Метки, так же как и смарт карты могут быть оснащены физическим каналом связи. Этот канал может быть использован для критичных функций или для *штампования*. Штампование – это процесс установки права собственности на неинициализированное устройство. Штампование метки, скорее всего, будет требовать некоторого времени, а также физического наличия метки. Плюс ко всему мы можем предположить, что метки будут содержать некоторую оптическую информацию, как-то: штрих коды или же обычные цифры. В своём предложении по защите Евро-банкнот, Жуелс и Паппу используют этот метод печатной информации как подкрепление к информации памяти метки.

Предполагается, что метки будут иметь некоторый механизм “отклика”, позволяющий обнаружить их присутствие. Подобное функционирование сродни работе меток класса 0 для электронного наблюдения за товарами (EAS). Кто угодно может окликнуть метку, она же отзовётся некоторым неуникальным сигналом. Также в метках будет реализована команда “деактивации” (“kill”). Деактивация метки будет медленным физическим процессом, походящим на штампование, после чего метка сразу же станет неработоспособной. Метка может быть деактивирована отсоединением антенны, коротким замыканием или подвержением мощному микроволновому излучению. Обобщённо наш проект базисной метки изображён на рисунке 4-3. Предложения, приведённые в главе 5, будут спроектированы с учётом этой спецификации.

- **Метка EPC 1 класса:** пассивное питание, ROM на 96 бит.
- **Радиус:** рабочий – 3м., прямой канал – 100м., обратный – 3м.
- **Антиколлизия:** Детерминистический, либо вероятностный алгоритм.
- **Быстродействие:** 100 операций чтения в секунду.
- **Количество тактов на операцию чтения:** 10 000.
- **Количество вентилей для обеспечения безопасности:** 200 – 2000.
- **Логические операции:** *чтение*, *оклик*.
- **Физические операции:** *штампование*, *деактивация*.

Рис. 4-3: Пример спецификации недорогой РЧ-метки.

Глава 5.

Предложения по защите.

Работая над ограничениями, указанными в главе 4, будем решать вопросы, поднятые в главе 3. Мы заключили, что метки уязвимы для атак на физическом уровне на аппаратную часть. В результате, основными проблемами являются активные атаки и подслушивание. Эти атаки могут быть направлены

против личной секретности, а так же против утечки важных данных инвентаря. Анализ трафика тоже представляет угрозу, потому как может быть направлен на вычисление местоположения людей, а так же на внутренние данные организации. Отказ в обслуживании так же является потенциально дорогим и разрушительным видом атак.

Разрешить проблему атак посредством активного опроса можно, ограничив доступ к чтению меток через введение контроля доступа. Оградиться от подслушивания можно, если убедиться, что данные меток не передаются через прямой канал. В разделе 5.1. представлен недорогой механизм контроля доступа, основывающийся на односторонних хеш-функциях. Этот механизм называют *хеш-замком*. В разделе 5.2. описан механизм хеш-замка, в который добавлена функция рандомизации, предотвращающая отслеживание метки. В разделе 5.3. определены необходимые свойства хеш-функции и развёрнуты различные подходы к построению недорогих хеш-функций, подходящих для хеш-замков. Раздел 5.4. предлагает два алгоритма антиколлизии методом обхода дерева, обеспечивающие хорошую защиту от подслушивания на большом расстоянии. Раздел 5.5. содержит несколько простых концепций, способных усилить защиту, в частности методом обнаружения и предотвращения атак типа “отказ в обслуживании”.

5.1. Хеш-замок.

Часто механизмы доступа основываются на шифровании с открытым ключом или на симметричном шифре, где требуется защищённое распространение секретного ключа. Освещаемым здесь РЧ-меткам для поддержки этих традиционных механизмов контроля доступа не хватает вычислительной мощности. Хеш-замки представляют собой простой механизм контроля доступа, основывающийся на односторонних хеш-функциях. Определение односторонней хеш-функции будет дано в разделе 5.3.1. Согласно схеме хеш-замка, каждая метка будет оснащена хеш-функцией. На практике будет достаточно аппаратно-оптимизированного криптографического хеша. Варианты проектов этих недорогих хеш-замков будут описаны в разделе 5.3. Каждая метка, в проект которой входит хеш-замок, будет иметь некоторый объём памяти, отведённый под временный *мета-ИН (Идентификационный Номер)*. Метки будут работать как в запертом, так и в открытом состоянии. Эти состояния можно произвольным образом определить для различных проектов меток. Предположим, что все функциональные возможности метки доступны для любого ближайшего считывателя при условии, что она открыта.

Владелец запирает метку, выбрав сначала произвольный ключ, а затем найдя от него хеш-функцию. Результат обозначается как *мета-ИН*, т.е. *мета-ИН = хеш(ключ)*. Затем владелец запишет мета-ИН в метку и переведёт её в закрытое состояние. Запись мета-ИН может происходить как посредством радиоканала, так и посредством физического канала для большей безопасности. После приёма мета-ИН, метка переключается в закрытое состояние. В закрытом состоянии метка будет отвечать на все запросы только собственным мета-ИН, при этом другие её возможности будут недоступны. После проведения этих процедур, владелец метки сохранит её ИН и мета-ИН в некоторой оконечной базе данных, проиндексированной по полю мета-ИН. Обобщённый вид этого протокола представлен на рисунке 5-1.

1. Считыватель R выбирает случайный *ключ* и вычисляет *мета-ИН* := *хеш(ключ)*.
2. R записывает *мета-ИН* в метку T.
3. T переходит в закрытое состояние.
4. R записывает пару (*мета-ИН, ключ*) в базу данных.

Рис. 5-1: Протокол запирания хеш-замка.

Для того чтобы открыть метку, владелец, прежде всего, опрашивает её, получает мета-ИН, затем, по этому номеру смотрит запись в оконечной базе данных и получает соответствующий ключ. После этого владелец передаёт ключ в метку, а последняя вычисляет хеш-функцию от полученного ключа и сверяет её со своим мета-ИН. Если значения совпадают, т.е. *хеш(ключ) = мета-ИН*, то метка отпирает сама себя и полный набор функций становится доступен для любого ридера в радиусе её вещания. Общий вид этого протокола проиллюстрирован на рисунках 5-2 и 5-3. Во избежание кражи незапертых меток, при чтении ИН их следует быстро отпереть, считать номер, затем быстро запереть снова.

1. Считыватель R опрашивает метку T и получает её мета-ИН.
2. R находит пару (*мета-ИН, ключ*) в оконечной базе данных.
3. R посылает *ключ* в T.
4. Если *хеш(ключ) = мета-ИН*, то T отпирается.

