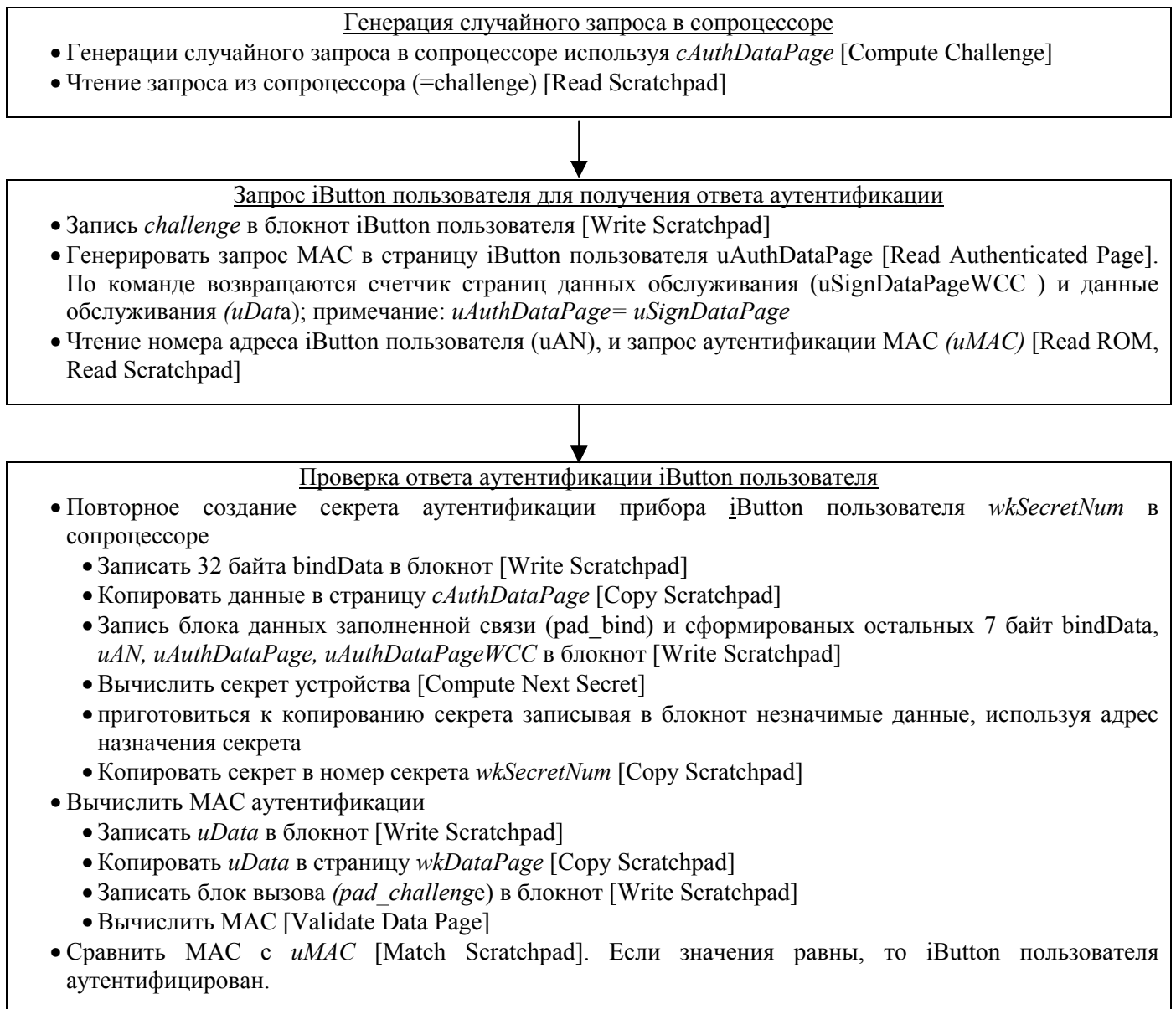
```
challenge = createChallenge(cAN,cAuthDataPage) // вычислить запрос в сопроцессоре

// запросить от iButton пользователя ответ аутентификации
// вызов API возвращает страницу данных обслуживания, счетчик страниц и ответный код MAC
// uData = uResp[0:31]; uSignDataPageWCC = uResp[32:35]; uMAC = uResp[36:55]
uResp = answerChallenge(uAN,uAuthDataPage,challenge)

// вновь создать секрет аутентификации iButton прибора пользователя в сопроцессоре
s = bindSecretToiButton(cAN,cAuthDataPage,wkSecretNum,bindData, uAuthDataPage,uAN)

// проверить ответный код MAC в сопроцессоре
s = verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage,
    uSignDataPageWCC,challenge,uMAC)
```

Рис. 7. Процесс аутентификации SHA iBUTTON пользователя



Дополненный блок данных привязки (*pad_bind*)

8 байт	4 байт	1 байт	7 байт	3 байт	9 байт
8 байт 00h	<i>bindData</i> [32:35]	<i>uAuthDataPage</i>	<i>uAN</i> [0:6]	<i>bindData</i> [36:38]	9 байт 00h

Дополненный блок данных запроса (*pad_challenge*)

8 дополняющих байт	4 байт	1 байт	7 байт	3 байт	9 дополняющих байт
8 байт 00h	<i>uAuthDataPageWCC</i>	<i>uAuthDataPage</i>	<i>uAN</i> [0: 6]	<i>challenge</i>	9 байт 00h

