Fault-атаки на алгоритм HMAC

Алексей Чиликов*

3 апреля 2010

Аннотация

В настоящей работе представлена атака на основе анализа сбоев (fault attack) на реализации алгоритмов HMAC и NMAC. Показано, что варианты алгоритмов, построенные на базе распространённых хэш-функций (SHA-1, MD5) уязвимы к данной атаке. Также обсуждаются возможные меры противодействия.

1 Введение

1.1 Метод анализа сбоев

Под атакой на основе анализа сбоев (fault attack) традиционно понимается активное воздействие противником на физическое устройство, реализующее тот или иной алгоритм. При этом определённым образом меняется ход исполнения алгоритма, в результате чего выходные данные становятся некорректными. Анализируя полученные выходные данные, противник получает возможность раскрыть секретную информацию, хранимую внутри устройства.

Атаки методом анализа сбоев были впервые предложены в работе [BDL97], применительно к алгоритму DES. В последующих работах обсуждался целый ряд различных моделей fault-атак как на симметричные – DES, AES, и т.д. ([BDL97, BS97, BS03]), так и на асимметричные – RSA, ECSM, и т.д. ([BMM00, Otto04, CT09]) алгоритмы. Также были приведены примеры конкретных методов физического воздействия на устройства, позволяющих реализовать некоторые из этих моделей (см., например [SA02]). Здесь мы не будем подробно останавливаться на физических аспектах атаки, а приведём лишь сокращённую классификацию математических моделей, следуя работе [Otto04].

Итак, атаки могут быть разделены:

- По степени контроля местоположения – на три класса: без контроля (в случае, когда местоположение повреждённых переменных неизвестно), со слабым контролем (когда известна повреждённая

*Passware, Research Department, chilikov@lostpassword.com

переменная) и с полным контролем (когда повреждаются заранее известные биты заданной переменной)

- По степени контроля времени – также на три класса: без контроля (в случае, когда время воздействия неизвестно), со слабым контролем (когда время воздействия известно с точностью до небольшого блока операций) и с полным контролем (когда время воздействия соответствует заранее заданной операции)
- По количеству повреждённых битов – один повреждённый бит, небольшое число повреждённых битов (например, один байт), произвольное число повреждённых битов заданной переменной

В данной работе будут рассмотрены атаки с точным контролем времени и слабым контролем местоположения. Различные варианты атаки будут иметь различные требования по количеству повреждённых битов.

1.2 Алгоритм HMAC

Алгоритм HMAC, описанный в [RFC2104], является широкораспространённым стандартом для выработки имитовставки (message authentication code, MAC). Он определяется следующим образом:

$$HMAC_K(M) = h((K \oplus \text{opad}) \,||\, h(K \oplus \text{ipad} \,||\, M)) \tag{1.1}$$

где opad и ipad – предопределённые константы, K – секретный ключ, а M – исходное сообщение. $||$ обозначает конкатенацию сообщений, а $\oplus$ – побитовое сложение аргументов. В качестве h может использоваться любая криптографическая хэш-функция (исходный стандарт рекомендует MD5, SHA-1 и RIPEMD-128/160).

Алгоритм HMAC является частным случаем алгоритма NMAC – менее распространённого, но в общем случае более стойкого. NMAC определяется следующим образом:

$$NMAC_{K_1,K_2}(M) = h((K_2) \,||\, h(K_1 \,||\, M)) \tag{1.2}$$

Будем обозначать через l длину полученной имитовставки (равную длине выхода используемой хэш-функции), а через b – длину блока хэш-функции.

1.3 Предыдущие работы

Алгоритмы вычисления имитовставок довольно часто реализуются аппаратно на различных устройствах (например, смарт-картах). Это делаемт их подходящим объектом для атак по побочным каналам, к числу которых относятся и fault-атаки. Отметим, например, работу [FLRV09], в которой рассмартивалась атака на HMAC при помощи техники electomagnetic template analysis (ETA).

Однако, насколько нам известно, fault-атаки на HMAC (и вообще на имитовставки) ранее не рассматривались.

2 Модель сбоев

Рассмотрим типичную реализацию алгоритма HMAC:

```
Constants: K - секретный ключ

Input:  M - сообщение
Output: H = HMAC( K, M )

 K1 = K XOR ipad    // produce inner key
 K2 = K XOR opad    // produce outer key

 H1 = h( K1 + M )   // "+" means concatenation
 H2 = h( K2 + H1 )  // "+" means concatenation

 return H2
```

Зафиксируем также реализацию хэш-функции h:

```
Constants: IV - начальные значение регистров

Input:  M - сообщение
Output: h( M )

 l = bitLength( M )

 // append padding to the message
 nBlocks = ( l + 65 ) / blockSize
 M.appendBit( 1 )       // append stop bit
 for( i = 0; i < ( nBlocks * blockSize - 64 ); i++ )
  M.addBit( 0 )         // append zero bits
 M.appendQword( l )     // append 64-bit message size

 // initialize accumulator
 for( i = 0; i < nRegisters; i++ )
  Acc[ i ] = IV[ i ]

 // update accumulator
 for( t = 0; t < nBlocks; t++ )
  for( i = 0; i < nRegisters; i++ )
   Regs[ i ] = Acc[ i ]    // copy registers
  Acc = U( Acc, getBlock( M, t ) )
  for( i = 0; i < nRegisters; i++ )
   Acc[ i ] += Regs[ i ]   // add 32-bit registers

 // return accumulator
 return Acc
```

Этот алгоритм универсален для всех рассматриваемых хэш-функций, отличаться будет лишь используемая функция U.

