Fault-атаки на алгоритм HMAC

Чиликов Алексей Анатольевич,
Passware

Fault-атаки

- Объект атаки - Аппаратная реализация
- Сценарий атаки
  - Умышленный сбой в процессе вычислений
  - Анализ искажённых результатов
  - Раскрытие секретной информации (ключа)
- Компоненты
  - Математическая модель
  - Физическая модель
  - Практическая реализация

История вопроса

- Boneh, DeMillo, Lipton – атака на DES (EUROCRYPT’1997)
- Fault-атаки на симметричные алгоритмы
  - DES, AES, stream ciphers
- Fault-атаки на асимметричные алгоритмы
  - RSA, Elliptic Curves
- Физические модели
  - Skorobogatov, Anderson (CHES’2002)
- Практические реализации
  - Есть (например, CHES’2009, атака на DES)

Алгоритм HMAC

- Популярный международный стандарт для выработки имитовставки (RFC2104, 1997)
- Основан на хэш-функции (SHA-1, MD5, RIPEMD)
- Криптостойкий (для стойких хэшей)
- Часто реализуется аппаратно (смарт-карты)

=> Может быть целью для side-channel атак

Side-Channel атаки на HMAC

- Electromagnetic Template Analysis
  - Fouque, Leurent, Real, Valette (CHES'2009)

- Автору неизвестны примеры Fault-атак на HMAC (и вообще на имитовставки)

- Цель работы – построить математическую модель эффективной Fault-атаки на HMAC

Описание алгоритма HMAC

$$HMAC(K, M) = h((K \oplus O) \,||\, h((K \oplus I) \,||\, M))$$

- Вход: K - ключ, M – сообщение
  - I, O – предопределённые константы
  - длина ключа = длина блока хэш-функции h
  - хэш-функция вызывается дважды
- Выход: HMAC
  - длина HMAC = длина выхода хэш-функции h

Описание алгоритма HMAC

- HMAC – частный случай алгоритма NMAC

$$NMAC(K1, K2, M) = h(K2 \,||\, h(K1 \,||\, M))$$

- $HMAC(K, M) = NMAC(K^I, K^O, M)$

Предложенная атака работает и для NMAC

Описание алгоритма HMAC

- Хэш-функция h – ядро алгоритма
- RFC 2104 рекомендует использовать алгоритмы SHA-1, MD5, RIPEMD
- Но можно использовать и другие – SHA-256/512, кандидаты для SHA-3

Мы используем для демонстрации MD5

Но! Атака работает и для других хэшей (SHA-1)

Типичная реализация HMAC

```
HMAC(K,M):
  K1 = K ^ I;
  K2 = K ^ O;
  H1 = h( K1 || M );
  H2 = h( K2 || H1 );
  return H2;

h( M ):
  дополняем M до границы полного блока;
  делим M на блоки: M = M[1] || M[2] || … ||M[n];

  ( A, B, C, D ) = ( A0, B0, C0, D0 ) // начальное значение аккумулятора - константы
  для i от 1 до n:
   ( A’, B’, C’, D’ ) = F( A, B, C, D, M[i] ); // F – функция сжатия
   A = A+A’; B = B+B’; C = C+C’; D = D+D’;  // 32-битное сложение

  return ( A, B, C, D ); // возвращаем финальное состояние аккумулятора
```

Модель сбоев

- Противник имеет физический доступ к устройству
- Ключ хранится в защищённой области (недоступен для чтения)
- Противник имеет возможность внести сбой в заданную промежуточную переменную
- Противник имеет возможность точно выбрать время сбоя
- Сбой приводит к изменению 1 бита целевой переменной (неизвестно, какого именно)

Обработка данных

- Противник сам выбирает сообщение M – считаем, что оно состоит из 1 блока
- Хэш-функция h вызывается два раза
- Функция сжатия F вызывается четыре раза – по два раза внутри каждого вызова h
- При первом вызове F обрабатывает только ключ – результат не зависит от сообщения
- При втором вызове F обрабатывает только блок, зависящий от сообщения, но не биты ключа
- Результат второго вызова F зависит от ключа лишь опосредованно – через результат первого вызова

Divide and Conquer!

Поток данных внутри устройства

- (A, B, C, D, K1) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- (A, B, C, D, M) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- H1 = (A, B, C, D)
- (A, B, C, D, K2) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- (A, B, C, D, H1) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- H2 = (A, B, C, D)
- Return H2

Поток данных при сбое

- (A, B, C, D, K1) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- (A, B, C, D, M) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- H1 = (A, B, C, D)
- (A, B, C, D, K2) -> (A’, B’, C’, D’)
- A += A’; B += B’; C += C’; D += D’;
- (A, B, C, D, H1) -> (A’, B’, C’, D’)
- Сбой в 1 бите: A’ = A’ ^ (1<<x)
- A += A’; B += B’; C += C’; D += D’;
- Новое A отличается исходного на 1 бит
- A* = A + (2^x) если i-й бит A’ был равен 0
- A* = A - (2^x) если i-й бит A’ был равен 1
- H2 = (A, B, C, D)
- Return H2

Анализ

- По знаку разности определяются все биты (A’, B’, C’, D’), кроме старшего
- Количество сбоев – по одному на раскрываемый бит
- При анализе нет перебора
- По A’, B’, C’, D’ вычисляются A, B, C, D

Анализ

- Раскрытие (A, B, C, D) перед последним вызовом h
- Раскрытие сообщения H1
- Раскрытие (A, B, C, D) перед вторым вызовом h
- Раскрытие сообщения K1
- Раскрытие K

Анализ

- Сложность:
- ~ O(|K| + |H|) сбоев
- ~ O(|K| + |H|) вычислений F

Выводы

- Атака работает против MD5, SHA-1, других алгоритмов
- Атака работает против HMAC и NMAC
- Модель сбоев может быть расширена (сбой в 1 байте вместо 1 бита)

- Работа готовится к публикации

Вопросы?

- chilikov@lostpassword.com