Генерация случайного запроса в сопроцессоре

Для генерации случайного запроса в сопроцессоре, TCU должен только затребовать, чтобы сопроцессор выполнил операцию вычисления запроса (Compute Challenge). TCU может взять любые 3 из 20-ти байт кода MAC, размещенных процессором SHA в блокнотной памяти по завершении выполнения команды. Функция API вызывается следующим образом (подробности реализации представлены в Приложении А):

challenge = *createChallenge(cAN, cAuthDataPage)*

Запрос iBUTTON пользователя для получения ответа аутентификации

TCU посылает запрос (*challenge*) в iButton пользователя с требованием вычислить ответный код MAC, основываясь на запросе (записанном в блокнотную память со смещением 20...22) и секрете аутентификации прибора. Реализация API представлена в Приложении А.

$$uResp = answerChallenge(uAN, uAuthDataPage, challenge)$$

После возврата uResp содержит данные обслуживания (*uData*), счетчик циклов записи страницы (*uAuthDataPageWCC*) и ответный код MAC (*uMAC*).

$$\begin{aligned} uData &= uResp[0:31] \\ uAuthDataPageWCC &= uResp[32:35] \\ uMAC &= uResp[36:55] \end{aligned}$$

Проверка ответа аутентификации iBUTTON пользователя

Для проверки ответа iButton пользователя, сначала в сопроцессоре должен быть вновь создан секрет аутентификации прибора iButton пользователя. Заметим, что вновь созданный секрет аутентификации не может считываться TCU, но этого и не требуется. Аутентификация достигается не непосредственным сравнением секретов аутентификации, а результатами вычислений SHA, основываясь на секрете аутентификации и других данных обслуживания и технических данных прибора. После того, как в сопроцессоре вновь создается секрет аутентификации пользователя, выполняется команда проверки достоверности страницы данных (Validate Data Page), чтобы вычислить код MAC для сравнения. Сопроцессор сохраняет результат вычисления (*cMAC*) в блокнотной памяти, скрывая его от считывания извне. Для сравнения *uMAC*, считанного из iButton пользователя, и *cMAC* из блокнотной памяти сопроцессора, передается команда сравнения блокнота (Match Scratchpad).

Первый этап процесса аутентификации — создание вновь секрета аутентификации прибора пользователя в сопроцессоре, — аналогичен установке секрета аутентификации прибора в iButton пользователя во время процесса установки обслуживания, за исключением того, что на этот раз прибором назначения является сопроцессор, и секрет устанавливается в неиспользуемый номер секрета сопроцессора. Реализация второго этапа по проверке ответного кода MAC iButton пользователя (*uMAC*) приводится в Приложении А.

// вновь создать уникальный секрет аутентификации прибора пользователя в сопроцессоре — в неиспользуемом номере секрета (wkSecretNum)

$$s = bindSecretToiButton(cAN, cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)$$

// проверить ответный MAC-код прибора пользователя в сопроцессоре

$$s = verifyAuthResponse(cAN, wkDataPage, uData, uAN, uAuthDataPage, uAuthDataPageWCC, challenge, uMAC)$$

Проверка встроенной сигнатуры данных обслуживания

Страница данных обслуживания проверяется путем сличения встроенной в нее сигнатуры (*sign*). Сначала блок сигнатуры страницы данных обслуживания устанавливается в известное начальное значение (*signInitial*), и в сопроцессоре вновь вычисляется сигнатура (*cSign*). Затем TCU сравнивает *sign* с *cSign*, и если они идентичны, то данные обслуживания являются достоверными. Заметим, что мы получили данные обслуживания и значение счетчика циклов записи страницы данных обслуживания во время аутентификации iButton пользователя.

Чтобы вновь создать сигнатуру, сопроцессор сначала сохраняет сигнатуру (*sign*), а затем инициализирует блок сигнатуры *uData* при помощи *signInitial* для восстановления первоначального блока (*uData_ini*), который был использован для генерации сигнатуры. Затем мы используем прибор пользователя и параметры обслуживания (номер адреса, номер страницы обслуживания и счетчик) для того, чтобы вновь создать сигнатуру:

$$cSign = createDataSignature(cAN, cSignPage, cSignSecretNum, uData_ini, signCode, uSignDataPage, uAN, uSignDataPageWCC - 1)$$

Переподписание обновленных данных обслуживания

При обслуживании динамических данных, данные обслуживания обновляются, и для новой подписи данных обслуживания вычисляется новая сигнатура. Этот процесс идентичен процессу установки начальных данных обслуживания в `iButton` пользователя. Если `uData_Update` являются обновленными данными обслуживания с блоком сигнатуры, установленным в `signInitial`, тогда новая сигнатура создается следующим образом:

```
sign = createDataSignature(cAN, cSignPage, cSignSecretNum, uData_Update,
                           signCode, uSignDataPage, uAN, uSignDataPageWCC)
```

Проведение транзакции с SHA `iBUTTON`: обобщение

Ниже приведено краткое изложение основных вызовов API для проведения транзакции.

