## Атака Бляйхенбахера

Данная атака ориентированна на паддинг PKCS#1 1.5.

Атака Блайхенбахера основана на побочном канале, который мы будем называть padding oracle. Атака срабатывает всякий раз, когда при специально сформированом запросе к оракулу заполнения PKCS#1 v1.5 оракул сообщает, что расшифрованное сообщение соответствует схеме заполнения. Атака, по сути, использует случаи, когда первыми двумя байтами в посланном сообщении являются 0x00 и 0x02.

Итак, обозначим за $k$ длину модуля в битах, тогда сообщение вместе с рандомной частью будет иметь $B = 2^{k-16}$ возможных вариантов, при этом если учесть, что первые 2 байта сообщения это '0x00' и '0x02', то мы можем сказать, что любое сообщение+паддинг для модуля данной длины будет лежать в интервале $[2B, 3B-1]$.

Следующим нашим шагом будет поиск такого числа $s_1$, для которого сообщение

$$s_1^e * m_0^e = (s_1 * m_0)^e (mod\ N) = m_1^e (mod\ N)$$

Расшифруется с корректным паддингом (то есть система примет ваше сообщение $m_1$ как сообщение с паддингом PKCS#1 v1.5). Поиск данного числа начинается с какого-то фиксированного уровня (в нашем случае с $\lceil \frac{N}{3B} \rceil$) и просто инкрементируется, пока мы не получим нужное $s_1$, которое удовлетворяет условию выше.

Найдя подходящее $s_1$, мы можем уменьшить интервал для $m_0$, составив следующее равенство:

$$m_0 * s_1 = m_1 + r * N$$
$$m_0 = \frac{m_1 + r * N}{s_1},$$

где $r$ - неизвестно.

В итоге мы можем составить границы для $m_0$:

$$2 * B \leq m_1 \leq 3 * B - 1, \rightarrow$$
$$\frac{2 * B + rN}{s_1} \leq m_0 \leq \frac{3 * B - 1 + rN}{s_1}, \rightarrow [r = \frac{m_0 s_1 - m_1}{N}] \rightarrow$$
$$\frac{2Bs_1 - 3B + 1}{N} \leq r \leq \frac{(3B-1)s_1 - 2B}{N}$$

(мы не можем оперировать в конечном виде формулы значениями $m_0$ и $m_1$, так как мы их не знаем).

Теперь мы выбираем все возможные значения $r$, удовлетворяющие условию выше, и для каждого $r$ мы рассчитываем интервал (потому что мы не знаем, сколько раз было совершено приведение по модулю, поэтому рассмотрим все возможные случаи):

$$\frac{2B + rN}{s_1} \le m_0 \le \frac{3B - 1 + rN}{s_1}$$

Каждый получившийся интервал пересекаем с интервалом $[2B, 3B-1]$.

В результате мы получили несколько интервалов, в одном из которых должно содержаться исходное сообщение $m_0$. Для того, чтобы нам выяснить, где именно оно находится, мы находим следующее $s_i$ таким же способом, как и $s_1$, только начиная перебор со значения $s_{i-1}$.

После нахождения $s_i$ мы сможем уточнить информацию для каждого из интервалов, полученных ранее. Для значения $s_i$ верно утверждение (по аналогии с $s_1$):

$$\frac{2B + rN}{s_i} \le m_0 \le \frac{3B - 1 + rN}{s_i}$$

Теперь рассчитаем информацию о новых возможных $r$, используя интервалы, которые мы уже имеем (то есть теперь мы будем уточнять не интервал $[2B, 3B-1]$, а по очереди все интервалы $[a, b]$, полученные на предыдущем шаге):

Выведем формулу границ для $r$ на интервале $[a, b]$ из формулы:

$$\frac{2Bs_1 - 3B + 1}{N} \le r \le \frac{(3B-1)s_1 - 2B}{N},$$

заменяя $2Bs_i$ на $as_i$ и $(3B-1)s_i$ на $bs_i$.

$$\frac{as_i - 3B + 1}{N} \le r \le \frac{bs_i - 2B}{N}$$

Вычисленные новые интервалы для $m_0$ пересекаются с интервалом $[a, b]$.