Рис. 5-2: Протокол отпирания хеш-замка.

Вследствие сложностей, связанных с инвертированием односторонней хеш-функции, эта схема не даёт нелегальным ридерам прочитать содержимое меток. По этой схеме можно определить попытки обмануть ридер, но их нельзя предотвратить. Злоумышленник может опросить метку, получить её мета-ИН, а затем передать законному ридеру этот ИН, проведя атаку на ответ. Законный ридер ответит ключом. Однако ридер может проверить содержимое метки (её ИН), сверив его с соответствующей мета-ИН записью в БД. Обнаруженное несоответствие, по крайней мере, предупредит ридер в том, что, возможно, была проведена атака.

Для реализации схемы хеш-замка нужно лишь внедрить хеш-функцию в метку и оперировать ключами в БД. В ближайшем будущем это может быть экономически выгодно, в этом могут помочь предложения из раздела 5.3. Хеш-замки можно использовать и для обеспечения доступа для нескольких лиц, или же для безопасности других возможностей меток, например доступа по записи. Тем не менее, запертые метки всё же можно относить к идентификаторам, потому как у них есть мета-ИН. Это позволяет сторонним пользователям создавать собственные БД и пользоваться преимуществами меток, при этом, не обязательно владея последними. К сожалению, по мета-ИН возможно отслеживание пользователей.

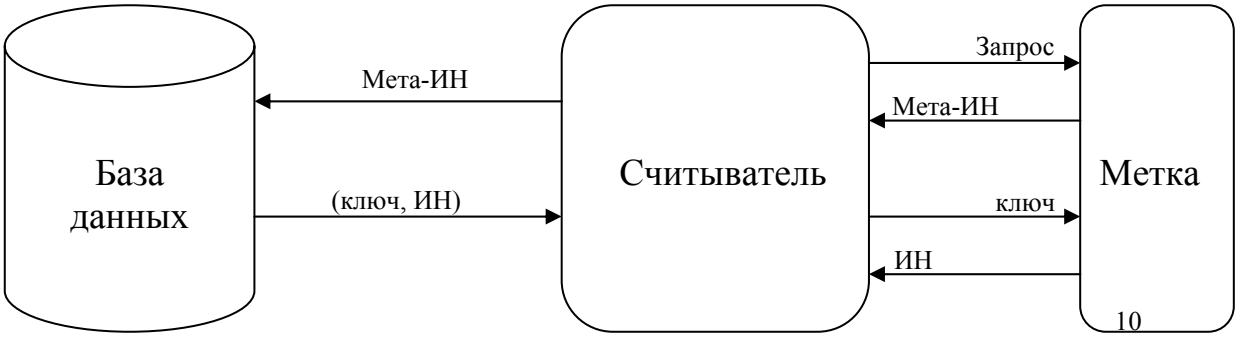


Рис. 5-3: Ридер отпирает запертую на хеш-замок метку.

5.2. Хеш-замок с рандомизацией.

Для предотвращения отслеживания меток требуется дополнительный режим работы. Находясь в этом режиме, метка должна непредсказуемым образом отвечать на запросы неавторизованных пользователей, но при этом должна идентифицироваться легальными ридерами. Мы представляем практический эвристический метод, основанный на односторонних хеш-функциях, лучше всего подходящих для покупателей с небольшим количеством меток. Так же мы предлагаем теоретически более сильный вариант, основанный на псевдослучайных функциях (PRF). Эта проблема в целом, так же как и другие похожие проблемы, будет подробнее раскрыта в разделе 7.1.

Как и в проекте раздела 5.1., метки будут оснащены односторонней хеш-функцией, однако теперь в них будет встроен ещё и генератор случайных чисел. Предположим, что легальный ридер будет иметь в базе данных ИН всех меток, с которыми он работает. Открытую метку можно запереть при помощи простой инструкции, при этом никакого протокола не нужно. Для того чтобы отпереть метку, считыватель сперва посылает простой запрос. В ответ на запрос, метки берут текущее сгенерированное по равномерному распределению число R. Затем метка хеширует конкатенацию этого числа и собственного ИН. Под конец метка посылает ридеру ответную посылку, состоящую из текущего случайного числа и результата хеш-функции (R, $h(ИН||R)$).

Когда легальный ридер принимает пару (R, $h(ИН||R)$), он предпринимает грубый поиск среди всех известных ему ИН, хеширует каждый из них в конкатенации с R и сверяет с полученным от метки значением. Вспомните, уже оговаривалось, что ридеры знают ИН меток, с которыми они работают. Если ридер находит соответствие, то он может отпереть метку, послав ей её ИН. Или же считыватель, уже зная ИН метки, может оставить её запертой. Этот протокол в обобщённом виде представлен на рисунке 5-4 и проиллюстрирован на рис. 5-5.

1. Считыватель R опрашивает метку T.
2. T генерирует текущее случайное число R и вычисляет $hash(ИН||R)$.
3. T посылает (R, $hash(ИН||R)$) на R.
4. R вычисляет $hash(ИН_i||R)$ для всех известных ему значений $ИН$.
5. Если R находит соответствие $hash(ИН_i||R) = hash(ИН||R)$, то он посылает $ИН$ метке.
6. Если принятый $ИН_i = ИН$, то метка отпирает себя.

Рис. 5-4: Протокол отключения хеш-замка с рандомизацией.

На практике, основанные на FPGA ядра SHA-1 имеют пропускную способность по данным примерно в 400 – 600 Мбит/сек при 512-битовых блоках, что составляет примерно 750 000 – 1 000 000 хеш-операций в секунду. Реализация на ASIC обеспечит большее быстродействие. В 1996 году сборочные версии кодов SHA-1, MD5 и RIPEMD работали на Пентиуме со скоростями 48.7, 113 и 82 Мбита/сек. соответственно. В 2003 году Пентиум,

работающий на частоте 1.6 ГГц, может выполнить 460 000 хеш-операций SHA-1 в секунду. В идеале хеш-функция должна требовать несколько вентилей, но много циклов при реализации в метке, но в то же время, немного циклов при реализации в ридере. На этом остановимся позже, в разделе 5.3.

Данная схема непрактична для владельцев большого количества меток, которым нужны скорости чтения порядка 100-200 меток в секунду. Однако это может оказаться полезным для владельцев небольшого числа меток. Потому как секретностью местонахождения более обеспокоены покупатели, чем магазины, последние могут применять обычный хеш-замок, вместо того, чтобы использовать его версию с рандомизацией.

Отдельным вопросом является то, как ИН метки попадает в БД легального ридера. При продаже продукта, его ИН должен прилагаться, иначе новый владелец не сможет прочитать метку. В разделе 5.5.6. раскрывается механизм, позволяющий новым владельцам получать доступ к своим меткам.