Предположим, рассмотренный алгоритм вычисления HMAC реализован аппаратно в некотором устройстве. Секретный ключ при этом содержится в защищённой памяти. Предположим, что противник имеет возможность модифицировать содержимое регистров, хранящих промежуточные переменные (например, внутреннее состояние аккумулятора в функции хэширования). Будем также считать, что время воздействия и изменяемая переменная могут быть выбраны точно.

3 Схема атаки

3.1 Простой вариант – сбой в одном бите

Начнём с рассмотрения простейшего случая, когда атакующий имеет возможность внести ошибку ровно в один бит выбранной переменной (неизвестно, в какой именно). Это требование весьма непрактично, однако оно позволяет проиллюстрировать суть атаки. Впоследствии будут рассмотрены варианты атаки с ослабленными возможностями противника, более применимые на практике.

При вычислении HMAC функция h вызывается дважды. Рассмотрим второй вызов. На вход ему приходит конкатенация ключа K_2 и H_1 – выхода первого вызова. Несложно заметить, что общая длина сообщения составляет $b + l$. Таким образом, хэшируются ровно два блока, и ровно дважды вызывается функция U. На вход первому вызову U приходит начальное значение IV и K_2. Выход этого вызова обозначим через Acc_2. Обозначим также через IV_2' промежуточное состояние аккумулятора после обработки первого блока. На вход второму вызову U приходит это значение IV_2' и H_1. Его выход обозначим через Acc_2'.

Противник имеет возможность выбрать сообщения, используемые для атаки. Потребуем, чтобы эти сообщения были короткими (менее $b - 65$ битов). В этом случае в первом вызове h также обрабатываются ровно два блока. Как мы увидим вопоследствии, для атаки почти всегда достаточно одного-двух выбранных сообщений (например, однобайтных).

На вход первому вызову U приходит IV и K_1. Выход U обозначим через Acc_1. Промежуточное состояние аккумулятора после обработки первого блока обозначим через IV_1'. На вход второму вызову U приходит IV_1' и M. Его выход обозначим через Acc_1'.

3.1.1 Первая фаза – восстановление Acc_2' и IV_2'

На выходе второго вызова аккумулятор содержит некоторый набор значений $Acc_2'[i]$. Затем эти значения складываются как 32-битные числа со значениями $IV_2'[i]$ и формируется итоговое значение хэша. Оно же является итоговым значением HMAC, и может быть получено противником на выходе устройства.

Целью первой фазы атаки является восстановление значений $Acc'_2[i]$ после последнего вызова U. Для этого противник вносит однобитовое искажение в регистр, содержащий $Acc[i]$, непосредственно перед последним сложением. Несложно заметить, что ошибка в j-м бите приведёт к изменению $Acc'_2[i]$ на $\pm 2^j$, и на ту же самую величину изменится выходное значение. Следовательно, зная истинное выходное значение H и искажённое значение H' (для одного и того же исходного сообщения), противник может легко установить, в каком из регистров произошёл сбой, в каком именно бите, и каково было истинное значение этого бита (при значении 1 разность $H'[i] - H[i]$ равна -2^j, а для 0: $+2^j$).

Замечание 3.1 *Для упрощения вычислений мы будем всегда говорить лишь о числе удачных сбоев, т.е. тех, которые действительно привели к изменению бита. Неудачные сбои могут быть легко обнаружены и отброшены при анализе.*

Знак разности однозначно определяет исходное значение бита $Acc'_2[i]$ во всех случаях, кроме случая $j = 31$ (поскольку $2^{31} = -2^{31} \mod 2^{32}$). Таким образом, противник не может определить старший бит ни одного из регистров аккумулятора. На выходе этой фазы атаки реально получается $2^{l/32}$ возможных пар состояний аккумулятора (Acc'_2, IV'_2), отличающихся друг от друга инверсией старших битов регистров. Это приводит к некоторому росту сложности анализа на последующем этапе, что будет подробно рассмотрено ниже.

