The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations

Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong and Yuval Yarom

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

UNIVERSITY OF MICHIGAN

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

DATA 61

THE UNIVERSITY of ADELAIDE

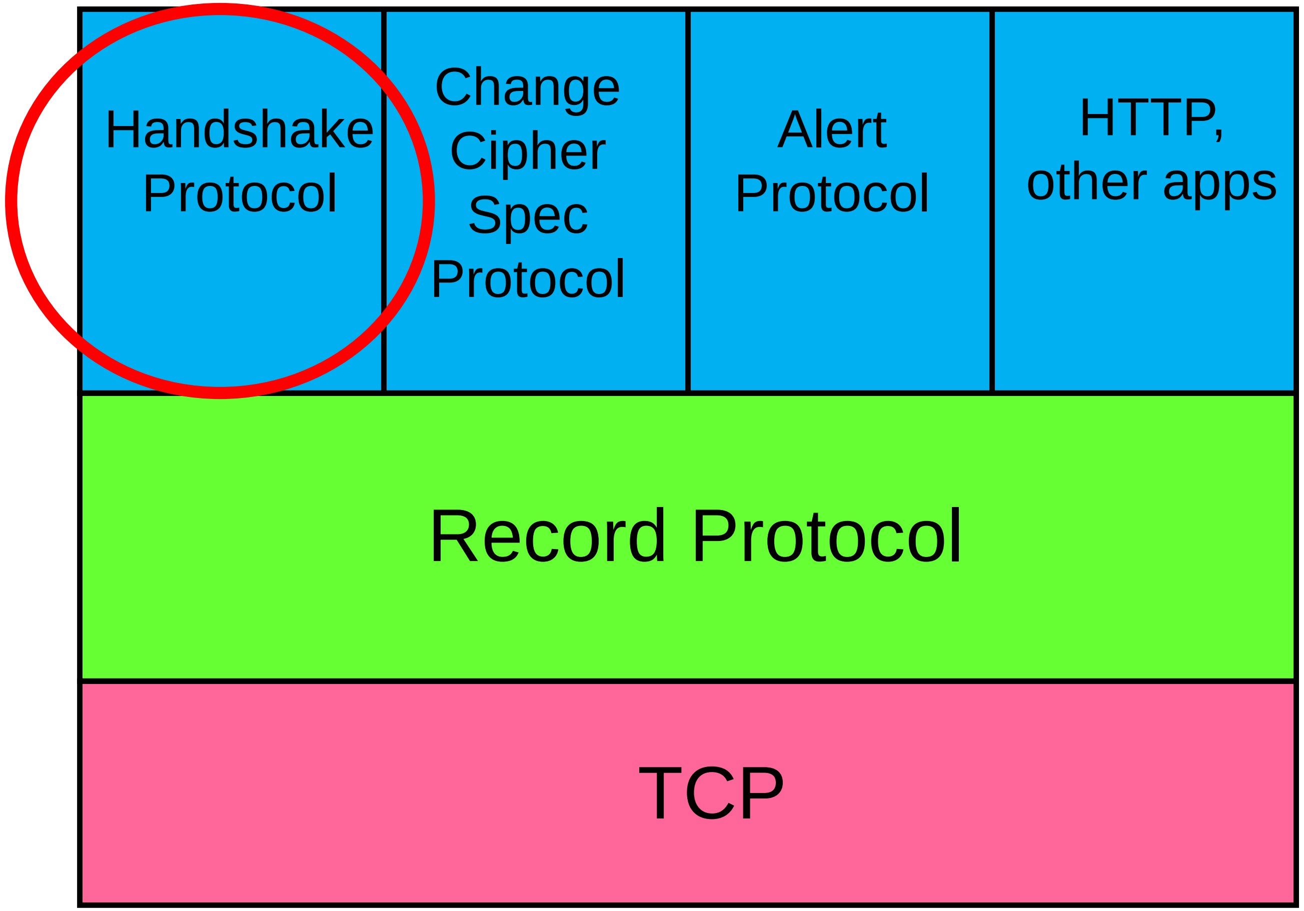
Talk Outline

1. Background
2. Attacking TLS and downgrade attack
3. RSA padding attack parallelization using CVP
4. Cache attacks on RSA padding
5. Conclusions

Transport Layer Security (TLS)

- The most widely used cryptographic protocol
- Provides communication security (https, VPN, etc.)
  - TLS handshake is used for authentication and secure key exchange
  - TLS Record layer protects the communication
  - Allows for cryptographic agility using different cipher suites

Transport Record Layer



RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme

RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme
- Once the most popular TLS key exchange option

RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme
- Once the most popular TLS key exchange option
- Long history of practical **implementation** attacks*

RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme
- Once the most popular TLS key exchange option
- Long history of practical **implementation** attacks*
- No forward secrecy

RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme
- Once the most popular TLS key exchange option
- Long history of practical **implementation** attacks*
- No forward secrecy
- Still widely used (Dec 2018)
    - ~6% by Mozilla's Telemetry and ICSI Certificate Notary

RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme
- Once the most popular TLS key exchange option
- Long history of practical **implementation** attacks*
- No forward secrecy
- Still widely used (Dec 2018)
  - ~6% by Mozilla's Telemetry and ICSI Certificate Notary
  - Better alternatives now available (e.g. Ephemeral ECDH)

RSA Key Exchange in TLS

- Uses the PKCS #1 v1.5 padding scheme
- Once the most popular TLS key exchange option
- Long history of practical **implementation** attacks*
- No forward secrecy
- Still widely used (Dec 2018)
  - ~6% by Mozilla's Telemetry and ICSI Certificate Notary
  - Better alternatives now available (e.g. Ephemeral ECDH)
  - Supported for backwards compatibility

9 lives of Bleichenbacher’s CAT

``Those who'll play with cats must expect to be scratched.'' -- Miguel de Cervantes, *Don Quixote*

9 lives of Bleichenbacher’s CAT

``Those who'll play with cats must expect to be scratched.'' -- Miguel de Cervantes, *Don Quixote*

- We tested 9 different implementations for vulnerably to cache based RSA padding attacks

9 lives of Bleichenbacher’s CAT

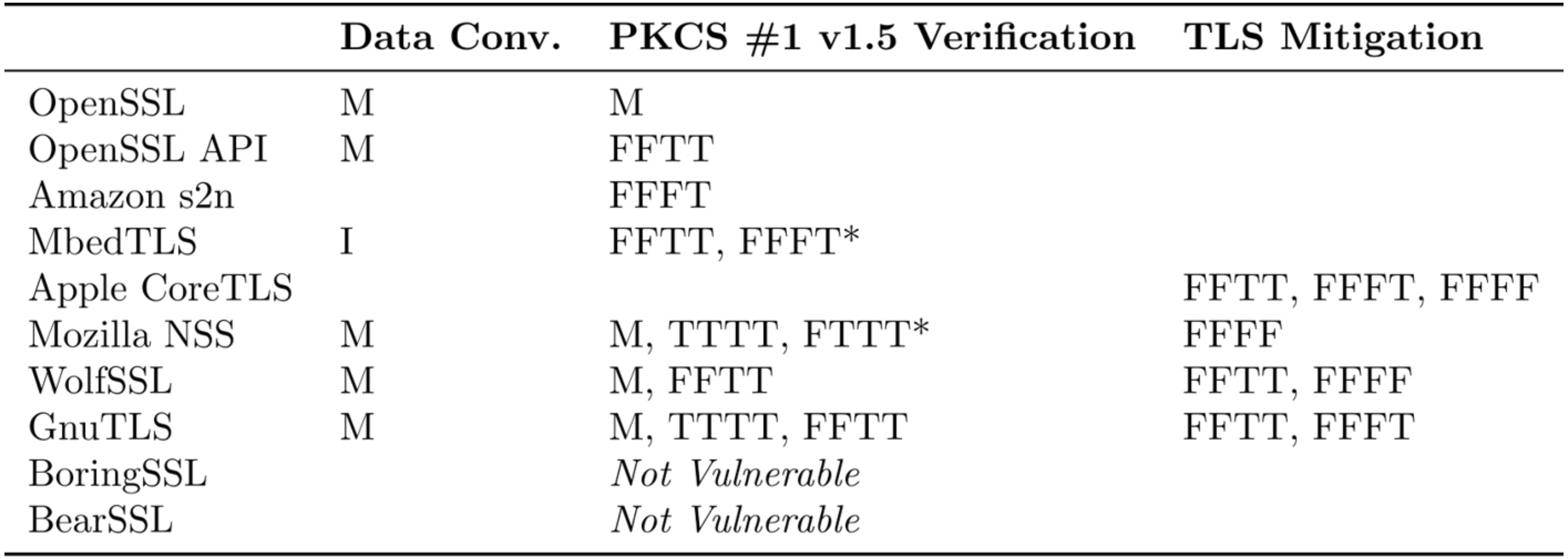
``Those who'll play with cats must expect to be scratched.'' -- Miguel de Cervantes, *Don Quixote*

- We tested 9 different implementations for vulnerably to cache based RSA padding attacks
  - Only BoringSSL and BearSSL were not vulnerable

9 lives of Bleichenbacher’s CAT

``Those who'll play with cats must expect to be scratched.'' -- Miguel de Cervantes, *Don Quixote*

- We tested 9 different implementations for vulnerably to cache based RSA padding attacks
  - Only BoringSSL and BearSSL were not vulnerable



	Data Conv.	**PKCS #1 v1.5 Verification**	**TLS Mitigation**
OpenSSL	M	M	
OpenSSL API	M	FFTT	
Amazon s2n		FFFT	
MbedTLS	I	FFTT, FFFT*	
Apple CoreTLS			FFTT, FFFT, FFFF
Mozilla NSS	M	M, TTTT, FTTT*	FFFF
WolfSSL	M	M, FFTT	FFTT, FFFF
GnuTLS	M	M, TTTT, FFTT	FFTT, FFFT
BoringSSL		*Not Vulnerable*	
BearSSL		*Not Vulnerable*	

9 lives of Bleichenbacher’s CAT

- We broke 6% of the Internet, so what?

9 lives of Bleichenbacher’s CAT

- We broke 6% of the Internet, so what?
- We show the feasibility of MiTM downgrade attack

9 lives of Bleichenbacher’s CAT

- We broke 6% of the Internet, so what?
- We show the feasibility of MiTM downgrade attack
  - Novel parallelization technique for RSA padding oracle attacks

9 lives of Bleichenbacher’s CAT

- We broke 6% of the Internet, so what?
- We show the feasibility of MiTM downgrade attack
  - Novel parallelization technique for RSA padding oracle attacks
  - Assume cache attack against multiple TLS servers

9 lives of Bleichenbacher’s CAT

- We broke 6% of the Internet, so what?
- We show the feasibility of MiTM downgrade attack
    - Novel parallelization technique for RSA padding oracle attacks
    - Assume cache attack against multiple TLS servers
    - Use BEAST to boost success probability

9 lives of Bleichenbacher’s CAT

- We broke 6% of the Internet, so what?
- We show the feasibility of MiTM downgrade attack
  - Novel parallelization technique for RSA padding oracle attacks
  - Assume cache attack against multiple TLS servers
  - Use BEAST to boost success probability
  - Break 100% of the connections that use vulnerable implantations

RSA Encryption

$$N = p \cdot q \qquad (p, q) \text{ are primes}$$

$$d \cdot e = 1 \mod \phi(N)$$

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

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

RSA Encryption

$$N = p \cdot q \qquad\qquad (p, q) \text{ are primes}$$
$$d \cdot e = 1 \mod \phi(N)$$
$$c = m^e \mod N$$
$$m = c^d \mod N$$

- Nice math, but how can we use it on real data?

RSA Encryption

$$N = p \cdot q \qquad (p, q) \text{ are primes}$$

$$d \cdot e = 1 \mod \phi(N)$$

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

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

- Nice math, but how can we use it on real data?
  - There are several real world problems

Why do we need padding

- Assume $e = 3$, $m = 1000$, $N \sim 2^{2048}$

Why do we need padding

- Assume $e = 3$, $m = 1000$, $N \sim 2^{2048}$
  - $m^e < N$, logarithm over the reals is easy
  - We need to make sure m is larger enough

Why do we need padding

- Assume $e = 3$, $m = 1000$, $N \sim 2^{2048}$
  - $m^e < N$, logarithm over the reals is easy
  - We need to make sure m is larger enough
- Assume I want to encrypt the answer to a Yes/No question – value 0 or 1

Why do we need padding

- Assume $e = 3$, $m = 1000$, $N \sim 2^{2048}$
    - $m^e < N$, logarithm over the reals is easy
    - We need to make sure m is larger enough
- Assume I want to encrypt the answer to a Yes/No question – value 0 or 1
    - Vulnerable to dictionary attack
    - Easy to detect repetitions
    - We need to make sure m is random

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext
  - Pads the plaintext to the RSA key length
  - Adds randomization

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext
  - Pads the plaintext to the RSA key length
  - Adds randomization
- Example for RSA key exchange in TLS 1.2

0x0002	[non-zero padding]	0x00	[48 bytes of premaster secret]

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext
  - Pads the plaintext to the RSA key length
  - Adds randomization
- Example for RSA key exchange in TLS 1.2

0x0002	[non-zero padding]	0x00	[48 bytes of premaster secret]

Encryption preamble

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext
  - Pads the plaintext to the RSA key length
  - Adds randomization
- Example for RSA key exchange in TLS 1.2

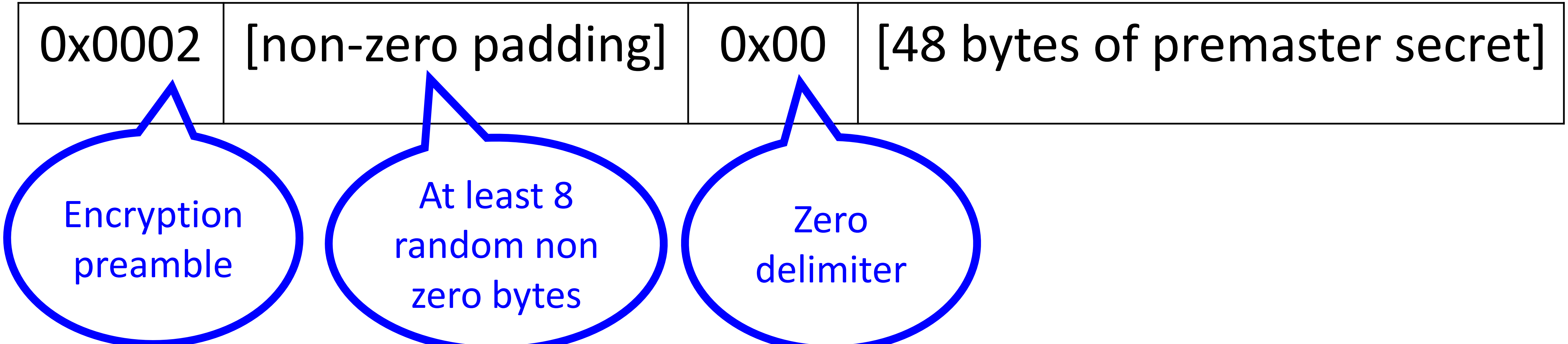
0x0002	[non-zero padding]	0x00	[48 bytes of premaster secret]