Хотя на практике эта схема достаточна, тем не менее, она не сильна теоретически. Формальное определение односторонней функции лишь устанавливает сложность инвертирования выходного значения функции. Но нельзя допустить утечки входных бит. Это подробно объясняется в разделе 5.3.1. Чтобы убедиться, что не произошло утечки битов ИН, следует использовать более сильный примитив.

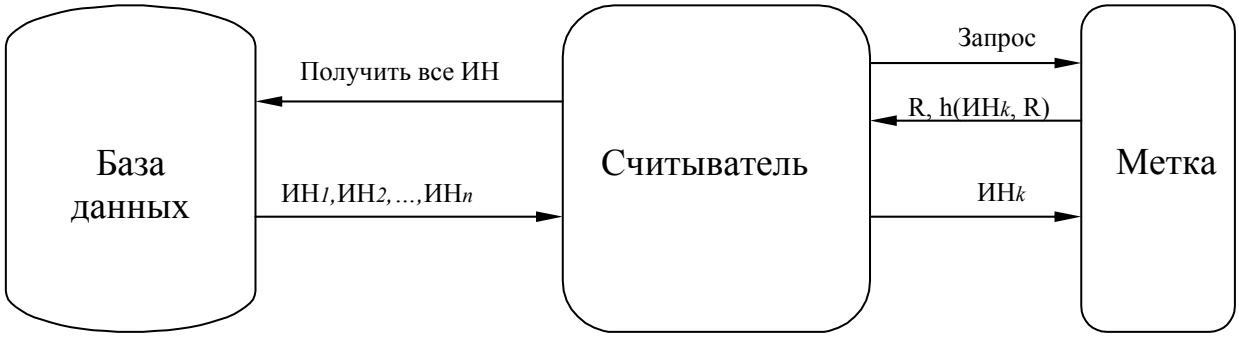


Рис. 5-5: Ридер отпирает метку с *k*-м ИН в схеме хеш-замка с рандомизацией.

Чтобы решить этот вопрос, допустим, что у каждой метки есть уникальный секретный ключ *k*. Его знает как метка, так и ридер. К тому же у метки имеется набор псевдослучайных функций, т.е. $U_{тн} = \sqrt{2P_N R_N}$. При опросе метки будут генерировать случайное число R и отвечать парой $\left(R, ИН \oplus f_k(R)\right)$. Затем ридер снова предпримет грубый поиск, используя все известные пары ИН/ключ, чтобы найти соответствие.

Небольшая поправка даёт возможность хранить в оконечной БД только лишь ключи метки, без необходимости хранить так же ИН метки. Метки могут прикрепить к своим ИН взятую от них хеш-функцию, а затем ответить последовательностью $\left(R, \left(ИН \,\|\, h(ИН)\right) \oplus f_k(R)\right)$. Для идентификации меток ридеры могут вычислять $f_k(R)$ для всех известных им ключей, затем складывать их по модулю 2 со второй частью ответа меток и проверять, подходит ли получившееся значение под форму $x \,\|\, h(x)$. Для ридера, не имеющего ключа, вещаемые меткой данные случайны и бессмысленны.

Остаётся неизвестным, можно ли наборы псевдослучайных функций реализовать с использованием значительно меньших ресурсов, чем при реализации симметричного шифрования. И, в контексте недорогих РЧ-меток, может не быть практической разницы. В проекте стиля Льюби-Ракоффа, многие симметричные алгоритмы шифрования используют ПСФ в качестве основного модуля ядра. Минимальная аппаратная сложность набора ПСФ остаётся открытой проблемой.

5.3. Недорогие хеш-функции.

Проекты из разделов 5.1. и 5.2. используют хеш-функции как фундаментальный модуль. Типичные коммерческие реализации стандартных хеш-функций, таких как SHA-1, требуют порядка 20 – 30 тыс. вентилей. Эта стоимость превышает доступные ресурсы всего проекта RFID. Как уже оговаривалось в разделе 4, мы предполагаем, что недорогие метки должны будут выполнять 100-200 операций чтения в секунду, и будут располагать 200 – 2 000 вентилей для целей безопасности. Многие коммерческие реализации хеш-функций оптимизированы под скорость, а не под количество вентилей. В этом смысле системы RFID допускают много большую гибкость. Здесь, под работы функций обеспечения безопасности, может быть выделено до 10 000 тактовых циклов. Вследствие некоторого изобилия тактовых циклов, при проектировании недорогих хеш-функций стоит пользоваться принципом *меньше вентилей, больше циклов*.

Всесторонний анализ криптографических хеш-функций можно найти в диссертации доктора Пренила. Бактиави, Сафави-Наини и Пиепрзик представляют общее исследование криптографических хеш-функций. В разделе 5.3.1. мы определим односторонние и устойчивые к коллизии хеш-функции. В разделе 5.3.2. представлены некоторые исторические подходы к проектированию, которые, тем не менее, нельзя использовать для недорогой RFID. В разделах 5.3.3. и 5.3.4. представлены два примера проектов недорогих функций RFID.

5.3.1. Определение хеша.

Определения, представленные в данном разделе в основном базируются на книге Менезиса, ван Орскота и Ванстоуна. Как минимум, хеш-функция h – это рационально исчислимая функция, производящая отображение из области значений произвольной длины в область значений фиксированной длины, т.е. $h:\{0,1\}^* \to \{0,1\}^n$. Под это широкое определение подходят многие тривиальные функции, поэтому от него мало пользы. Гораздо точнее определяют хеш-функцию следующие её свойства:

- *Необратимость* – для любого значения функции *y*, невозможно найти ни одно такое *x*, которое удовлетворяло бы уравнению *h(x) = y*, при условии, что не известно ни одно из соответствующих входных значений.
- *Необратимость второго рода* – при известном *x*, невозможно найти *x'* != *x*, такое, что *h(x) = h(x')*.
- *Устойчивость к коллизии* – невозможно найти такую пару входных значений *x* и *x'*, что *h(x) = h(x')*. Обратите внимание на свободу выбора обоих входных значений.

Односторонней хеш-функцией (ОСХФ) называют такую функцию, которая обладает свойствами необратимости первого и второго рода. Другими словами можно считать, что эту функцию “трудно инвертировать”. *Устойчивой к коллизии хеш-функцией* (УКХФ) называют функцию, обладающую свойствами необратимости второго рода и устойчивости к коллизии. Хоть это и необязательно, но на практике большинство УКХФ – тоже односторонние.

