Bleichenbacher's RSA Attack on PKCS#1

Instructor: Sruthi Sekar
Submitted by: Tanmay Patil, Jaswin Reddy Mandavyapuram
210260058@iitb.ac.in, 23B1289@iitb.ac.in

November 27, 2024

1. Brief Overview of the Topic

In this report, we analyze the following situation. Let n, e be an RSA public key, and let d be the corresponding secret key. Assume that an attacker has access to an oracle that, for any chosen ciphertext c, indicates whether the corresponding plaintext $c^d \mod n$ has the correct format according to the RSA encryption standard PKCS #1.

We show how to use this oracle to decrypt or sign a message. The attacker carefully prepares ciphertexts that are sent to the oracle. Combining the returns from the oracle, the attacker gradually gains information on c^d. The chosen ciphertexts are based on previous outcomes of the oracle. Thus, this technique is an example of an *adaptive chosen-ciphertext attack*. Usually, a chosen-ciphertext attack is based on the theoretical assumption that the attacker has access to a decryption device that returns the complete decryption for a chosen ciphertext.

Hence, if a public-key cryptosystem is susceptible to a chosen-ciphertext attack, that often is considered to be only a theoretical weakness. However, the attack shown in this report is *practical*, because it is easy to get the necessary information corresponding to the oracle reply. The attack can be carried out if, for example, the attacker has access to a server that accepts encrypted messages and returns an error message depending on whether the decrypted message is PKCS conforming.

This report is organized as follows. We describe a parity oracle attack example in Section 2. Then, we describe the RSA encryption standard PKCS #1 in Section 3. In Section 4, we describe and analyze our chosen-ciphertext attack. Different situations in which this attack can be carried out are listed in Section 5.

2. Parity Oracle Attack Example

Before illustrating padding oracle attacks, we give a simple example of a side channel in which a single bit of the plaintext is revealed: its *parity*. It is well-known that leaking the parity of an RSA encrypted message allows the recovery of the whole message. This fact is sometimes regarded as a good property of the cipher: if there is no way to compute the plaintext, we are also guaranteed that it is not even possible to compute the parity bit of the plaintext (as otherwise, we could use the attack to reconstruct the whole message).

Example 1: The Even Eggs Farm

For packaging reasons, the **EEGGs Farm** only sells an even number of eggs to customers. To guarantee the highest privacy standard possible, all orders are encrypted under a 1024-bit RSA public key. Using OpenSSL, let us try to place two encrypted orders of 0x10 and 0x11 eggs, respectively:

```
$ echo -ne "\x10" | openssl rsautl -encrypt -inkey key  | ./EEGGs
OK
$ echo -ne "\x11" | openssl rsautl -encrypt -inkey key  | ./EEGGs
Sorry, only an even number of eggs can be ordered
```

This is an example of a side channel. The application reveals partial information about the plaintext. The EEGGs service acts as an *oracle* that reveals the parity of the plaintext. Although parity seems innocuous, it can be exploited to recover the entire encrypted message.

Description of the Attack

Consider the following ciphertext:

```
$ hexdump -e ' 128/1  "%02x" "\n" ' enc_message
67d39413a1d1cd5a46d634a925bc8dad372f38c93c24330a45ac824c962ff9fd2b4936374ece2c
8eecbdefc3b4cacdf29f342efd5cc11fe9f1e797c265804b1ddf17d72cbc4bc1326bc02e484b6d
369d2b1afd60e2fa007f1c510086296c3e0a7b64d1297dd948fb560f94605197da6252febe3e97
c160bd91902efbc62a8e36
```

We can check the parity of the plaintext using the EEGGs parity oracle:

```
$ cat enc_message | ./EEGGs
OK
```

Thus, we know that the plaintext m ends with a 0 bit. Now consider multiplying m by 2 modulo n. Depending on whether $m < n/2$ or $m > n/2$, we have:

$$\begin{array}{cccc} \text{Range for } m & 2m & \mod n & \text{Parity} \\ 0 \leq m < n/2 & 2m & & \text{even} \\ n/2 \leq m < n-1 & 2m-n & & \text{odd} \end{array}$$

If we multiply the ciphertext by the encryption of 2 and query the parity oracle on the new ciphertext, we can determine whether m lies in the first or second half of the interval $[0, n-1]$. This procedure can be extended to $2^i m \mod n$ for the i-th step, allowing for a binary search over the range.