Оценим количество сбоев, необходимых на данной фазе атаки. Каждый из удачных сбоев даёт противнику достоверную информацию об одном из битов целевого регистра. Для раскрытия одного регистра необходимо получить 31 сбой в различных битах, а для раскрытия всего состояния аккумулятора за исключением старших битов – 124 или 155 сбоев в различных битах (для MD5 и SHA-1 соответственно). Разумеется, есть вероятность, что различные сбои будут воздействовать на повторяющиеся биты (и, следовательно, не дадут никакой новой информации для анализа). Предположим, что при сбое в заданном регистре любой из битов изменяется с равной вероятностью, и что различные сбои независимы между собой. Тогда вероятность того, что конкретный бит не изменится ни разу за N (удачных) испытаний, равна $(1 - 1/32)^N$. Вероятность того, что хотя бы один из битов не будет затронут в ходе N испытаний, не превосходит $31 \cdot (1 - 1/32)^N \approx 31e^{-N/32}$. Таким образом, для получения фиксированной вероятности успеха ε достаточно провести порядка $31 \ln(32/\varepsilon)$ сбоев в заданном регистре. Среднее же количество необходимых сбоев будет порядка $31 \ln 32 \approx 107,5$. Для восстановления всех $l/32$ регистров аккумулятора потребуется в среднем $l \ln 2$ сбоев (< 430 для MD5 и < 540 для SHA-1).

Зная значения $Acc'_2[i]$ и выходной хэш H, противник может легко вычислить значения $IV'_2[i]$ – промежуточного состояния аккумулятора после первого этапа. Эти значения, в свою очередь, равны сумме $Acc_2[i]$

и известных констант $IV[i]$. Таким образом, противнику теперь известно также и Acc_2.

Помимо этого, знание IV_2' и Acc_2' позволяет перейти к следующей фазе атаки – восстановлению внутреннего хэша H_1.

3.1.2 Вторая фаза – восстановление H_1

Для восстановления истинного значения H_1 потребуется несколько более детально рассмотреть функцию U. Эта функция различна для различных алгоритмов хэширования. Мы начнём с рассмотрения MD5, как наиболее простого случая.

Для начала заметим, что размер H_1 составляет 16 байт. Остальные 48 байт второго блока во внешнем вызове h заранее известны (так как известен общий размер сообщения в битах). Функция U в алгоритме MD5 обрабатывает блок как 16 последовательных 32-битных значений (DWORD). Первые 4 из них соответствуют искомому H_1, а остальные 12 – известны заранее.

Рассмотрим теперь отдельный раунд функции U для алгоритма MD5. Все они устроены по единому образцу:

```
A = A + M[ q[ r ] ] + T[ r ] + F( B, C, D ) // r - round index
A = A <<< s
A += B
```

Здесь (A, B, C, D) – регистры аккумулятора (порядок зависит от номера раунда), $M[i]$ – i-й DWORD входного блока, q, T – предопределённые таблицы, F – одна из четырёх предопределённых функций (в зависимости от раунда). Все сложения производятся по модулю 2^{32}, $X <<< s$ обозначает циклический сдвиг влево 32-разрядного X на s бит. Переменная r обозначает номер текущего раунда.

Несложно заметить, что если $M[q[r]]$ известно, то соответствующий раунд легко обратим (т.е., зная финальное состояние A, B, C, D можно однозначно установить начальное). В противном же случае, можно ожидать, что ошибка, внесённая в регистр A перед исполнением раунда, даст некоторую информацию о начальном содержимом A. Если при этом финальное содержимое известно (или легко вычислимо), то можно получить информацию о недостающем слагаемом $M[q[r]]$. Эти соображения позволяют эффективно определить первые 4 DWORD входного сообщения по аналогии с предыдущей фазой.

Рассмотрим алгоритм анализа более подробно. Имеет место следующая

Лемма 3.2 *Пусть известны значения IV_2' и $M[q[t]]$ для всех $t > r$. Тогда противник может (при обработке второго вызова U во внешнем вызове h) внести серию сбоев в любой из регистров аккумулятора до начала исполнения раунда r и на основе полученного выхода однозначно вычислить значения $(A_{r+1}, B_{r+1}, C_{r+1}, D_{r+1})$ после исполнения r-го раунда.*

▷ Действительно, раунд обратим, если известно значение $M[q[t]]$. Таким образом, все раунды, начиная с $r+1$, обратимы. Следовательно, для вычисления достаточно знать лишь состояние аккумулятора на выходе из последнего раунда (т.е. Acc'_2). Оно легко вычисляется по выходному значению и IV'_2 для заданного вычисления (с учётом внесённого сбоя). Выходное значение известно противнику. Значение же IV'_2 совпадает со значением для истинного вычисления, поскольку сбой вносится позже. А истинное значение уже определено на предыдущей фазе атаки. □

Замечание 3.3 *Те же рассуждения справедливы и в случае, когда сбой вообще не вносился. При этом нет необходимости производить какие-либо манипуляции с устройством – все операции производятся аналитически.*

Для раскрытия искомого значения $M[q[r]]$ можно применить тот же метод, что и в предыдущей фазе атаки. Действительно, противник может вычислить истинное значение аккумулятора $(A_{r+1}, B_{r+1}, C_{r+1}, D_{r+1})$ после исполнения раунда и модифицированное значение A'_{r+1}, полученное в результате сбоя (значения остальных регистров совпадают с истинными). Обращая последние два шага раунда (B и s известны), и вычитая известное значение $T[r] + F(B, C, D)$, противник узнаёт истинное значение суммы $A_r + M[q[r]]$ и значение изменённой суммы $A'_r + M[q[r]]$ (A'_r – значение, полученное в результате сбоя). Их разность равна $A'_r - A_r = \pm 2^j$, где j – индекс бита, изменённого в результате сбоя. Знак разности, как и в предыдущей фазе, однозначно определяет истинное содержимое j-го бита A_r.