Encryption preamble

At least 8 random non zero bytes

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext
  - Pads the plaintext to the RSA key length
  - Adds randomization
- Example for RSA key exchange in TLS 1.2



0x0002	[non-zero padding]	0x00	[48 bytes of premaster secret]

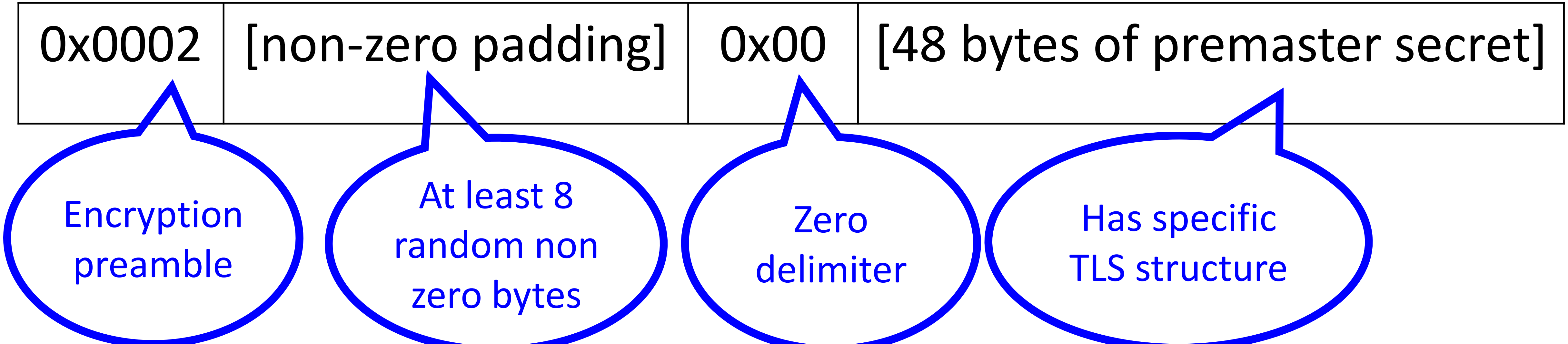
Encryption preamble

At least 8 random non zero bytes

Zero delimiter

PKCS #1 v1.5 padding scheme

- Used to pad and encrypt the plaintext
  - Pads the plaintext to the RSA key length
  - Adds randomization
- Example for RSA key exchange in TLS 1.2



0x0002	[non-zero padding]	0x00	[48 bytes of premaster secret]

Encryption preamble

At least 8 random non zero bytes

Zero delimiter

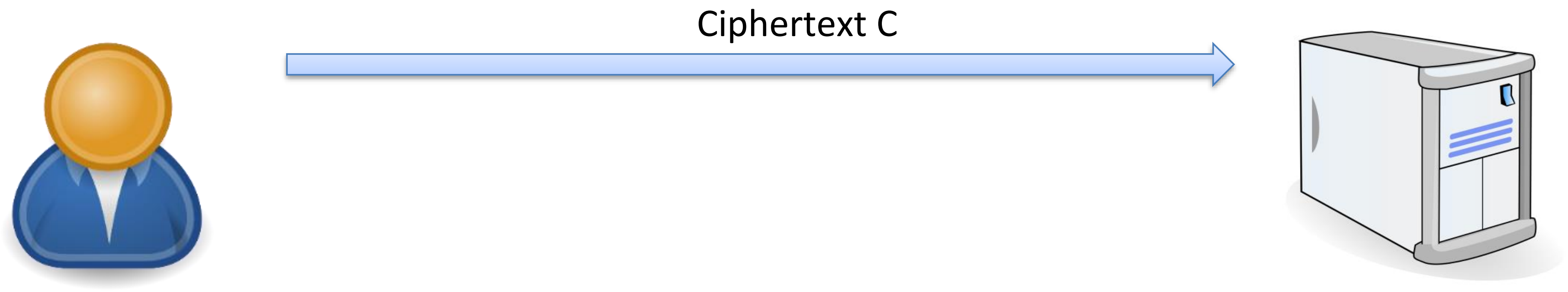
Has specific TLS structure

Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation

Bleichenbacher’s Attack

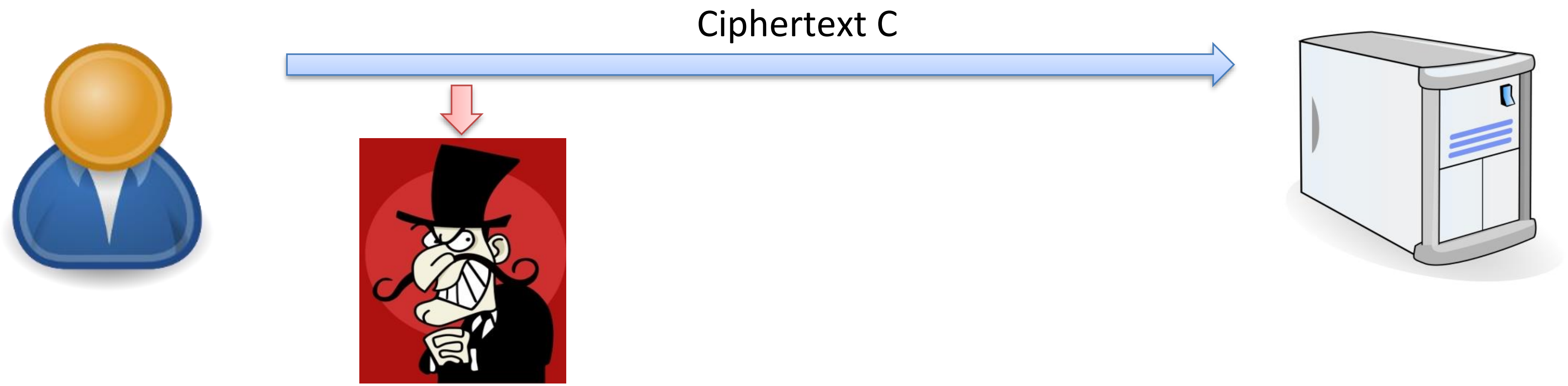
- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Ciphertext C

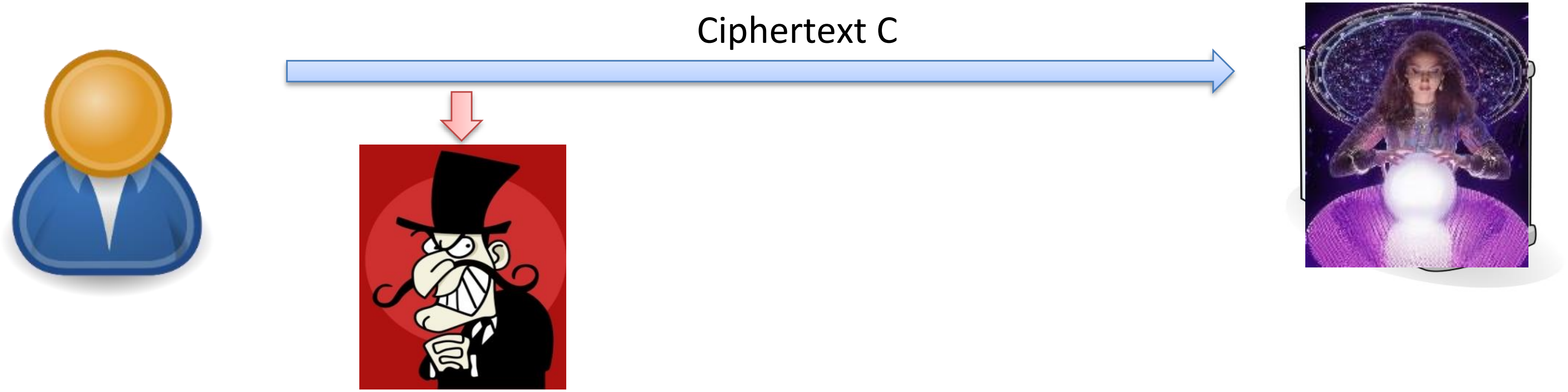
Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



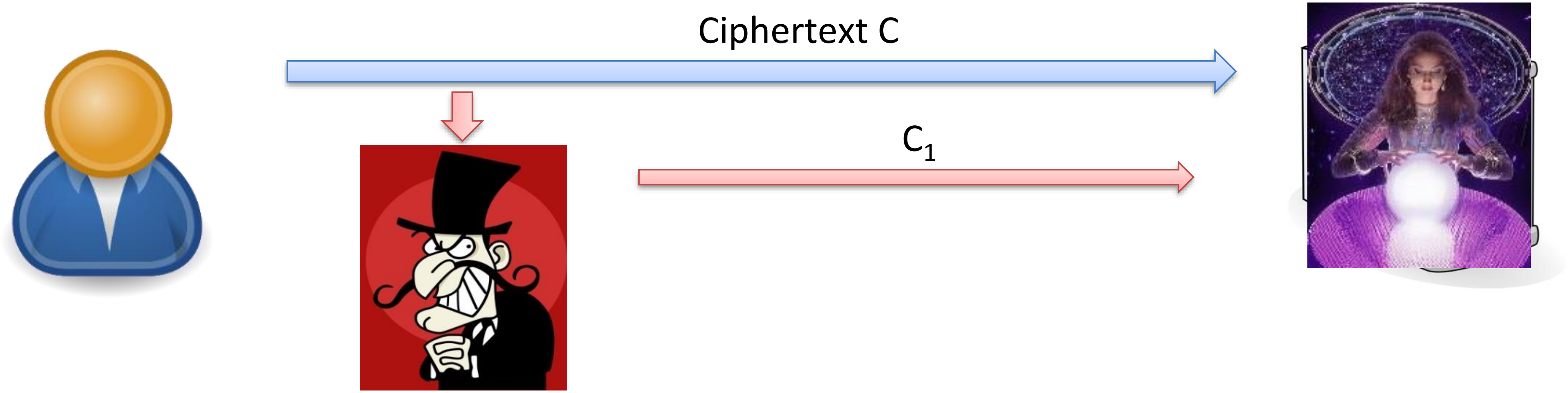
Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



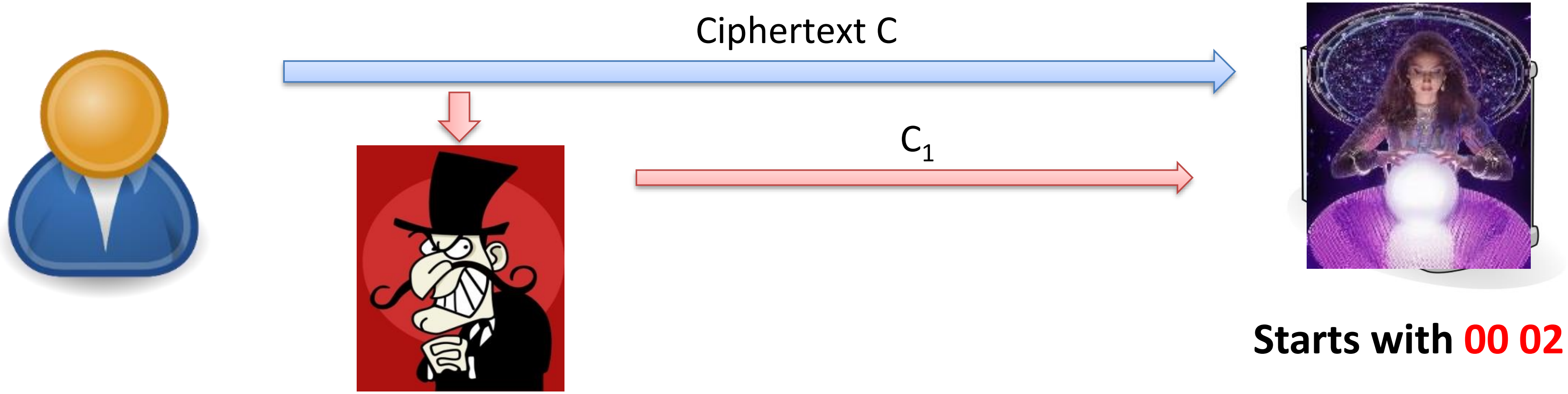
Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



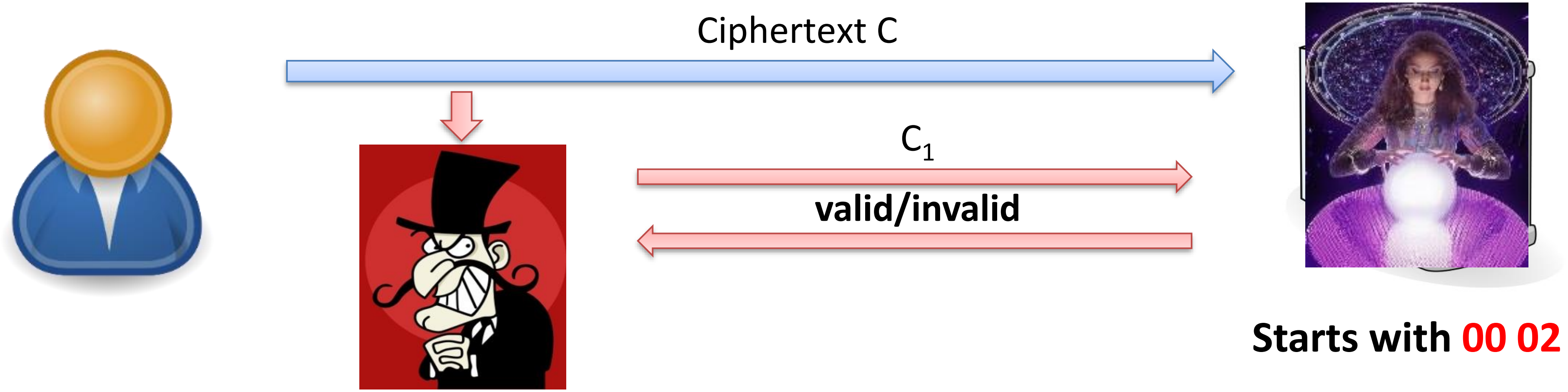
Ciphertext C

C_1

Starts with 00 02 ?

Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Ciphertext C

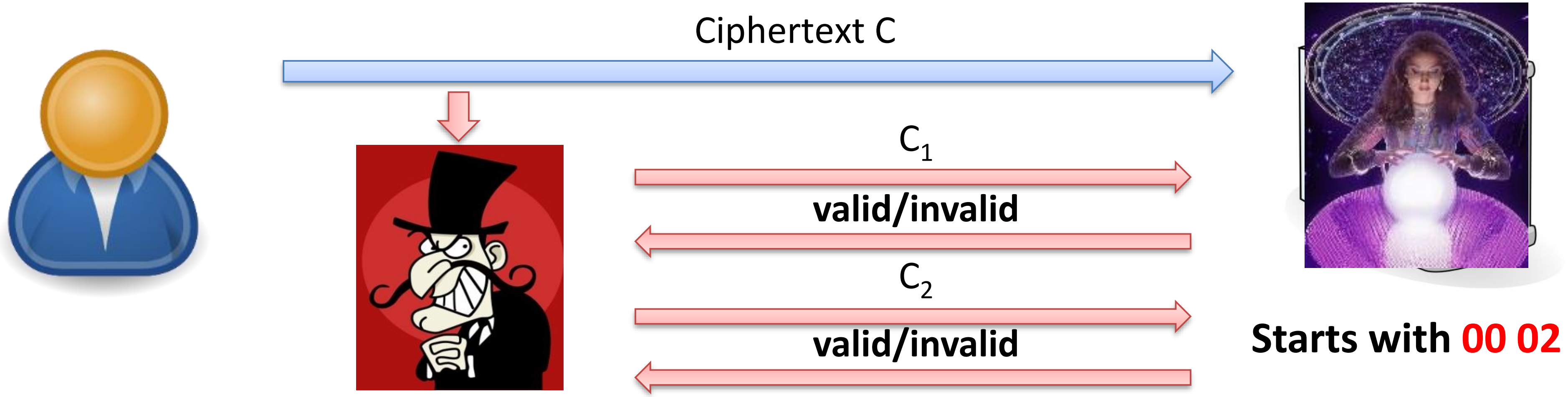
C_1

valid/invalid

Starts with 00 02 ?

Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Ciphertext C

C_1

valid/invalid

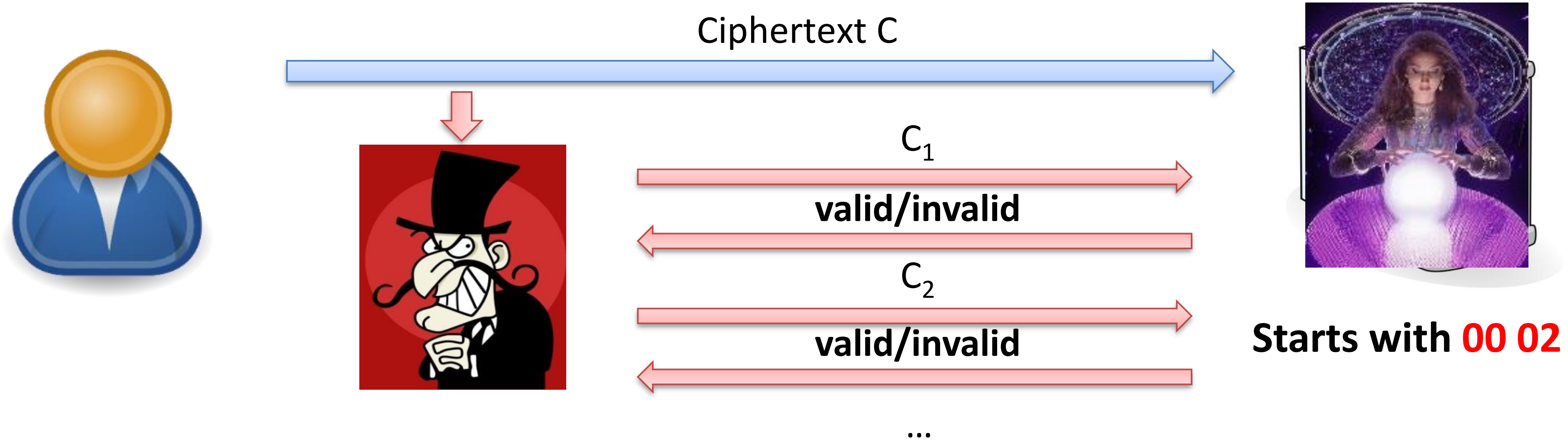
C_2

valid/invalid

Starts with 00 02 ?

Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Ciphertext C

C_1

valid/invalid

C_2

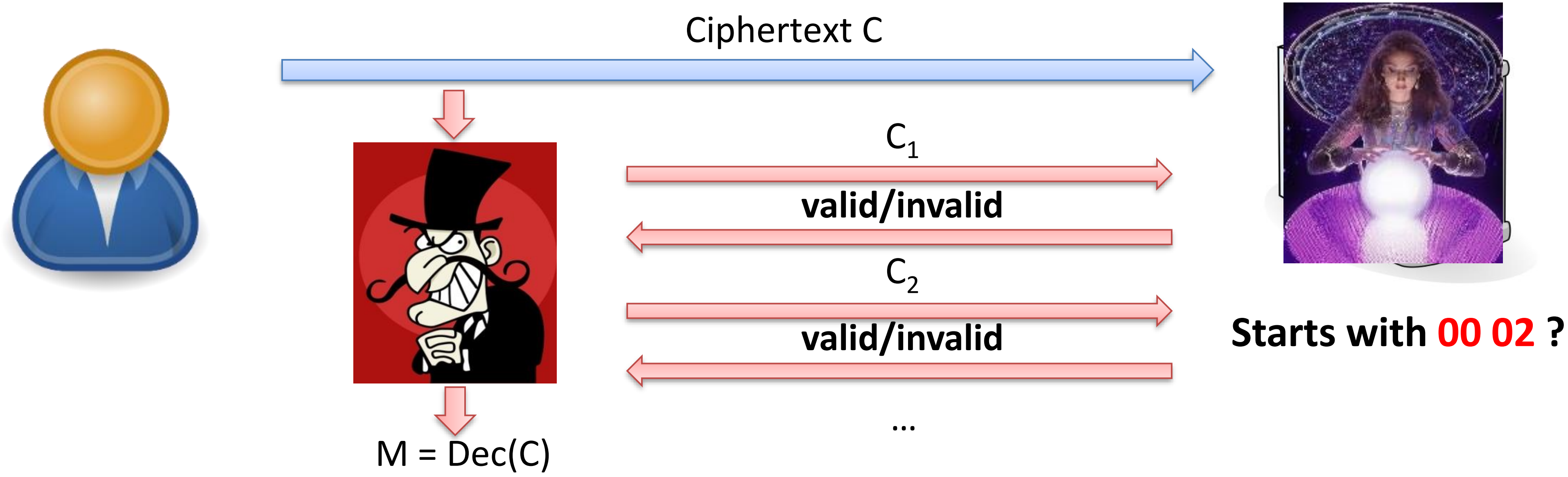
valid/invalid

…

Starts with 00 02 ?

Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Ciphertext C

C_1

valid/invalid

C_2

valid/invalid

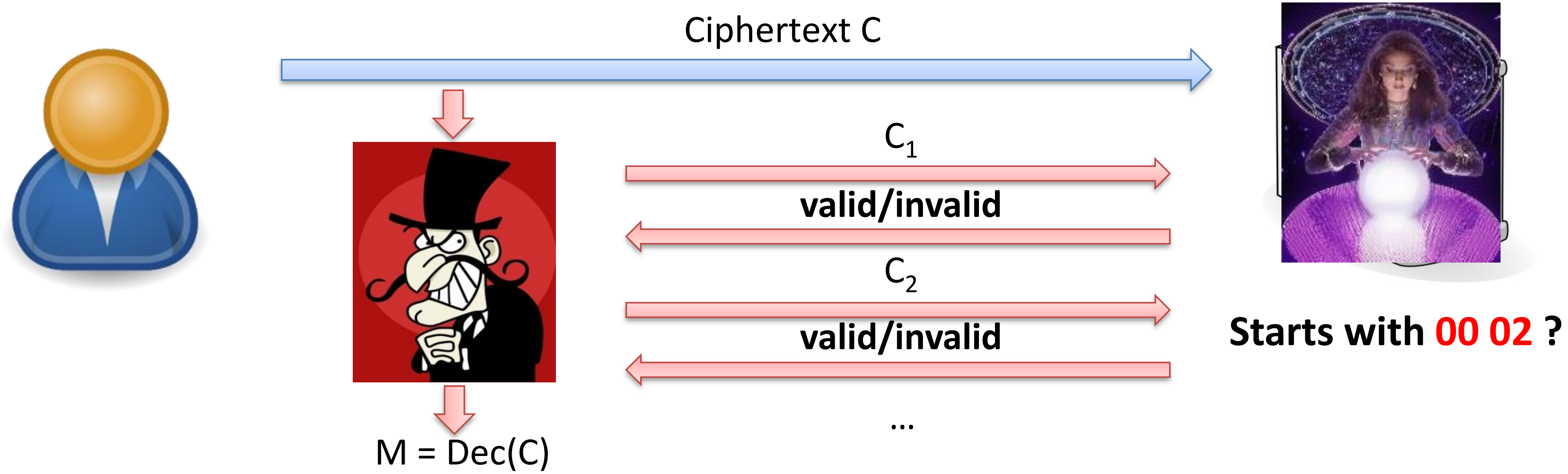
…

M = Dec(C)

Starts with 00 02 ?

Bleichenbacher’s Attack

- 1998: Adaptive chosen-ciphertext attack
- Exploits strict RSA PKCS#1 v1.5 padding validation



Ciphertext C

C_1

valid/invalid

C_2

valid/invalid

…

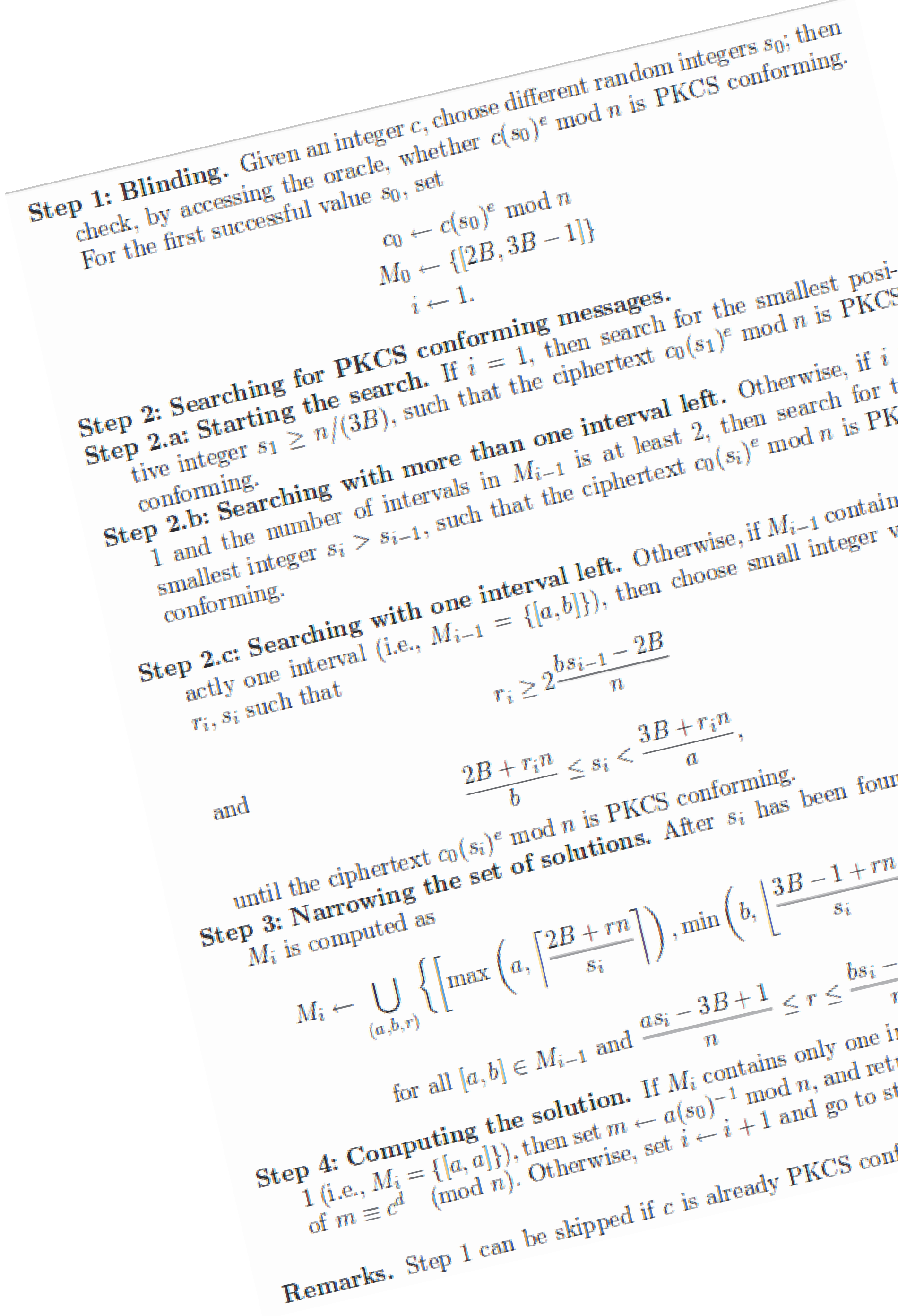
M = Dec(C)

Starts with 00 02 ?