Python Implementation

The following Python script performs the attack using the parity oracle:

```
import math, binascii
from decimal import *
from subprocess import *

# modulus and encryption exponent
n = 0xa58bfc62abe40a57d287860f395a7705ab0534eeb2bb591131a93d627dd2564b6
1ebde2a43082a50fbd84d308d1f70b7cd7dc4aca6239abe46ff76d2a7135034c2f8d477
d5e1438b57230b157c71c1eb44b62cbf5e2cca14b956979c3b48e7ae4475dd4db8e72eb
d5559bde3f281c8ee75c08e23c6960e1e1669fac91a815c67
e = 0x10001

# reads the encrypted message
f = open('enc_message')
data = f.read()
f.close()
y = int(binascii.hexlify(data), 16)

enctwo = (2 ** e) % n # the encryption of 2

# The parity oracle: calls EEGGs and returns True when it gets an OK
    answer
def parity(y, n):
    p = Popen(['./EEGGs'], stdin=PIPE, stdout=PIPE)
    bin_y = binascii.unhexlify('%0256x' % y)  # converts long int into
        bytes
    ret = p.communicate(bin_y) # calls EEGGs
    return ret[0] == 'OK\n'

# do the binary search
def partial(y, n):
    k = int(math.ceil(math.log(n, 2)))  # n. of iterations
    getcontext().prec = k  # allows for 'precise enough' floats
    l, u = Decimal(0), Decimal(n)
    for _ in range(k):
        h = (l + u) / 2
        if parity(y, n):
            u = h  # left interval
        else:
            l = h  # right interval
        y = (y * enctwo) % n
    return int(u)

print('%0256x' % partial((y * enctwo) % n, n))
```

When run, the script recovers the plaintext:

```
$ python parity.py
00023572e4bcc8df4c7e53f931c1d79addcc3d0412db8723e28c84a5f6340d0f95f9836b079076
69f2527eda22e1f80e6e2e4dac062326f5716fca45004c65616b696e672070617269747920616c
6c6f777320666f722064656372797074696e6720612077686f6c652052534120656e6372797074
6564206d6573736167650a
```

The plaintext message is revealed to be:

```
Leaking parity allows for decrypting a whole RSA encrypted message.
```

3. PKCS #1 Overview

In this section, we describe briefly the RSA encryption standard PKCS #1. Currently, there are three block formats: Block types 0 and 1 are reserved for digital signatures, and block type 2 is used for encryption. We describe only block type 2, because it is relevant for this report.

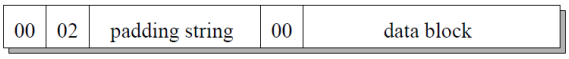

Figure 1: PKCS #1 block format for encryption. The first two bytes in this format are constant. The length of the padding block can vary.

The first two bytes in this format are constant. The length of the padding block can vary.

Let n, e be an RSA public key, and let p, q, d be the corresponding secret key (i.e., $n = pq$ and $d \equiv e^{-1} \mod \phi(n)$). Moreover, let k be the byte length of n. Hence, we have $2^{8(k-1)} \leq n < 2^{8k}$. A data block D, consisting of $|D|$ bytes, is encrypted as follows. First, a padding string PS, consisting of $k - 3 - |D|$ nonzero bytes, is generated pseudo-randomly. Here, $|D|$ must not exceed $k - 11$; in other words, the byte length of PS is at least 8. Now, the encryption block $EB = 00\|02\|PS\|00\|D$ is formed (Figure 1), converted into an integer x, and encrypted with RSA, giving the ciphertext $c \equiv x^e \mod n$. The representation of the ciphertext is not important for this report.

We are, however, interested in how the receiver parses a ciphertext. First, they get an integer x' by decrypting the ciphertext with their private key. Then, they convert x' into an encryption block EB'. Now they look for the first zero byte, which indicates the ending of the padding string PS and the start of the data block D. The following definition specifies when this parsing process is successful.