Если в результате уточнения всех имеющихся интервалов мы получили больше одного промежутка, то мы находим новое $s_i$ и повторяем процедуру снова.

Для случая, когда у нас остался один промежуток $[a, b]$ для $m_0$, мы можем выполнить важную оптимизацию, которая делает атаку сходящейся очень быстро. Мы опишем её ниже.

### Бинарный поиск с одним оставшимся интервалом

Из формулы для новых интервалов для $m_0$ на шаге $i$

$$\frac{2B + rN}{s_i} \le m_0 \le \frac{3B - 1 + rN}{s_i}$$

мы непосредственно выводим

$$\frac{2B + rN}{m_0} \le s_i \le \frac{3B - 1 + rN}{m_0}$$

Поскольку $m_0$ находится в $[a, b]$ (у нас есть только один возможный интервал), мы получаем ограничение пространства поиска для $s_i$

$$\frac{2B + rN}{b} \le s_i \le \frac{3B - 1 + rN}{a}$$

если мы выберем $r \ge \dfrac{2(bs_i - 2B)}{N}$, то получим

$$\frac{2B + 2(bs_{i-1} - 2B)}{b} \approx 2s_{i-1} \le s_i$$

Таким образом, новая $s_i$ по меньшей мере вдвое превосходит предыдущую ступень. Обратите внимание, что все новые интервалы для $m_0$ на шаге $i$

$$\frac{2B + rN}{s_i} \le m_0 \le \frac{3B - 1 + rN}{s_i}$$

имеют размер $B/s_i$. Как следствие, удвоение $s_{i-1}$ даёт интервал, который в два раза меньше интервала предыдущего шага. Это напоминает двоичный поиск, который мы реализовали с помощью побочного канала чётности, и заставляет атаку сходиться в несколько шагов, которые являются линейными по количеству битов.

Получая 'бинарным поиском' новые значения интервала $[a, b]$, мы его уменьшаем и в конечном счёте он сходится к интервалу нулевой длины, границами которого и будет являться дешифрованное сообщение.

### Источники информации

http://secgroup.dais.unive.it/...Oracle-Attacks-on-RSA.html
https://www.youtube.com/watch?...&index=6

### Решение таска

Теперь рассмотрим решение нашего таска. Он отличается от обычной атаки Бляйхенбахера тем, что вместо байта 0x2 для паддинга используется id (байт, который известен). То есть нам ничего не мешает оценивать сообщение в рамках отрезка $[id * B, (id + 1) * B]$. Мы будем 'паддить' имеющиеся после каждого предсказания промежутки с помощью нужного на следующем шаге id и вычислять новые промежутки для сообщения (после каждой итерации мы 'снимаем' паддинг с промежутков, чтобы на следующей итерации наложить новый паддинг).

Мы должны прийти в результате нескольких 'обращений к оракулу' к единственному промежутку, после чего мы начнем проводить нашу версию 'бинарного поиска'.

Нам даны значения id и предсказанное значение $s_{i+1}$ - по ним, если посмотреть на алгоритм бинарного поиска, мы сможем восстановить значение переменной $r$, а после этого мы сможем высчитать новые границы для промежутка $[a, b]$. Значение $r$ определяется пределах

$$r \in (\frac{s_i * a + 1 - (id + 1) * B}{N}, \frac{s_i * b - id * B}{N})$$

Для каждой итерации поиска данное $r$ может принимать одно значение (проверено автором). На крайний случай, можно проверять промежуток для каждого из $r$, пересекая его с единственным имеющимся промежутком. Таким образом, даже имея итерации с разным id в паддинге, мы можем получать информацию из оракула.

Также хочется отметить, что нас интересуют только первые 32 байта открытого текста, так как именно в них находится флаг - поэтому нам дано такое количество запросов, которое позволяет восстановить только верхние 32 байта ( то есть после учета всех предсказаний длина промежутка будет не более $2^{30*8} = 2^{240}$).