- Similar attack on PKCS #1 v2 OEAP padding scheme [Manger 2001]

Bleichenbacher’s Attack

- The attack needs some math
  - Not going into details here



Step 1: Blinding. Given an integer c, choose different random integers s_0; then check, by accessing the oracle, whether $c(s_0)^e \bmod n$ is PKCS conforming. For the first successful value s_0, set

$$c_0 \leftarrow c(s_0)^e \bmod n$$
$$M_0 \leftarrow \{[2B, 3B-1]\}$$
$$i \leftarrow 1.$$

Step 2: Searching for PKCS conforming messages.

Step 2.a: Starting the search. If $i = 1$, then search for the smallest posi- tive integer $s_1 \geq n/(3B)$, such that the ciphertext $c_0(s_1)^e \bmod n$ is PKCS conforming.

Step 2.b: Searching with more than one interval left. Otherwise, if i 1 and the number of intervals in M_{i-1} is at least 2, then search for t smallest integer $s_i > s_{i-1}$, such that the ciphertext $c_0(s_i)^e \bmod n$ is PK conforming.

Step 2.c: Searching with one interval left. Otherwise, if M_{i-1} contain actly one interval (i.e., $M_{i-1} = \{[a, b]\}$), then choose small integer v r_i, s_i such that

$$r_i \geq 2\frac{bs_{i-1} - 2B}{n}$$

and

$$\frac{2B + r_i n}{b} \leq s_i < \frac{3B + r_i n}{a},$$

until the ciphertext $c_0(s_i)^e \bmod n$ is PKCS conforming.

Step 3: Narrowing the set of solutions. After s_i has been foun M_i is computed as

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

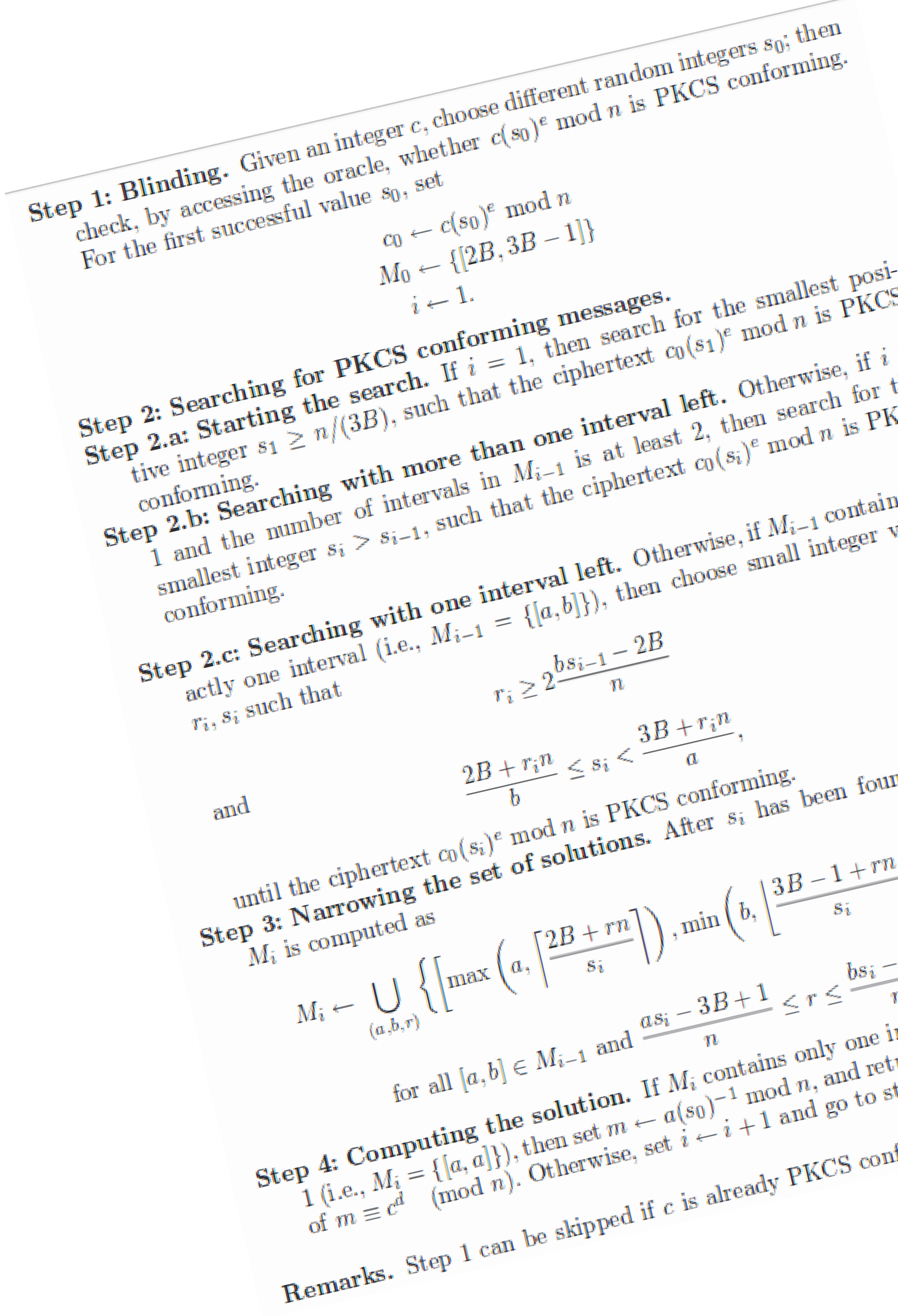
$$\text{for all } [a, b] \in M_{i-1} \text{ and } \frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i -}{n}$$

Step 4: Computing the solution. If M_i contains only one i 1 (i.e., $M_i = \{[a, a]\}$), then set $m \leftarrow a(s_0)^{-1} \bmod n$, and ret of $m \equiv c^d \pmod{n}$. Otherwise, set $i \leftarrow i + 1$ and go to st

Remarks. Step 1 can be skipped if c is already PKCS conf

Bleichenbacher's Attack

- The attack needs some math
  - Not going into details here
- "Million message attack"
  - In general performance depends on the oracle properties



Step 1: Blinding. Given an integer c, choose different random integers s_0; then check, by accessing the oracle, whether $c(s_0)^e \bmod n$ is PKCS conforming. For the first successful value s_0, set

$$c_0 \leftarrow c(s_0)^e \bmod n$$
$$M_0 \leftarrow \{[2B, 3B-1]\}$$
$$i \leftarrow 1.$$

Step 2: Searching for PKCS conforming messages.
Step 2.a: Starting the search. If $i = 1$, then search for the smallest posi-
tive integer $s_1 \geq n/(3B)$, such that the ciphertext $c_0(s_1)^e \bmod n$ is PKCS
conforming.
Step 2.b: Searching with more than one interval left. Otherwise, if i
1 and the number of intervals in M_{i-1} is at least 2, then search for t
smallest integer $s_i > s_{i-1}$, such that the ciphertext $c_0(s_i)^e \bmod n$ is PK
conforming.

Step 2.c: Searching with one interval left. Otherwise, if M_{i-1} contain
actly one interval (i.e., $M_{i-1} = \{[a, b]\}$), then choose small integer v
r_i, s_i such that

$$r_i \geq 2\frac{bs_{i-1} - 2B}{n}$$

and

$$\frac{2B + r_i n}{b} \leq s_i < \frac{3B + r_i n}{a},$$

until the ciphertext $c_0(s_i)^e \bmod n$ is PKCS conforming.
Step 3: Narrowing the set of solutions. After s_i has been foun
M_i is computed as

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

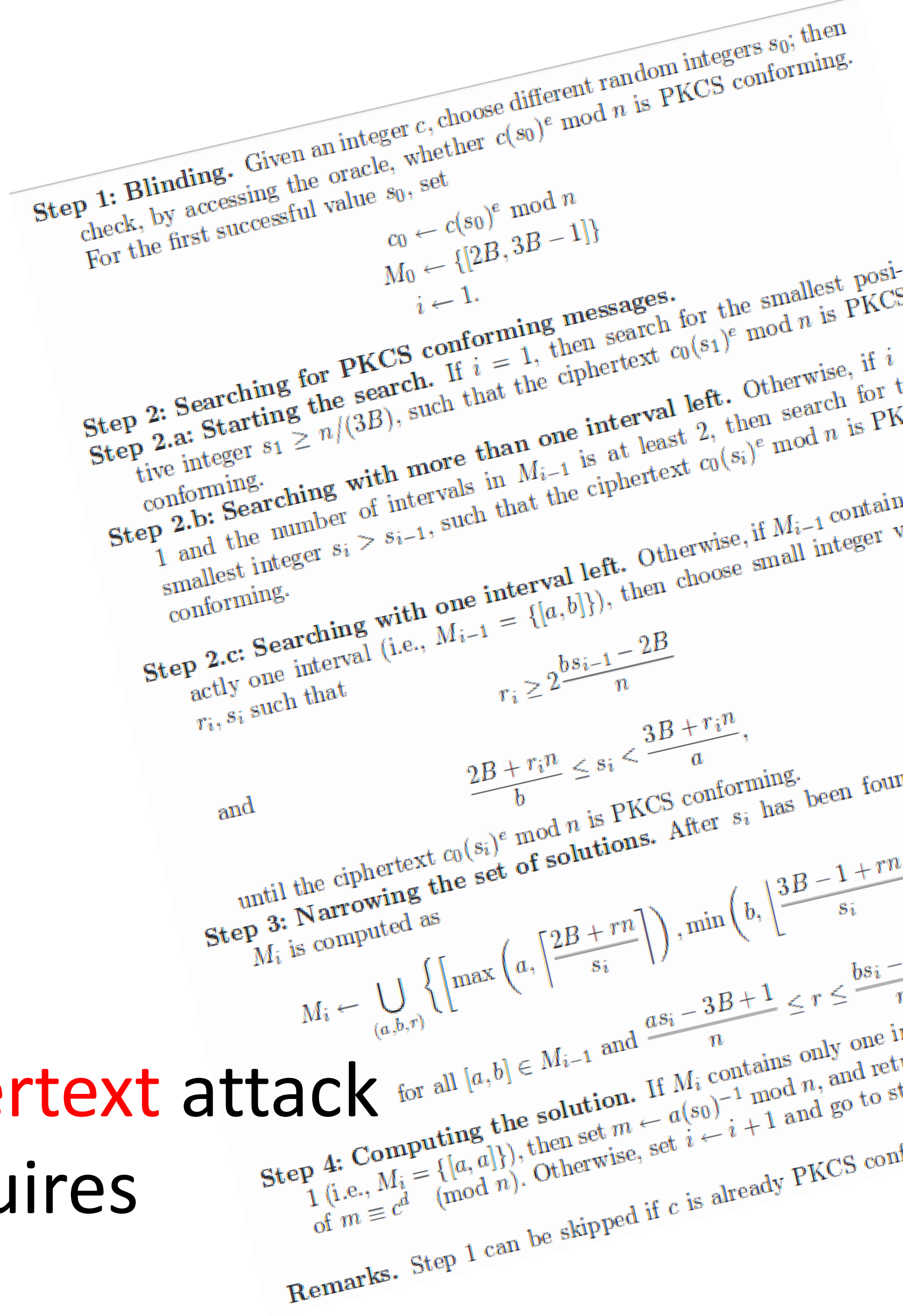
$$\text{for all } [a, b] \in M_{i-1} \text{ and } \frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i -}{n}$$

Step 4: Computing the solution. If M_i contains only one i
1 (i.e., $M_i = \{[a, a]\}$), then set $m \leftarrow a(s_0)^{-1} \bmod n$, and ret
of $m \equiv c^d \pmod{n}$). Otherwise, set $i \leftarrow i + 1$ and go to st

Remarks. Step 1 can be skipped if c is already PKCS conf

Bleichenbacher's Attack

- The attack needs some math
  - Not going into details here
- "Million message attack"
  - In general performance depends on the oracle properties
- For this talk we need to know
  - The attack is and adaptive chosen ciphertext attack
  - Decrypting 2048 bit RSA encryption requires at least 2048 sequential oracle queries



Step 1: Blinding. Given an integer c, choose different random integers s_0; then check, by accessing the oracle, whether $c(s_0)^e \bmod n$ is PKCS conforming. For the first successful value s_0, set

$$c_0 \leftarrow c(s_0)^e \bmod n$$
$$M_0 \leftarrow \{[2B, 3B-1]\}$$
$$i \leftarrow 1.$$

Step 2: Searching for PKCS conforming messages.
Step 2.a: Starting the search. If $i = 1$, then search for the smallest positive integer $s_1 \geq n/(3B)$, such that the ciphertext $c_0(s_1)^e \bmod n$ is PKCS conforming.
Step 2.b: Searching with more than one interval left. Otherwise, if i 1 and the number of intervals in M_{i-1} is at least 2, then search for t smallest integer $s_i > s_{i-1}$, such that the ciphertext $c_0(s_i)^e \bmod n$ is PK conforming.

Step 2.c: Searching with one interval left. Otherwise, if M_{i-1} contain actly one interval (i.e., $M_{i-1} = \{[a,b]\}$), then choose small integer v r_i, s_i such that

$$r_i \geq 2\frac{bs_{i-1} - 2B}{n}$$

and

$$\frac{2B + r_i n}{b} \leq s_i < \frac{3B + r_i n}{a},$$

until the ciphertext $c_0(s_i)^e \bmod n$ is PKCS conforming.
Step 3: Narrowing the set of solutions. After s_i has been foun M_i is computed as

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

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

Step 4: Computing the solution. If M_i contains only one i 1 (i.e., $M_i = \{[a,a]\}$), then set $m \leftarrow a(s_0)^{-1} \bmod n$, and ret of $m \equiv c^d \pmod{n}$. Otherwise, set $i \leftarrow i + 1$ and go to st

Remarks. Step 1 can be skipped if c is already PKCS conf

ME WANT COOKIE!

- Session cookies give access to the users’ data
    - Are sent in the beginning of each TLS connection

ME WANT COOKIE!

- Session cookies give access to the users’ data
  - Are sent in the beginning of each TLS connection
- Attack scenario for RSA KX:

ME WANT COOKIE!

- Session cookies give access to the users’ data
  - Are sent in the beginning of each TLS connection
- Attack scenario for RSA KX:
  - Sniff TLS handshake and first message

ME WANT COOKIE!

- Session cookies give access to the users’ data
  - Are sent in the beginning of each TLS connection
- Attack scenario for RSA KX:
  - Sniff TLS handshake and first message
  - Use Bleich. to decrypt premaster secret

ME WANT COOKIE!