Хоть хеш-функции и трудно инвертировать, но нельзя сказать, что они обязательно прячут информацию. Например, пусть у нас есть односторонняя функция *h’*. Определим другую хеш-функцию $h(x||y) = h'(x)||y$. Из-за сложности инвертирования *h’*, *h* – так же трудно инвертировать. Однако ясно, что во второй части уравнения происходит утечка входного значения первой части. В контексте RFID, вспомним схему хеш-замка с рандомизацией из раздела 5.2. Согласно этой схеме, метка посылает пару (R, $h(R||ИН)$). Если данную здесь функцию *h* рассматривать как удовлетворяющую уравнению, приведённому выше, то это значение станет равным (R, $h'(R)||ИН$). Очевидно, что эта утечка информации сводит на нет всю полезность использования хеш-функции на первом месте. Теоретически, хеш-замок с рандомизацией должен основываться на псевдослучайной функции или на совершенной односторонней функции. Несмотря на теоретическую небезопасность, на практике эвристические хеш-функции значительно скрывают информацию.

5.3.2. Подходы к проектированию.

На практике были введены несколько различных подходов к построению хеш-функций. Многие из этих подходов недопустимо дороги для дешёвых меток. На практике используются два основных класса хешей, для которых существует теоретическое доказательство сложности обратимости. Это хеши, основанные на модульной арифметике или на ЭнПи-заполненности, а так же эвристические хеши. Это разделение аналогично разнице между работающими на открытом ключе и симметричными криптосистемами. Фактически для каждого из классов хешей существуют примеры, в основе которых лежат те же асимметричные и симметричные примитивы.

Хеши, разработанные теоретически, включают хеши, основанные на модульной арифметике, алгебраических матрицах и на “сложных” проблемах, как, например “задача о ранце”. Хеши, полученные из области модульной арифметики, основаны на сложности факторизации и нахождения дискретных логарифмов в полях Галоиса. В основе криптосистем RSA и Эль-Гамаля лежат, соответственно, те же сложности. Общий размер хешей, основанных на модульной математике зависит от размера самих модулей. Довольно крупные модули будут занимать намного больше доступного на РЧ-метке места. Операции модульной арифметики так же требуют интенсивных вычислений и не могут быть реализованы на том количестве вентилей, которое доступно в РЧ-метке. Возможность в ближайшем будущем использования в РЧ-метках шифрования методами колец, сеток и эллиптических кривых также представляется маловероятной.

Теоретически, защищённые хеши так же могут быть основаны на алгебраических матрицах. Например, пусть дана секретная матрица K размерности $n \times n$. Тогда хеш от сообщения M можно определить как

$H(M) = M^t KM$. К сожалению, хеширование сообщения длиной, скажем, 128 бит, потребует матрицу ключей размером примерно 512 байт. Операции с матрицами такого размера за гранью возможного для недорогих меток. Однако, небольшие матричные функции, такие как S-коробка или подхеш могут быть полезными фундаментами при проектировании хеш-функций.

Третий класс хешей основывается на трудности решения “задачи о ранце”. Суть этой задачи в следующем: пусть дан ряд целых значений $S = S_1, \ldots, S_k$, а так же целое n; найти некоторое подмножество $T \subseteq S$, такое, что $\sum_{t_i \in T} t_i = n$. Меркл и Хелман вначале использовали эту задачу в криптосистеме на открытом ключе. Позже, Харари, Дамгард и Земур предложили аддитивный и мультипликативный хеши, основанные на этой задаче. Схема Дамгарда была взломана Камионом и Патариным. Пренил указал на слабости в схеме Харари. Из-за больших требований по вычислительной мощности и размеру памяти, хеши, основанные на “задаче о ранце”, также за гранью возможностей недорогих РЧ-меток.

Эвристические хеши могут быть в виде специальных функций, или могут использовать блочные шифры, как фундаментальный модуль. С целью реализации хеш-функции были предложены многие схемы. В их число входят: N-хеш, “механизм интегрирования данных” ISO, а так же “криптографическая контрольная сумма” Лаи, Рупеля и Вуливена. Пренил, Говартс и Вандеваль исследовали 64 различных способа сборки хеш-функции из блочных шифров. Для 54 способов они продемонстрировали некоторые варианты атак. Позже, при помощи анализа методом чёрного ящика, Блэк, Рогавэй и Шримптон показали защищённость оставшихся 12. Потому как мы пришли к выводу, что реализовывать блочные шифры – слишком дорого, они не являются стоящим вариантом. Однако, в будущих проектах возможно более эффективное использование ресурсов, если блочный шифр будет использоваться как для хешей, так и для шифрования.

Обычно на практике используются функции, специально созданные, чтобы удовлетворять свойствам хешей. Существует семейство хеш-функций, носящих названия MD2, MD4 и MD5. Серьёзные ошибки были найдены в MD2. Доббертин показал, как при помощи обычного ПК можно в течение нескольких минут обнаружить коллизии в MD4. MD5 в общем считается защищённой, хотя метод Доббертина может быть расширен для обнаружения коллизий в компрессии функции MD5. Хоть эта атака и не найдёт коллизий в MD5, однако она может служить первым шагом на пути к этому. SHA-1, так же как и HAVAL и RIPEMD, - это ещё одна широко используемая специальная хеш-функция. В основном эти алгоритмы были оптимизированы под скорость, а так же под лёгкость программной реализации. Большинство цен на реализацию уменьшают ресурсы, доступные для RFID. В разделах 5.3.3. и 5.3.4. мы предлагаем два подхода к проектированию, которые могут быть приемлемы для недорогих систем.

5.3.3. Клеточные автоматы.

Клеточные автоматы (КА) – это конечные автоматы, переходы в которых зависят исключительно от ближайших “соседей” данного состояния. Научный труд на эту тему создал Вольфрам. Простейшие системы клеточных автоматов –

двоичные и одномерные. Каждое следующее состояние ячейки зависит от её текущего состояния и от текущих состояний соседних ячеек. Т.о. значение *i*-й ячейки в момент времени *t+1*, т.е. $c_{i,t+1}$, зависит от состояний $c_{i-1,t}, c_{i,t}, c_{i+1,t}$. Так как каждое из восьми возможных состояний имеет два возможных значения, всего существует 256 одномерных систем КА.

Многие из этих систем ведут себя предсказуемо. Однако некоторые из них проявляют “случайные” или “хаотические” свойства. Вольфрам проводил исследования, используя эти свойства в криптографии и при генерации случайных чисел. В частности Вольфрам провёл анализ отдельной ячейки КА, Правило #30, определённой как $c_{i,t+1} = c_{i-1,t} \oplus \left(c_{i,t} \vee c_{i+1,t}\right)$, реализованного в циклическом регистре.