Раскрыв истинное значение A_r и уже зная $A_r + M[q[r]]$, противник легко вычисляет искомое значение $M[q[r]]$.

Как и раньше, имеет место *проблема старшего бита*. Это приводит к росту числа вариантов (для MD5 4 старших бита в каждом DWORD останутся неопределёнными при помощи указанного сценария анализа). Кроме того, имеется 16 возможных вариантов IV'_2, которые должны быть проанализированы отдельно. Всё это увеличивает сложность аналитической части алгоритма (максимум в $16 \cdot 16 = 256$ раз). Однако же при этом не требуется увеличение количества сбоев, поскольку информация, полученная в результате конкретного сбоя позволяет раскрыть бит сообщения для каждого из возможных наборов IV'_2 и ранее определённых $M[q[r]]$. Количество сбоев, необходимое для раскрытия одного DWORD оценивается также, как и в предыущей фазе.

Теперь осталось лишь явно определить последовательность, в которой будут раскрываться значения $M[q[r]]$. Изначально известны все значения $M[i]$ при $4 \geq i \geq 15$. Первое из неизвестных значений ($M[2]$) участвует в 78-м раунде. В качестве изменяемого регистра выступает регистр номер 2. Таким образом, сбой вносится в регистр 2 перед 78-м раундом. После раскрытия $M[2]$ следующей целью будет $M[1]$, задействованный в 71-м

раунде (изменяемый регистр номер 1). Далее – $M[3]$ в раунде 69 (регистр 3) и наконец, $M[0]$ в раунде 64 (регистр 0).

Всего противнику необходимо раскрыть 4 DWORD. Все оценки аналогичны полученным для предыдущей фазы. Общее количество необходимых сбоев будет вновь порядка 430 (для MD5).

3.1.3 Решение проблемы старших битов

Вернёмся теперь к рассмотрению проблемы старшего бита. Как несложно заметить, на входе второй фазы на самом деле имелось не одно возможное значение IV_2' и Acc_2', а целых 16 взаимосвязанных пар таких значений. Разумеется, каждое из значений Acc_2' приводило при 'обратном ходе' к различным промежуточным состояниям аккумулятора. При анализе изменённого состояния, в процессе 'обратного хода' необходимо пользоваться некоторым заранее выбранным истинным состоянием Acc_2', и сравнивать полученное значение $A_r' + T[q[r]]$ со значением $A_r + T[q[r]]$, построенным для того же Acc_2'. При правильном выборе состояния Acc_2' указанные суммы будут различаться в точности на $\pm 2^j$. Поскольку при обратном ходе используются достаточно сложные функции, вероятность того, что при ложном выборе Acc_2' также будут различаться на $\pm 2^j$ очень мала. На практике, уже на этом этапе анализа весьма вероятно устранение всех ошибочных вариантов Acc_2' (и, соответственно, IV_2').

Однако даже в худшем из возможных случаев, когда каждый из 16 вариантов остаётся возможным, и при раскрытии очередого $M[q[r]]$ старший бит остаётся неопределённым – число возможных вариантов всё ещё не превосходит 256. В каждом из них однозначно определены IV_2', Acc_2' и блок сообщения. При этом при раскрытии блока сообщения значения IV_2' не используются вовсе. Соответственно, можно просто проверить выполнение соотношения $U(IV_2', M) = Acc_2'$. Поскольку первые раунды U содержат сложные функции, которые никак не использовались при анализе, вероятность выполнения этого соотношения при ошибочном выборе IV_2' крайне мала. Такой способ анализа очень прост в реализации, а по вычислительной сложности сравним с хэшированием 256 блоков.

Действуя подобным образом, противник может раскрыть и все остававшиеся неопределёнными старшие биты.

На выходе данной фазы противник выявляет всё исходное сообщение, т.е. искомый H_1. Также теперь ему известны единственно верные значения IV_2' и Acc_2'.

3.1.4 Особенности второй фазы для SHA-1

Теперь рассмотрим, как данная фаза атаки может быть применена к алгоритму SHA-1.