- Session cookies give access to the users’ data
  - Are sent in the beginning of each TLS connection
- Attack scenario for RSA KX:
  - Sniff TLS handshake and first message
  - Use Bleich. to decrypt premaster secret
  - Decrypt first message

ME WANT COOKIE!

- Session cookies give access to the users' data
  - Are sent in the beginning of each TLS connection
- Attack scenario for RSA KX:
  - Sniff TLS handshake and first message
  - Use Bleich. to decrypt premaster secret
  - Decrypt first message
  - COOKIE!

Attack Scenario RSA KX:
Sniff + Cache timing side channel

Attack Scenario RSA KX:
Sniff + Cache timing side channel

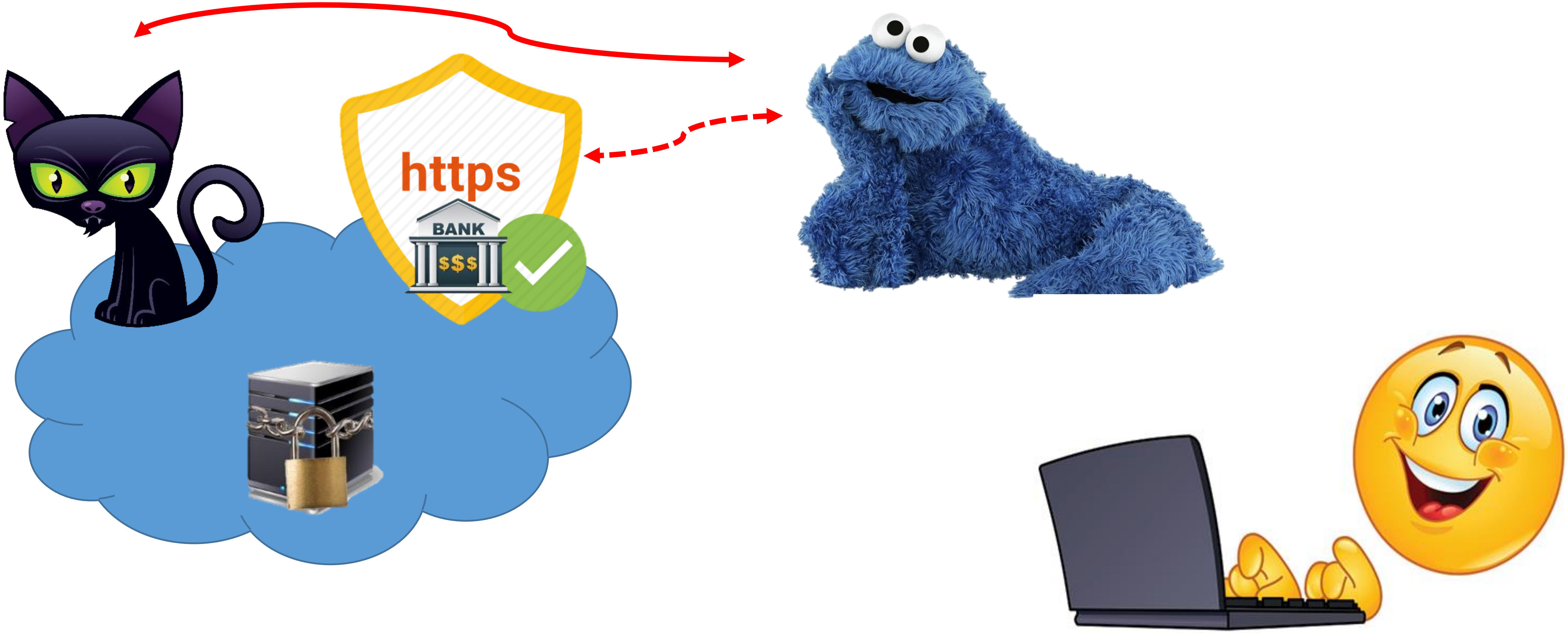
Attack Scenario RSA KX:
Sniff + Cache timing side channel

Attack Scenario RSA KX:
Sniff + Cache timing side channel

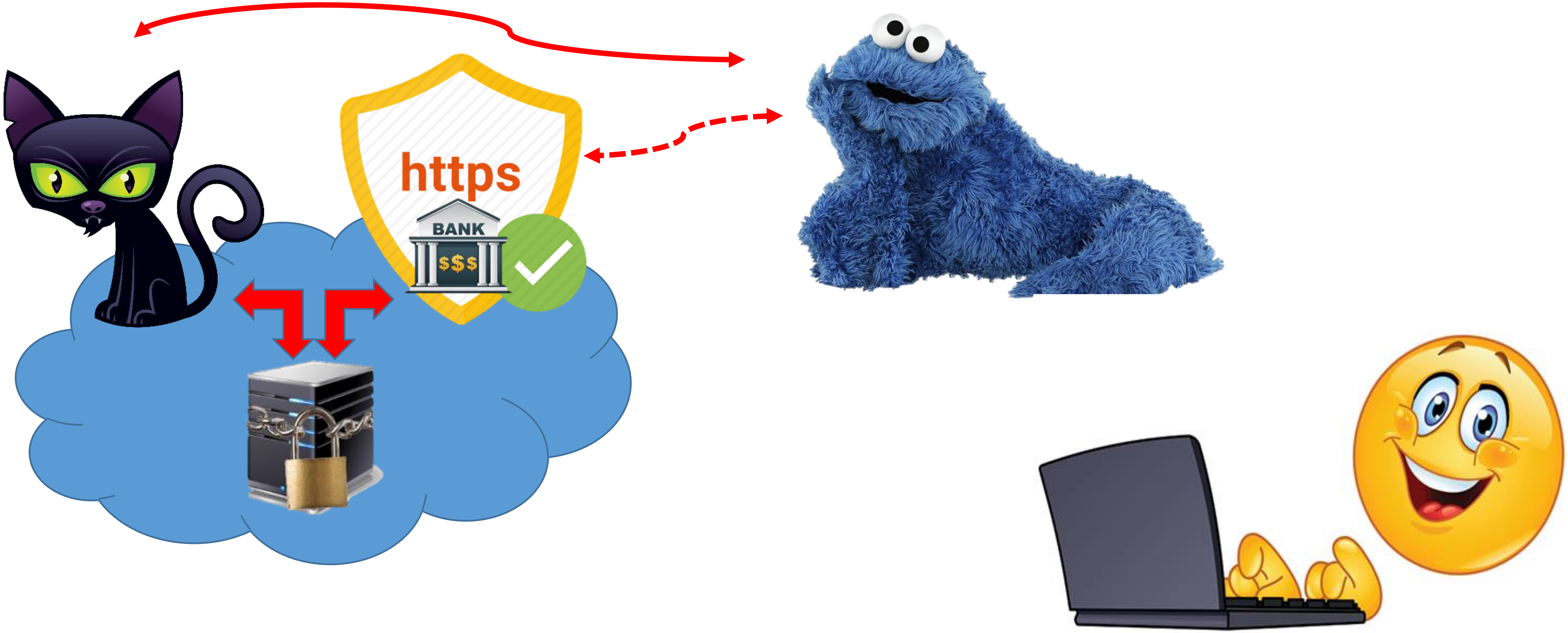
Attack Scenario RSA KX:
Sniff + Cache timing side channel

Attack Scenario RSA KX:
Sniff + Cache timing side channel

Attack Scenario RSA KX:
Sniff + Cache timing side channel



Attack Scenario RSA KX:
Sniff + Cache timing side channel



Attack Scenario RSA KX:
Sniff + Cache timing side channel

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX
- Use RSA KX vulnerability for downgrade attack

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX
- Use RSA KX vulnerability for downgrade attack
  - Only requires server support for RSA KX

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX
- Use RSA KX vulnerability for downgrade attack
  - Only requires server support for RSA KX
  - Works also on TLS 1.3 [JSS 15]

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX
- Use RSA KX vulnerability for downgrade attack
  - Only requires server support for RSA KX
  - Works also on TLS 1.3 [JSS 15]
  - Require active MiTM attack

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX
- Use RSA KX vulnerability for downgrade attack
  - Only requires server support for RSA KX
  - Works also on TLS 1.3 [JSS 15]
  - Require active MiTM attack
  - COOKIE?

ME WANT COOKIE! ALL COOKIES!

- Only 6% of connections use RSA KX
- Use RSA KX vulnerability for downgrade attack
  - Only requires server support for RSA KX
  - Works also on TLS 1.3 [JSS 15]
  - Require active MiTM attack
  - COOKIE?
- Time to finish attack < 30 sec

Downgrade attack on Firefox

- We can prevent timeout in Firefox’s TLS handshakes using TLS warning alerts [ABDG+15]

Downgrade attack on Firefox

- We can prevent timeout in Firefox’s TLS handshakes using TLS warning alerts [ABDG+15]
- Do MiTM downgrade attack

Downgrade attack on Firefox

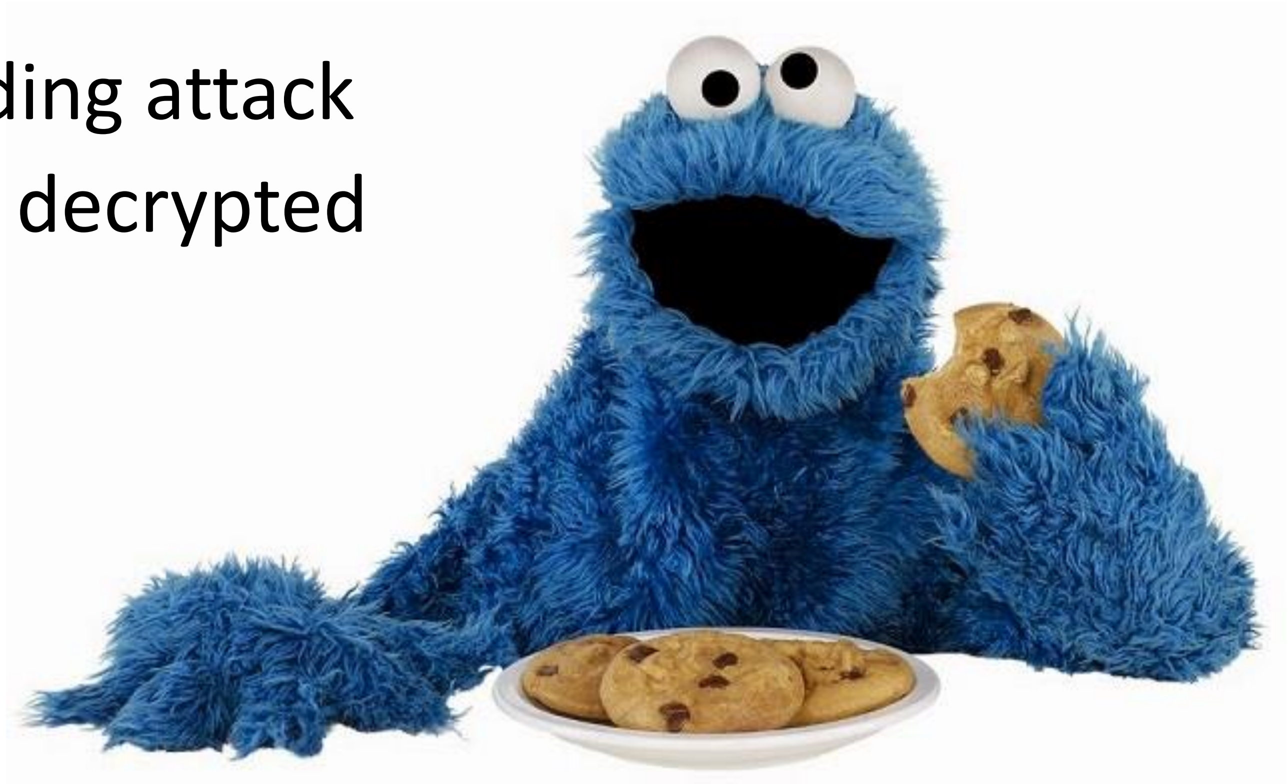
- We can prevent timeout in Firefox's TLS handshakes using TLS warning alerts [ABDG+15]
- Do MiTM downgrade attack
  - Keep session alive during padding attack

Downgrade attack on Firefox

- We can prevent timeout in Firefox’s TLS handshakes using TLS warning alerts [ABDG+15]
- Do MiTM downgrade attack
  - Keep session alive during padding attack
  - Finish the TLS handshake with decrypted premaster secret

Downgrade attack on Firefox

- We can prevent timeout in Firefox’s TLS handshakes using TLS warning alerts [ABDG+15]
- Do MiTM downgrade attack
  - Keep session alive during padding attack
  - Finish the TLS handshake with decrypted premaster secret
  - Cookie?



Downgrade attack on Firefox

- We can prevent timeout in Firefox's TLS handshakes using TLS warning alerts [ABDG+15]
- Do MiTM downgrade attack
  - Keep session alive during padding attack
  - Finish the TLS handshake with decrypted premaster secret
  - Cookie?
- The user will notice the delay

The Boost of the BEAST

- BEAST like attack can help!

The Boost of the BEAST

- BEAST like attack can help!
  - JavaScript in browser allows the attacker to repeatedly reopen connections in the background, without the user’s knowledge.