Обслуживание статических данных без проверки достоверности данных

```
// создать случайно изменяющийся код в сопроцессоре
challenge = createChallenge(cAN, cAuthDataPage)

// затребовать от iButton пользователя ответ на запрос
uResp = answerChallenge(uAN, uAuthDataPage, challenge)
uData = uResp[0:31]
uSignDataPageWCC = uResp[32:35]
uMAC = uResp[36:55]

// вновь создать секрет аутентификации прибора в сопроцессоре
// в неиспользуемом номере секрета (wkSecretNum)
s = bindSecretToiButton(cAN, cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)

// проверить ответный MAC-код прибора пользователя в сопроцессоре
status = verifyAuthResponse(cAN, wkDataPage, uData, uAN, uAuthDataPage,
                            uAuthDataPageWCC, challenge, uMAC)
...

```

Обслуживание статических данных с проверкой достоверности данных

```
// создать случайно изменяющийся код в сопроцессоре
challenge = createChallenge(cAN, cAuthDataPage)

// затребовать от iButton пользователя ответ на запрос
uResp = answerChallenge(uAN, uAuthDataPage, challenge)
uData = uResp[0:31]
using = uData[4:23]
uSignDataPageWCC = uResp[32:35]
uMAC = uResp[36:55]

```

```
// вновь создать уникальный секрет аутентификации прибора пользователя в сопроцессоре
// в неиспользуемом номере секрета (wkSecretNum)
s = bindSecretToiButton(cAN,cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)
```

```
// проверить ответный MAC-код прибора пользователя в сопроцессоре
status = verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage,
uAuthDataPageWCC,challenge,uMAC)
```

```
// вновь создать сигнатуру данных в сопроцессоре
// инициализировать uData с помощью signInitial
uData_ini=...
cSign = createDataSignature(cAN,cSignPage, cSignSecretNum, uData_ini, signCode,
uSignDataPage, uAN, uSignDataPageWCC - 1)
```

```
// сравнить uSign и cSign
...
```

Обслуживание динамических данных

```
// создать случайно изменяющийся код в сопроцессоре
challenge = createChallenge(cAN, cAuthDataPage)
```

```
// затребовать от iButton пользователя ответ на запрос
uResp = answerChallenge(uAN,uAuthDataPage,challenge)
uData = uResp[0:31]
uSignDataPageWCC = uResp[32:35]
uMAC = uResp[36:55]
```

```
// вновь создать уникальный секрет аутентификации прибора пользователя в сопроцессоре
// в неиспользуемом номере секрета (wkSecretNum)
s = bindSecretToiButton(cAN,cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)
```

```
// проверить ответный MAC-код прибора пользователя в сопроцессоре
s = verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage,
uAuthDataPageWCC,challenge,uMAC)
```

```
// вновь создать сигнатуру данных в сопроцессоре
cSign = createDataSignature(cAN,cSignPage, cSignSecretNum, uData_ini, signCode,
uSignDataPage, uAN, uSignDataPageWCC - 1)
```

```
// сравнить uSign и cSign
...
```

```
// обновить данные обслуживания и вновь вычислить сигнатуру
sign = createDataSignature(cAN, cSignPage,cSignSecretNum, uData_Update,
signCode,uSignDataPage, uAN, uSignDataPageWCC)
```

```
// записать новые данные обслуживания и сигнатуру (upData) в iButton пользователя
s = writeDataPage(uAN,uAuthDataPage,upData)
```

```
// аутентифицировать iButton пользователя и снова проверить данные обслуживания
challenge = createChallenge(cAN, cAuthDataPage)
```

```
// затребовать от iButton пользователя ответ на запрос
uResp = answerChallenge(uAN, uAuthDataPage, challenge)
uData = uResp[0:31]
uSignDataPageWCC = uResp[32:35]
uMAC = uResp[36:55]
```

```
// убедиться, что iButton пользователя получил обновленные данные, сравнить uData и upData
```

```
...
```

```
// проверить ответный MAC-код прибора пользователя в сопроцессоре
status = verifyAuthResponse(cAN, wkDataPage, uData, uAN, uAuthDataPage,
uAuthDataPageWCC, challenge, uMAC)
```

ПРИЛОЖЕНИЕ А: Реализация API

В этом разделе рассматриваются подробности реализации различных API. Разработчик должен уметь адаптировать их к выбранному языку программирования. Следует иметь в виду, что это не листинги реальных или функционирующих компьютерных программ. Они предназначены лишь для демонстрации последовательностей основных команд и потока данных. Для краткости в листинге игнорируются проверки ошибок и циклы повторения. При фактической реализации всегда необходимы проверки ошибок и циклы повторения.

Основные операции ввода/вывода приборов шины 1-Wire

Предполагается, что необходимые функции доступа к прибору для шины 1-Wire обеспечиваются драйвером под соответствующую платформу. Названия реальных функций могут отличаться от перечисленных ниже, однако их функциональные возможности должны оставаться такими же. Мы предполагаем, что обеспечиваются следующие вспомогательные функции.