Раундовая функиця SHA-1 устроена несколько иначе, чем в MD5. Однако она также обратима, что позволяет использовать для анализа похожую технику. Наиболее важной особенностью алгоритма SHA-1

является то, что на каждом из раундов обрабатывается не DWORD блока, а DWORD из *расширенной таблицы*. Эта таблица, в свою очередь, формируется из исходного блока сообщения. Таким образом, в этом случае атакующий не знает заранее ни одного из DWORD, задействованных в последних раундах (поскольку они зависят в том числе и от неизвестных ему DWORD сообщения). Следовательно, он должен применять сбои, начиная с последнего раунда, до тех пор, пока не соберёт достаточно информации для восстановления всех неизвестных DWORD.

Алгоритм формирования расширенной таблицы для SHA-1 выглядит так:

```
При 0 <= t <= 15
 w[t] = M[t]
При 16 <= t <= 79
 w[t] = ( w[t-3] ^ w[t-8] ^ w[t-14] ^ w[t-16] ) <<< 1
```

Очевидно, зависимость между битами расширенной таблицы и исходным сообщением является линейной, и восстановление сообщения по расширенной таблице не представляет никакой сложности.

Однако оценки сложности атаки зависят от того, сколько именно и каких сбоев необходимо внести противнику для получения достаточной информации. Поэтому имеет смысл разобрать получаемые зависимости более подробно.

Все DWORD, кроме первых 5 известны противнику. Таким образом, ему необходимо определить 160 бит. Пусть он действует по аналогии с ранее изложенным сценарием, считая неизвестными все значения $w[i]$ при $75 \leq i \leq 79$. В этом случае он последовательно раскрывает по 31 биту соответствующих $w[i]$, применяя серии сбоев перед соответствующими раундами. Неопределённый старший бит доопределяется обоими возможными значениями (т.е. число ветвей анализа после каждого рацнда растёт вдвое). С учётом этого доопределения противник получает в итоге систему из 160 линейных уравнений от 160 неизвестных бит. Такая система будет однозначно разрешима (что легко проверяется явно).

Вспоминая, что перед началом анализа противник имеет 32 возможных варианта пар (IV'_2, Acc'_2), получаем, что общее число ветвей анализа не превосходит $32 \cdot 32 = 1024$. Каждая из них приводит к некоторому варианту H_1. Истинный вариант может быть выбран из всех возможных применением техники из предыдущего пункта.

3.1.5 Третья фаза – восстановление Acc'_1 и IV'_1

Теперь, зная значение H_1, противник может приступить к восстановлению значений Acc'_1 и IV'_1. Для этого противник вносит сбой в один из регистров $Acc[i]$ при первом вызове h, во время обработки второго блока, непосредственно перед последним сложением с IV.

Несложно заметить, что изменение одного бита $Acc[i]$ приведёт к изменению выходного значения H_1 на $\pm 2^j$, где j – индекс изменившегося

бита, а знак однозначно определяется исходным значением этого бита в регистре $Acc[i]$. Таким образом, 63 значениявида $H_1[i] \pm 2^j$ исчерпывают все возможные варианты изменённых значений H_1, но реально среди них могут встретиться лишь 32.

Выходное значение имитовставки однозначно определяется по промежуточному состоянию аккумулятора IV_2' и обрабатываемому блоку H_1. Промежуточное состояние уже определено в первой фазе. Поэтому противник может заранее построить все возможные варианты изменённых значений H_2. В результате конкретного сбоя будет получено одно из этих значений, и противник может легко установить, какому именно из изменённых значений H_2 оно соответствует (путём поиска в заранее построенной таблице). Далее он может действовать по уже проверенной схеме – рассмотреть разность истинного и изменённого значения H_2 и на её основе выявить, какой именно бит изменился, и чему он был равен до изменения. Получив сбои во всех возможных битах, противник восстанавливает значения $Acc_1'[i]$, и вычитая их из известных $H_1[i]$, получает $IV_1'[i]$.

Теоретически возможно, что различным H_1 будут соотвествовать одинаковые H_2. Однако это означает, что будет найдена коллизия для функции сжатия U. Такое событие имеет пренебрежимо малую вероятность. Кроме того, эта проблема легко может быть легко устранена взятием другого исходного сообщения M. При этом изменится значение $H_1 = h(K_1 \mid\mid M)$, множество $H_1 \pm 2^j$ и множество соответствующих им H_2, но искомые значения Acc_1' и IV_1' останутся теми же, поскольку зависят только от K_1.

Как и в первой фазе атаки, возникает проблема старшего бита. На выходе данной фазы противник получает не единственный вариант значений IV_1' и Acc_1', а несколько вариантов ($2^{l/32}$), отличающихся инверсией старших битов регистров. Правильный вариант может быть установлен, например, при помощи *бесключевой эмуляции*, которая рассмотрена в следующем пункте. Отметим, что при этом не требуется внесения дополнительных сбоев в работу устройства.

Как и в первой фазе атаки, противнику требуется восстановить 4 или 5 регистров (в зависимости от алгоритма хэширования). Для восстановления одного DWORD требуется получить успешные сбои во всех битах. Поэтому справедливы те же самые оценки на количество необходимых успешных сбоев: ≈ 430 для MD5 и ≈ 540 для SHA-1.