Definition 1. An encryption block EB consisting of k bytes – that is,

$$EB = EB_1\| \cdots \|EB_k$$

is called *PKCS conforming* if it satisfies the requirements of block type 2 in PKCS #1. In particular, EB must satisfy the following conditions:

- $EB_1 = 00$.
- $EB_2 = 02$.
- EB_3 through EB_{10} are nonzero.
- At least one of the bytes EB_{11} through EB_k is 00.

We also call a ciphertext c *PKCS conforming* if its decryption is PKCS conforming.

Note that the definition of conforming does not include possible integrity checks. We show in Section 4 that it should not be possible for an attacker to decide whether a chosen ciphertext is PKCS conforming. It is sometimes possible for an attacker to do so even if the data block contains further integrity checks.

4. Chosen-Ciphertext Attack Description

In a chosen-ciphertext attack, the attacker selects the ciphertext, sends it to the victim, and is given in return the corresponding plaintext or some part thereof. A chosen-ciphertext attack is called *adaptive* if the attacker can choose the ciphertexts based on previous outcomes of the attack.

It is well known that plain RSA is susceptible to a chosen-ciphertext attack. An attacker who wishes to find the decryption $m \equiv c^d \pmod{n}$ of a ciphertext c can choose a random integer s and ask for the decryption of the innocent-looking message $c_0 \equiv s^e c \pmod{n}$. From the answer $m_0 \equiv (c_0)^d$, it is easy to recover the original message, because $m \equiv m_0 s^{-1} \pmod{n}$.

Another well-known result is that the least significant bit of RSA encryption is as secure as the entire message. In particular, there exists an algorithm that can decrypt a ciphertext if there exists another algorithm that can predict the least significant bit of a message given only the corresponding ciphertext and the public key. Håstad and Näslund recently extended this result to show that all individual RSA bits are secure .

Hence, it is not necessary for an attacker to learn the complete decrypted message in a chosen-ciphertext attack: Single bits per chosen ciphertext may be sufficient.

The result reported in this report is similar. We assume that the attacker has access to an oracle that, for every ciphertext, returns whether the corresponding plaintext is PKCS conforming. We show that we can use this oracle to compute $c^d \pmod{n}$ for any chosen integer c. Theoretically, we can use Håstad's and Näslund's algorithm to find c. In this report, we describe a different algorithm that aims to minimize the number of chosen ciphertexts, thus demonstrating the practicality of the attack.

That is, we are not trying to generalize the attack; rather, we take advantage of specific properties of PKCS #1. In particular, the algorithm relies on the facts that the first two bytes of the PKCS #1 format are constant, and that these two bytes are known with certainty when a ciphertext is accepted. Additionally, we use heuristic arguments in our analysis of the algorithm to approximate the number of expected chosen ciphertexts rather than finding an upper bound.

3.1 Description of the Attack

First, we give a short overview of the attack; then, we describe the attack in detail. Assume that the attacker wants to find $m \equiv c^d \pmod{n}$, where c is an arbitrary integer. Basically, the attacker chooses integers s, computes:

$$c_0 \equiv c \cdot s^e \pmod{n},$$

and sends c_0 to the oracle. If the oracle says that c_0 is PKCS conforming, then the attacker knows that the first two bytes of ms are 00 and 02. For convenience, let:

$$B = 2^{8(k-2)},$$

where k is the length of n in bytes. Thus, if ms is PKCS conforming, it implies:

$$2B \leq ms \pmod{n} < 3B.$$

By collecting several such pieces of information, the attacker can eventually derive m. Typically, 2^{20} chosen ciphertexts are sufficient, but this number varies depending on implementation details.

The attack can be divided into three phases:

- **Phase 1: Blinding.** The message is blinded, giving a ciphertext c_0 that corresponds to an unknown message m_0.
- **Phase 2: Searching.** The attacker finds small values s_i for which the ciphertext $c_0 \cdot s_i^e \pmod n$ is PKCS conforming. Each successful s_i narrows the possible range of m_0.
- **Phase 3: Narrowing.** When only one interval remains, the attacker has sufficient information to choose s_i such that $c_0 \cdot s_i^e \pmod n$ is highly likely to be PKCS conforming, eventually reducing the range to a single value.

Detailed Steps of the Attack: The variable M_i always represents a set of (closed) intervals, such that m_0 is contained within one of the intervals of M_i.

