УДК

Д. Ю. Боборухин

Атаки по времени и методы защиты от них

Любой алгоритм, имеющий зависимость по времени между исходным текстом и ключом, содержит уязвимость. Связав время работы программы со значением ключа, криптоаналитик получит инструмент для атаки. В результате будет обнаружен ключ либо создан алгоритм, позволяющий дешифровать любое зашифрованное сообщение. Атаки по времени представляют собой значительную угрозу безопасности криптографических систем. Понимание принципов, лежащих в основе атак по времени, и реализация соответствующих мер противодействия — важные шаги в защите конфиденциальной информации от злоумышленников.

Ключевые слова: атаки по времени, RSA, постоянное время исполнения, метод ослепления, криптоанализ

1. Введение

П. Кочер [1] первым заговорил об атаках по времени в 1996 году на конференциях "RSA Data Security"и "CRYPTO". Атака по времени — это атака по побочным каналам, в которой атакующий пытается скомпрометировать криптосистему с помощью анализа времени, затрачиваемого на исполнение криптографических алгоритмов. Атаки через побочные каналы используют информацию о времени, энергопотреблении, электромагнитных излучениях или даже звуке для восстановления секретной информации о криптосистеме.

Каждая операция в криптографической системе исполняется какое-то число процессорных тактов и занимает время на исполнение, которое может отличаться в зависимости от входных данных или значения секретного параметра(ключа). Прежде всего, это связано с внутренними особенностями системы. Атаки по времени могут быть особенно разрушительными, поскольку для них не требуется физический доступ к целевой системе, их можно проводить удалённо по сети [3].

Это является действительно важной проблемой, так как атаке по времени исполнения подвержены реализации известных алгоритмов шифрования, например RSA [2, 12, 15], AES [13, 14], DES, RC5, IDEA и другие.

Чтобы предотвратить атаки на время, разработчики должны применять методы безопасной разработки. Один из подходов заключается в обеспечении реализации криптографических систем в режиме постоянного времени исполнения [7]. Он подразумевает исключение зависимости времени от входных данных. Это устраняет уязвимости для атак по времени, которыми могут воспользоваться злоумышленники. Кроме того, введение случайных задержек или методов ослепления может помочь скрыть зависимость времени исполнения от входных данных.

2. Причины появления уязвимости

Зависимости времени работы алгоритма, или числа занимаемых процессорных тактов от входных данных может зависеть от нескольких факторов. [4]

- Прежде всего, это арифметические операции: некоторые арифметические операции могут занимать разное время в зависимости от операндов. Например, операции умножения или деления могут иметь разное время выполнения в зависимости от умножаемых или делимых значений.

- Условные ветвления [5]: если алгоритм содержит условные операторы, время выполнения может меняться в зависимости от исполняемого участка. Кроме того, предсказатель ветвлений часто создаёт потенциальную угрозу(Spectre угроза). [15] За счёт анализа спекулятивного исполнения и анализа по времени криптоаналитик получает возможность извлечь секретную информацию из системы.

- Оптимизация алгоритмов: некоторые криптографические оптимизации, ускоряющие выполнение операций для определенных входных значений могут хорошо работать лишь для некоторых параметров, что порождает расхождение во времени исполнения алгоритма.

- Шаблоны доступа к памяти: время доступа к памяти может меняться в зависимости от расположения данных. Если алгоритм при определённых входных параметрах обращается в основную память, а при других в кэш, это породит разницу во времени работы. Криптоаналитик может использовать эту вариацию для получения информации о шаблонах доступа к памяти, что, в свою очередь, может раскрыть секретную информацию.

3. Атака по времени на алгоритм RSA

RSA - это криптосистема с открытым ключом, которая использует открытую экспоненту e для шифрования и закрытую экспоненту d для дешифрования. Она использует модуль N, который представляет собой произведение двух больших простых чисел p и q, то есть $N = p * q$. Экспоненты e и d должны быть выбраны таким образом, чтобы удовлетворять условию $e * d = 1 mod (p-1)(q-1)$. Таким образом, ключевая пара RSA состоит из открытого ключа (N, e) и закрытого ключа d.

Сила RSA заключается в том, что факторизация больших чисел является сложной задачей. Самые известные методы факторизации до сих пор работают очень медленно. Мы можем предположить, что RSA надежно защищена от факторизационной атаки для ключа подходящей длины.

3.1. Алгоритм работы RSA

Чтобы зашифровать сообщение M, вычисляется $C = M^e mod N$, где C - зашифрованное сообщение, или шифротекст. Чтобы расшифровать шифротекст C, вычисляется $M = C^d mod N$, что дает исходное сообщение M. Расшифровка таким образом возможна благодаря результату из теории чисел, известному как теорема Эйлера.

