IMQ MINDED SECURITY

How to Teach Complex Ideas

AN IMQ GROUP COMPANY

together toward excellence

$ whoami

```
[leo@archlinux .emacs.d]$ groups

 Human
 Teacher
 Computer Scientist
 Security Consultant and Trainer
```

$ whoami - work

Software Security Consultant and Trainer

$ whoami - personal

- Esadecimale

  https://www.youtube.com/@esadecimale/videos

- Hexdump

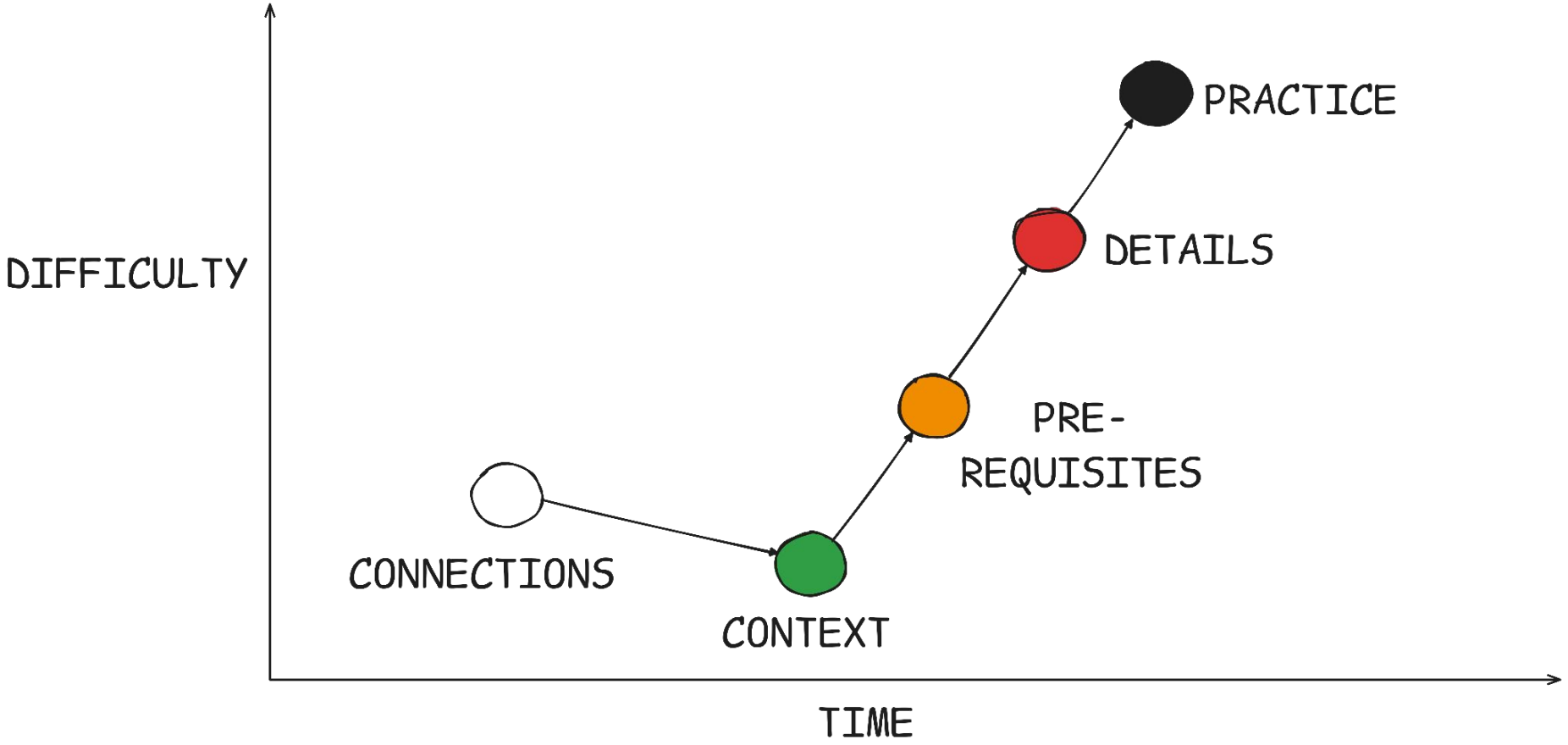
  https://www.youtube.com/@hexdump/videos

$ whoami - personal

What I like doing:

- Find a “cool” idea
- Study it
- Implement a simple PoC
- Teach Others

How to Teach Complex Ideas


DIFFICULTY
TIME
PRACTICE
DETAILS
PRE-
REQUISITES
CONNECTIONS
CONTEXT

Bleichenbacher
Padding Oracle Attack

1 + 1 = 0

0x00 - Connections

On Teaching - Connections

Where is the meaning?

Meaning is driven by connections to things we already know

Connections

To protect **e-commerce** digital communications, in 1995 the **Netscape** released

SSL – Secure Socket Layer

Connections

Connections

SSL is a **network protocol** designed to offer **cryptographic services**.

- **Confidentiality**
- **Integrity**
- **Mutual Authentication**
- **…**

Connections

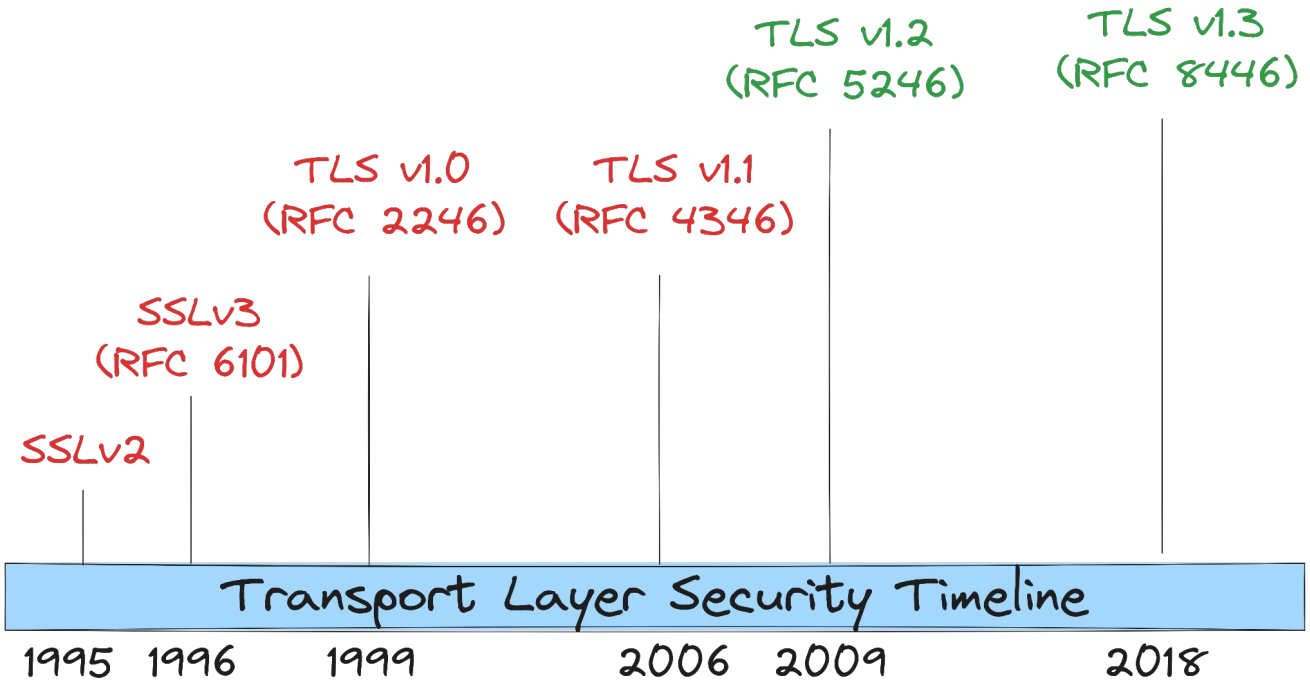
In 1999 **SSL** was standardized by the **IETF**.

SSL ⟶ **TLS**

- SSL -> Secure Socket Layer
- IETF -> Internet Engineering Task Force
- TLS -> **Transport Layer Security**

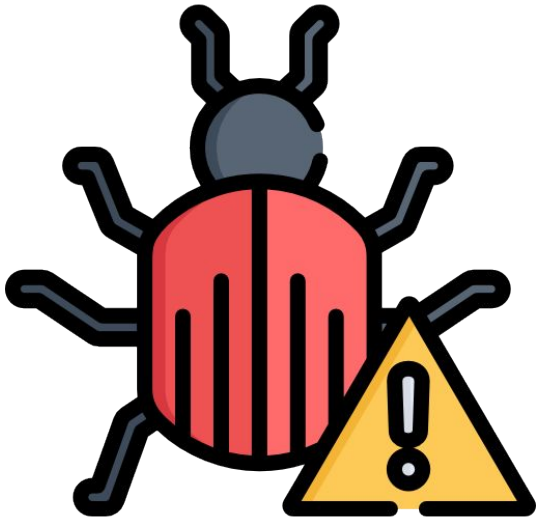
Connections

Over the years new versions of **TLS** were developed.



Connections

Over the years various **vulnerabilities** have been discovered within the **SSL/TLS stack**.