Чтобы реализовать КА, в регистре, размером *n*, должно быть приблизительно *2n* логических вентилей. На рисунке 5-6 отображена схема регистра, реализовывающего Правило #30 на единичной ячейке. Рисунок 5-7 изображает временную эволюцию работы Правила #30 на циклическом регистре фиксированного размера в течение нескольких сотен итераций. Вначале регистр содержал единичную ячейку в состоянии “вкл.”.

Ранняя реализация хеша, основанного на КА, была представлена Дамгардом. Деймен, Говертс и Вандевель показали незащищённое место в схеме Дамгарда. Они предлагают собственный, основанный на КА, хеш, называемый “Хеш-ячейка” и ещё одну, улучшенную версию, под названием “Подхеш”. В то время как никакой информации о трещинах в схеме Хеш-ячейки опубликовано не было, Пренил всё же нашёл в ней несколько недостатков.

Не смотря ни на что, Хеш-ячейка и другие, основанные на КА, хеши могут служить подходящей парадигмой для недорогих РЧ-меток. Выгодно то, что метки могут уже содержать регистр, используемый для антиколлизии, и этот регистр также можно использовать для хеширования.

Одним из недостатков применения хешей КА в РЧ-метках является то, что при большом количестве параллельных вычислений может потребляться слишком много энергии. К счастью, КА можно случайным образом сериализовать в ущерб быстродействию. КА можно сделать даже из сдвигового регистра с обратной связью и одной пары вентилей. О более сложных сдвиговых регистрах с обратной связью поговорим в разделе 5.3.4.

Второй проблемой, связанной с хешами, основанными на КА, является то, что они обязательно требуют большого числа тактов работы. Явного пути уменьшения количества циклов не существует. Фактически Фольфрам предполагал, что количество вычислений, необходимых для работы КА, невозможно сократить – результат можно получить, лишь пройдя через каждую ступень вычислений. Ридер с хеш-замком на рандомизации, использующий хеш КА, должен отвести под счёт большое количество тактов для каждой из известных ему меток. Конечно же, это может быть неотъемлемой слабостью всех хеш-замков с рандомизацией. Другие потенциальные подходы изложены в разделе 7.1.

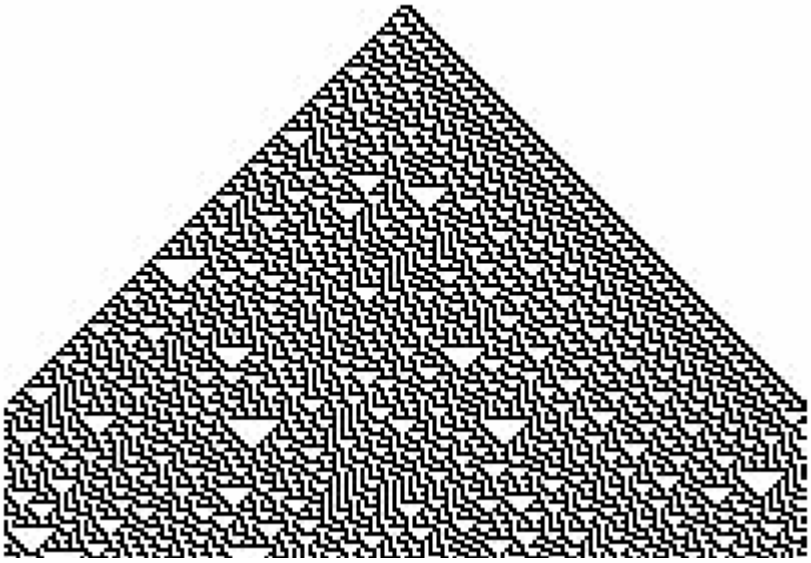


Рис. 5-7: пример “шумовой” выходной последовательности системы клеточного автомата.

5.3.4. Сдвиговые регистры с нелинейной обратной связью.

Сдвиговый регистр с обратной связью состоит из собственно сдвигового регистра и функции обратной связи. Некоторые разряды сдвигового регистра отводятся и подаются на входы функции обратной связи. Содержимое регистра сдвигается каждый раз, когда нужно получить выходной бит. Результат функции обратной связи – один бит – подаётся на вход регистра. Это проиллюстрировано на рис. 5-8.

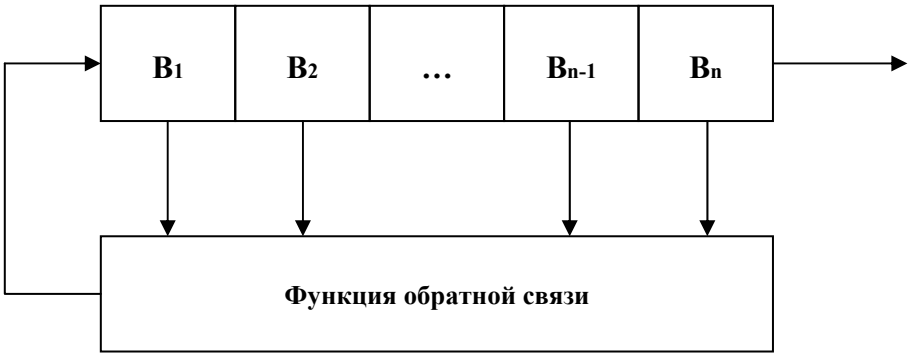


Рис. 5-8: Сдвиговый регистр с обратной связью.

Функция обратной связи может быть линейной и нелинейной. Соответствующие структуры носят названия Сдвигового Регистра с Линейной Обратной Связью (СРЛОС) и Сдвигового Регистра с Нелинейной Обратной Связью (СРНОС). Эрнст Селмер провёл раннюю работу, касающуюся математики, стоящей за СРЛОС. Криптограф национального агенства

безопасности, Соломан Коломб, соединил выводы Селмера со своими собственными и написал принесшую много плодов книгу.

Применяются также более сложные функции с обратной связью. Например, множество бит можно заменить *блоком подстановок* или *блоком перестановок*. Так же как и в случае с клеточными автоматами, недорогая хеш-функция будет основываться на простой функции обратной связи, прошедшей через множество итераций.

Более 50 лет назад Клодом Шенноном были представлены две полезные концепции в проектировании эвристических хешей. Эти концепции называются *конфузия* и *диффузия*. Первое из этих свойств, конфузия, заключается в том, что статистическое отношение между входными и выходными данными хеша должно быть слишком сложным, чтобы противник мог им воспользоваться. Свойство диффузии означает, что влияние одного разряда на входе распространяется на многие разряды на выходе. Другими словами, выходная последовательность должна скрывать характер входной.