The Boost of the BEAST

- BEAST like attack can help!
  - JavaScript in browser allows the attacker to repeatedly reopen connections in the background, without the user’s knowledge.
  - At the start of each connection, the same session cookie is sent in the first packet

The Boost of the BEAST

- BEAST like attack can help!
  - JavaScript in browser allows the attacker to repeatedly reopen connections in the background, without the user’s knowledge.
  - At the start of each connection, the same session cookie is sent in the first packet
  - Need to break just one connection

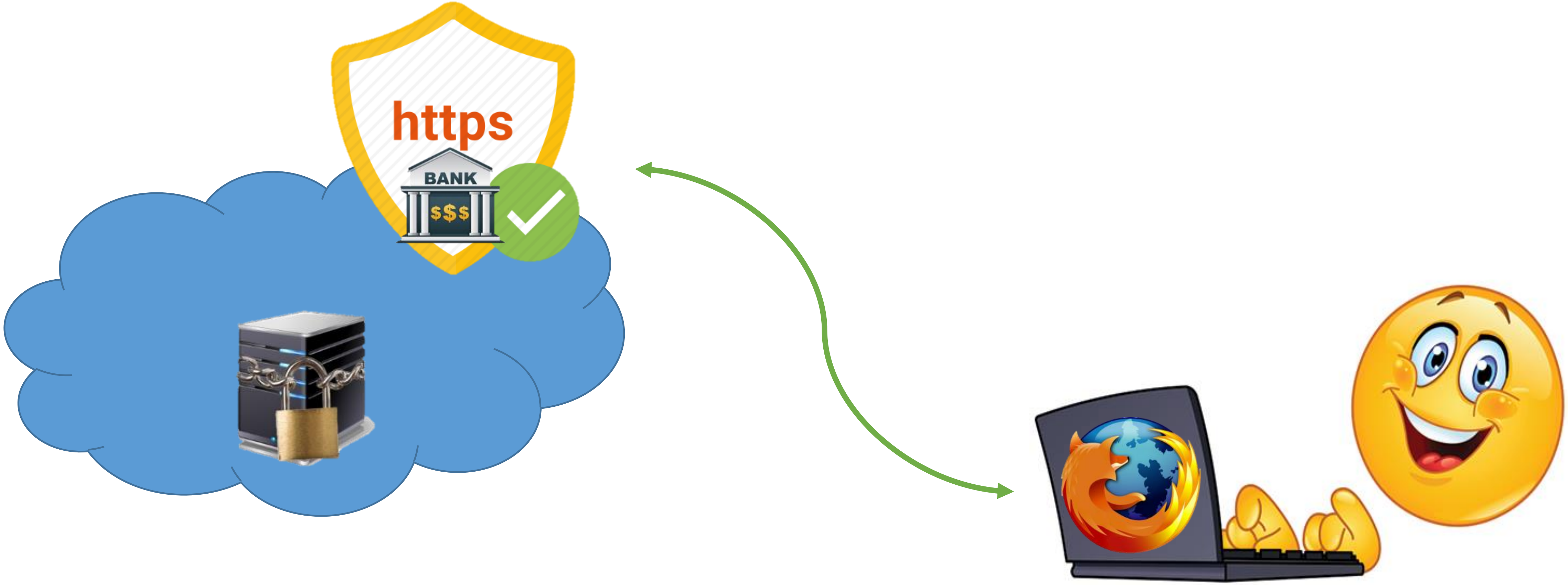
The Boost of the BEAST

- BEAST like attack can help!
  - JavaScript in browser allows the attacker to repeatedly reopen connections in the background, without the user's knowledge.
  - At the start of each connection, the same session cookie is sent in the first packet
  - Need to break just one connection
  - COOKIE!

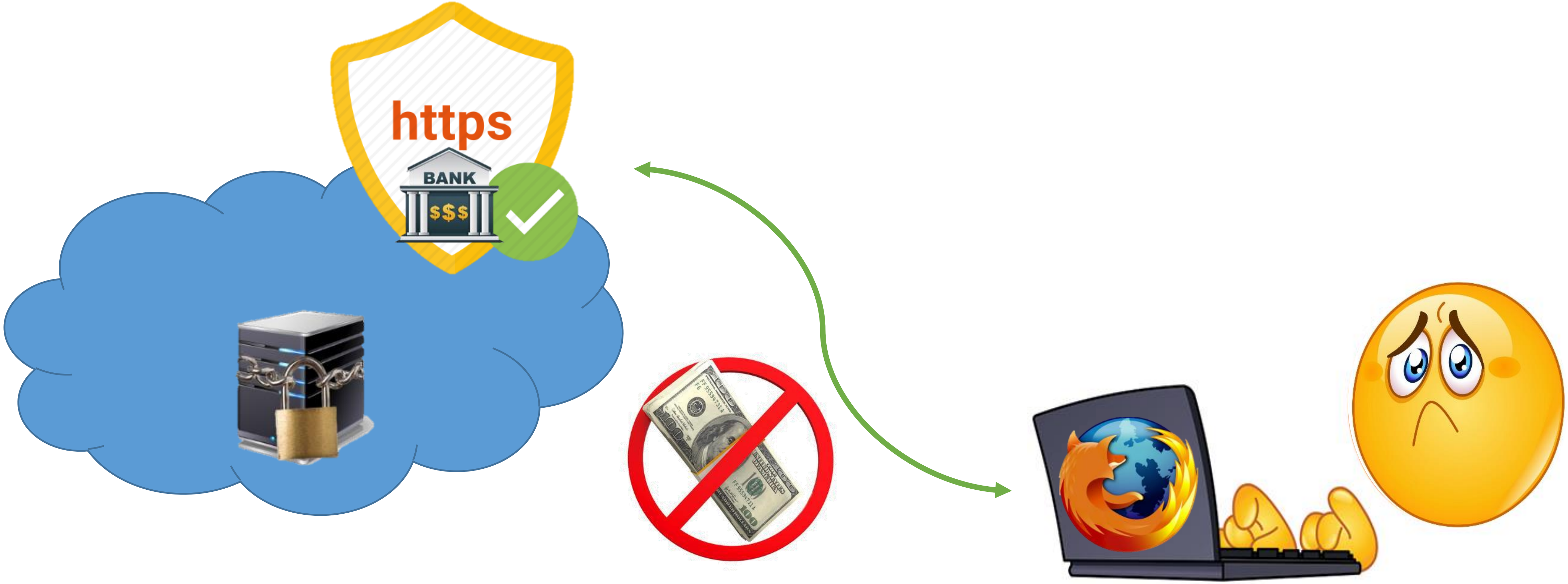
Attack Scenario Firefox:
MiTM + Cache timing side channel

Attack Scenario Firefox:
MiTM + Cache timing side channel

Attack Scenario Firefox: MiTM + Cache timing side channel



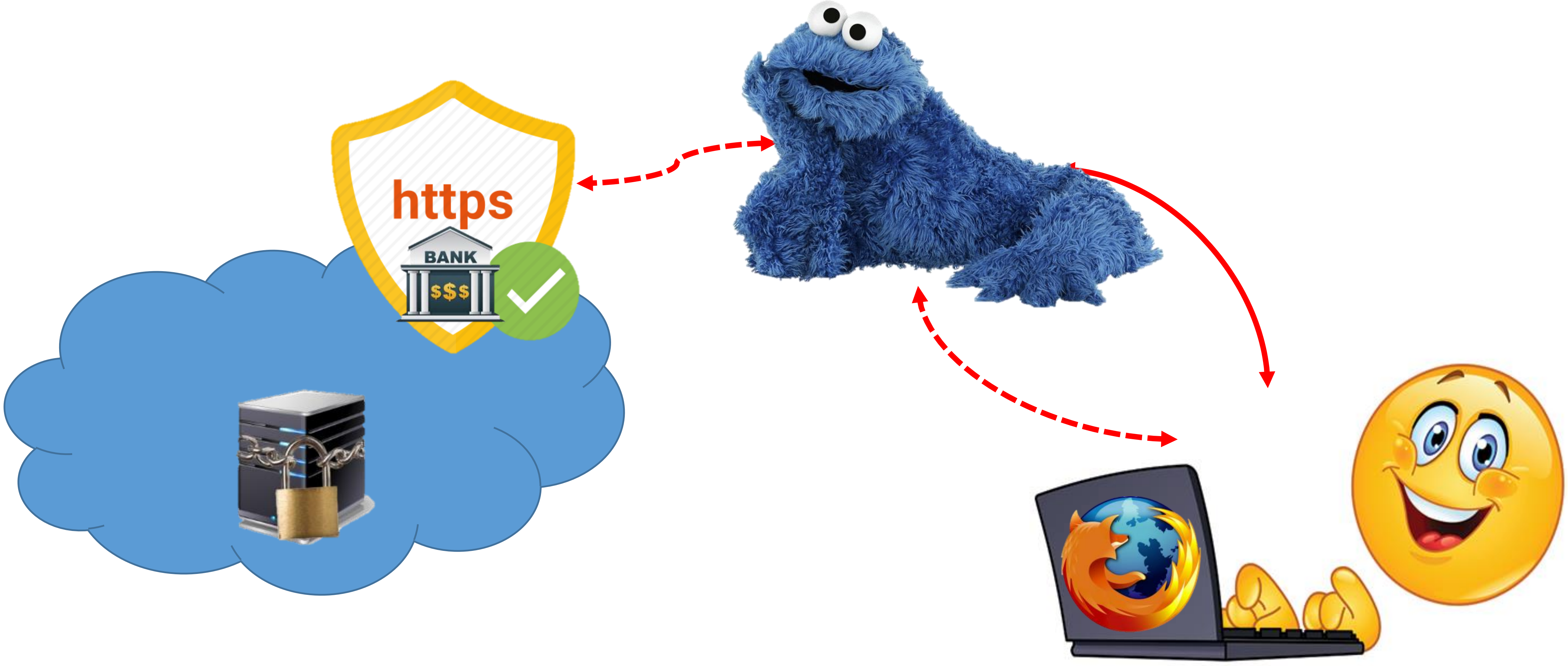
Attack Scenario Firefox: MiTM + Cache timing side channel



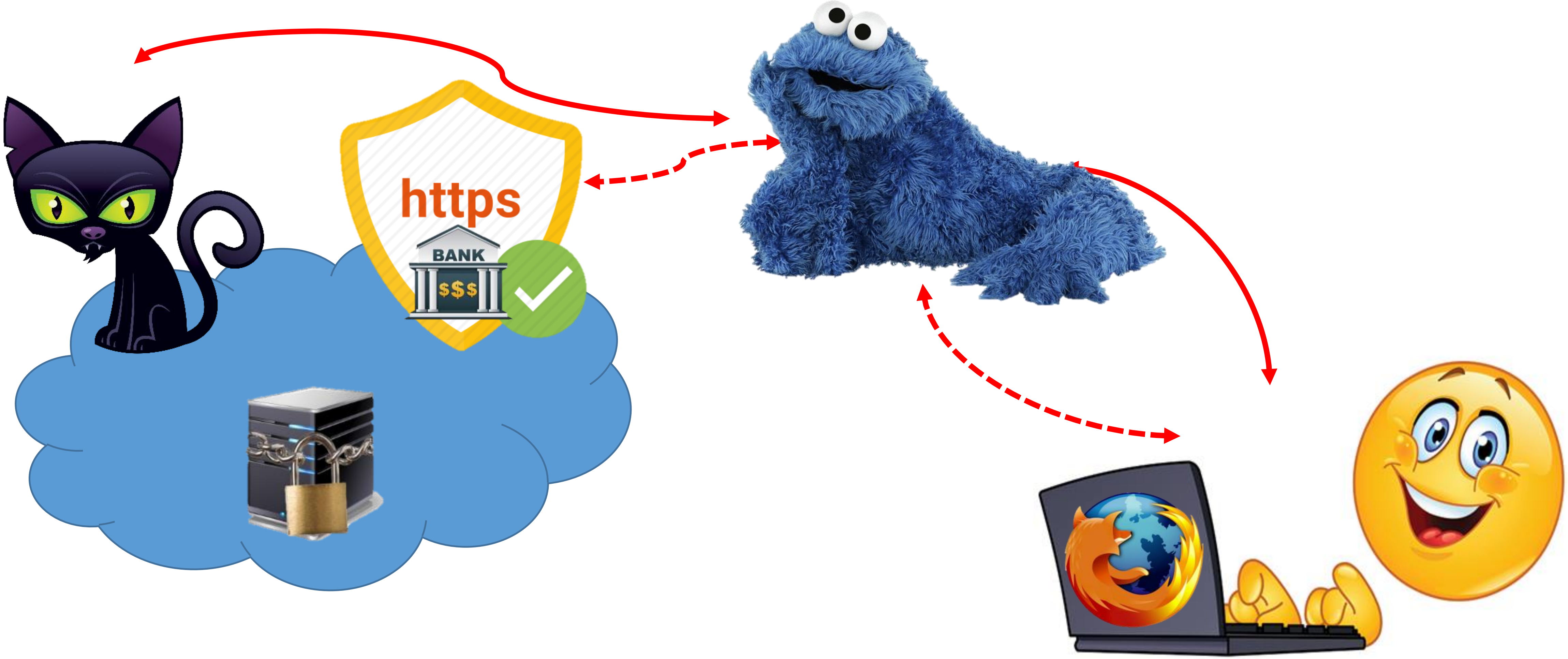
Attack Scenario Firefox: MiTM + Cache timing side channel

https
BANK
$$$
Win Big Prizes .COM

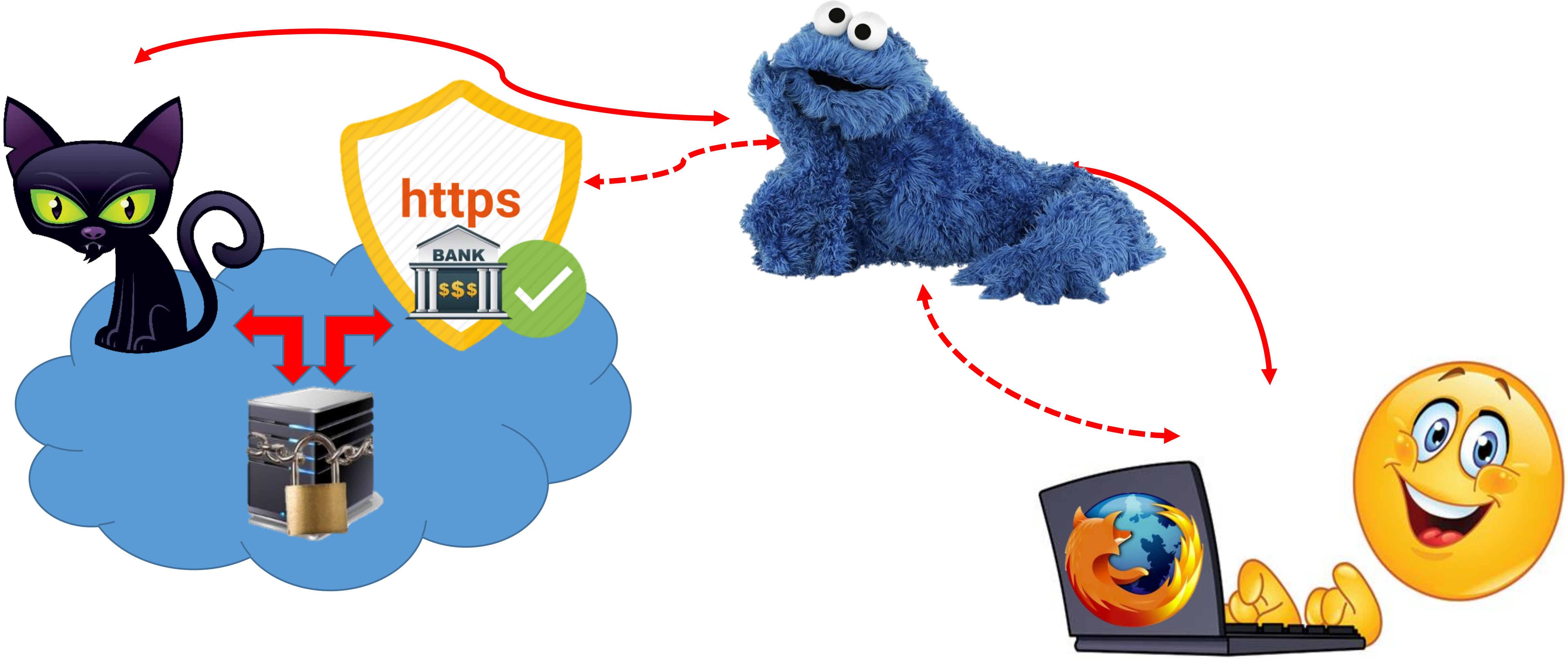
Attack Scenario Firefox:
MiTM + Cache timing side channel