Таким образом, расшифровка шифротекста представляет собой модульное экспонирование $M = C^d mod N$, где N - модуль RSA, C - шифротекст, а d - закрытый

ключ. Цель атакующего - найти d. Для атаки по времени атакующий должен заставить целевую систему вычислить $C^d mod N$ для нескольких тщательно выбранных значений C. Точно измерив необходимое время и проанализировав колебания времени, криптоаналитик может восстановить биты закрытого ключа. Атака по времени является проблемой обнаружения сигнала. Сигнал состоит из вариаций синхронизации, вызванных целевым битом экспоненты, а шум - из неточностей в измерениях синхронизации и случайных флуктуаций синхронизации, особенно в сети. В итоге, количество необходимых выборок синхронизации пропорционально количеству бит в экспоненте d, а также опрделяется свойствами сигнала и шума. Поскольку число бит в секретной экспоненте d ограничено, атака практична с вычислительной точки зрения.

```
x = C
for j = 1 to n
    x = mod(x^2, N)
    if d_j == 1 then
        x = mod(x*C, N)
    end if
next j
return x
```

Рис. 1. Алгоритм быстрого возведения в степень [8]

Модульное экспонирование в RSA состоит из возведения большого числа в большую экспоненту, что является трудоемкой операцией. Простым и эффективным алгоритмом вычисления $C^d mod N$ является алгоритм быстрого возведения в степень. Его алгоритм приведен на Рис. 1 , где $d = d_0 d_1 .. d_n$ в двоичном исчислении, причем $d_0 = 1$.

В стандартных реализациях RSA для выполнения операций умножения и возведения в квадрат используется алгоритм Монтгомери. При использовании алгоритма Монтгомери умножение занимает постоянное время, не зависящее от размера множителей. Однако если промежуточный результат умножения превышает модуль N, выполняется дополнительное вычитание [2]. Именно этот дополнительный шаг вызывает разницу во времени для разных входов и раскрывает информацию о секретном ключе.

Другим типичным способом оптимизации реализации RSA является использование китайской теоремы об остатке(KToO). При использовании KToO функция $M = C^d mod N$ вычисляется сначала путем вычисления $M_1 = C^{d_1} mod N$ и $M_2 = C^{d_2} mod N$, где d_1 и d_2 предварительно вычисляются из d. Затем M_1 и M_2 объединяются для получения M. RSA с KToO делает оригинальную атаку Кочера неработоспособной. Тем не менее, атака по времени может раскрыть множители N, а затем и сам секретный ключ [3].

3.2. Пример атаки на RSA

Данная атака использует уязвимость, порождённую наличием условных переходов в алгоритме. То есть, при определённых входных данных, выполняются разные участки кода.

Пусть C - сообщение для расшифровки, d - закрытый ключ для восстановления, обозначаемый $d_0 d_1 .. d_n$, где $d_0 = 1$.

```
if x >= N
    x = x % N
end if
return x
```

Рис. 2. Алгоритм взятия модуля [8]

Предположим, что для вычисления модульного экспонтирования используется алгоритм быстрого возведения в степень (Рис. 1), а операция $mod(x, N)$ реализована, как показано на Рис. 2.

Рассмотрим алгоритм на Рис. 1. Если если $d_j = 1$, то выполняются две операции, $x = mod(x^2, N)$ и $x = mod(x * C, N)$, вместро одной в противном случае. Другими словами, при $d_j = 1$ время вычислений должно быть больше. Также обратим внимание на Рис. 2, что модульное сокращение выполняется только в том случае, если промежуточный результат умножения больше модуля N. Данная атака использует эти два факта. Таким образом, осуществляется анализ двух наборов сообщений, один из которых требует сокращения для вычисления $mod(x * C, N)$, а другой — нет.

Предположим, что мы можем заставить систему расшифровать сообщения по нашему выбору. Начнем атаку на d_1 с выбора двух сообщений Y и Z, где $Y^3 < N$ и $Z^2 < N < Z^3$. Если d_1 действительно равно 1, то будет выполнена операция $x = mod(x * x^2, N)$ и время для расшифровки сообщения Z будет больше, поскольку модульное сокращение не выполняется для сообщения Y. С другой стороны, если реальное значение d_1 равно 0, операция $x = mod(x * x^2, N)$ не будет выполняться, и вычисления для Y и Z должны занимать примерно одинаковое время. Таким образом мы можем восстановить бит d_1, и продолжить атаку для следующих бит.
Так как в этом случае разница во времени очень мала, используют набор сообщений Y_i и Z_i для $i = 0, 1, .., m - 1$, удовлетворяющих начальным условиям. Пусть y_i - время, необходимое для вычисления $Y_i^d mod N$, z_i - время, необходимое для вычисления $Z_i^d mod N$. Вычисление среднего времени y и z по всем i позволяет сгладить шумы и заметить разницу даже на небольших отклонениях.