Таблица 4. Основные операции ввода/вывода для шины 1-Wire

$S = select(devAN)$	Выбирает прибор, номер адреса которого ($devAN$) приводится в аргументе. Вызов возвращает 0, если выбор прибора прошел успешно
$S = select()$	Выбирает прибор, у которого $devAN$ такой же, как и у последнего прибора, к которому был доступ. Вызов возвращает 0, если выбор прибора прошел успешно
$S = resume()$	Возобновляет обмен данными с прибором сети, который последним участвовал в обмене данными. Вызов возвращает 0, если выбор прибора прошел успешно
$S = reset()$	Сбрасывает сеть 1-Wire, вызов возвращает 0, если операция была успешной
$Data = readBytes(len)$	Читает байты длины len из выбранного прибора
$S = writeBytes(block, from, len)$	Записывает байты длины len (начиная со смещения $from$ массива данных $block$) в выбранный прибор. Вызов возвращает 0, если операция была успешной
$TA1 = lowAddress(page)$	Возвращает младший байт адреса номера страницы данных
$TA2 = highAddress(page)$	Возвращает старший байт адреса номера страницы

	данных
$TA1S = lowSecretAddress(secretNum)$	Возвращает младший байт адреса номера секрета
$TA2S = highSecretAddress(secretNum)$	Возвращает старший байт адреса номера секрета
$C = CRC16(block, from, len, seed)$	Вычисляет CRC-16 заданных элементов данных длины len (начиная со смещения $from$ массива $block$) с заданным начальным числом $seed$
$C = invCRC16(block, from, len, seed)$	Вычисляет инвертированный CRC-16 заданных элементов данных длины len (начиная со смещения $from$ массива $block$) с заданным начальным числом $seed$

Запись на страницу данных (WRITEdatapage)

Этот API записывает 32-байтный блок данных на страницу данных.

```
int status = writeDataPage(byte[] devAN, byte pageNum, byte[] data)
```

Переменная	Тип	Описание
<i>devAN</i>	byte[8]	номер адреса iButton
<i>pageNum</i>	byte	номер страницы
<i>data</i>	byte[32]	записываемый блок данных
<i>status</i>	int	0 = нет ошибки состояние $\neq 0$ означает, что произошла ошибка. В зависимости от фактической реализации, значения статуса могут отражать различные источники ошибок

$TA1 = lowAddress(pageNum)$

$TA2 = highAddress(pageNum)$

Команда	Поток данных	Примечания
select(...)	devAN	Выбирает iButton для обмена данными
Erase Scratchpad	[W]: {C3h, TA1, TA2}	Сбрасывает флаг HIDE для последующего ввода/вывода
readBytes(len)		Считывает достаточное число байтов состояния (например, len = 5), чтобы проверить, завершилась ли операция. Значение AAh указывает на завершение
reset()		Сбрасывает предыдущую операцию
resume()		Возобновляет обмен данными с прибором
Write Scratchpad	[W] {0F, TA1, TA2, данные}	Записывает блок данных в блокнотную память
readBytes(2)		Считывает инвертированный CRC-16 предыдущего потока данных. TCU должен вычислить свою версию CRC-16 для того же потока данных и сравнить со значением, считанным из прибора. Несовпадение указывает на ошибки ввода/вывода
reset()		
resume()		
Read Scratchpad	[W] AAh [R] {TA1, TA2, ES}	Считывает адресные регистры и регистр ES. TCU должен проверить, корректно ли их чтение
reset()		
resume()		
Copy Scratchpad	[W] {55h, TA1, TA2, ES}	Копирует данные блокнотной памяти в указанную страницу
readBytes(len)		Проверяет состояние операции. Значение AAh в возвращаемом блоке указывает на завершение операции копирования
reset()		Сброс сети и возврат

Упрощенный способ обнаружения ошибок и повторения предустановленного числа циклов состоит в том, чтобы включить вышеупомянутые этапы в цикл и позволить потоку команд переходить в начало цикла каждый раз при возникновении ошибки. Этот метод прост в реализации, но имеет невысокую скорость

выполнения. Более тщательный метод обнаружения ошибок заключался бы в проверке на ошибку каждого вызова команды, при этом итерация выполнялась бы только при нарушающем работу вызове, когда происходит ошибка. Таблица, приведенная ниже, иллюстрирует упрощенный метод.

Команда	Поток данных	Примечания
<i>loop = 0</i>		
start:		
<i>loop = loop + 1</i>		
<i>do while loop <= loopLimit</i>		
<i>k = select(...)</i>	devAN	Выбирает <i>iButton</i> для обмена данными
<i>if (k <> 0) goto start</i>		
Erase Scratchpad	[W]: {C3h, TA1, TA2}	Сбрасывает флаг HIDE для последующего ввода/вывода
readBytes(len)	{байты состояния}	Читает достаточное число байтов состояния (например, len = 5), чтобы проверить, завершилась ли операция. Значение AAh указывает на завершение
<i>if(последний байт состояния не AAh) goto start</i>		
<i>s1 = reset()</i>		Сбрасывает предыдущую операцию
<i>s2 = resume()</i>		Возобновляет обмен данными с прибором
<i>if (s1 <> 0 или s2 <> 0) goto start</i>		
Write Scratchpad	[W] {0F, TA1, TA2, данные}	Записывает блок данных в блокнотную память
readBytes(2)	{инв. байты CRC-16}	Считывает инвертированный CRC-16 предыдущего потока данных. TCU должен вычислить свою версию CRC-16 для того же потока данных и сравнить со значением, считанным из прибора. Несовпадение указывает на ошибки ввода/вывода
<i>if (байты CRC не совпадают) goto start</i>		
<i>s1 = reset()</i>		
<i>s2 = resume()</i>		
<i>if (s1 <> 0 или s2 <> 0) goto start</i>		
Read Scratchpad	[W] AAh [R] {TA1, TA2, ES}	Считывает адресные регистры и регистр ES. TCU должен проверить, корректно ли их чтение
<i>if (TA1, TA2 и ES не совпадают) goto start</i>		
<i>reset()</i>		
<i>resume()</i>		
Copy Scratchpad	[W] {55h, TA1, TA2, ES}	Копирует данные блокнотной памяти в указанную страницу
readBytes(len)	{байты состояния}	Проверяет состояние операции. Значение AAh в возвращаемом блоке указывает на завершение операции копирования
<i>if (последний байт состояния <> AAh) goto start</i>		
<i>s1 = reset()</i>		Сбрасывает сеть
<i>end loop:</i>		
<i>if (loop > loopLimit)</i>		
<i>status = 1</i>		
<i>else</i>		
<i>status = 0</i>		