3.1.6 Бесключевая эмуляция

Уже в этот момент, зная значения $Acc_1 = U(IV, K_1)$ и $Acc_2 = U(IV, K2)$, противник может решить основную свою задачу – вычислить HMAC для произвольного сообщения. Покажем это.

Имеет место следующая

Лемма 3.4 *Пусть произвольная хэш-функция h определена как*

$$h(M) = f(C(\dots C(C(\hat{IV}, M_0), M_1), \dots, M_{k-1}))$$

где $\mathbb{B} = \{0,1\}$, $C : \mathbb{B}^n \times \mathbb{B}^b \mapsto \mathbb{B}^n$ – произвольная функция сжатия, $f : \mathbb{B}^n \mapsto \mathbb{B}^l$ – произвольная функция усложнения, M – хэшируемое сообщение, k – его длина в блоках, $\hat{IV}$ – фиксированная константа (длины n). Тогда для вычисления $h(M)$ достаточно знать промежуточное состояние $IV' = C(\dots C(C(\hat{IV}, M_0), M_1), \dots, M_{m-1})$ и последующие блоки $M_m, \dots, M_{k-1}$.

Это довольно очевидное утверждение, поскольку первые блоки сообщения не участвуют явно в обработке последующих блоков, и всё их влияние заключается в изменении состояния промежуточного аккумулятора IV'. Если же его значение после обработки m блоков известно, то знание самих этих блоков $M_0, \dots, M_{m-1}$ для вычисления хэш-функции уже не требуется.

Опираясь на Лемму 3.4, легко показать возможность выработки имитовставки для произвольного сообщения M.

Во-первых, заметим, что все рассматриваемые хэш-функции удовлетворяют условиям Леммы 3.4. Для этого достаточно взять в качестве функции $C(IV, M)$ композицию $U(IV, M)$ и порегистрового сложения с IV. Функция f будет тождественной.

Далее рассмотрим внутренний вызов h. Ключ K_1 составляет в точности один блок хэшируемого сообщения. Он неизвестен, но известо значение IV_1' после его обработки. Все остальные блоки сообщения известны, что позволяет, в соответствии с Леммой 3.4, вычислить значение внутреннего хэша H_1.

Теперь рассмотрим внешний вызов h. Снова имеем неизвестный ключ K_2, размер которого равен размеру блока. Промежуточное состояние IV_2' после его обработки известно, а оставшийся фрагмент сообщения H_1 – вычислен. Опять же, в соответствии с Леммой 3.4, противник имеет возможность вычислить хэш H_2, который и будет искомой имитовставкой.

Бесключевая эмуляция требует точного знания параметров IV_1' и IV_2'. Однако предыдущая фаза атаки даёт $2^{l/32}$ возможных вариантов для IV_1'. Соответственно, применение указанной схемы приведёт к нескольким возможным вариантам имитовставки. Пусть при подаче на вход устройства некоторого сообщения M' (отличного от исходного M) получен некоторый результат. Он обязан совпадать с одним из возможных результатов бесключевой эмуляции для M'. То самое значение IV_1', для которого произошло это совпадение, и будет истинным. как правило, оно единственно. Таким образом, бесключевая эмуляция позволяет решить проблему старших битов для третьей фазы атаки. Заметим, что при этом не требуется внесения каких-либо дополнительных сбоев в работу устройства, а только обращение к нему в обычном режиме.

3.1.7 Четвёртая фаза – восстановление ключа

Итак, основная задача противника – построение имитовставки для произвольного сообщения – уже решена. Однако нет причин не попытаться

восстановить непосредственно ключевую информацию.

Для восстановления ключа K_2 противник может применить сбой при обработке первого блока во время вычисления внешнего хэша (второй вызов h). Поскольку уже известно истинное значение IV_2', а исходное значение IV является известной константой, то противник может легко вычислить Acc_2 (состояние аккумулятора после обработки всех раундов, но до последнего сложения). Далее схема атаки аналогична второй фазе, за исключением того, что теперь ни один из битов блока сообщения заранее неизвестен. Однако же можно предвычислить все возможные значения аккумулятора после сбоя в заданном раунде. Для последнего раунда их будет в точности 63 (поскольку состояние регистра до вычисления изменится не более чем 63 способами), и лишь 32 из них будут возникать при конкретном вычислении. На основе этих изменённых значений можно методом бесключевой эмуляции вычислить все возможные значения имитовставки. Значение, выданное устройством после сбоя, должно совпадать с одним из этих возможных вариантов. Путём поиска значения в таблице можно установить, какое именно изменение в регистре произошло и, как и раньше, установить значение сбойного бита. Единственным исключением будет старший бит, который не может быть установлен таким образом. Оба его значения должны рассматриваться как возможные, что приведёт к соответствующему ветвлению алгоритма.