Attack Scenario Firefox: MiTM + Cache timing side channel



Attack Scenario Firefox: MiTM + Cache timing side channel



Attack Scenario Firefox:
MiTM + Cache timing side channel

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high
- With low probability, require much less

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high
- With low probability, require much less
- BEAST - Try many MiTM downgrade attack

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high
- With low probability, require much less
- BEAST - Try many MiTM downgrade attack
  - Need to break just 1 out of 1000

General Downgrade attack

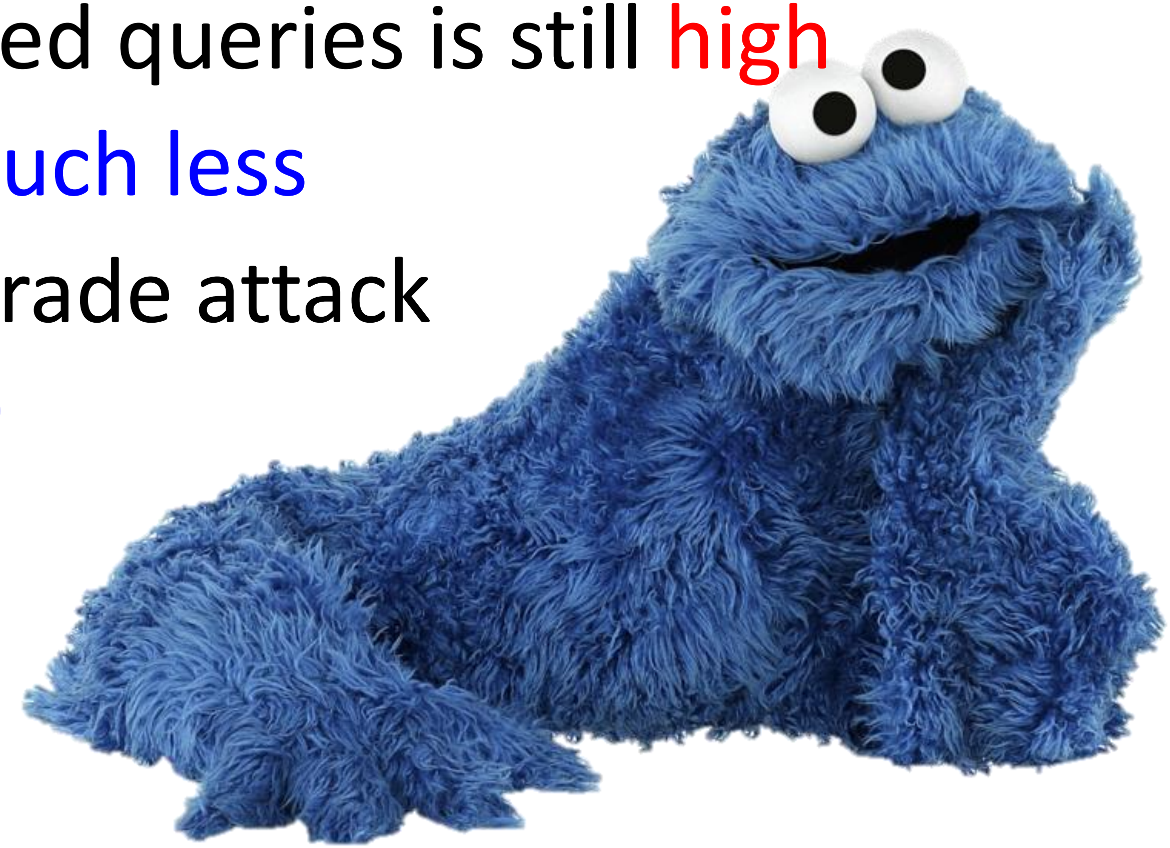
- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high
- With low probability, require much less
- BEAST - Try many MiTM downgrade attack
  - Need to break just 1 out of 1000
  - Cookie?

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high
- With low probability, require much less
- BEAST - Try many MiTM downgrade attack
  - Need to break just 1 out of 1000
  - Cookie?
- Need at least 2048 queries

General Downgrade attack

- In most browsers we only have 30 seconds to finish the TLS handshake
- The expected number of required queries is still high
- With low probability, require much less
- BEAST - Try many MiTM downgrade attack
  - Need to break just 1 out of 1000
  - Cookie?
- Need at least 2048 queries
  - Have time for < 600



Parallel Downgrade attack

- Many companies reuse certificate on multiple servers

Parallel Downgrade attack

- Many companies reuse certificate on multiple servers
- We can parallelize the attack across multiple servers

Parallel Downgrade attack

- Many companies reuse certificate on multiple servers
- We can parallelize the attack across multiple servers
  - Each server is a separate oracle

Parallel Downgrade attack

- Many companies reuse certificate on multiple servers
- We can parallelize the attack across multiple servers
  - Each server is a separate oracle
  - Many previous works mention parallelization

Parallel Downgrade attack

- Many companies reuse certificate on multiple servers
- We can parallelize the attack across multiple servers
  - Each server is a separate oracle
  - Many previous works mention parallelization
  - Cookie?

Parallel Downgrade attack

- Many companies reuse certificate on multiple servers
- We can parallelize the attack across multiple servers
  - Each server is a separate oracle
  - Many previous works mention parallelization
  - Cookie?
- Need at least 2048 sequential adaptive queries

Parallel Downgrade attack

- Many companies reuse certificate on multiple servers
- We can parallelize the attack across multiple servers
  - Each server is a separate oracle
  - Many previous works mention parallelization
  - Cookie?
- Need at least 2048 sequential adaptive queries
  - Have time for < 600

A little Manger background

A little Manger background

- Assume we have the following Manger oracle

$$\mathsf{Ma}(c) = \begin{cases} 1 \text{ if } c^d \bmod N \text{ starts with } \mathsf{0x00} \\ 0 \text{ otherwise} \end{cases} .$$

A little Manger background

- Assume we have the following Manger oracle

$$\mathsf{Ma}(c) = \begin{cases} 1 \text{ if } c^d \bmod N \text{ starts with } \mathsf{0x00} \\ 0 \text{ otherwise} \end{cases} .$$

- We start with a blinding phase to find s such that

$$\mathsf{Ma}(c \cdot s^e \mod N) = \mathsf{Ma}((m \cdot s)^e \mod N) = 1$$

$$m \cdot s \mod N < 2^{\log_2 N - 8}$$

A little Manger background

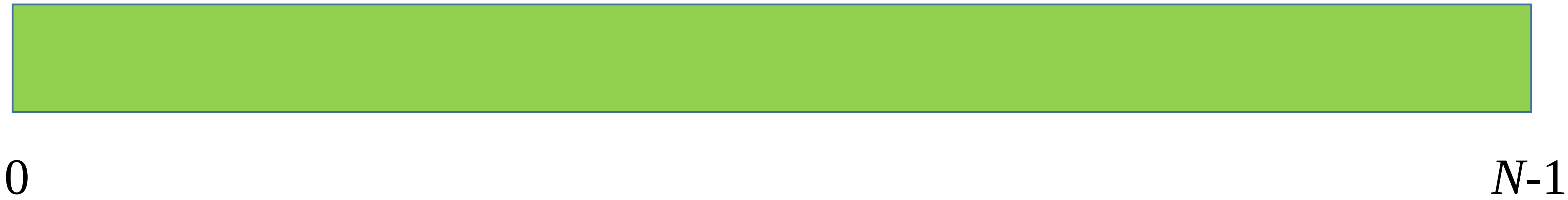
- Assume we have the following Manger oracle

$$\mathsf{Ma}(c) = \begin{cases} 1 \text{ if } c^d \mod N \text{ starts with } \mathsf{0x00} \\ 0 \text{ otherwise} \end{cases} .$$

- We start with a blinding phase to find s such that

$$\mathsf{Ma}(c \cdot s^e \mod N) = \mathsf{Ma}((m \cdot s)^e \mod N) = 1$$

$$m \cdot s \mod N < 2^{\log_2 N - 8}$$



A little Manger background

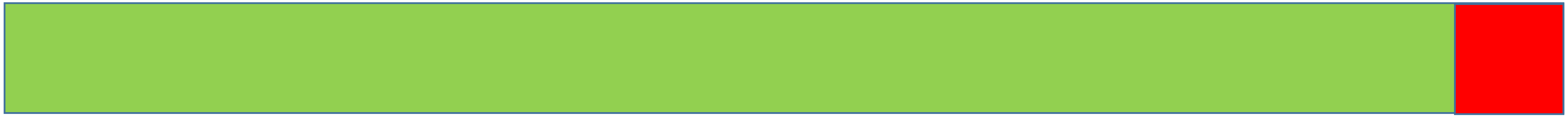
- Assume we have the following Manger oracle

$$\text{Ma}(c) = \begin{cases} 1 \text{ if } c^d \bmod N \text{ starts with 0x00} \\ 0 \text{ otherwise} \end{cases}.$$

- We start with a blinding phase to find s such that

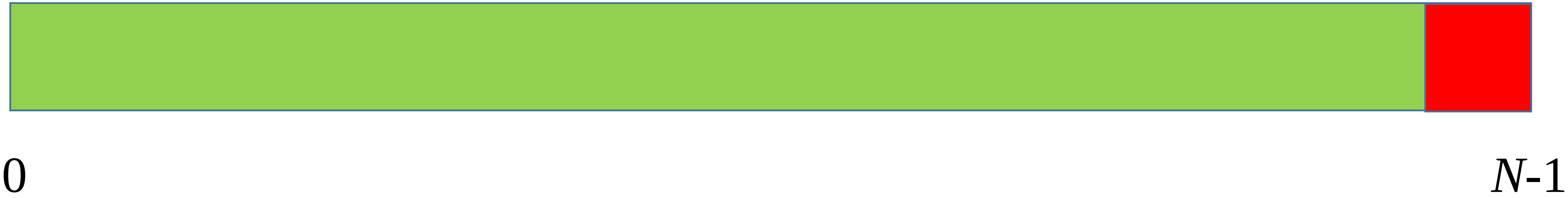
$$\text{Ma}(c \cdot s^e \mod N) = \text{Ma}((m \cdot s)^e \mod N) = 1$$

$$m \cdot s \mod N < 2^{\log_2 N - 8}$$



A little Manger background

- Iteratively reduce size of possible interval

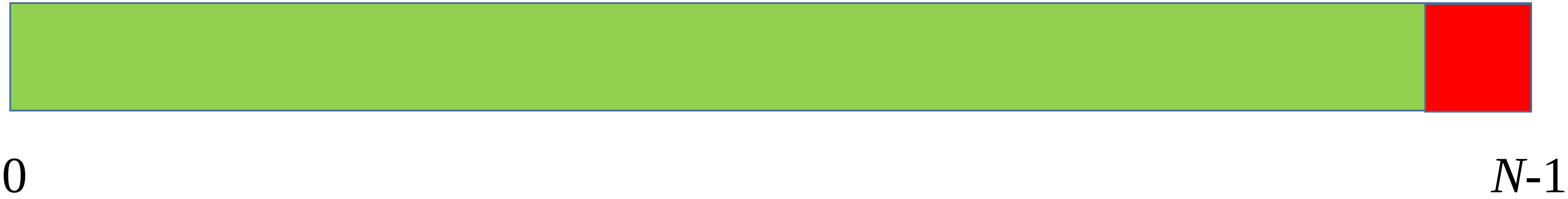


A little Manger background

- Iteratively reduce size of possible interval
- After additional i sequential queries we learn that

$$m \cdot s \mod N \in [a_i, b_i]$$

$$r_i = m \cdot s - a_i \mod N < 2^{\log_2 (b_i - a_i)}$$

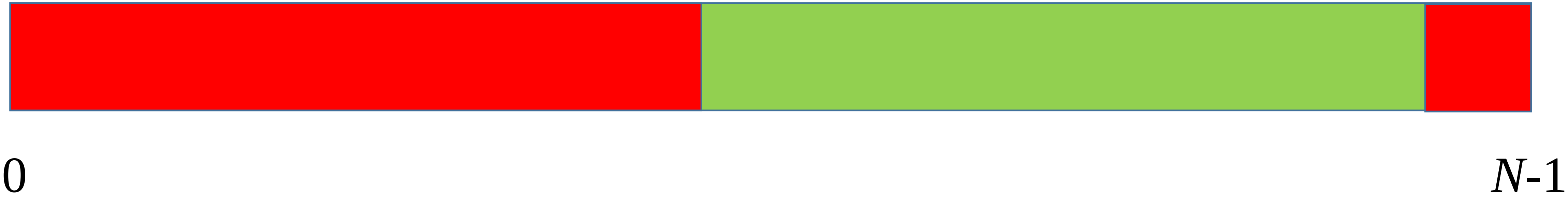


A little Manger background

- Iteratively reduce size of possible interval
- After additional i sequential queries we learn that

$$m \cdot s \mod N \in [a_i, b_i]$$

$$r_i = m \cdot s - a_i \mod N < 2^{\log_2 (b_i - a_i)}$$



A little Manger background

- Iteratively reduce size of possible interval
- After additional i sequential queries we learn that

$$m \cdot s \mod N \in [a_i, b_i]$$

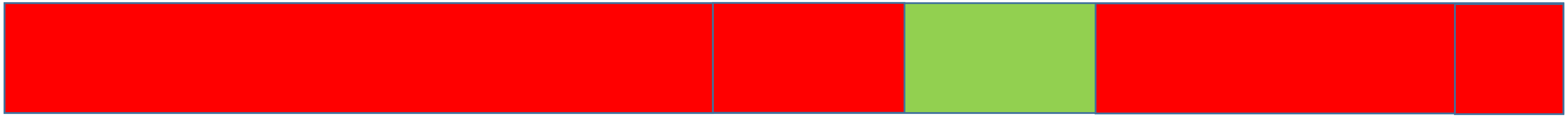
$$r_i = m \cdot s - a_i \mod N < 2^{\log_2 (b_i - a_i)}$$