Стирание страницы данных (ERASEDATAPAGE)

Этот API стирает содержимое страницы данных, заполняя ее 32-мя байтами FFh.

```
int status = eraseDataPage(byte[] devAN, byte pageNum)
```

Переменная	Тип	Описание
<i>devAN</i>	byte[8]	номер адреса iButton
<i>pageNum</i>	byte	номер страницы
<i>status</i>	int	см. описание выше для возвращаемой переменной <i>status</i>

```
set data[32] = {32 байт FFh}
status = writeDataPage(devAN, pageNum, data)
```

Копирование кода MAC на страницу секретов (COPYMACTOSECRET)

Этот API копирует 8 байт результирующего кода MAC из блокнотной памяти в память по адресу секрета. Данная функция часто используется после команд вычисления первого секрета (Compute First Secret) и вычисления следующего секрета (Compute Next Secret) для копирования неполного кода MAC, размещенного в блокнотной памяти, в память секретов.

```
int status = copyMACtoSecret(byte[] devAN, byte secretNum)
```

Переменная	Тип	Описание
<i>devAN</i>	byte[8]	номер адреса iButton
<i>secretNum</i>	byte	номер секрета

```
TA1S = lowSecretAddress(secretNum)
TA2S = highSecretAddress(secretNum)
set tmp_data[32] = {32 00h bytes}
```

Команда	Поток данных	Примечания
select(devAN)		Выбирает iButton для обмена данными
Write Scratchpad	[W] {0Fh, TA1S, TA2S, tmp_data} [R] { инв. байты CRC-16}	Устанавливает флаги адреса для следующей операции копирования. Возвращаемые байты CRC используются для обнаружения ошибок
reset()		
resume()		
Read Scratchpad	[W] AAh [R] {TA1S,TA2S,ES}	Считывает адресные регистры и регистр E/S. TCU проверяет, корректны ли эти регистры
reset()		
resume()		
Copy Scratchpad	[W] {55h, TA1S, TA2S, ES}	Копирует секрет из блокнотной памяти в номер секрета <i>secretNum</i> . Заметим, что только 8 байт из блокнотной памяти копируются в память секретов
readBytes(len)		Считывает достаточное число байтов состояния для проверки, завершена ли операция копирования
reset()		

Установка секрета системы (INSTALLSYSTEMSECRET)

Установка секрета системы в SHA iButton выполняется и для сопроцессора, и для iButtons пользователя, и происходит во время установки обслуживания и проведения транзакций. Процессы установки секрета аутентификации системы и секрета подписи системы идентичны, за исключением того, что неполные фразы обычно отличаются, и что секрет подписи системы должен быть установлен в секрет 0 сопроцессора. Для этой цели в общем случае используется API *installSystemSecret*.

```
int status = installSystemSecret(byte[] devAN, byte pageNum, byte secretNum,
                                int numPartials, byte[] partial)
```

Переменная	Тип	Описание
<i>devAN</i>	byte[8]	номер адреса iButton
<i>pageNum</i>	byte	номер страницы данных прибора для записи неполных фраз
<i>secretNum</i>	byte	номер секрета для промежуточного и конечного секретов
<i>numPartials</i>	int	число неполных фраз
<i>partial</i>	byte[size]	матрица неполных фраз. Определена как одномерная матрица $size = 47 * numPartials$

TA1 = *lowAddress(pageNum)*

TA2 = *highAddress(pageNum)*

TA1S = *lowSecretAddress(secretNum)*

TA2S = *highSecretAddress(secretNum)*

Таблица 5. Дополнение неполной фразы (PAD_PARTIAL[J])

8 дополняющих байт	15 байт	9 дополняющих байт)
8 байт 00h	<i>partial[j][32:46]</i>	9 байт 00h

Команда	Поток данных	Примечания
Loop j = 0 to numPartials – 1		
записать partial[j] на страницу данных pageNum		
WriteDataPage(...)	devAN, pageNum, partial[j][0:31]	Записывает первые 32 байт partial[j] на страницу данных pageNum
записать дополненную неполную фразу (pad_partial) в блокнот		
resume()		
Write Scratchpad	[W] {0Fh, TA1, TA2, pad_partial[j]} [R] {инв. байты CRC-16}	Записывает дополненный код данных привязки в блокнотную память. Возвращаемые инвертированные байты CRC используются для обнаружения ошибок
reset()		
resume()		
If (j = 0) ComputeFirstSecret else ComputeNextSecret	If (j = 0) [W]{33h,TA1,TA2,0F} else [W]{33h,TA1,TA2,F0} [R] {инв. байты CRC-16}	Вычислить код MAC (секрет) Запускает функцию вычисления первого или следующего секрета в зависимости от значения j. Неполный результат MAC размещается в блокнотной памяти
readBytes(len)		Считывает достаточное число байтов состояния (например, len = 5) для проверки, завершена ли операция. Значение AAh указывает на успешное завершение
reset()		
copyMACToSecret(...)	devAN, secretNum	Копирует 8 байт кода MAC в секрет назначения
reset()		
End of loop		