1. **Step 1: Blinding.** Given an integer c, choose different random integers s_0, then check, using the oracle, whether $c \cdot s_0^e \pmod n$ is PKCS conforming. For the first successful s_0, set:
$$\begin{aligned} c_0 &\equiv c \cdot s_0^e \pmod n, \\ M_0 &= \{[2B, 3B-1]\}, \\ i &= 1. \end{aligned}$$
2. **Step 2: Searching for PKCS conforming messages.**
   (a) **Step 2.a: Starting the search.** If $i = 1$, search for the smallest positive integer $s_1 \geq \lceil n/(3B) \rceil$, such that $c_0 \cdot s_1^e \pmod n$ is PKCS conforming.
   (b) **Step 2.b: Searching with multiple intervals.** If $i > 1$ and M_{i-1} contains at least two intervals, search for the smallest integer $s_i > s_{i-1}$, such that $c_0 \cdot s_i^e \pmod n$ is PKCS conforming.
   (c) **Step 2.c: Searching with one interval.** If M_{i-1} contains exactly one interval $[a, b]$, choose small integer values r_i and s_i such that:
$$r_i \geq \frac{2b \cdot s_{i-1} - 2B}{n}, \tag{1}$$
   and
$$\frac{2B + r_i n}{b} \leq s_i < \frac{3B + r_i n}{a}. \tag{2}$$
   Repeat until $c_0 \cdot s_i^e \pmod n$ is PKCS conforming.
3. **Step 3: Narrowing the set of solutions.** After finding s_i, compute the set:
$$M_i = \bigcup_{[a,b] \in M_{i-1}} \left\{ \left[\max\left(a, \left\lceil \frac{2B + rn}{s_i} \right\rceil \right), \min\left(b, \left\lfloor \frac{3B - 1 + rn}{s_i} \right\rfloor \right) \right] \right\}, \tag{3}$$
   for all r satisfying:
$$\frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i - 2B}{n}.$$
4. **Step 4: Computing the solution.** If M_i contains only one interval $[a, a]$, set:
$$m \equiv a \cdot s_0^{-1} \pmod n,$$
   and return m as the solution for $m \equiv c^d \pmod n$. Otherwise, increment i and return to Step 2.

Remarks:

- Step 1 can be skipped if c is already PKCS conforming, in which case $s_0 = 1$.
- In Step 2.a, we start with $s_1 = \lceil n/(3B) \rceil$, as smaller values result in $m_0 \cdot s_1$ being non-PKCS conforming.
- The attack can be improved by using additional information, such as the presence of at least one zero byte in any PKCS conforming message.

3.2 Analysis of the Attack

We now analyze the correctness of the attack and approximate the complexity of, and, in particular, the number of oracle accesses necessary for this attack. We must make a few heuristic assumptions; hence, we cannot give a rigorous proof of our result.

First, we approximate the probability $\Pr(P)$ that a randomly chosen integer $0 \leq m < n$ is PKCS conforming. Let $\Pr(A) = \frac{B}{n}$ be the probability that, for a randomly chosen integer, the first two bytes are 00 and 02, respectively. Since we have $2^{16}B < n < 2^8 B$, it follows that:

$$2^{-16} < \Pr(A) < 2^{-8}.$$

The RSA modulus is usually chosen to be a multiple of 8; hence, $\Pr(A)$ will usually be close to 2^{-16}. The probability that the padding block PS contains at least 8 non-zero bytes followed by a zero byte is:

$$\Pr(P|A) = \left(\frac{255}{256}\right)^8 \left(1 - \left(\frac{255}{256}\right)^{k-10}\right),$$

where k is the number of bits in the modulus n. Assuming a modulus n of at least 512 bits (i.e., $k \geq 64$), we have:

$$0.18 < \Pr(P|A) < 0.97.$$

Thus, we have:

$$0.18 \cdot 2^{-16} < \Pr(P) < 0.97 \cdot 2^{-8}.$$

Next, we explain why our algorithm finds m_0 and thus m. We prove that $m_0 \in M_i$ for all i by induction over i. Since m_0 is PKCS conforming, we have $2B \leq m_0 \leq 3B - 1$, and so trivially, $m_0 \in M_0$.