Connections

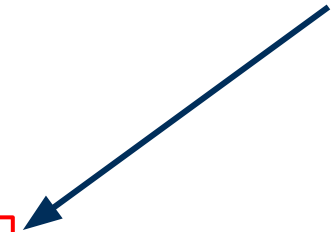
Classes of SSL/TLS Vulnerabilities:

- **Design**
  - Insecure Renegotiation
- **Implementation**
  - Heartbleed
  - Early CCS
- **Cryptography Usage**
  - CBC Padding Oracle
  - BEAST Attack

Connections

Classes of SSL/TLS Vulnerabilities:

- **Design**
  - Insecure Renegotiation
- **Implementation**
  - Heartbleed
  - Early CCS
- **Cryptography Usage**
  - CBC Padding Oracle
  - BEAST Attack



Let us focus on these ones!

Connections

Let us focus on the cryptographic attack known as

Bleichenbacher's Oracle Attack

also known as

The Million Message Attack

Connections

Discovered in 1998 by **Daniel Bleichenbacher**.

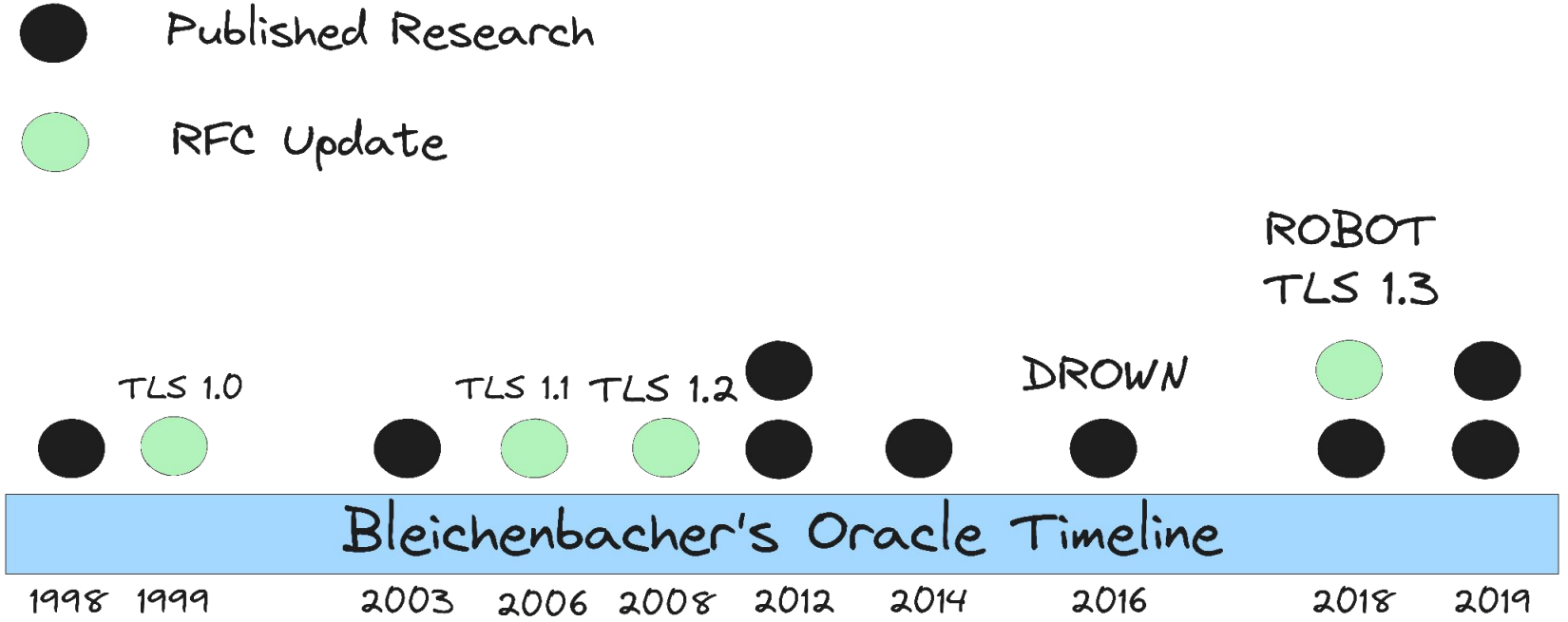
Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1

Daniel Bleichenbacher

Bell Laboratories
700 Mountain Ave., Murray Hill, NJ 07974
bleichen@research.bell-labs.com

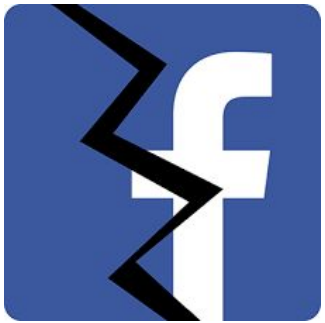
Connections

It has **resurfaced multiple times** over the years.



Connections

In 2018, **Hanno Böck**, **Juraj Somorovsky** and **Craig Young** used a version of this attack to use facebook's private key!



Can you actually prove that Facebook was vulnerable?

We were able to sign a test message with Facebook's private key.

You don't have to take our word for it; we have cryptographic proof. Just use these commands:

Connections

It has **resurfaced multiple times** over the years.

Windows into the Past: Exploiting Legacy Crypto in Modern OS's Kerberos Implementation

Michal Shagam and Eyal Ronen, *Tel Aviv University*

https://www.usenix.org/conference/usenixsecurity24/presentation/shagam

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

0x01 - Context

On Teaching - Context

A well-defined **context** allows to the mind to reach the **correct meaning** with **less effort**.

Context

Main ideas behind the **TLS protocol**

TLS Handshake

- Key Exchange
- Authentication

TLS Session

- Confidentiality
- Integrity

Context - TLS Handshake

- Exchange of Capabilities
- Authentication (Public Key Infrastructure)
- Key Exchange
  - RSA
  - DHE
  - ECDHE

Context - TLS Handshake

Context - TLS Handshake

The attack can be applied only when **RSA** is used to encrypt the **ClientKeyExchange** message.

RFC 5246 TLS August 2008

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa:
            EncryptedPreMasterSecret;
        case dhe_dss:
        case dhe_rsa:
        case dh_dss:
        case dh_rsa:
        case dh_anon:
            ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```

RFC 5246
TLS v1.2

Context

The **Bleichenbacher's Oracle Attack** is an

Adaptive Chosen Ciphertext Attack

Context

Adaptive Chosen Ciphertext Attack

- **Adaptive**: the attack has various phases, where each phase depends on the outcome of the previous.

- **Chosen Ciphertext**: the attack works by constructing specific ciphertext messages.

Context

It allows an attacker to
forcefully decrypt an encrypted message.

*** no need to know the private key of the server.**

Context

The attack scenario on TLS

1. Sniff encrypted handshake + session.

2. **Use padding oracle oracle to decrypt shared symmetric key.**

3. Decrypt entire TLS session.

Context

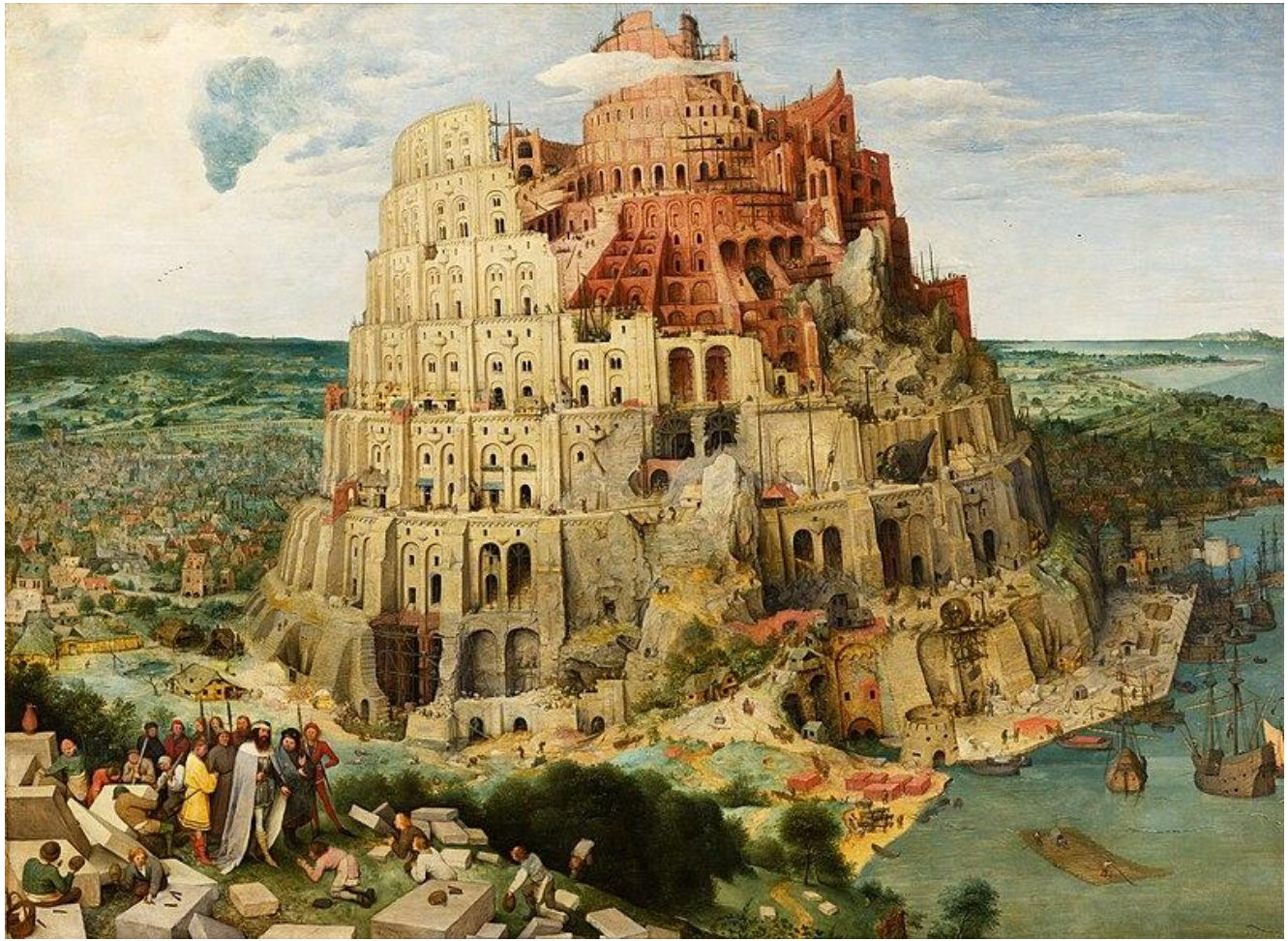
In the latest version (**TLS v1.3**), the **RSA** key exchange option has been removed.