Привязка секрета к `IBUTTON` пользователя (`BINDSECRETTOIBUTTON`)

Уникальный секрет аутентификации прибора создается путем привязки секрета аутентификации системы к номеру страницы данных обслуживания и номеру адреса прибора.

```
int status = bindSecretToiButton(byte[] devAN, byte pageNum, byte secretNum,
byte[] bindData, byte uAuthDataPage, byte[] uAN)
```

Переменная	Тип	Описание
<code>devAN</code>	byte[8]	номер адреса прибора
<code>pageNum</code>	byte	номер страницы данных
<code>secretNum</code>	byte	номер секрета аутентификации
<code>bindData</code>	byte[39]	39-байтный блок данных привязки системы
<code>uAuthDataPage</code>	byte	номер страницы данных для данных привязки
<code>uAN</code>	byte[8]	номер адреса прибора пользователя

$TA1 = lowAddress(pageNum)$

$TA2 = highAddress(pageNum)$

$TA1S = lowSecretAddress(secretNum)$

$TA2S = highSecretAddress(secretNum)$

Таблица 6. Дополнение данных привязки (`PAD_BIND`)

8 дополняющих байт	4 байт	1 байт	7 байт	3 байт	9 дополняющих байт
8 байт 00h	<code>bindData[32:35]</code>	<code>uAuthDataPage</code>	<code>uAN[0:6]</code>	<code>bindData[36:38]</code>	9 байт 00h

Команда	Поток данных	Примечания
<code>writeDataPage(...)</code>	<code>devAN</code> , <code>pageNum</code> , <code>bindData[0:31]</code>	Записывает данные привязки <code>bindData[0:31]</code> на страницу <code>pageNum</code>
Запись дополненных данных привязки (<code>pad_bind</code>) в блокнотную память		
<code>resume()</code>		
<code>Write Scratchpad</code>	[W] {0Fh, TA1, TA2, <code>pad_bind</code> } [R] {байты CRC-16}	Записывает дополненные данные привязки в блокнотную память. TCU проверяет возвращаемый CRC-16 байтов состояния операции для обнаружения ошибок
<code>reset()</code>		
вычислить код MAC (секрет прибора)		
<code>resume()</code>		
<code>Compute Next Secret</code>	[W] {33h, TA1, TA2, F02} [R] {байты инвертированного CRC-16}	Запускает функцию вычисления следующего секрета. Неполный результат MAC сохраняется в блокнотной памяти для следующей операции копирования
<code>readBytes(len)</code>		Считывает достаточное число байтов состояния для проверки, завершилась ли операция
<code>reset()</code>		
<code>copyMACtoSecret(...)</code>	<code>devAN</code> , <code>secretNum</code>	Копирует секрет прибора из блокнотной памяти в указанный секрет

Создание сигнатуры данных обслуживания (CREATEDATASIGNATURE)

Этот API вычисляет сигнатуру для данных обслуживания, используя данные обслуживания, секрет подписи системы, номер страницы данных обслуживания и ее счетчик циклов записи, а также номер адреса `iButton` пользователя.

```
byte sig[] = createDataSignature(byte[] devAN, byte pageNum, byte secretNum, byte[] uData,
    byte[] signCode, byte uSignDataPage, byte[] uAN, byte[] uSignDataPageWCC)
```

В случае успеха вызов (функции) возвращает 20-байтный код MAC.

Параметры и данные, используемые для создания сигнатуры данных обслуживания, перечислены ниже:

Переменная	Тип	Описание
<code>devAN</code>	byte[8]	номер адреса <code>iButton</code> (сопроцессор)
<code>pageNum</code>	byte	номер страницы данных, на которую записываются данные обслуживания для вычисления сигнатуры
<code>secretNum</code>	byte	номер секрета, где хранится секрет подписи системы (должен быть 0)
<code>uData</code>	byte[32]	подписываемый блок данных пользователя
<code>signCode</code>	byte[3]	код системы, используемый при вычислении сигнатуры
<code>uSignDataPage</code>	byte	номер страницы данных обслуживания <code>iButton</code> пользователя, используемый для вычисления сигнатуры
<code>uAN</code>	byte[8]	номер адреса прибора пользователя, используемый для вычисления сигнатуры
<code>uSignDataPageWCC</code>	byte[4]	счетчик циклов записи страницы данных обслуживания (<code>uSignDataPage</code>) в <code>iButton</code> пользователя

$TA1 = lowAddress(pageNum)$

$TA2 = highAddress(pageNum)$

Для эффективности ввода/вывода, `signCode` и другие параметры обслуживания дополняются до 32-байтного блока и сразу записываются в блокнотную память.