Для предпоследнего раунда возможных значений будет уже 126 (с учётом неопределённости старшего бита на предыдущем шаге). Однако те из них, которые отвечают ошибочному выбору старшего бита, с большой вероятностью могут быть отброшены при анализе, поскольку соответствующая им разность истинного и изменённого состояния не будет равна $\pm 2^j$ (и следовательно, выданное устройством значение всегда будет лежать в 'правильной' половине таблицы). Таким образом, на данном шаге можно восстановить старший бит предыдущего DWORD, и 31 бит текущего DWORD (за исключением старшего бита).

Аналогичный сценарий может быть повторён и для всех оставшихся DWORD. Для полного вычисления блока достаточно (в случае MD5) рассмотреть 16 последних раундов. На выходе будут восстановлены 511 битов блока. Оставшийся неопределённым бит (старший бит DWORD, задействованного в последнем из рассмотренных раундов) может быть найден простым подбором.

Для алгоритма SHA-1 можно применить тот же сценарий анализа. Единственное различие заключается в том, что вместо непосредственно ключа будут раскрыты последние 16 DWORD расширенной таблицы. Однако, как несложно заметить, алгоритм построения расширенной таблицы позволяет однозначно вычислить по 16 последовательным DWORD не только следующий DWORD, но и предыдущий. Таким образом, зная последние 16 DWORD, противник может вычислить первые, которые и являются искомым ключом. Заметим, что при этом не требуется обращений к устройству.

Теперь оценим количество необходимых сбоев. Для полного раскрытия

ключа необходимо последовательно раскрыть 16 DWORD. Для раскрытия одного DWORD требуется получить успешные сбои во всех битах. Таким образом, средняя оценка количества сбоев для раскрытия одного DWORD та же, что и для первой фазы – порядка $107,5$. Это приводит к общей оценке – порядка 1720 успешных сбоев.

В случае алгоритма HMAC по восстановленному K_2 однозначно устанавливается исходный ключ $K = K_2 \oplus \text{opad}$ и внутренний ключ $K_1 = K \oplus \text{ipad}$. В случае же NMAC K_1 ключ необходимо вычислять отдельно.

Для вычисления K_1 можно использовать аналогичную процедуру. Бои при этом будут вноситься во время обработки первого блока при вычислении внутреннего хэша (первый вызов h). Как и раньше, противник начинает с последнего раунда. Сбой в одном бите приведёт к появлению одного из 63 возможных вариантов изменённого IV_1'. Противник вычисляет таблицу 63 возможных значений имитовставки, соответствующих этим сбоям, и ищет в ней значение, выданное устройством. На основе этого, он определяет, какой именно сбой произошёл, и раскрывает значение одного из битов обрабатываемого DWORD. Раскрываются все биты, кроме старшего. Затем аналогичная процедура применяется к предпоследнему раунду, что позволяет раскрыть недостающий старший бит предыдущего DWORD, и 31 бит текущего. Далее процедура повторяется для предшествующих раундов. После обработки 16 раундов раскрываются 511 битов блока сообщения (т.е. K_1). Последний бит ищется подбором. Все оценки для количества необходимых сбоев совпадают с оценками для K_2.

После завершения анализа противник восстанавливает секретный ключ целиком.

3.1.8 Оценка сложности атаки

Общее количество сбоев, необходимых для успешной атаки, зависит от конкретного алгоритма. Как мы видели выше, на каждой из первых трёх фаз атаки для хэш-функций MD5 необходимо раскрыть 4 DWORD. Раскрытие каждого из DWORD требует в среднем $\approx 31 \ln 32 \approx 107,5$ сбоев. Таким образом, средняя сложность трёхфазной атаки (дающей возможность бесключевой эмуляции) равна примерно $\approx 12 \cdot 31 \ln 32 \approx 1290$. Для SHA-1 необходимо раскрыть по 5 DWORD на каждой фазе, поэтому сложность увеличивается до $\approx 15 \cdot 31 \ln 32 \approx 1612$.

Для непосредственного восстановления ключей требуется провести четвёртую фазу атаки. Она требует восстановления 16 DWORD в случае алгоритма HMAC и 32 – для NMAC. Ожидаемая средняя сложность вычисляется аналогично и составляет $\approx 16 \cdot 31 \ln 32 \approx 1719$ для HMAC. Для алгоритма NMAC эта оценка будет удвоена.

3.2 Более практичная атака – сбой в одном байте

Очевидно, что предложенная модель однобитовых сбоев малопригодна на практике. Однако предложенная схема атаки может быть в ряде

случаев распространена и на более практичные модели. Например, если противник имеет возможность изменять целиком содержимое некоторого байта в одном из регистров аккумулятора. Такая модель сбоев вполне допустима в случае, когда алгоритм реализован в устройстве с 8-разрядной архитектурой.