RFC 8446

TLS v1.3

0x02 - Pre-Requisites

You cannot teach it all



Pre-Requisites

To fully understand the attack we need

- **RSA Encryption**
- **PKCS #1 v1.5 Padding Scheme**
- **Padding Oracle**

RSA

RSA

Public-key cryptography scheme used to:

- **Encrypt** messages (confidentiality)
- **Sign** messages (integrity, authenticity)

RSA

In RSA, messages are numbers

```
Hello World          -> 2342312312333…
This is a secret!    -> 7192391239112…
Lisp is cool btw     -> 5123123123123…
```

RSA

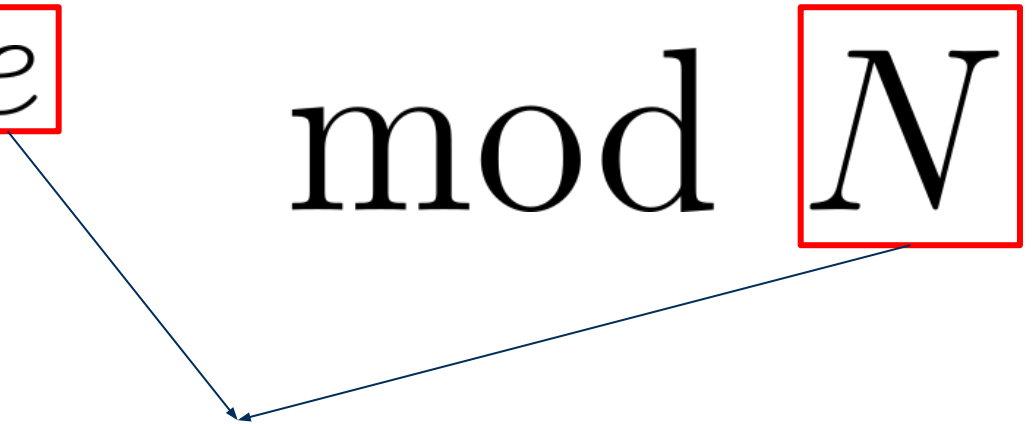
Modular Arithmetic

$$9 + 9 \bmod 12 = 6$$
$$7 + 7 \bmod 12 = 2$$
$$1 + 6 \bmod 12 = 7$$

RSA

Encryption Computation

$$c = m^{e} \mod N$$



Public key

RSA

Decryption Computation

$$m = c^{d} \mod N$$

Private key

PKCS #1 v1.5

PKCS #1 v1.5

Textbook RSA is completely deterministic

```
t1:This is a secret! -> 719239123911…
t2:This is a secret! -> 719239123911…
```

We lose **semantic security**!

PKCS #1 v1.5

In 1993 the scheme **PKCS #1 v1.5** was standardized.

Obsoleted by: 2437 INFORMATIONAL

Network Working Group B. Kaliski
Request for Comments: 2313 RSA Laboratories East
Category: Informational March 1998

PKCS #1: RSA Encryption
Version 1.5

PKCS #1 v1.5

The rules of **PKCS #1 v1.5**

$$0\text{x } 00\ 02 \mid \underbrace{R_1 \ \ldots \ R_i}_{\geq 8} \mid 00 \mid \underbrace{M_1 \ \ldots \ M_j}_{\leq k-11}$$

***k = byte length of modulus N**

PKCS #1 v1.5

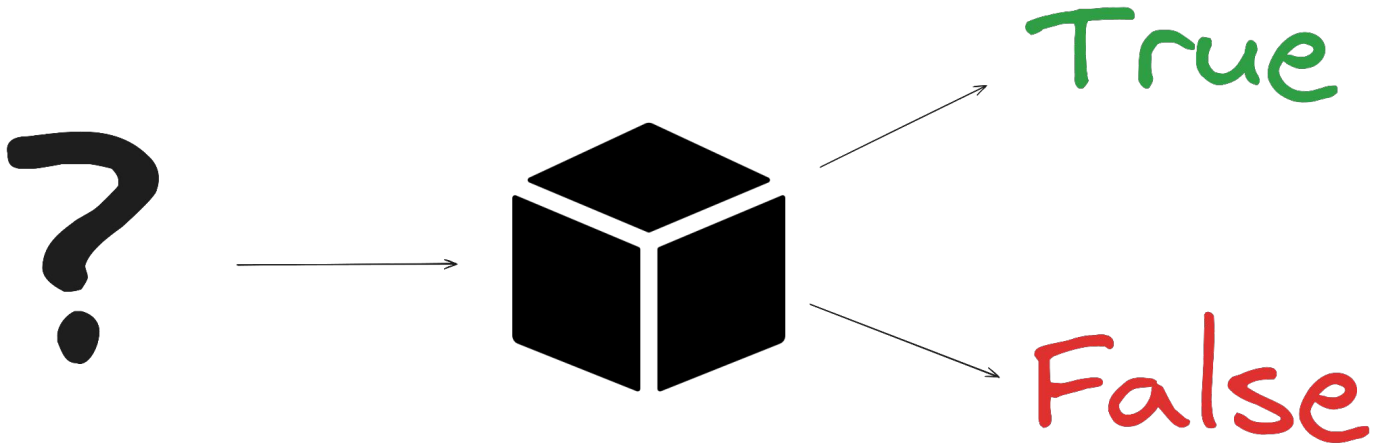
In code

```
def pkcs(msg):
    padded_msg = b""
    random_len = K - len(msg) - 3

    if random_len < 8:
        print(f"[ERROR] - Message is too long for given K")
        exit()

    padded_msg += b"\x00" + b"\x02"
    padded_msg += secrets.token_bytes(random_len)
    padded_msg += b"\x00"
    padded_msg += msg.encode()

    return padded_msg
```

Padding Oracles

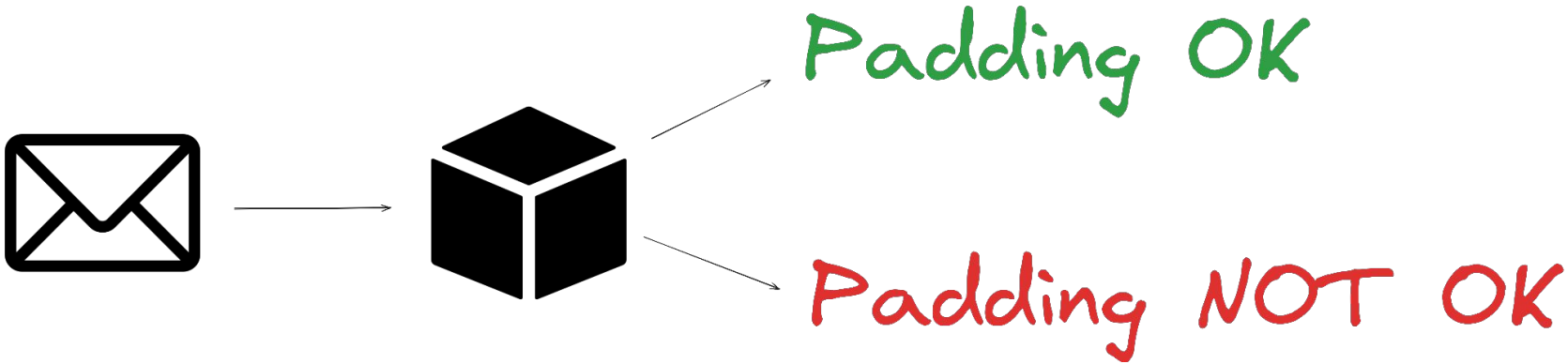
Padding Oracles

An **oracle** is a **black box** that can **answer a specific question**.



Padding Oracles

A **padding oracle** answers questions related to the **padding of messages**.



Padding Oracles

Example

```
m1 -> 0x 00 02 4A 20 FA BC …

m2 -> 0x 05 FF 02 03 04 05 …
```

Padding Oracles

Example

Padding Oracles

In code

```
def oracle(msg_hex):
    global D, N

    # transform hex into number
    encrypted_msg = int("0x" + msg_hex, 0)

    # raw decrypt using RSA
    decrypted_msg = pow(encrypted_msg, D, N)
    decrypted_hex = f"%0{PADDING_VALUE}x" % decrypted_msg

    # check for padding
    if decrypted_hex[0:4] != "0002":
        return False
    else:
        return True
```

Padding Oracles

Padding is checked on plaintext!