В контексте проекта хеша, основанного на СРНОС, расчёт идёт на то, чтобы усложнённая функция обратной связи вызывала конфузию. Проведение множества итераций и правильное размещение точек ответвления поможет рассеять влияние входных разрядов. Мы предлагаем конструктивное решение, состоящее из небольшого блока перестановок, с точками ответвления, расположенными в разрядах, номера которых представляют геометрическую прогрессию с начальным значением 1 и множителем 2. Сей блок перестановок будет подключен к рабочему регистру, размер которого в два раза превышает конечную выходную последовательность хеша. После итерации в течение нескольких циклов, половина рабочего регистра сбрасывается. Оставшаяся половина – и есть результат хеш-функции. Этот проект приведён на рис. 5-9.

Рис. 5-9 изображает хеш на СРНОС и общим размером 128 разрядов. Здесь функция F – это 7-разрядовая перестановка. Во время одной итерации содержание разрядов 1, 2, 4, 8, 16, 32 и 64 ответвляется и подаётся на F. Результат перестановки записывается обратно в те же разряды. И, наконец, содержимое регистра сдвигается на одну позицию вправо. После довольно большого количества итераций вторая половина регистра не рассматривается, а первая служит выходным значением хеша.

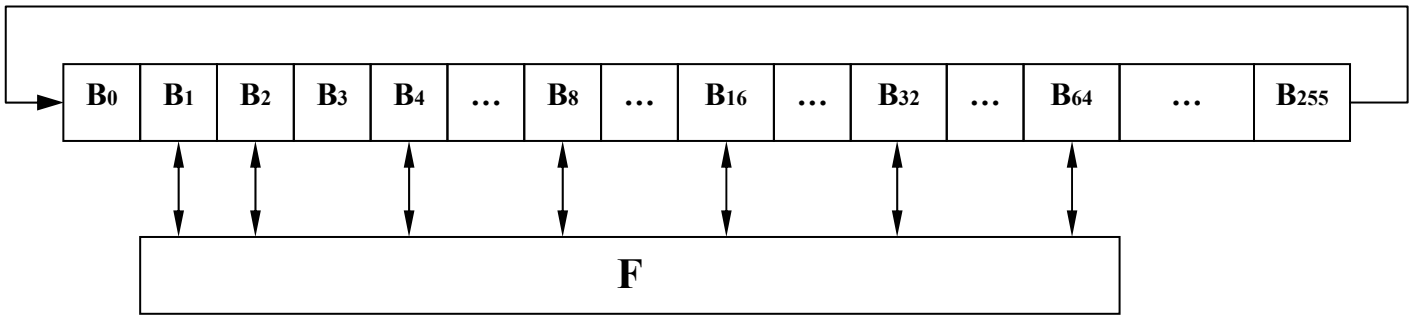


Рис. 5-9: Предлагаемая архитектура СРНОС.

Последним этапом перед получением результата хеш-функции является перестановка. Инвертирование блока перестановок повлечёт работу функции в обратном порядке. Отсюда следует, что информация не “теряется” во время итераций, что могло бы урезать область определения функции. Как минимум две точки ответвления в разрядах 1 и 2 гарантируют, что каждый бит на входе

влияет на каждый бит на выходе. Дополнительные точки ответвления предназначены для более быстрого рассеивания влияния каждого бита.

Выбор пал на семь точек ответвления исключительно из-за ограниченности ресурсов. Для перестановки 7 на 7 требуется таблица соответствия, размером 128 x 7, что составляет примерно 900 бит. Стандартизованная перестановка может быть зашита в метку относительно небольшой ценой. При условии, что каждая итерация занимает два цикла, то эта функция может быть вызвана около 5 000 раз кряду. При этом содержимое 256-разрядного регистра совершит примерно 20 полных оборотов. Это должно быстро рассеять каждый входной бит ровно по всей выходной последовательности.

Это пример одного из возможных недорогих проектов. Сама функция *F*, точки ответвления и размер регистра могут меняться. Нереальным может оказаться лишь одно допущение – это то, что метки будут содержать 256-разрядный сдвиговый регистр. Сегодняшние метки располагают лишь 96 разрядами памяти, доступной лишь для чтения. Однако уже разрабатываются стандарты для 128 и 256-разрядных меток ЭКП (Электронный Код Продукта - EPC).

Так же как и при использовании хешей на КА, при хеш-замках с рандомизацией, построенных на CPHOC, существует потенциальная проблема, которая заключается в том, что ридерам придётся вычислять хеш-функцию для каждой известной метки. В отличие от систем на КА, CPHOC можно подключать параллельно на стороне ридера, чтобы увеличить быстродействие. Мы предполагаем, что ридеры будут иметь по сравнению с метками намного более обширные ресурсы. Спараллеленные CPHOC представляют существенный компромисс между большим количеством вентилей и увеличением быстродействия.

5.4. Защищённая антиколлизия.

Алгоритм антиколлизии методом обхода бинарного дерева, уже обсуждавшийся в разделе 2.2.4., имеет врождённый порок безопасности вследствие асимметрии мощностей прямого и обратного каналов. Каждый бит всех выделяемых меток вещается ридером посредством прямого канала. На некоторых рабочих частотах при помощи расположенного на расстоянии до 100 м. от ридера подслушивающего устройства можно проследить эти передачи и получить содержимое каждой метки. В разделах 5.4.1. и 5.4.2. представлены два варианта защиты обычной схемы обхода дерева.

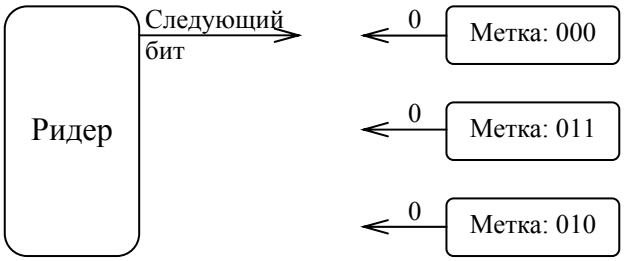
5.4.1. Обход дерева вслепую.