3.3. Реализации атаки на практике

Ж. Дем и Ф. Кёун провели аналогичную атаку на раннюю версию смарт-карты CASCADE [2]. Они сообщили о взломе 512-битного ключа за несколько минут с помощью 350 000 измерений времени. 128-битный ключ можно было взломать со скоростью 4 бит/с, используя 10 000 измерений.

Д. Брумли и Д. Боне разработали атаку по времени на OpenSSL, криптографическую библиотеку с открытым исходным кодом, используемую в веб-серверах и других SSL-приложениях, и успешно извлекли закрытый ключ d [3]. До того как эта атака была обнародована, считалось, что атаки по времени не сработают против серверов общего назначения, поскольку колебания времени будут маскировать время расшифровки криптографического алгоритма. Тем не менее атака сработала в локальной сети между машинами, разделенными несколькими маршрутизаторами. Для удалённого извлечения 1024-битного ключа с сервера OpenSSL 0.9.7 потребовалось около миллиона запросов, примерно за два часа [3].

Таким образом, в сетях с малым разбросом задержек, таких как локальные или корпоративные сети, атака по времени вполне осуществима. В случае удаленной атаки, криптоаналитик должен учитывать и другие переменные, помимо задержки в сети, например нагрузку на сервер. То есть, в периоды низкой активности сервер может быть более уязвим.

4. Методы защиты от атак по времени

Чтобы избежать вышеперечисленные уязвимости для атак по времени существует несколько методов защиты, наиболее распространёнными их них являются програмирование с постоянным временем исполнения и ослепление. Эти техники могут быть реализованны как на аппаратном уровне, так и на уровне програмного стека.

4.1. Программирование с постоянным временем исполнения

При программировании с постоянным временем исполнения, производительность программы становится независимой от входных параметров [7].

Программирование с постоянным временем исполнения подразумевает использование принципов, исключающих перечисленные выше причины появления уязвимости. Существует множество реализаций обхода потенциально опасных конструкций, мы перечислим основные из них, относящиеся к програмным решениям.

Чтобы арифметические операции выполнялись за постоянное время, независимо от операндов, можно использовать вычисления с фиксированной точкой. Для отказа от условных ветвлений часто применимы выражения с битовыми сдвигами, позволяющие получить тот же результат, но избежать зависимости от выбранной ветки. Единообразность шаблонов доступа к памяти может быть получена путём обращения ко всем элементам структуры данных по фиксированной схеме, независимо от фактических обрабатываемых данных.

Чтобы проиллюстрировать эти принципы, рассмотрим пример функции сравнения строк на языке C.

Потенциальная уязвимость

```
int strcmp(char *s1, char *s2) {
    while (*s1 && (*s1 == *s2)) {
        s1++;
        s2++;
    }
    return *(char *)s1 - *(char *)s2;
}
```

Данная реализация выходит из цикла, как только обнаруживает несоответствие, что приводит к изменению времени исполнения в зависимости от входных данных. Злоумышленник может использовать эти колебания времени для получения информации о сравниваемых строках.

Код с постоянным временем исполнения

```
int ct_strcmp(char *s1, char *s2) {
    unsigned char result = 0;
    while (*s1 && *s2) {
        result |= *s1 ^ *s2;
        s1++;
        s2++;
    }
    return result | (*s1 ^ *s2);
}
```

В данной реализации цикл выполняется на протяжении всей длины строк, а результат вычисляется с помощью побитовых операций, которые не вносят временных изменений в зависимости от входных значений.

В примере алгоритма RSA можно избавиться от уязвимости, всегда выполняя дополнительное сокращение в алгоритме Монтгомери [2]. Однако необходимо следить за тем, чтобы дополнительное сокращение не было оптимизировано компилятором.

4.2. Метод ослепления

Данный метод подразумевает введение случайности в алгоритм вычисления, чтобы устранить зависимость времени исполнения от входных данных. Часто это реализуется путём дополнительных вычислений со случайной величиной.

Для случая RSA также реализован метод ослепления, это приводит к накладным расходам от 2% до 10%. После работ Боне и Брумли [3], OpenSSL 0.9.7 включает в своей реализации ослепление по умолчанию.

5. Проблемы и ограничения методов защиты

Программирование с постоянным временем исполнения и метод ослепления являются мощными техниками для защиты от атак по времени, однако они не лишены своих проблем и ограничений [6].