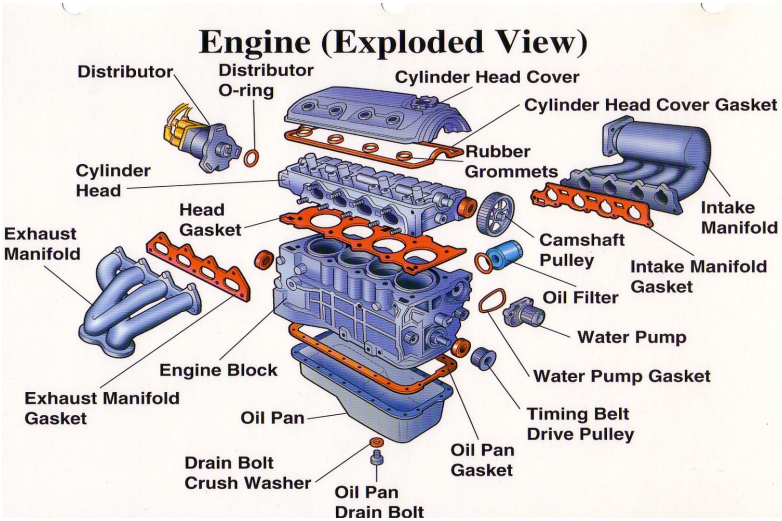
$$\texttt{encrypted message} \rightarrow \texttt{padded message}$$
$$\rightarrow \texttt{plaintext message}$$

0x04 - Details

On Teaching - Details

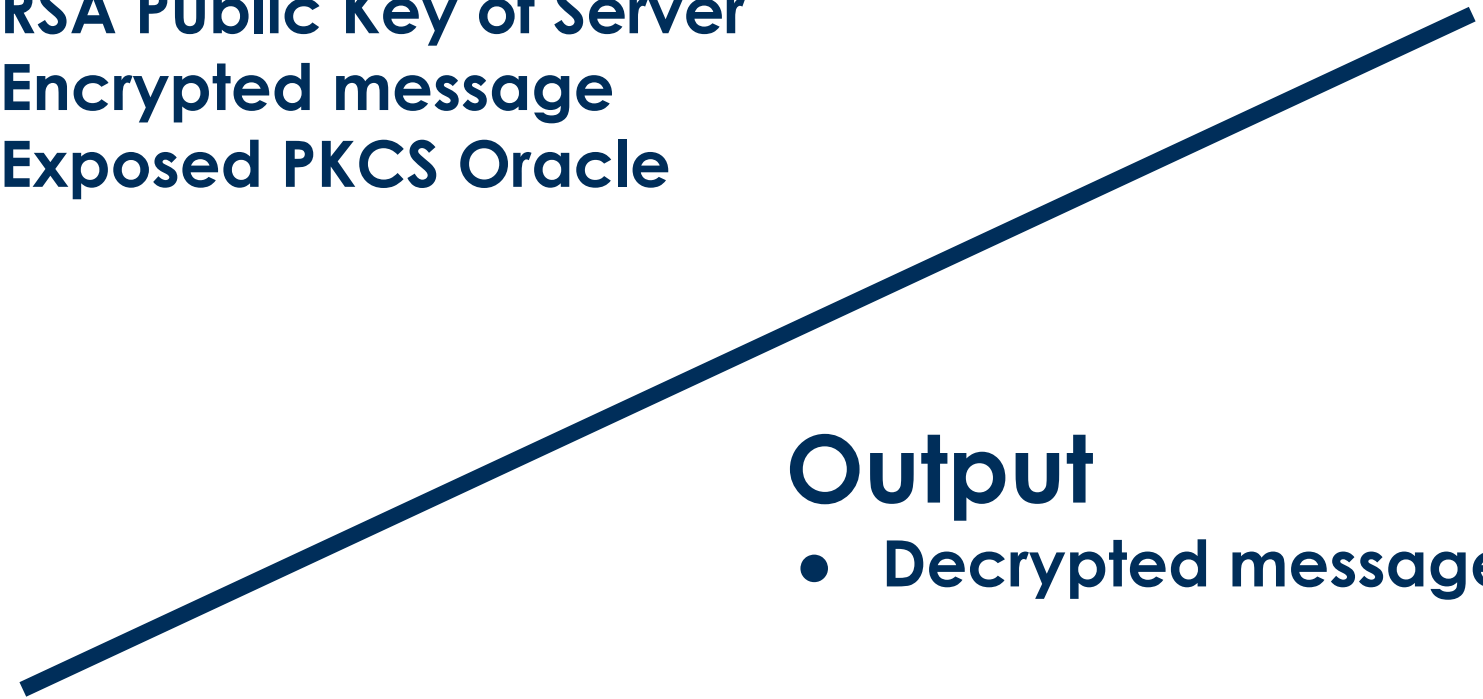
Abstractions

Details



Inputs

- RSA Public Key of Server
- Encrypted message
- Exposed PKCS Oracle



Output

- Decrypted message

Objective

Find the original message **m**.

$$c \rightarrow \mathrm{PKCS}(m) \rightarrow m$$

Consequences of RSA

Consequences of RSA

The **laws of exponents** in algebra states that

$$a^c \cdot b^c = (a \cdot b)^c$$

Consequences of RSA

In RSA **encryption is implemented with exponentiation**.