Таблица 7. Дополнение данных подписи (PAD_SIGNCODE)

8 дополняющих байт	4 байт	7 байт	1 байт	3 байт	9 дополняющих байт
8 байт 00h	<code>uSignDataPageWCC + 1⁷</code>	<code>uAN[0:7]</code>	<code>uSignDataPage</code>	<code>signCode</code>	9 байт 00h

Команда	Поток данных	Примечания
<code>writeDataPage(...)</code>	<code>devAN, pageNum, uData</code>	Записывает <code>uData</code> на страницу <code>pageNum</code>
Записать дополненные данные подписи (pad_signCode) в блокнотную память		
<code>resume()</code>		
Write Scratchpad	[W] {0Fh, TA1, TA2, pad_signCode} [R] {инв. байты CRC-16}	Записать дополненные данные подписи (pad_signCode) в блокнотную память
<code>reset()</code>		
Вычислить сигнатуру		
<code>resume()</code>		
Sign Data Page	[W] {33h, TA1, TA2, C3} [R] {инв. байты CRC-16}	Запускает команду подписи страницы данных. Результирующий код MAC сохраняется в блокнотной памяти

⁷ WCC + 1 становится новым счетчиком циклов записи страницы данных обслуживания после того, как на нее были записаны подписанные данные обслуживания, поскольку каждая запись на страницу увеличивает значение счетчика цикла на 1 (на страницах 8...15).

readBytes(len)		Считывает достаточное число байтов состояния, чтобы проверить, завершена ли операция
reset()		
resume()		
Read Scratchpad	[W] AAh [R]{TA1S, TA2S, ES, данные блокнотной памяти и байты CRC}	Считывает адрес, регистры ES и данные блокнотной памяти. 20-байтный блок сигнатуры сохраняется, начиная со смещения 8 в блокнотной памяти
reset()		Сброс сети 1-Wire и возврат

Создание байтов запроса (CREATECHALLENGE)

Этот API возвращает 3-байтный запрос, который часто генерируется сопроцессором. Запрос имеет случайную природу в том смысле, что при вычислении SHA в качестве одного из входных параметров используется счетчик процессора SHA (счетчик увеличивает свое значение каждый раз, когда в приборе выполняется вычисление SHA, и пользователь не получит те же самые байты запроса дважды). Поскольку при вычислении SHA входные данные берутся со страницы данных и из блокнотной памяти, запрос можно было бы действительно сделать случайным, используя в качестве входных данных условия внешней среды (такие как температура, влажность, уровень шума, сила нажатия на считывающее устройство и т.д.). Вызов этого API производится по определенному значению счетчика процессора SHA или в ответ на любое содержимое в странице памяти и блокнота, чтобы обеспечить постоянно меняющийся код MAC.

byte[] challenge = createChallenge(byte[] devAN, byte pageNum)

Переменная	Тип	Описание
<i>devAN</i>	byte [8]	номер адреса iButton
<i>pageNum</i>	byte	номер страницы данных. В этой простой реализации используется только секрет, связанный с этой страницей

TA1 = lowAddress(pageNum)

TA2 = highAddress(pageNum)

Команда	Поток данных	Примечания
select(...)	devAN	Выбирает прибор для обмена данными
Erase Scratchpad	[W] {C3h, TA1, TA2}	Сбрасывает флаг HIDE, чтобы можно было прочитать данные блокнотной памяти
reset()		
resume()		
Compute Challenge	[W]{33h, TA1, TA2, CCh}	Запускает команду вычисления запроса
readBytes(2)	[R] {инв. байты CRC-16}	Считывает два байта инвертированного CRC-16 команды, TA1, TA2 и байт управления
readBytes(len)		Проверяет, завершено ли вычисление SHA. Значение AAh указывает на успех
reset()		
resume()		
Read Scratchpad	[R] {b0, b1, ..., b31}	Считывает 32 байт из блокнотной памяти. Результат вычисления SHA сохраняется между смещением 8 и 27. Берет любые три байта в качестве желаемого запроса (=challenge)
reset()		Сброс сети 1-Wire и возврат

Ответ на запрос аутентификации (ANSWERCHALLENGE)

Когда `iButton` пользователя получает запрос от местного хоста, он отвечает на это кодом MAC, вычисленным из выбранной страницы данных и своего секрета аутентификации прибора.

```
byte[] resp = answerChallenge(byte[] devAN, byte pageNum, byte[] challenge)
```

Вызов функции возвращает нуль, если произошла ошибка, а в случае успеха — следующие элементы данных, упакованные в одномерную матрицу байтов.

```
uData[32] = resp[0:31]           данные со страницы pageNum
pageWCC [4] = resp[32:35]       счетчик цикла записи страницы pageNum
uMAC[20] = resp[36:55]         ответный MAC-код аутентификации
```

```
TA1 = lowAddress(pageNum)
TA2 = highAddress(pageNum)
```

Переменная	Тип	Описание
<code>devAN</code>	byte[8]	номер адреса прибора
<code>pageNum</code>	byte	номер страницы данных. Заметим, что этот номер страницы должен использоваться в паре с номером секрета аутентификации прибора
<code>challenge</code>	byte[3]	байты запроса
<code>resp</code>	byte[56]	одномерная матрица данных, упакованная согласно приведенной выше таблице

Для эффективности ввода/вывода байты запроса часто дополняются до полного 32-байтного блока для записи в блокнотную память (`pad_challenge`):

Таблица 8. Дополнение байтов запроса (PAD_CHALLENGE)

