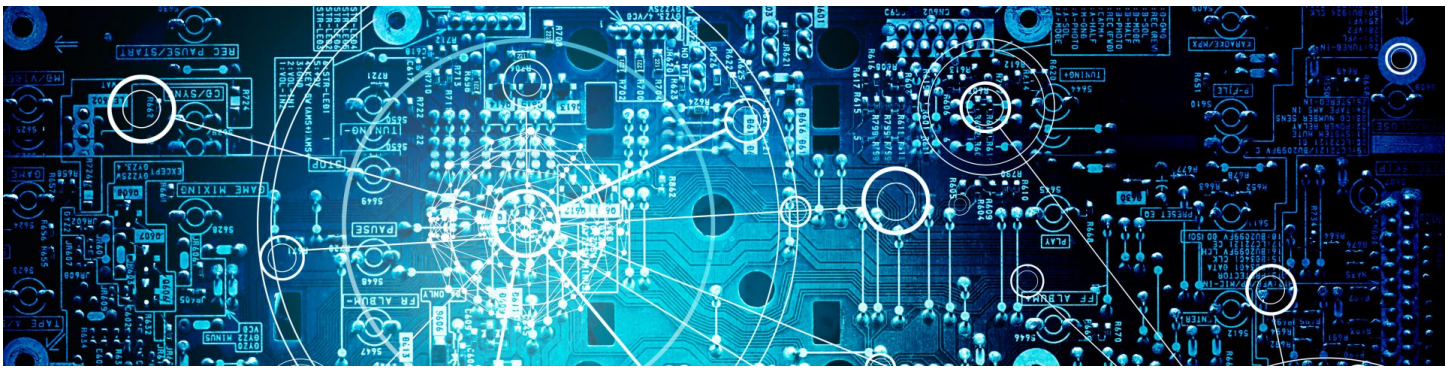
KUDELSKI SECURITY | RESEARCH

The Latest News from Research at Kudelski Security


HOME CATEGORIES



# BREAKING RSA OAEP WITH MANGER'S ATTACK

April 5, 2018 Yolan Romailler Crypto Leave a comment

The RSA cryptosystem has had its fair share of attacks over the years, but among the most impressive, you can find the infamous Bleichenbacher attack [Ble98], which doomed PKCS v1.5 in 1998. Nineteen years later, the ROBOT attack proved that the Bleichenbacher attack was still a concern today. Now, what alternatives to RSA PKCS v1.5 do we have? Well, its successor, RSA OAEP also known as RSA PKCS v2.1 is obviously a good candidate.

SEARCH

Sear

CATEGORIES

Select Ca

ARCHIVES

Select Mo

RSA OAEP is an interesting scheme because it has been mathematically proven to be secure against a chosen-ciphertext attack in the random oracle model.

Guess what? An attack against "weak implementations" of RSA OAEP also exists. This attack, while less well known than Bleichenbacher's because it never makes the headlines, is known as "Manger's attack" [Man01] after the name of its creator, James Manger, who published it in 2001.

In this blog post, I will first explain each part of the attack and then guide you through their implementation in Go. We will end up with a fully functional attack just waiting for a so-called "oracle" to be plugged in. Typical oracles, in practice, are provided by timing information, but sometimes an error message eases the job.

First, let me first introduce the notion of oracle as I'll use it in this post:

> ***Definition:* oracle**
> *An* oracle *is a black-box that will take an input and answer a specific, fixed question regarding its input, without being restricted in terms of knowledge. It can be used by any user indifferently, including any adversary.*

In cryptography, the most common sorts of oracles used to attack different schemes are *padding oracles* and *timing oracles*.

- The *padding oracles,* as in Bleichenbacher's attack, take ciphertexts as input and reveal whether or not the padding is correct after their decryption.
- The *timing oracles*, as used by Kocher [Koc96] for example, usually report the time needed to decrypt a specific ciphertext.

In the end, timing oracles can often be used as padding oracles, typically when the implementation returns an early error if the padding is wrong, leading an attacker to build a padding oracle out of the return timings of an implementation.

## Manger's Attack

Let $n$ be an RSA modulus, with $e$ and $d$ its public and private exponents, respectively, and $k = \lceil \log_{256} n \rceil$ be its byte length. Then as long as the attacker has access to an oracle able to tell them whether $y \equiv c^d \mod n$ is less than $B = 2^{8(k-1)}$ or not, then they are able to narrow down the range of possible messages corresponding to the ciphertext $c = m^e \mod n$ to only a single message, $m$, in $\mathcal{O}(\log(n))$ queries.

Furthermore, the attack crucially relies on a well-known property of the RSA cryptosystem, namely its malleability [DDN91]:

> ***Definition:*** **malleability**
> *For an encryption $E(m)$ of the plaintext $m$, the scheme $E$ is* malleable *if it is possible for a given function $f_1$ to generate another ciphertext $f_1(E(m))$ which yields the plaintext $f_2(m)$, for a function $f_2$ without requiring knowledge of $m$ at any point. The property of being malleable is called* malleability.

This is the case for the RSA cryptosystem, since we operate in the quotient ring $\mathbb{Z}/n\mathbb{Z}$. For a given public key $(e, n)$, the plaintext $m$ is encrypted as $E(m) \equiv m^e \mod n$. We can construct the ciphertext $m \cdot x$ for any $x$, as being $E(m) \cdot x^e \mod n = (mx)^e \mod n \equiv E(mx)$.

Now let me formally describe the attack:
Initially we only know one ciphertext, $c$, we would like to decrypt without access to the private key, but with access

to an oracle able to tell us, when we send it a value $a$, whether the decrypted value $b \equiv a^d \mod n$ satisfies $b \geq B$ or not.

I'll now describe the attack as presented in [Man01] when $2B < n$, which is usually the case since the size of most RSA moduli is an exact multiples of $8$ bits, rendering $n$ at least $128$ times bigger than $B$.

If this is not the case, then the attack is still possible but has to deal with multiple intervals, cf. [Man01] section 3.2.

So, when that assumption holds, the attack is performed in three steps as follows:

## Step 1

We firstly have to try to multiply the unknown plaintext $m$ with

$f_{1,1} = 2,\ f_{1,2} = 2^2,\ \ldots,\ f_{1,i} = 2^i,\ \ldots$

by sending the values $f_{1,i}^e \cdot c \mod n$ to the oracle for $i = 1, 2, \ldots$ until it returns that $f_{1,i} \cdot m \geq B \mod n$. This is true as soon as $f_{1,i}^e \cdot c \geq B \mod n$ thanks to the malleability of the RSA cryptosystem.

This ensures us that $f_{1,i} \cdot m \in [B, 2B[$, so we know that $\frac{f_{1,i}}{2} \cdot m \in \left[\frac{B}{2}, B\right[$.

This can easily be done in Go as follows:

```
f1 := new(big.Int).SetInt64(int64(2))
for !tryOracle(f1, c, e, N, ourOracle) {
  f1.Mul(two, f1)
}
```

Where the tryOracle(f, c, e, N, ourOracle) function is sending the ciphertext $f^e \cdot c \mod n$ to our oracle which returns, for $c = m^e \mod n$ whether $f \cdot m \geq B \mod n$ or not.

## Step 2

We now try to multiply the unknown plaintext $m$ with

$f_{2,j} = \left(\left\lfloor \frac{n+B}{B} \right\rfloor + j\right) \cdot \frac{f_{1,i}}{2}, \quad \text{for } j \in \mathbb{N}^*$

by exploiting the malleability of the RSA cryptosystem.

To do so, we send $f_{2,j}^e \cdot c \mod n$ to the oracle, for $j = 1, 2, \ldots$, until it returns that $f_{2,j} \cdot m < B$.

It then necessarily implies that the modulo reduction

"wrapped" $f_{2,j} \cdot m$ into something smaller than $B$, since by definition of $f_{1,i}$ and $f_{2,j}$, we know that $f_{2,j} \cdot m \geq B$.
In turn, it then implies that $f_{2,j} \cdot m \in [n, n + B[$.

Note that this step necessarily terminates, taking at most $\left\lceil \frac{n}{B} \right\rceil$ oracle queries, since $n$ will always be exceeded when $f_{2,j} = \left\lceil \frac{2n}{B} \right\rceil \cdot \frac{f_{1,i}}{2}$.

This is also easily implemented in Go:

```
nB := new(big.Int).Add(N, B)
nBB := new(big.Int).Div(nB, B)
f2 := new(big.Int).Mul(nBB, f12)

for tryOracle(f2, c, e, N) {
  f2.Add(f2, f12)
}
```

## Step 3

We finally want to narrow down the range of possible messages to just one.
This is done iteratively, approximately dividing the range by two at each step. It uses a heuristic approach to define its parameters and has not been formally proven by Manger in his article.
This implies defining $m_{min,1} = \left\lceil \frac{n}{f_{2,j}} \right\rceil$ and $m_{max,1} = \left\lfloor \frac{n+B}{f_{2,j}} \right\rfloor$ and then as long as the range $[m_{min,k}, m_{max,k}]$ is containing more than one value, one can do the following:

1. Choose a temporary multiple $f_{tmp,k} = \left\lfloor \frac{2B}{m_{max,k} - m_{min,k}} \right\rfloor$;
2. Compute a boundary point $i_k = \left\lfloor \frac{f_{tmp,k} \cdot m_{min,k}}{n} \right\rfloor$;
3. Compute a value $f_{3,k}$, which is so that $f_{3,k} \cdot m$ is spanning a single boundary point at $i_k n + B$, as follows: $f_{3,k} = \left\lceil \frac{i_k n}{m_{min,k}} \right\rceil$;
4. Send the value $f_{3,k}^e \cdot c$ to the oracle, if it returns $f_{3,k} \cdot m \geq B$, then one can set $m_{min,k+1} = \left\lceil \frac{i_k n + B}{f_{3,k}} \right\rceil$ and go back to 1. Else, if it returns $f_{3,k} \cdot m < B$, once can set $m_{max,k+1} = \left\lfloor \frac{i_k n + B}{f_{3,k}} \right\rfloor$ and go back to 1.

This is the longest part of the computation and can be implemented as follows in Go (using notations as close to Manger's article as possible):

```
mmin := divCeil(N, f2)
mmax := new(big.Int).Div(nB, f2)
BB := new(big.Int).Mul(two, B)
diff := new(big.Int).Sub(mmax, mmin)

for diff.Sub(mmax, mmin).Cmp(zero) > 0 {
 ftmp := new(big.Int).Div(BB, diff)
 ftmpmmin := new(big.Int).Mul(ftmp, mmin
 i := new(big.Int).Div(ftmpmmin, N)
 iN := new(big.Int).Mul(i, N)
 f3 := divCeil(iN, mmin)
 iNB := new(big.Int).Add(iN, B)
 if tryOracle(f3, c, e, N) {
  mmin = divCeil(iNB, f3)
 } else {
  mmax.Div(iNB, f3)
 }
}
```

When this iterating process terminates, the range of possible messages $[m_{min,k}, m_{max,k}]$ only spans one value, $m$ the secret message. On average, the attack requires approximately $\ell$ queries, for $\ell$ the bit-size of the RSA modulus.

## The code

My complete, working, heavily commented implementation of Manger's attack in Go can be found on Github and is compatible with generic oracles!

Note that in this implementation, I provide a test file attack_test.go where I am crafting the oracle to tell you whether the most significant byte of the decrypted, padded data is zero or not. This is equivalent to testing whether the decrypted value $b \equiv a^d \mod n$ satisfies $b \geq B$ or not, by definition of $B$ and $k$. This allows me to attack a modified version of Go's crypto library (to be found in the package moddedrsa). This file is a black-box test and it is probably a good example (excepted for the infamous dot import) if you want to implement the attack using your own oracle.

In the end, in order to use it with your very own oracle, you just need to implement the "Oracle" interface in your own Go program and call the attack using your oracle.

Let me know if you used this code, I'm curious about the oracles you are using!

# Bibliography


| | |
|---|---|
| [Ble98] | Daniel Bleichenbacher. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1". In: CRYPTO 1998. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998. |
| [Man01] | James Manger. "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0". In: CRYPTO 2001. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001. |
| [Koc96] | Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: CRYPTO 1996. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996. |
| [DDN91] | Danny Dolev, Cynthia Dwork, and Moni Naor. "Non-Malleable Cryptography (Extended Abstract)". In: 23rd ACM STOC. ACM Press, May 1991. |


FEATURED IMAGE HACKING ORACLE RSA SECURITY

« Wire applications audit (with X41 D-Sec)

Critical Vulnerabilities Cisco Smart Install Actively Exploited to Cause Mass Network Outages (CVE-2018-0171 & CVE-2018-0156) »

## LEAVE A REPLY

Enter your comment here...