$$c = m^e \mod N$$

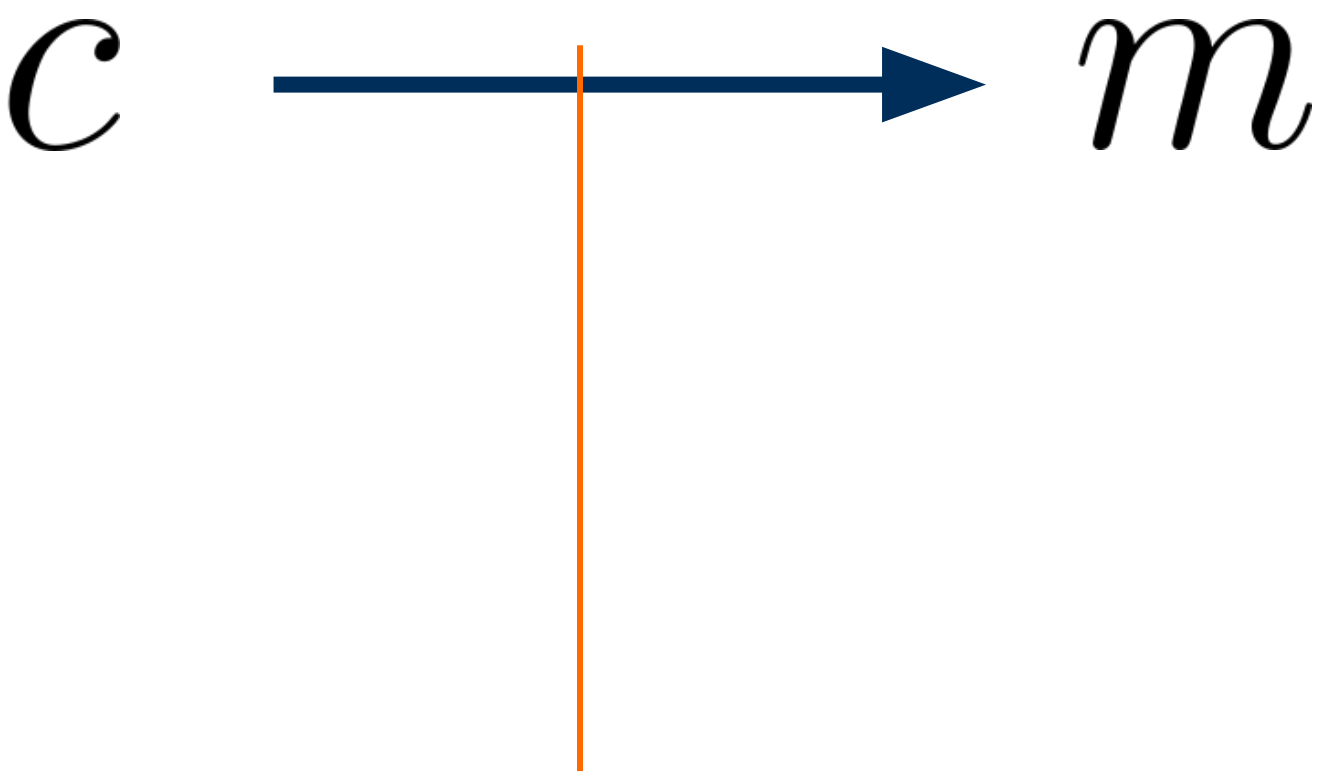
Attacker PoV

Server PoV

c

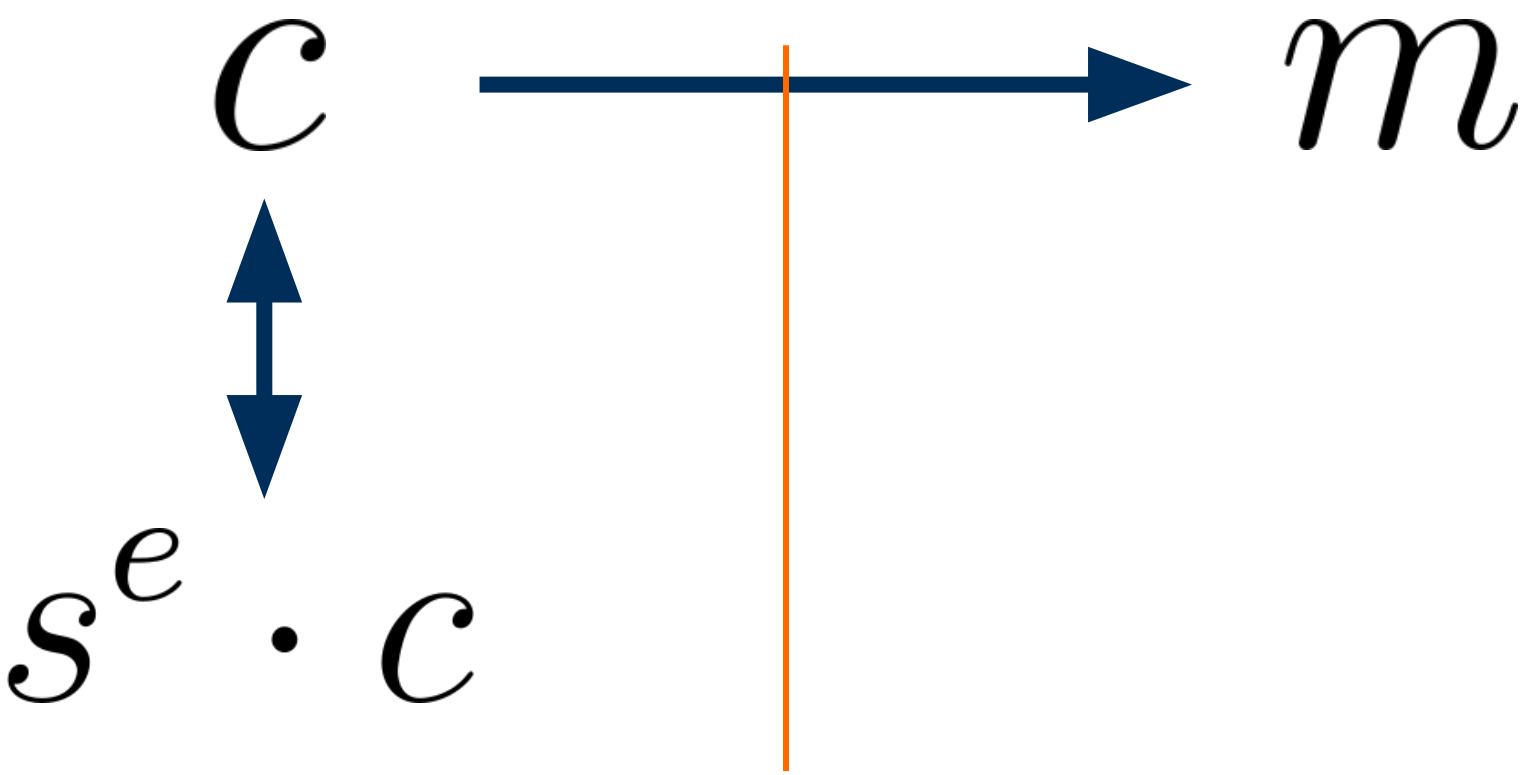
Attacker PoV

Server PoV



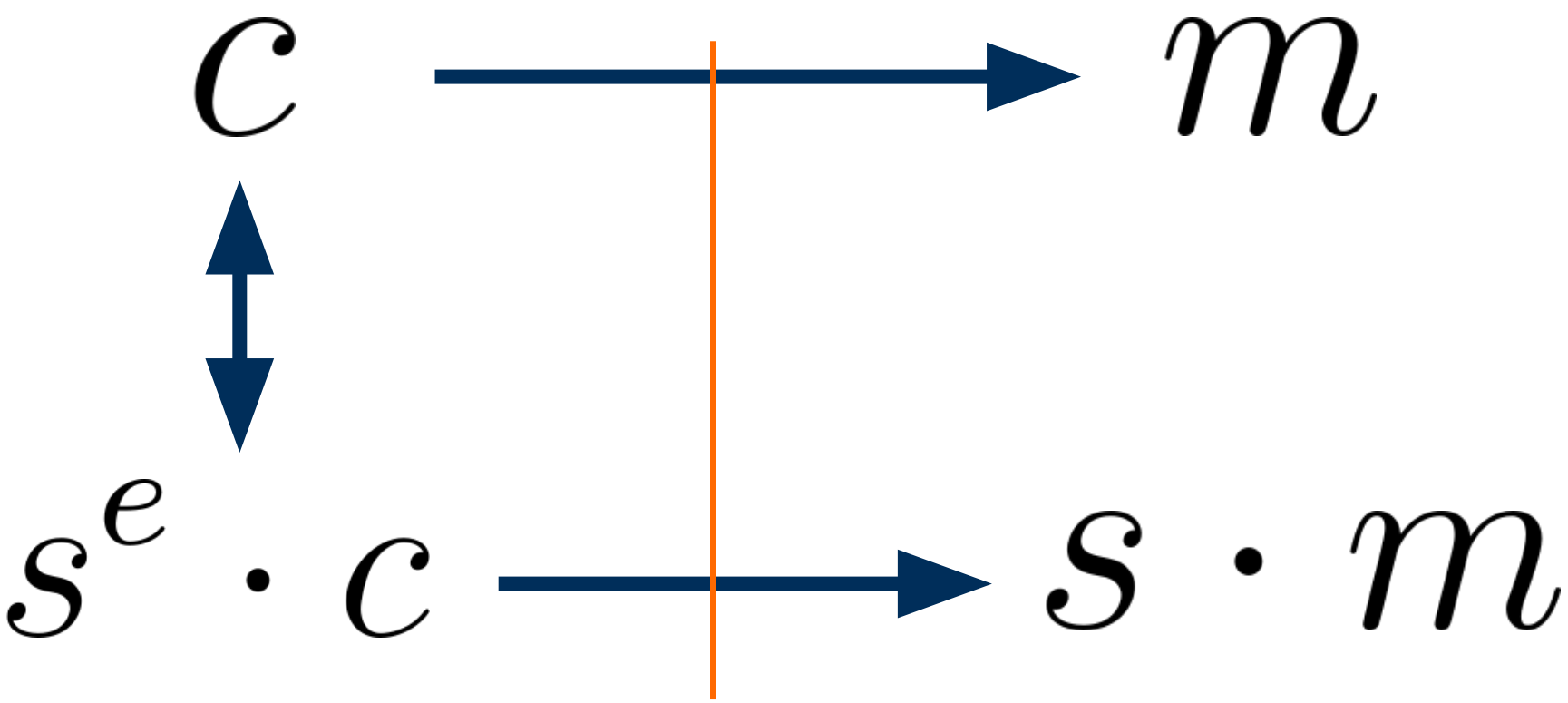
Attacker PoV

Server PoV



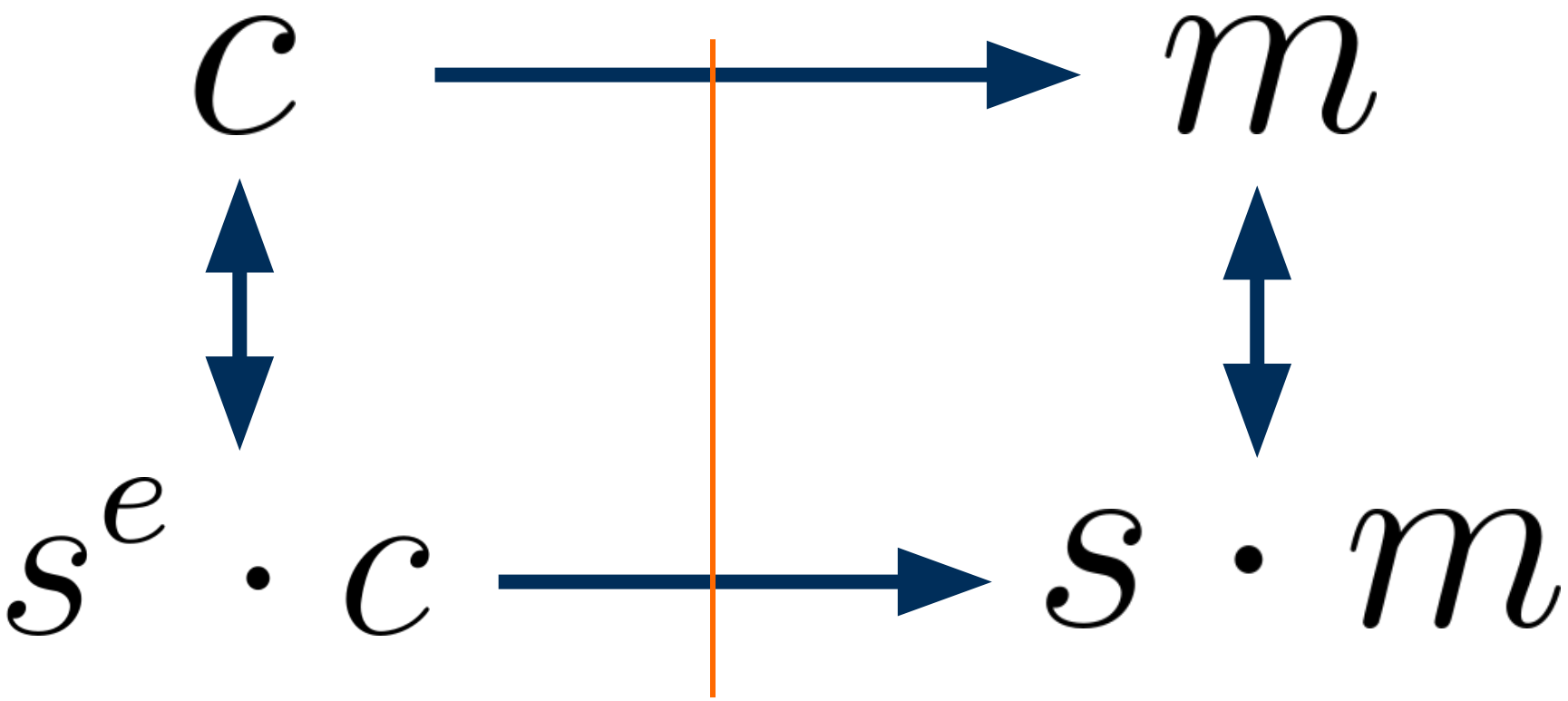
Attacker PoV

Server PoV



Attacker PoV

Server PoV



Consequences of PKCS#1 v1.5

Consequences of PKCS #1 v1.5

Remember the rules of **PKCS #1 v1.5**

$$0\text{x } 00\ 02 \mid R_1 \ \ldots \ R_i \mid 00 \mid M_1 \ \ldots \ M_j$$

Consequences of PKCS #1 v1.5

Remember the rules of **PKCS #1 v1.5**

$$\boxed{0\text{x } 00\ 02} \mid R_1 \ \ldots \ R_i \mid 00 \mid M_1 \ \ldots \ M_j$$

Valid messages start with 0x 00 02!