Now assume that $m_0 \in M_{i-1}$. Hence, there exists an interval $[a, b] \in M_{i-1}$ with $a \leq m_0 \leq b$. Since $m_0 s_i$ is PKCS conforming, there exists an integer r such that:

$$2B \leq m_0 s_i - rn \leq 3B - 1,$$

and hence:

$$as_i - (3B - 1) \leq rn \leq bs_i - 2B.$$

We also have:

$$\frac{2B + rn}{s_i} \leq m_0 \leq \frac{3B - 1 + rn}{s_i}.$$

Hence, it follows from the definition of M_i that m_0 is contained in one of the intervals.

Now we analyze the complexity of the attack. The messages in Step 1 are chosen randomly; therefore, this step needs about $1/\Pr(P)$ accesses to the oracle on average to

find s_0. We assume again that, on average, we need $1/\Pr(P)$ accesses to the oracle to find s_i for $i \geq 1$ in Step 2.a and 2.b.

Let ω_i be the number of intervals in M_i. Using heuristic arguments, we can expect that ω_i will satisfy the following equation for $i \geq 1$:

$$\omega_i \leq 1 + 2^{i-1} s_i \left(\frac{B}{n}\right)^i.$$

Indeed, the length of an interval in M_i is upper-bounded by $\left\lceil \frac{B}{s_i} \right\rceil$. The knowledge that $m_0 s_i$ is PKCS conforming alone would lead to:

$$\left\lceil \frac{s_i B}{n} \right\rceil$$

intervals of the form:

$$I_r = \left[\frac{2B + rn}{s_i}, \frac{3B - 1 + rn}{s_i}\right],$$

since r can take at most $\left\lceil \frac{s_i B}{n} \right\rceil$ values in equation (3). These intervals, or a fraction of them, are included in M_i if I_r overlaps with one interval of M_{i-1}. No interval I_r can overlap with two intervals in M_{i-1}. If intervals I_r were randomly distributed, then the probability that one intersects with M_{i-1} would be upper-bounded by:

$$\left(\frac{1}{s_i} + \frac{1}{s_{i-1}}\right) \omega_{i-1}.$$

Hence, we get equation (4) by taking into account that one interval must contain m_0. In our case, we expect s_2 to be approximately $2/\Pr(P)$, and we have:

$$\frac{2(B/n)^2}{\Pr(P)} = \frac{2B}{n \Pr(P|A)} < \frac{2B}{0.18n} < \frac{1}{20}.$$

Thus, ω_2 is 1 with high probability. Hence, we expect that Step 2.b will be executed only once.

Now we analyze Step 2.c. We have $M_i = \{[a, b]\}$; hence, $a \leq m_0 \leq b$, and thus:

$$\frac{2B + r_i n}{b} \leq \frac{2B + r_i n}{m_0} \leq s_i \leq \frac{3B - 1 + r_i n}{m_0} \leq \frac{3B - 1 + r_i n}{a}.$$

The length of the interval $\left[\frac{2B+r_i n}{b}, \frac{3B-1+r_i n}{a}\right]$ is:

$$\frac{3B - 1 + r_i n}{a} - \frac{2B + r_i n}{b} \geq \frac{B - 1}{m_0} \geq \frac{1}{3}\frac{B - 1}{B}.$$

Therefore, we can expect to find a pair r_i, s_i that satisfies (2) for about each third value of r_i that is tried. Thus, it seems easy to find such pairs r_i, s_i that satisfy (1) and (2) just by iterating through possible values for r_i.

The probability that $s_i \in \left[\frac{2B+r_i n}{m_0}, \frac{3B-1+r_i n}{m_0}\right]$ is roughly 1/2. Thus, we will find a PKCS-conforming s_i after trying about $2/\Pr(P|A)$ chosen ciphertexts. Since the remaining interval in M_i is divided in half in each step 2.c, we expect to find m_0 with about:

$$\frac{3}{\Pr(P)} + \frac{16k}{\Pr(P|A)}$$

chosen ciphertexts, where k denotes the size of the modulus in bytes. For $\Pr(P) = 0.18 \cdot 2^{-16}$ and $k = 128$ (which corresponds to a 1024-bit modulus), we expect that the attack needs roughly 2^{20} chosen ciphertexts to succeed. The bit length of the modulus is usually a multiple of 8; hence, $\Pr(P)$ is close to $0.18 \cdot 2^{-16}$, as assumed previously.