Серьёзной проблемой является довольно сильный сигнал прямого канала ридер-метка, обсуждавшийся ранее в разделе 4. В зависимости от рабочей частоты метки, можно прослушать этот канал в сотнях метров от источника и, возможно, просчитать содержимое метки. Отдельным вопросом является алгоритм антиколлизии методом обхода бинарного дерева, оговоренный в

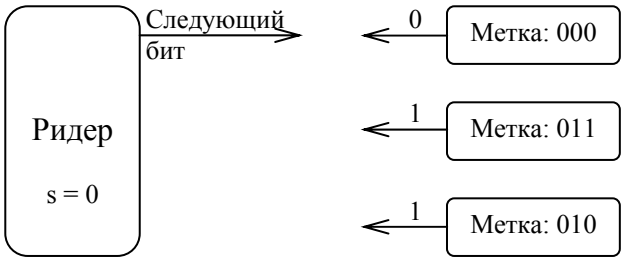
разделе 2.2.4. При работе этого алгоритма, ридер вещает каждый бит ИН метки, подвергающейся выделению, через “громкий” прямой канал.

Мы представляем вариант бинарного обхода дерева, при котором не происходит передачи незащищённого ИН метки по прямому каналу и не причиняется ущерба быстродействию. В начале своего существования эта схема имела название “Тихий Обход Дерева”. Предположим, что некоторая группа меток несёт в своём ИН определённый префикс, будь то: код продукта или ИН производителя. С целью выделения меток, ридер запрашивает их следующий бит. Если при этом не происходит коллизии, то это значит, что в данном разряде ИН меток совпадают.

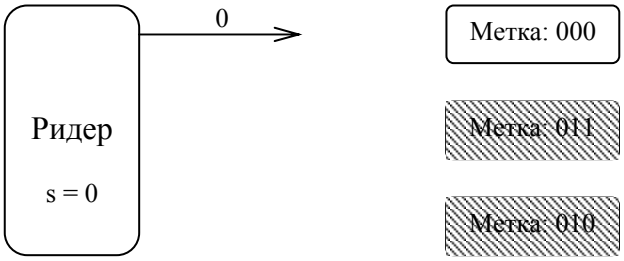
Отдалённое устройство сможет отслеживать лишь прямой канал и не будет слышать ответа меток. Таким образом, ридер и метки эффективно передают значение секретного бита. При возникновении коллизии ридер должен определить, с какой подгруппой меток продолжить работу. При отсутствии коллизии ридер просто запрашивает следующий бит.



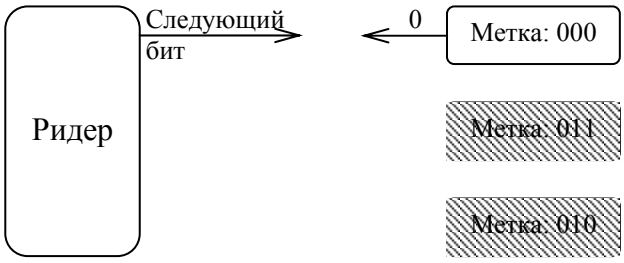
1. Ридер опрашивает первый бит. Коллизии не происходит.



2. Ридер записывает первый секретный бит и обнаруживает коллизию во втором бите.



3. Ридер выбирает метки с 0 во второй битовой позиции, посылая значение (0 XOR s). Остальные метки отключаются.



4. Ридер выделяет оставшиеся метки.

Рис. 5-10: Ридер выделяет Метку 000 при помощи алгоритма обхода дерева вслепую.

Поскольку мы предположили, что метки имеют некоторый общий префикс, ридер может получить его как общий секрет на обратном канале. Общий секретный префикс можно использовать для того, чтобы скрыть значение уникальной группы ИН. Пусть мы имеем две метки со значениями $b_1 b_2$ и $b_1 \overline{b_2}$. Ридер примет b_1 от обеих меток, без коллизии. На следующем же бите он обнаружит коллизию. Потому как для расположенного на удалении подслушивающего устройства, b_1 – неизвестен, то ридер для того, чтобы выделить метку, не открывая никаких бит, может отправить либо $b_1 \oplus b_2$, либо $b_1 \oplus \overline{b_2}$. На рис. 5-2 изображена работа ридера с двумя битами по алгоритму обхода дерева вслепую.

Очевидно, что при помощи подслушивающих устройств, расположенных в радиусе распространения обратного канала, можно получить весь ИН целиком. Тем не менее, немного усложнённая, схема обхода дерева вслепую эффективно предохраняет от сбора полезной информации прямого канала на большом расстоянии. По быстродействию данная схема схожа с обычным обходом дерева, потому как выделение метки совершается лишь тогда, когда она передаст целиком свой ИН посредством обратного канала.

5.4.2. Обход дерева с рандомизацией.

Алгоритм обхода дерева вслепую, приведённый в разделе 5.4.1., можно упростить при помощи рандомизации. Согласно Ривесту, основная идея метода обхода дерева с рандомизацией заключается в том, что при новом обходе дерева каждая метка генерирует временный случайный *псевдо-ИН*. Ридер будет работать по схеме обычного обхода дерева, но уже со значениями псевдо-ИН. В тот момент, когда подойдёт к концу выделение метки, она перешлёт свой настоящий ИН через обратный канал.

В этой схеме биты псевдо-ИН будут по обыкновению передаваться по прямому каналу. Здесь нет необходимости делать предположение о некотором общем префиксе меток или же о работе с секретными ключами на стороне ридера. Рандомизация вводится в ущерб быстродействию вследствие дополнительных затрат на линии связи, что мы обсудим далее в этом разделе. Быстродействие можно обменять на небольшой изъян в безопасности.

Чтобы не пропустить ни одной метки в радиусе действия, ридер должен поддерживать их питание до тех пор, пока все их не прочтёт. Существенным моментом является то, что метки будут менять свой псевдо-ИН при каждом новом обходе дерева. В тот момент, когда питание будет выключено, метки сразу же забудут свои псевдо-ИН.

Вообще биты псевдо-ИН можно генерировать налету. Ридер будет генерировать случайный бит и опрашивать по нему все метки. При обнаружении коллизии ридер подаст всем меткам с некоторым конкретным битом ИН команду уснуть. Метки должны запомнить ту битовую позицию, на которой их погрузили в сон. Приняв несколько бит без коллизии, ридер убедится, что метка была выделена. В наихудшем случае, когда метки генерируют одинаковые случайные биты, ридер будет всё же способен

обнаружить коллизию в обычном ИН. Вероятность этого может быть уменьшена ценой скорости коммуникации.

Алгоритм обхода ридера показан на рис. 5-11. Функция “Обход” имеет два аргумента: *i* – текущая битовая позиция и *count* – количество следующих друг за другом бит без коллизии. Когда величина *count* превысит некоторый априорный порог, ридер сделает вывод, что метка выделена и попытается прочитать её ИН.

Главным недостатком обхода дерева с рандомизацией является дополнительная коммуникационная стоимость, связанная с передачей псевдо-ИН. Вспомните, что как при обычном обходе дерева, так и при обходе дерева вслепую, происходит вещание лишь длины ИН метки. Ещё одним небольшим вопросом является то, что меткам потребуется бит состояния, чтобы отследить момент, когда они будут приостановлены (т.е. значение *i* на рис. 5-11).