Consequences of PKCS #1 v1.5

```
        0x 00 02 00 00 00 00 …
        …
m -> 0x 00 02 02 03 04 05 …
        …
        0x 00 02 FF FF FF FF …
```

Consequences of PKCS #1 v1.5

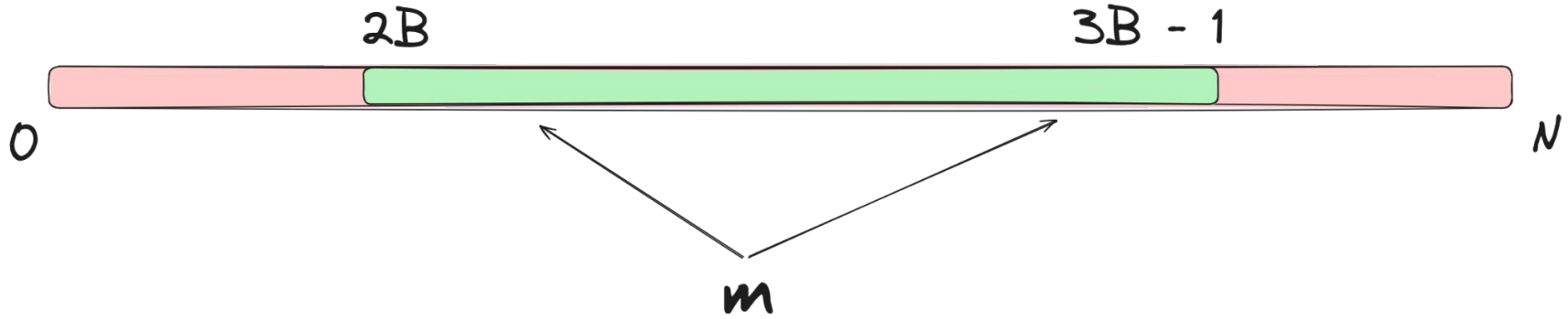
If **m** is properly padded,
we have a **numerical bound on the plaintext**!

$$2B \leq m \leq 3B - 1$$

```
KEY_BIT_SIZE = 1024
B = 2 ** (8 * (KEY_BYTE_SIZE - 2))
B2, B3 = 2*B, 3*B
```

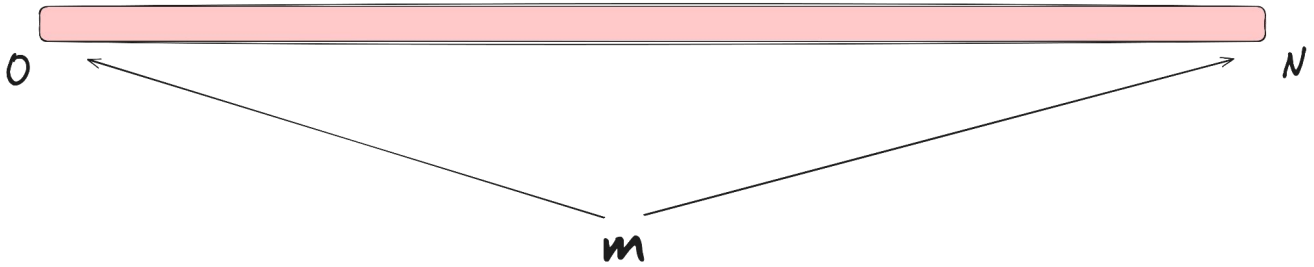
Consequences of PKCS #1 v1.5

If **m** is properly padded,
we have a **numerical bound on the plaintext**!

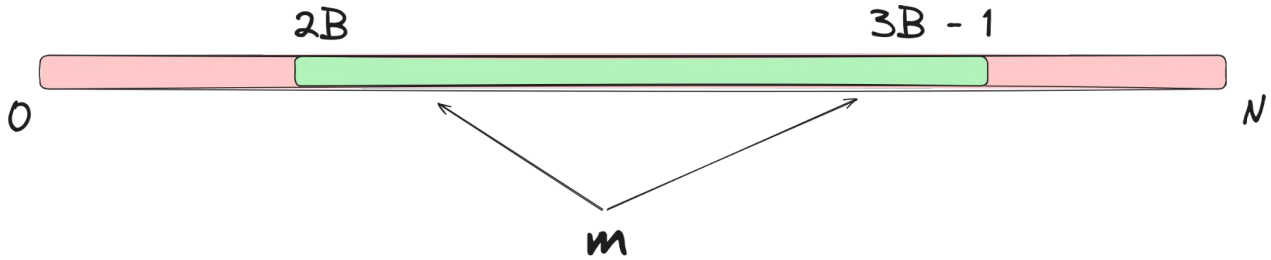


Consequences of PKCS #1 v1.5

Before knowledge of valid PKCS padding



After knowledge of valid PKCS padding



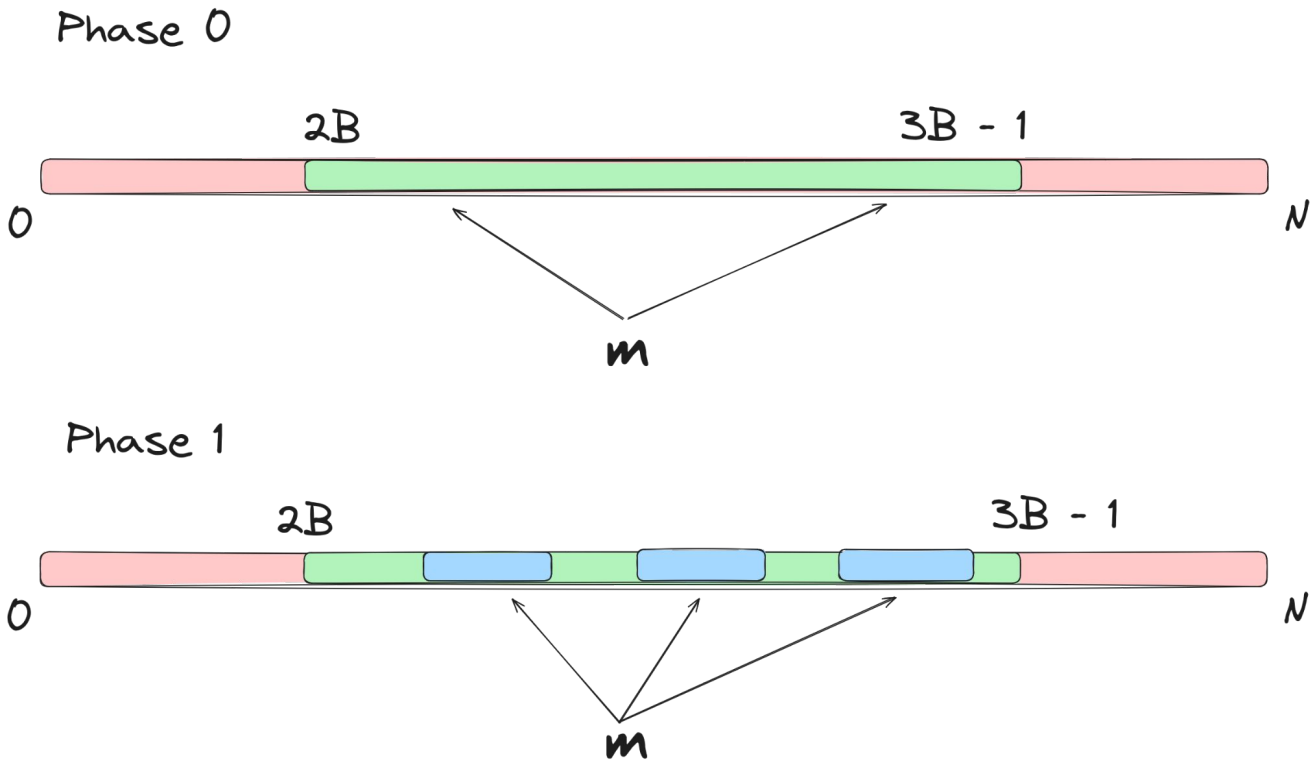
Decryption Algorithm

Decryption Algorithm

The decryption algorithm works in **phases**.

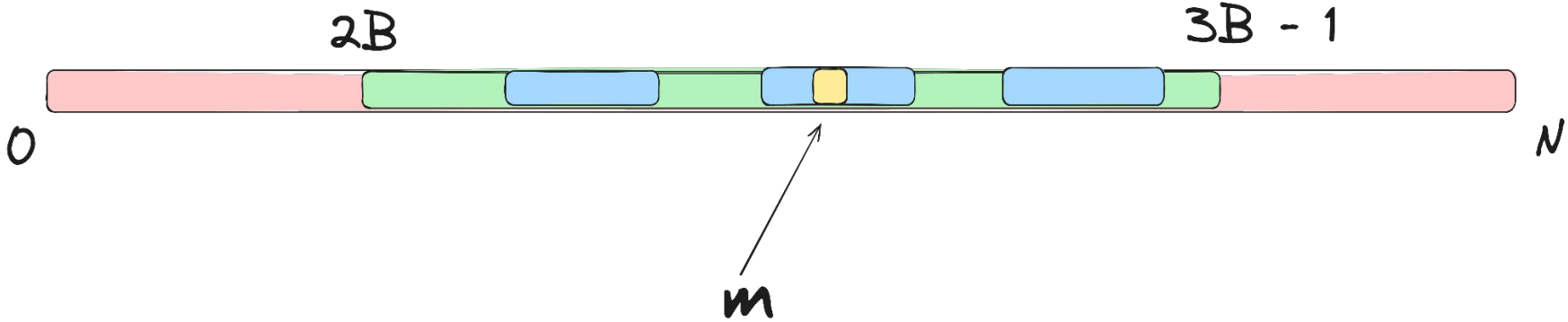
Each phase computes a set of intervals that contain the plaintext.

Decryption Algorithm



Decryption Algorithm

In the last phase we have a single valid interval that contains a single number.



This remaining number is the decrypted plaintext!

Decryption Algorithm

Each phase has two steps

- **Step 1:** Find a number

- **Step 2:** Construct a set of intervals

Decryption Algorithm - Initialization

Initialization

$$s_0 = 2$$

$$M_0 = \{ \ [2B, 3B] \ \}$$

Decryption Algorithm - Step 1

The first step finds a **value** such that

$$\boxed{s_i} \cdot m \mod N$$

is **PKCS#1 v1.5** compliant.

Decryption Algorithm - Step 1

The attacker sends the **modified ciphertext**

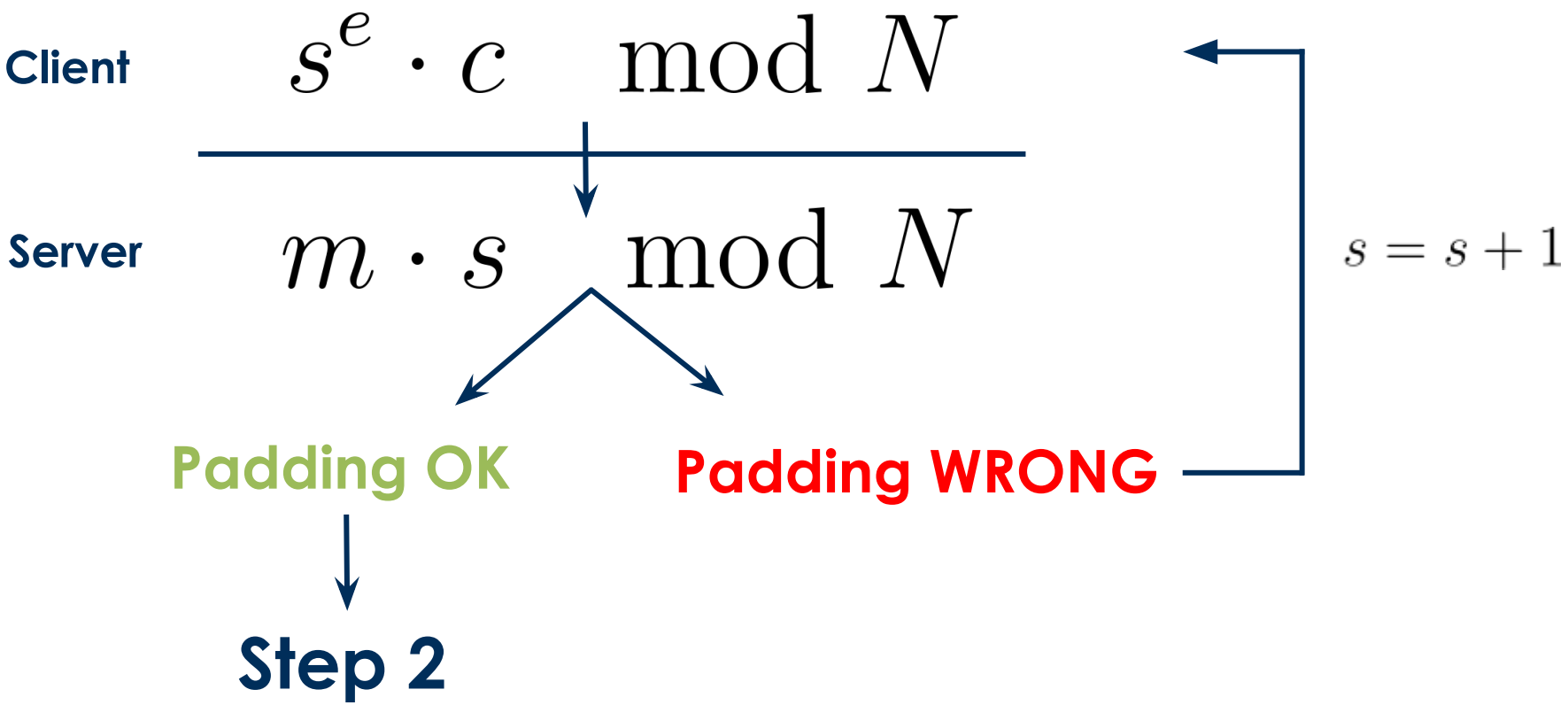
$$s^e \cdot c \mod N$$

The server will decrypt and **check the padding**

$$m \cdot s \mod N$$

Decryption Algorithm - Step 1

Decryption Algorithm - Step 1



Decryption Algorithm - Step 1

```
def bb_step_1(s):
  global E, N, c
  s = s + 1
  while True:
    new_c = (s^E * c) mod N
    if oracle(new_c):
      return s
    s = s + 1
```

Decryption Algorithm - Step 2

The second step constructs a **set of intervals** that includes the original plaintext.

$$\exists [a, b] \in M_i : \quad m \in [a, b]$$

Decryption Algorithm - Step 2

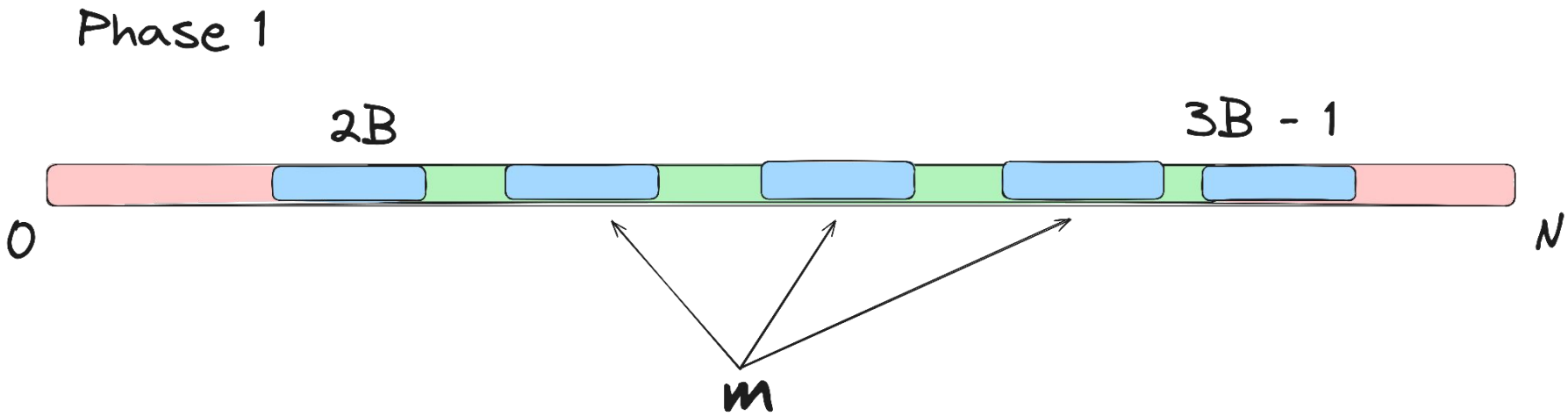
Formulas are **formal**, but not **intuitive**.

$$M_i \leftarrow \bigcup_{(a,b,r)} \left\{ \left[\max\left(a, \left\lceil \frac{2B + rn}{s_i} \right\rceil\right), \min\left(b, \left\lfloor \frac{3B - 1 + rn}{s_i} \right\rfloor\right)\right]\right\} \quad (3)$$

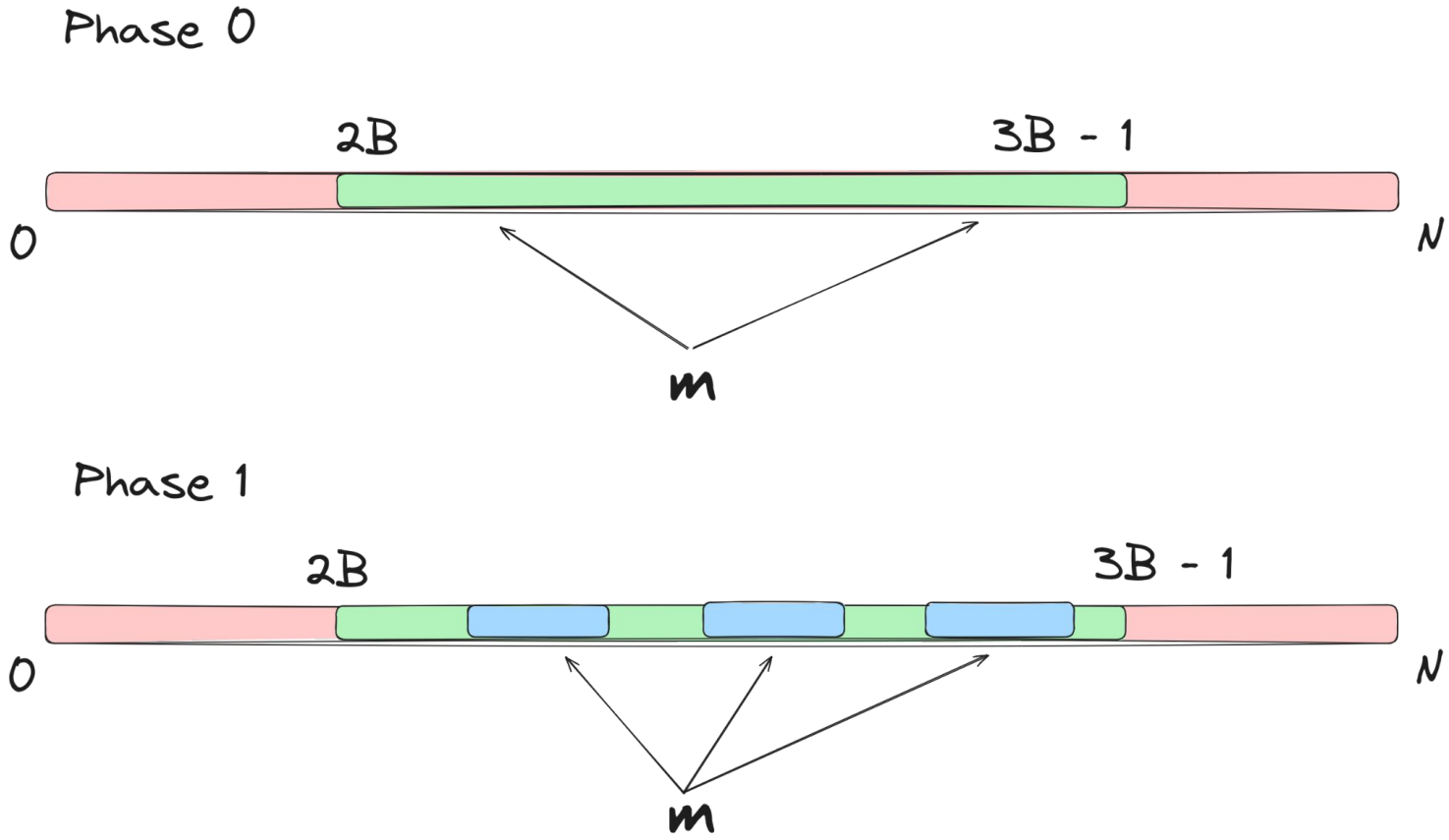
$$\text{for all } [a,b] \in M_{i-1} \text{ and } \frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i - 2B}{n}.$$

Decryption Algorithm - Phase 2

Images can help



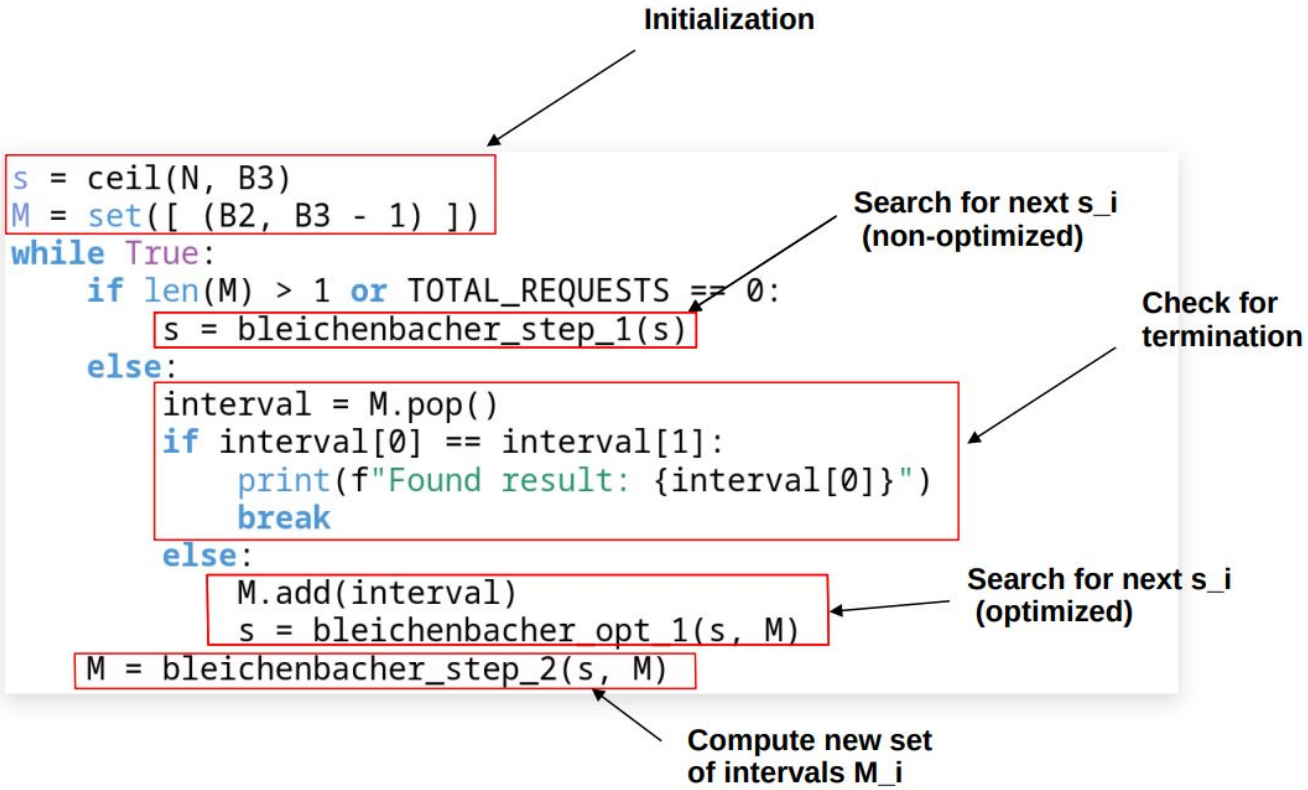
Decryption Algorithm - Step 2



Decryption Algorithm - Step 2

```
def bb_step_2(s, old_M):
    new_M = set([])
    for (a, b) in old_M:
        r1 = ceil((a * s - B3 + 1), N)
        r2 = floor((b * s - B2), N) + 1
        for r in range(r1, r2):
            aa = ceil(B2 + r*N, s)
            bb = floor(B3 - 1 + r*N, s)
            newa = max(a, aa)
            newb = min(b, bb)
            if newa <= newb:
                new_M |= set([ (newa, newb) ])
    return new_M
```

Decryption Algorithm - Full Code



Decryption Algorithm

The **bottleneck** is in the **phase 1**.

Decryption Algorithm

How many phases?

$$2^{20} \approx 10^{6}$$

Decryption Algorithm

Over the years,
optimizations techniques were discovered.

- Tiger Bounds
- Beta Method
- Parallel Threads Methods
- Skipping Holes
- Trimmers

Decryption Algorithm

What I did not cover (for time):

- Blinding step
- Optimized search for next s
- …

0x05 - Practice

Practice

Develop a **safe environment** to try the ideas of the lecture against reality.

Practice

Reality is Brutal.

*but rewarding (sometimes)

Practice

Host: bb.esadecimale.it
Port: 1337
Description: Ask, and I shall answer.

---

nc bb.esadecimale.it 1337

Practice

CVE-2016-0704,
CVE-2016-0800
CVE-2017-6168,
CVE-2017-17305
CVE-2018-16868
CVE-2019-1563
CVE-2023-46809
…

0x06 - Takeaways

Takeways

DO NOT USE RSA WITH PKCS#1 V1.5!

If you want to learn applied crypto

- https://cryptohack.org/
- https://cryptopals.com/

the cryptopals crypto challenges

Set 1: Basics

Set 2: Block crypto

Welcome to the challenges

We can't introduce these any better than Maciej Ceglowski did, so read that blog post first.

We've built a collection of exercises that demonstrate attacks on real-world crypto.

0xFF - References

Research Literature

- Original Bleichenbacher (CRYPTO 1998)
- Klima et al. (CHES 2003)
- Bleichenbacher's Attack Strikes Again: Breaking PKCS#1 v1.5 in XML Encryption (ESORICS 2012)
- Efficient Padding Oracle Attacks on Cryptographic Hardware (CRYPTO 2012)
- Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks (USENIX 2014)
- Return Of Bleichenbacher's Oracle Threat (USENIX 2018)
- The Dangers of Key Reuse: Practical Attacks on IPsec IKE (USENIX 2018)
- The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations (2019)
- Marvin Attack (Timing sidechannels 2023)

References

- **Platforms**
  - https://cryptohack.org/
  - https://cryptopals.com/
- **Tooling**
  - https://testssl.sh/
  - https://github.com/tlsfuzzer/tlsfuzzer
  - https://github.com/tls-attacker/TLS-Attacker
- **Reading**
  - https://blog.leonardotamiano.xyz/tech/bleichenbacher-oracle/
  - https://leonardotamiano.xyz/thesis.pdf
  - https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/bachelors-thesis_livia-capol.pdf

forum.esadecimale.it

Q?

Thanks!

moca 2024