Remarks. The probabilities in this section were computed under the assumption that the values s_i are independent of each other. We made that assumption to allow a heuristic analysis of the algorithm. However, the assumption may be wrong in special cases. For example, let us assume that m_0 and $s_i m_0$ are both PKCS conforming with padding strings of similar length; that is, we have, for some integer j:

$$m_0 = 2^{28(k-2)} + 28jPS + D,$$

$$s_i m_0 = 2^{28(k-2)} + 28jPS' + D'.$$

Then, $(2s_i - 1)m_0$ is PKCS conforming with high probability, since:

$$(2s_i - 1)m_0 = 2^{28(k-2)} + 28j(2PS' - PS) + 2D' - D,$$

which often is PKCS conforming too. We believe that such relations generally help the attacker, but in certain situations, the attack might require many more chosen ciphertexts than our analysis indicates.

Usually, the bit size of the RSA modulus is a multiple of 8. This choice is a good one because, for such a modulus, $\Pr(P)$ is small. A modulus with a bit length of $8k - 7$ would make the attack much easier, because, in that case, only about 2^{13} chosen messages would be necessary.

4 Access to an Oracle

In this section, we describe three situations in which an attacker could get access to an oracle.

4.1 Plain Encryption

Let us assume that a cryptographic protocol starts as follows. Alice generates a message m (e.g., a randomly chosen key). She encrypts it with PKCS #1, without applying any further integrity checks, and sends the ciphertext to Bob. Bob decrypts the message. If the format of the message is not PKCS conforming, then he returns an error; otherwise, he proceeds according to the protocol.

If Eve impersonates Alice, she can easily send messages to Bob and check them for conformance. Note that Eve's attack works even when the protocol includes strong authentication at a later step, since Eve has obtained useful information before she has to respond with an authenticated message.

Note that the RSA encryption standard PKCS #1 recommends that a message digest be included before an RSA operation, but only for the signing procedure. Even though the standard mentions that an encrypted message does not ensure integrity by itself, the standard does not indicate where such an integrity check should be included.

4.2 Detailed Error Messages

Thus far, we have shown that a reliable integrity check is an important part of an RSA encryption. One way to include such a check is to let the sender sign the message with his private key, before he encrypts it with the receiver's public key. Then, an attacker can no longer hope to create a correct message by accident. However, the attack will nonetheless be successful when, in the case of a failed verification, the receiver returns an error message that gives detailed information about where the verification failed.

In particular, it would compromise security to return different error messages for a message that is not PKCS conforming and for a message where only the signature verification failed.

4.3 A Timing Attack

Certain applications combine encryption and signatures. In such cases, a reliable integrity check often is part of the signature, but is not included in the encryption.

Let us assume that an encrypted message c is decrypted and verified as shown in the following pseudo-code:

```
1. Let m  c^d \pmod{n} be the RSA-decryption of c.
2. If m is not PKCS conforming, then reject.
3. Otherwise, verify the signature of m.
4. If the signature is not correct, then reject; otherwise, accept.
```

An attacker will not be able to generate a chosen ciphertext c such that this message has a correct signature. However, she will be able to generate messages such that c sometimes passes the check in Step 2 and is rejected only after the signature is checked. Hence, by measuring the server's response time, an attacker could determine whether c is PKCS conforming.

This timing attack is much easier to perform than Kocher's timing attack , which measures the time difference of single modular multiplications— a small fraction of the time used for one exponentiation. In our case, however, we have to distinguish between performing only a decryption and performing both a decryption and a signature verification.

In the worst case, the time for the signature verification could be significantly longer than the time for the decryption - when, for example, we have a 512-bit encryption key because of export restrictions, but we use a 2048-bit key to ensure strong authentication. In addition, the attacker can choose what signing key is sent to the server.

References

[1] S. Fluhrer, M. C. D. F. Anca, "Practical Padding Oracle Attacks on RSA," *University of Venice*, November 2012. https://secgroup.dais.unive.it/wp-content/uploads/2012/11/Practical-Padding-Oracle-Attacks-on-RSA.html.

[2] D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1," *Stanford University*, 1998.

[3] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, E. Tews, "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks," *USENIX Security Symposium*, 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer.