RSA "Greater Than Half" Oracle Attack

1 Background and terminology

1.1 Oracles

The attack scenario we explore here is one where a server which uses RSA encryption is listening for connections and we can interact with it as an "attacker" in order to retrieve information. In particular, interactions with the server include the client (in our case, the attacker) sending encrypted messages to the server, and the server responding according to the message it receives. The idea of this attack (and of other attacks) is that according to the server's reply — its content or other characteristics of it — an attacker can obtain certain information and use it to calculate or find out things which they "shouldn't".

The server whose behavior allows a client to deduce secret information is called an "oracle". A client — an attacker — can initiate interactions with the server, sending it messages about which they want to learn something, and the server — the oracle — responds in a manner that allows the client to learn something they shouldn't. We often refer to the attacker's messages as "queries" and to the oracle's answers as "responses".

1.2 RSA encryption

A quick recap of "textbook" RSA encryption:

An RSA key consists of a public modulus N, a public exponent e, and a private exponent d which is the inverse of e in the multiplicative group modulo N. The public key is therefore (N, e) and the private key is d. To encrypt a message m, which is an integer modulo N, one calculates $c = m^e \bmod N$. The party with the secret key (in this case, the server) can then decrypt c by calculating $c^d \equiv m^{e \cdot d} \equiv m \pmod{N}$.

2 This scenario

2.1 The oracle

In this scenario we have a server which listens for connections, and once a connection is made (a socket is opened), it receives an message encrypted using its RSA key. The encryption is "textbook" RSA, meaning that it is of the form $m^e \bmod N$, where N is the public modulus, e is the public exponent, and m is the message (in this case, the message is a number modulo N). The server responds with a single byte with the value 1 or 0, which indicates whether the encrypted message is greater than $\frac{N}{2}$ or smaller; in other words, we can describe the server as an oracle O which operates thus:

$$O\left(m^e \bmod N\right) = \begin{cases} 1 & m > \frac{N}{2} \\ 0 & \text{otherwise} \end{cases}$$

2.2 The attack

We now show how to use the oracle described above in order to decrypt any message m encrypted using the server's public key. We denote by N the public modulus and by e the public exponent of the server's RSA key; recall that the encryption of m is $m^e \bmod N$. We denote by O the oracle provided by the server, i.e. if

we send the server a ciphertext c as a query to the server, its response is $O(c)$; c is an encrypted message and $O(c)$ is 1 or 0 (true or false).

Notice that given this oracle, we can find out whether the encrypted message m is greater than $\frac{N}{2}$ or not by querying the oracle; if $O(m^e \bmod N) = 1$ then $m > \frac{N}{2}$.

We continue with the following observation:

Proposition 1. Given an encrypted message $c = m^e \bmod N$ (and the server's public parameters N and e), we can create a new encrypted message c' which encrypts the message $2m$; in other words, we can calculate $c' = (2m)^e \bmod N$.

Proof. Using the public parameters N and e, we can encrypt the message 2: $2^e \bmod N$. By multiplying this ciphertext by the given ciphertext c (and taking the result modulo N), we obtain:

$$2^e \cdot c \equiv 2^e \cdot m^e \equiv (2m)^e \equiv c' \pmod{N}$$

□

We call the function that "doubles" an encrypted message "double", that is:

$$\text{double}(m^e \bmod N) = (2m)^e \bmod N$$

We therefore have a way to double encrypted messages without being able to decrypt them. We can then query the oracle on these doubled messages, revealing more information on the original message. Notice that to begin with, by querying $O(c)$, we narrowed the range of m by half: according to the oracle's response, the message is either in $\left[0, \frac{N}{2}\right]$ or in $\left(\frac{N}{2}, N\right)$. By doubling m and querying the oracle again, we narrow the possible range for m again.

Suppose for example that m is greater than $\frac{N}{2}$, meaning that $O(c) = 1$. In this case,

$$2m \in \begin{cases} \left(N, \frac{3}{2}N\right] & m \in \left(\frac{N}{2}, \frac{3}{4}N\right] \\ \left(\frac{3}{2}N, 2N\right) & m \in \left(\frac{3}{4}N, N\right) \end{cases}$$

and so

$$2m \bmod N \in \begin{cases} \left(0, \frac{N}{2}\right] & m \in \left(\frac{N}{2}, \frac{3}{4}N\right] \\ \left(\frac{N}{2}, N\right) & m \in \left(\frac{3}{4}N, N\right) \end{cases}$$

meaning that

$$O(\text{double}(c)) = \begin{cases} 0 & m \in \left(\frac{N}{2}, \frac{3}{4}N\right] \\ 1 & m \in \left(\frac{3}{4}N, N\right) \end{cases}$$

and indeed, "doubling" c and querying the oracle again has halved the possible range for m; if $O(\text{double}(c)) = 0$, then m is between $\frac{N}{2}$ and $\frac{3}{4}N$, and if not, then m is between $\frac{3}{4}N$ and N.

This observation leads to the attack:

Proposition 2. Given an encrypted message $c = m^e \bmod N$ (and the server's public parameters N and e) and the oracle O described above, we can find m using approximately $\log(N)$ oracle queries.

Proof. Using what we've seen above, we can query the oracle, apply the doubling function to the ciphertext and repeat the process. In each step, the known range of possible values for m is halved, meaning that in approximately $\log(N)$ steps, the range will consist of a single possible value for m. □

2.3 Attack implementation

We describe the steps of the attack shown above. Once again, $c = m^e \bmod N$ is given, as are N and e, and the oracle O (the server) can be queried.

Algorithm 1 RSA "Greater Than Half" Oracle Attack

```
1:  procedure DOUBLE(c, N, e)
2:      output ((2^e mod N) · c) mod N
3:  end procedure

4:  procedure ATTACK(c, N, e, O)
5:      low ← 0
6:      high ← N
7:      while low < high − 1 do
8:          if O(c) = 1 then
9:              low ← AVERAGE(low, high)
10:         else
11:             high ← AVERAGE(low, high)
12:         end if
13:         c ← DOUBLE(c)
14:     end while
15:     output low
16: end procedure
```

As binary-search-like algorithms like the above can be tricky to get exactly right, we recommend that on completion of the searching loop (on lines 7 to 14), the algorithm check that it has indeed reached the correct value of m by calculating $low^e \bmod N$ and comparing it to the ciphertext given (the original value of the parameter c).

Furthermore, when the range of possible values of m is sufficiently small, one can go over all values in the range, calculate their ciphertexts and compare them to the given ciphertext c, thus requiring fewer queries to the server and possibly eliminating small errors in the search implementation.

2.4 The challenge's server

A script that implements the oracle interaction with the server is provided, so this section is not necessary in order to solve the challenge, and is given here for the sake of completeness.

Each TCP session with the server consists of one oracle query. The format in which the server expects the query is length-value, where the length field is a two-byte big endian integer and the value is the encrypted message, encoded in big endian in the length given in the previous field. The server responds with a single byte, 0 or 1, according to the oracle described above, and closes the connection.