Обход(*i, count*)
  b_i := Случайный бит *i*, прочитанный из всех активных меток.
  Если обнаружена коллизия в b_i, то:
    Приостановить все метки с $b_i == 1$.
    Каждая приостановленная метка сохраняет *i*.
    Обход(i+1, 0).
    Разбудить все метки, приостановленные на бите *i*.
    Обход(i+1, 0).
  Иначе Если в b_i коллизии не обнаружено, то:
    Если (*count* > порог) **Обойти** оставшиеся метки.
    Иначе Обход(i+1, count+1).

Рис. 5-11: Алгоритм Обхода Дерева с Рандомизацией.

Выбор длины псевдо-ИН зависит от общего количества меток, работающих с ридером. Пусть для группы из *n* меток используются псевдо-ИН длины *m* бит. Количество меток, случайным образом выбирающих некоторый конкретный псевдо-ИН, подпадает под распределение Пуассона. Возьмём $\lambda = \frac{n}{2^m}$. Тогда количество псевдо-ИН, имеющих *k* хозяинов, будет около $2^m e^{-\lambda} \frac{\lambda^k}{k!}$. Пусть $n = 2000$, а $m = 16$. Тогда $\lambda = 0.03$. Вероятное количество псевдо-ИН, имеющих *k* владельцев, приведено в таблице 5.1.

При использовании 96-битных ИН, передача 16-битных псевдо-ИН снижает быстродействие примерно на 17%. Отметим, что, так как коллизии среди псевдо-ИН разрешаются методом обычного обхода дерева, то при коллизии *k* меток при одинаковом псевдо-ИН, происходит утечка *k* бит информации собственно ИН. Для того чтобы скрыть часть этих данных, можно использовать Тихий Обход Дерева, хотя этот метод предполагает наличие некоторого общего префикса.

В примере, приведённом в таблице 5.1., 2000 меток с 96-битными ИН содержат примерно 192 000 бит данных. При возникновении коллизий, будет происходить утечка примерно 30 бит этой информации на каждый следующий обход дерева. Данная ситуация может быть приемлема во многих ситуациях, однако при большом числе обходов может накопиться порядочная утечка. Выбор длины псевдо-ИН и способа обработки коллизий среди псевдо-ИН будет

зависеть от требований по секретности и по быстродействию, предъявляемых будущими пользователями.

k	Количество псевдо-ИН, имеющих k хозяев	Комментарии
0	63599	Большая часть всех возможных псевдо-ИН не будет выбрана
1	1907	Большинство меток сгенерируют уникальный псевдо-ИН
2	28	Из-за одинакового псевдо-ИН возникнет коллизия нескольких пар меток
3	0.29	Очень редко будет возникать коллизия более чем двух меток

Таблица 5.1.: ожидаемое распределение 2000 меток над случайными 16-битными псевдо-ИН.

5.5. Другие предложения.

5.5.1. Соглашение по Асимметричному Ключу.

Преимущество асимметрии прямого и обратного каналов может быть использована ридерами для передачи такой ценной информации, как ключи. Пусть ридеру требуется передать значение v в выделяемую метку. Эта метка может сгенерировать случайное значение r в качестве одноразового блока и открыто передать его по обратному каналу. Теперь ридер может передать $v \oplus r$ по прямому каналу. В том случае, если подслушивающие устройства расположены вне зоны обратного канала, то они уловят лишь $v \oplus r$, где v – теоретически скрытая информация.

5.5.2. Засорение и Просеивание.

Ещё одним средством против прослушивания прямого канала является вещание ложных “сорных” команд от ридера, с целью запутать или разбавить информацию, собираемую подслушивающим устройством. При работе по некоторому общему секретному алгоритму, эти команды могут быть отфильтрованы или “просеяны” метками, использующими простой “MAC”.

5.5.3. Устройства Обнаружения.

В средах, где используется RFID, можно разместить несколько устройств обнаружения неавторизованных чтений или аномальных передач на рабочих частотах меток. Вследствие большой мощности прямого канала, задача обнаружения посторонних ридеров – довольно проста. Вмонтирование устройств обнаружения неавторизованного запроса и глушения в “умные полки” поможет обнаружить и идентифицировать атаки отказа в обслуживании в условиях розничной торговли.

5.5.4. Кричащие Метки.

Функциональные способности устройств обнаружения могут быть расширены до обнаружения ситуации вывода из строя меток. Сармой была предложена идея разработать метки, которые “кричат” при выводе последних из строя. Крик должен быть в виде сигнального импульса на некоторой частоте. Включение функции обнаружения крика в “умные полки” в дальнейшем поможет в обнаружении атак отказа в обслуживании.

5.5.5. Агенты Безопасности.

Устройства обнаружения можно встроить, как стандартные элементы, в ридеры, возможно даже в телефонные будки, или в КПК (Карманные Переносные Компьютеры). Законные считывающие устройства могут обнаруживать, протоколировать и фильтровать попытки чтения других ридеров. Ридер может действовать как “вышибала”, экранируя “плохие” запросы и пропуская “хорошие”. В сущности метки и ридеры могут представлять собой Беспроводную Персональную Сеть (Wireless Personal Area Network). В этом случае ридеры действовали бы как шлюзы между БПС и внешним миром.

Некоторое локальное устройство может даже имитировать содержимое расположенных рядом меток, полностью снимая необходимость опрашивать метки и увеличивая их радиус действия. Жуелс, Ривест и Зидлоу предлагают похожий подход. Их идея заключается в использовании “Блокирующей Метки”, с целью имитировать большое количество обычных РЧ-меток и эффективно блокировать неавторизованные ридеры.

5.5.6. Печать Главного Ключа.

Для того чтобы пользователи также могли располагать функциями меток, встроенных в приобретённые ими предметы, на упаковке продукта можно напечатать главный ключ, в виде штрих-кода или в виде десятичного номера. После приобретения продукта, потребитель может использовать главный ключ, чтобы переключить метку из режима хеш-замка из раздела 5.1. в режим с рандомизацией из раздела 5.2. Главный ключ также может функционировать как механизм восстановления ключа, когда пользователям нужно отпереть метки, к которым они потеряли ключи. Потому как главный ключ должен визуально считываться с внутренней стороны упаковки, злоумышленники не могут получить его, не обладая собственно упаковкой. Для большей безопасности все функции, использующие главный ключ, могут нуждаться в наличии физического канала связи с устройством чтения, а не радиочастотного.