Основными отличиями такой усложнённой атаки от рассмотренного выше варианта будет некоторое усложнение анализа разностей. Вместо очевидной формулы $X' - X = \pm 2^j$ будет иметь место следующая: $X' - X = \Delta \cdot 2^{8j}$, где $0 \leq j \leq 3$ и $\Delta \in [-X_j, 255 - X_j]$, X_j – истинное значение j-го разряда переменной. Как несложно заметить, в этом случае номер изменённого разряда очевиден, а истинное его значение может быть определено анализом границ разностей $X' - X$ для наиболее удалённых значений. При этом несколько растёт сложность атаки по числу сбоев (приблизительно в $256/8 = 32$ раза).

Что касается вычислительной сложности анализа, то здесь главной проблемой является *проблема старшего байта* – аналог рассмотренной выше проблемы старшего бита. Причина её всё та же – старший байт при анализе разностей не может быть определён. Однако масштаб проблемы значительно больше – неопределённость результата после первого этапа составляет уже 2^{32} для MD5 и 2^{40} для SHA-1. Однако множества возможных вариантов могут быть легко описаны аналитически, и некоторые более тонкие техники дальнейшего их анализа позволяют обойтись без столь масштабного перебора вариантов, сократив его до вполне практических значений (ценой ешё небольшого увеличения числа сбоев).

Детальное изложение атаки на основе однобайтовых сбоев будет опубликовано в расширенном варианте работы.

4 Выводы

Предложена новая fault-атака на реализации алгоритма HMAC. Показано, что алгоритм HMAC на базе широкораспространённых функций хэширования (SHA-1, MD5) является уязвимым к данной атаке.

Эффективность указанной атаки подтверждена компьютерным моделированием. Атака позволяет находить секретный ключ за реальное время на обычном персональном компьютере.

Сценарий атаки применим также для алгоритма NMAC. Варианты HMAC и NMAC, построенные на основе иных хэш-функций, также могут оказаться уязвимыми к данным атакам при выполнении следующих условий:

1. Хэш-функция использует схему Меркла-Дамгарда
2. Раундовая функция обратима
3. Финальной операцией в функции сжатия и раундовой функции является обратимая функция, действующая на регистры, и отличная от XOR

Этим условиям удовлетворяет большое число популярных хэш-функций.

Список литературы

[BDL97] *D. Boneh, R. A. DeMillo, and R. J. Lipton.*
On the Importance of Checking Cryptographic Protocols for Faults (extended abstract).
W. Funny, editor, Advances in Cryptology (EUROCRYPT 1997), volume 1233 of Lecture Notes in Computer Science, pages 37-51. Springer-Verlag, 1997.

[BDL01] *D. Boneh, R.A. DeMillo and R.J. Lipton.*
On the Importance of Eliminating Errors in Cryptographic Computations.
Journal of Cryptology, pp. 101-120, 2001.

[BMM00] *I. Biehl, B. Meyer, and V. Müller.*
Differential Fault Attacks on Ellitic Curve Cryptosystems.
M. Bellare, editor, Advances in Cryptology (CRYPTO 2000), volume 1880 of Lecture Notes in Computer Science, pages 131-146. Springer-Verlag, 2000.

[BS97] *Eli Biham and Adi Shamir.*
Differential fault analysis of secret key cryptosystems.
B. S. Kaliski, Jr., editor, Advances in Cryptology (CRYPTO 1997), volume 1294 of Lecture Notes in Computer Science, pages 513-525. Springer-Verlag, 1997.

[BS03] *J. Blomer and J. P. Seifert.*
Fault Based Cryptanalysis of the Advanced Encryption Standard (AES).
Financial Cryptography (FC 2003), volume 2742 of Lecture Notes in Computer Science, pages 162-181. Springer-Verlag, 2004.

[CT09] *A. Chilikov and O. Taraskin.*
New Fault Attack on Elliptic Curve Scalar Multiplication.
eprint.iacr.org/2009/528.

[FLRV09] *P-A. Fouque, G. Leurent, D. Réal, and F. Valette.*
Practical Electromagnetic Template Attack on HMAC.
C. Clavier, K. Gaj, editors, Cryptographic Hardware and Embedded Systems (CHES 2009), volume 5747 of Lecture Notes in Computer Science, pages 66-80. Springer-Verlag, 2009.

[Otto04] *M. Otto.*
Fault Attack and Countermeasures.
Dissertation, Facultät für Elektrotechnik, Informatik und Mathematik, Institut für Informatik, Universität Paderborn, 2004.

[RFC2104] *H. Krawczyk, M. Bellare, R. Canetti.*
RFC2104. HMAC: Keyed-Hashing for Message Authentication.
http://www.ietf.org/rfc/rfc2104.txt .

[SA02] *S. Skorobogatov and R. Anderson.*
Optical Fault Induction Attacks.
G. Goos, J. Hartmanis, J. van Leeuwen, editors, Cryptographic Hardware and Embedded Systems (CHES 2002), volume 2523 of Lecture Notes in Computer Science, pages 31-48. Springer-Verlag, 2003.