20 дополняющих байт	3 байт	9 дополняющих байт
20 байт 00h	<code>challenge</code>	9 байт 00h

Команда	Поток данных	Примечания
Записать дополненные байты запроса (pad_challenge) в блокнотную память		
<code>select(...)</code>	<code>devAN</code>	Выбирает <code>iButton</code> пользователя для обмена данными
<code>Erase Scratchpad</code>	<code>[W]{C3h,TA1,TA2}</code>	Стирает блокнотную память, устанавливает прибор в состояние готовности для приема данных
<code>reset()</code>		
<code>resume()</code>		
<code>Write Scratchpad</code>	<code>[W]{0Fh, TA1, TA2, pad_challenge}</code>	Записывает дополненные данные запроса в блокнотную память
<code>reset()</code>		
Вычислить ответный код MAC		
<code>resume()</code>		
<code>Read Auth. Page</code>	<code>[W]{A5h, TA1, TA2}</code> <code>[R]{32 байта данных со страницы данных (uData), счетчика страницы данных (pageWCC), счетчика страницы секретов, инвертированный CRC-16 команды, адреса, данных и байтов счетчика}</code>	(1) Эта команда заставляет прибор пользователя вычислять код MAC, основываясь на своем секрете аутентификации, данных выбранной страницы и запросе из блокнотной памяти. (2) TCU должен считать содержимое страницы данных, счетчика страницы данных, счетчика страницы секретов и инвертированный CRC-16 команды, адреса, данных и байтов счетчика
<code>readBytes(len)</code>		Считывает байты состояния, чтобы проверить, завершена ли операция
<code>reset()</code>		
Считать ответный код MAC (uMAC)		

resume()		
Read Scratchpad	R: {TA1, TA2, ES, 32 байта данных блокнотной памяти} uMAC начинается со смещения 8 и заканчивается смещением 27	Считывает результат вычисления из блокнотной памяти. Заметим, что требуемый код MAC (20 байт) хранится, начиная со смещения 8
reset()		Сброс сети 1-Wire и возврат

Проверка ответа аутентификации (VERIFYAUTHRESPONSE)

Этот API вызывается, чтобы проверить, является ли ответ аутентификации `iButton` пользователя корректным. Эта функция должна вызываться после того, как секрет аутентификации прибора `iButton` пользователя вновь создан в секрете рабочей области сопроцессора.

```
int status = verifyAuthResponse(byte[] devAN, byte wkDataPage, byte[] uData, byte[] uAN, byte
uPageNum, byte[] uPageWCC, byte[] challenge, byte[] uMAC)
```

```
status = 0      ответный MAC-код пользователя (uMAC) достоверен
      1         ответ пользователя является недостоверным
     -1         произошла ошибка
```

<i>Переменная</i>	Тип	Описание
<i>devAN</i>	byte[8]	номер адреса запрашиваемого прибора
<i>wkDataPage</i>	byte	номер страницы данных рабочей области
<i>uData</i>	byte[32]	содержимое страницы данных прибора пользователя (страницы <i>uPageNum</i>), полученное при запросе прибора пользователя для ответа аутентификации
<i>uAN</i>	byte[8]	номер адреса прибора пользователя
<i>uPageNum</i>	byte	номер страницы данных аутентификации прибора пользователя
<i>uPageWCC</i>	byte[4]	счетчик циклов записи страницы данных аутентификации прибора пользователя
<i>challenge</i>	byte [3]	байты запроса, используемые при запросе <code>iButton</code> пользователя для получения MAC-кода аутентификации
<i>uMAC</i>	byte [20]	MAC-код аутентификации <code>iButton</code> пользователя

```
TA1W = lowAddress(wkDataPage)
```

```
TA2W = highAddress(wkDataPage)
```

Для эффективности ввода/вывода, запрос и другие параметры обслуживания и параметры прибора дополняются до полной 32-байтной страницы для записи в блокнотную память.

Таблица 9. Дополнение запроса и параметров обслуживания (PAD_AUTH)

8 дополняющих байт	4 байт	1 байт	7 байт	3 байт	9 дополняющих байт
8 байт 00h	<i>uPageWCC</i>	<i>uPageNum</i>	<i>uAN[0:6]</i>	<i>challenge</i>	9 байт 00h

Команда	Поток данных	Примечания
writeToDataPage(...)	devAN,wkDataPage, uData	Записать uData в страницу данных рабочей области
Записать параметры обслуживания и байты запроса в блокнотную память		
resume()		
Write Scratchpad	[W] {0FH, TA1W, TA2W, ad_auth} [R] {инв. байты CRC-16}	Записывает дополненные байты запроса и параметры обслуживания
reset()		
Вычислить ответный код MAC		
resume()		
Validate Data Page	[W] {33h,TA1W, TA2W,3Ch} [R] {инв. CRC-16}	Результаты вычисления размещаются в блокнотной памяти, между смещением 8 и 27
readBytes(len)		Считывает достаточное число байтов состояния (например len = 5), чтобы проверить, завершилась ли операция
reset()		
Проверить ответный MAC-код iButton пользователя		
resume()		
Match Scratchpad	[W] {3Ch, uMAC} [R] {инв. байты CRC-16, байты состояния}	Проверка последнего байта состояния: AAh = данные совпадают FFh = данные не совпадают
reset()		Сброс сети и возврат