- нагрузка на производительность: программирование с постоянным временем исполнения и использование методов ослепления может привести к накладным расходам на производительность, поскольку часто требует дополнительных операций. Это может быть компромиссом между безопасностью и производительностью, и он может не подходить для всех систем.

- сложность: написание алгоритмов с постоянным временем исполнения может быть более сложным и подверженным ошибкам, чем написание наивной реализации. Разработчики должны знать о потенциальных источниках временных отклонений и тщательно проектировать свой код, чтобы избежать их.

- верификация: проверка того, что код действительно работает в режиме постоянного времени, может оказаться непростой задачей. Требуется тщательный анализ и тестирование, чтобы убедиться в отсутствии скрытых вариаций синхронизации. Автоматизированные инструменты и формальные методы верификации в общем случае не являются безотказными.

- оптимизации компилятора: современные компиляторы могут вносить изменения во временные параметры за счет оптимизаций, которые упорядочивают или устраняют код. Для того чтобы скомпилированный код оставался безопасным, требуется тщательный контроль над настройками компилятора и иногда ручная проверка сгенерированного машинного кода.

6. Практические методы устранения уязвимостей для атак по времени исполнения

Реализация средств защиты от атак по времени представляет собой многогранную задачу, включающую в себя баланс между безопасностью, производительностью и сложностью системы. Для решения этой задачи требуется глубокое понимание как аппаратного обеспечения, так и программного стека. Ранее были рассмотрены программные методы устранения уязвимости, к примерам таких реализаций можно отнести OpenSSL и OpenSSL Constant-Time Implementations, Compiler-Based Mitigations в LLVM [11]. К примерам аппаратных решений можно отнести Intel Cache Allocation Technology (CAT) [9], ARM TrustZone [10].

Безопасность данных является важным направлением, которое постоянно развивается. Новые тенденции для защиты от атак по времени: применение машинного обучения для анализа по времени, гомоморфное шифрование, програмно-аппаратный дизайн, формальная верификация.

7. Заключение

Атаки через побочные каналы свидетельствуют о том, что даже если криптографический алгоритм математически силён, на практике он может оказаться небезопасен в зависимости от способа реализации и дизайна системы в целом. Проектирование безопасной системы должно охватывать все её аспекты, включая криптографические и некриптографические. Криптографические алгоритмы, основанные на модульном экспоненте, такие как RSA, могут быть уязвимы для атак по времени. Если операция экспоненцирования, в которой участвует секретный ключ, может быть засечена злоумышленником с достаточной точностью, ключ может быть восстановлен с помощью тщательно отобранных входных значений, количество которых пропорционально длине ключа. Существуют практические реализации атаки, поэтому она представляет настоящую важность. Защита от таких атак возможна и это является необходимостью для безопасности шифровальных систем.

Литература

1. *P.Kocher, https://paulkocher.com/doc/TimingAttacks.pdf* Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems
2. *J.F. Dhem, F. Koeune, P.-A. Leroux, J.-L. Willems, https://www.researchgate.net/publication/220962876* A practical implementation of the timing attack
3. *D.Brumley and D. Boneh, https://crypto.stanford.edu/ dabo/papers/ssl-timing.pdf* Remote Timing Attacks are Practical
4. *European Information Technologies Certification Academy* What is a timing attack?
5. *European Information Technologies Certification Academy* What role does the branch predictor play in CPU timing attacks, and how can attackers manipulate it to leak sensitive information?
6. *European Information Technologies Certification Academy* What are some of the challenges and trade-offs involved in implementing hardware and software mitigations against timing attacks while maintaining system performance?
7. *European Information Technologies Certification Academy* How can constant-time programming help mitigate the risk of timing attacks in cryptographic algorithms?
8. *Wing H. Wong, https://www.cs.sjsu.edu/faculty/stamp/students/article.html* Timing Attacks on RSA: Revealing Your Secrets through the Fourth Dimension
9. *Intel technical articles* Introduction to Cache Allocation Technology
10. *Arm technologies* Arm TrustZone
11. *llvm devmtg slides* Does LLVM implement security hardenings correctly?
12. *Yuval Yarom, Daniel Genkin, and Nadia Heninger* CacheBleed: A Timing Attack on OpenSSL Constant Time RSA

13. *National Institute of Standards and Technology* FIPS 197, Advanced Encryption Standard (AES)

14. *Daniel J. Bernstein, The University of Illinois at Chicago* Cache-timing attacks on AES

15. *European Information Technologies Certification Academy* CacheBleed: A Timing Attack on OpenSSL Constant Time RSA