A little Manger background

- Iteratively reduce size of possible interval
- After additional i sequential queries we learn that

$$m \cdot s \mod N \in [a_i, b_i]$$

$$r_i = m \cdot s - a_i \mod N < 2^{\log_2 (b_i - a_i)}$$

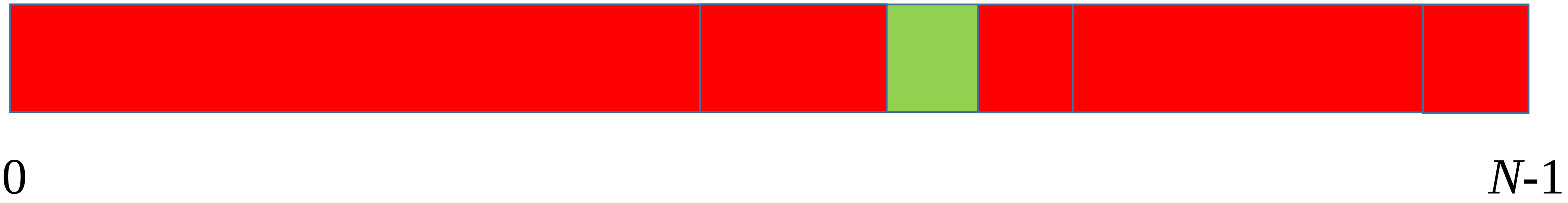


A little Manger background

- Iteratively reduce size of possible interval
- After additional i sequential queries we learn that

$$m \cdot s \mod N \in [a_i, b_i]$$

$$r_i = m \cdot s - a_i \mod N < 2^{\log_2 (b_i - a_i)}$$

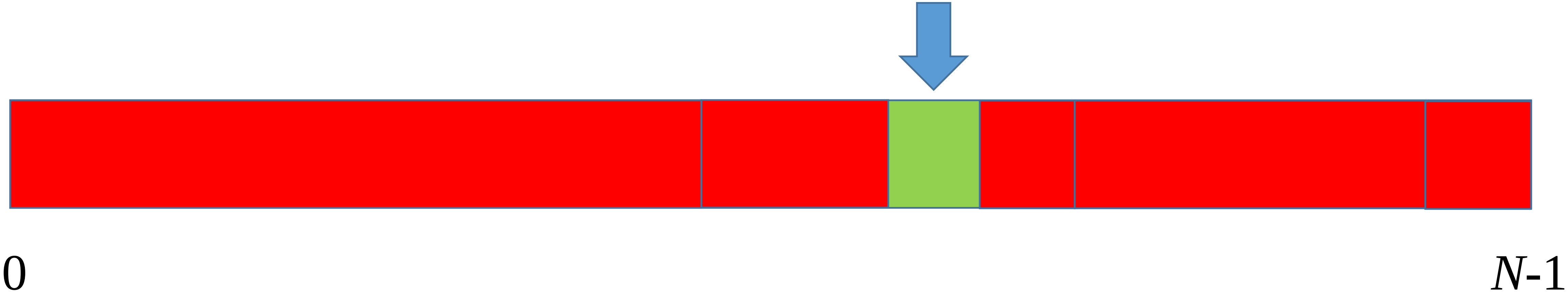


A little Manger background

- Iteratively reduce size of possible interval
- After additional i sequential queries we learn that

$$m \cdot s \mod N \in [a_i, b_i]$$

$$r_i = m \cdot s - a_i \mod N < 2^{\log_2 (b_i - a_i)}$$



The Cookie Lattice

- Assume we can run k attacks in parallel with i sequential queries each, for each attack j we know that

$$r_{j,i} = m \cdot s_j - a_{j,i} \mod N < 2^{\log_2 (b_{j,i} - a_{j,i})}$$

The Cookie Lattice

- Assume we can run k attacks in parallel with i sequential queries each, for each attack j we know that

$$r_{j,i} = m \cdot s_j - a_{j,i} \mod N < 2^{\log_2 (b_{j,i} - a_{j,i})}$$

- Similar Boneh & Venkatesan's Hidden Number Problem

The Cookie Lattice

- Assume we can run k attacks in parallel with i sequential queries each, for each attack j we know that

$$r_{j,i} = m \cdot s_j - a_{j,i} \mod N < 2^{\log_2 (b_{j,i} - a_{j,i})}$$

- Similar Boneh & Venkatesan's Hidden Number Problem
- Finding m is reduced to CVP that we can embed in a SVP lattice and solve with LLL

$$M^i = \begin{bmatrix} s_1 & s_2 & s_3 & \dots & s_k & 0 \\ N & 0 & 0 & \dots & 0 & 0 \\ 0 & N & 0 & \dots & 0 & 0 \\ 0 & 0 & N & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & N & 0 \\ a_{1,i} & a_{2,i} & a_{3,i} & \dots & a_{k,i} & N \cdot (k-1)/k \end{bmatrix}$$

The Cookie Lattice

- Assume we can run k attacks in parallel with i sequential queries each, for each attack j we know that

$$r_{j,i} = m \cdot s_j - a_{j,i} \mod N < 2^{\log_2 (b_{j,i} - a_{j,i})}$$

- Similar Boneh & Venkatesan’s Hidden Number Problem
- Finding m is reduced to CVP that we can embed in a SVP lattice and solve with LLL

$$M^i = \begin{bmatrix} s_1 & s_2 & s_3 & \dots & s_k & 0 \\ N & 0 & 0 & \dots & 0 & 0 \\ 0 & N & 0 & \dots & 0 & 0 \\ 0 & 0 & N & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & N & 0 \\ a_{1,i} & a_{2,i} & a_{3,i} & \dots & a_{k,i} & N \cdot (k-1)/k \end{bmatrix}$$

- We need just 5 servers to decrypt 2048 bit RSA using a Manger oracle

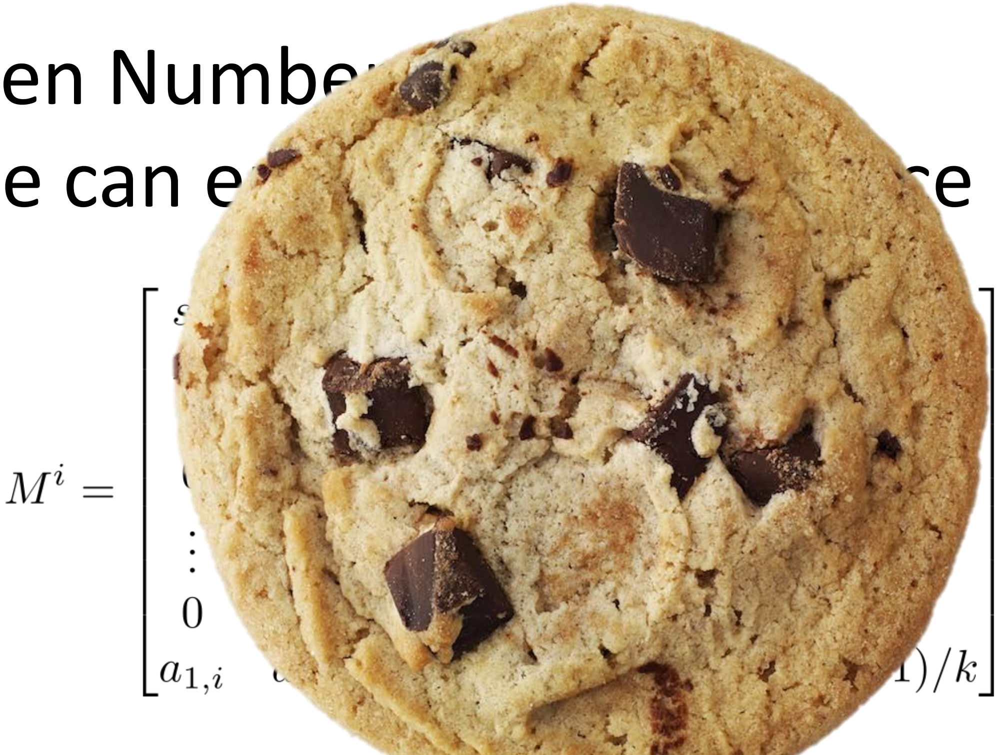
The Cookie Lattice

- Assume we can run k attacks in parallel with i sequential queries each, for each attack j we know that

$$r_{j,i} = m \cdot s_j - a_{j,i} \mod N < 2^{\log_2 (b_{j,i} - a_{j,i})}$$

- Similar Boneh & Venkatesan's Hidden Numbe
- Finding m is reduced to CVP that we can e ce and solve with LLL

- We need just 5 servers to decrypt 2048 bit RSA using a Manger oracle



The Cookie Lattice Tradeoff

- The initial blinding phase is more “expensive” per bit

The Cookie Lattice Tradeoff

- The initial blinding phase is more “expensive” per bit

- The parallel attack requires more queries!

The Cookie Lattice Tradeoff

- The initial blinding phase is more “expensive” per bit
- The parallel attack requires more queries!
- So why do we do it?

The Cookie Lattice Tradeoff

- The initial blinding phase is more “expensive” per bit

- The parallel attack requires more queries!

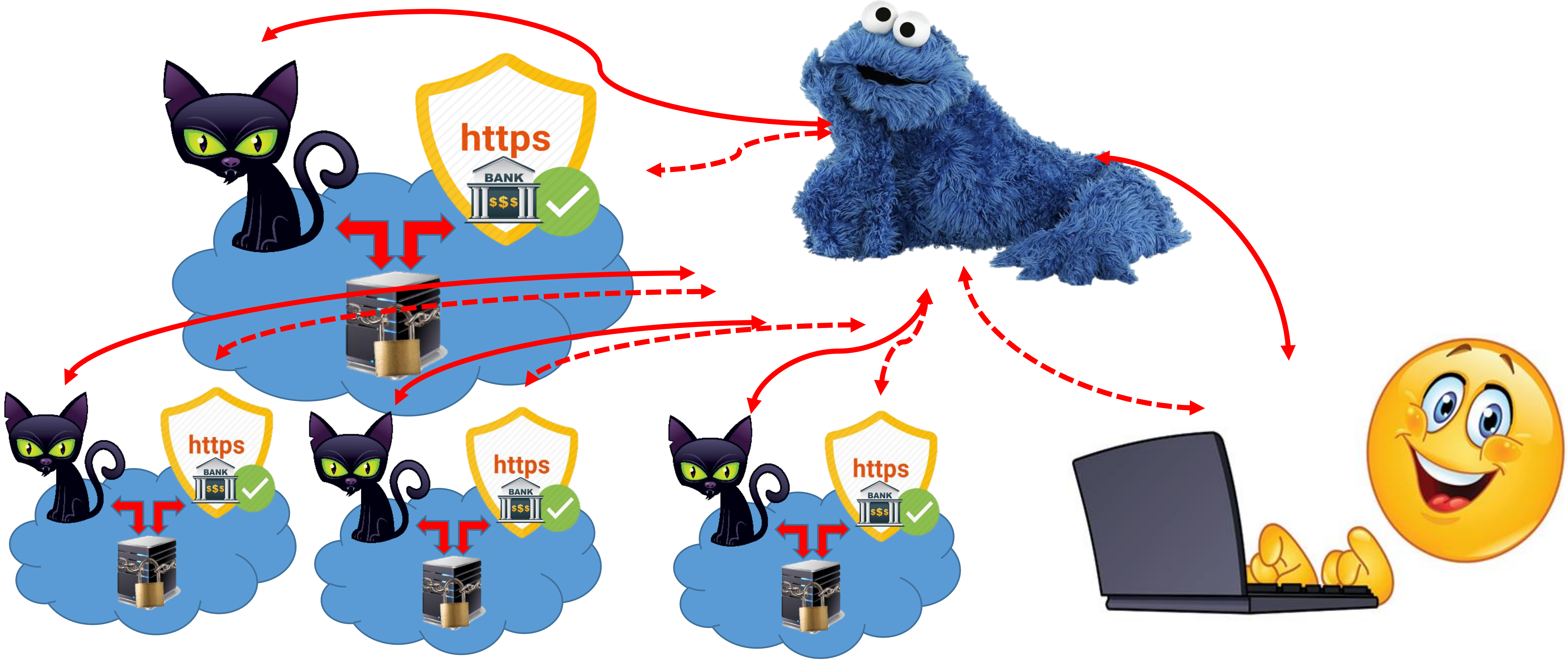
- So why do we do it?

    - Tradeoff between the total number of queries and number of sequential queries

The Cookie Lattice Tradeoff

- The initial blinding phase is more “expensive” per bit
- The parallel attack requires more queries!
- So why do we do it?
  - Tradeoff between the total number of queries and number of sequential queries
  - Allows us to finish attack in less than 30 seconds

Attack Scenario Parallel: MiTM + Cache timing side channel



Attack Scenario Parallel: MiTM + Cache timing side channel

$$M^i = \begin{bmatrix} s_1 & s_2 & s_3 & \dots & s_k & 0 \\ N & 0 & 0 & \dots & 0 & 0 \\ 0 & N & 0 & \dots & 0 & 0 \\ 0 & 0 & N & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & N & 0 \\ a_1^i & a_2^i & a_3^i & \dots & a_k^i & N \cdot (k-1)/k \end{bmatrix}$$

Attack Scenario Parallel:
MiTM + Cache timing side channel

Why is fixing Bleichenbacher so hard?

- Hard to reduce time variability in RSA KX

Why is fixing Bleichenbacher so hard?

- Hard to reduce time variability in RSA KX

  But *most* implementation managed

Why is fixing Bleichenbacher so hard?

- Hard to reduce time variability in RSA KX

  But *most* implementation managed
- Very hard to implement RSA KX in constant time

Why is fixing Bleichenbacher so hard?

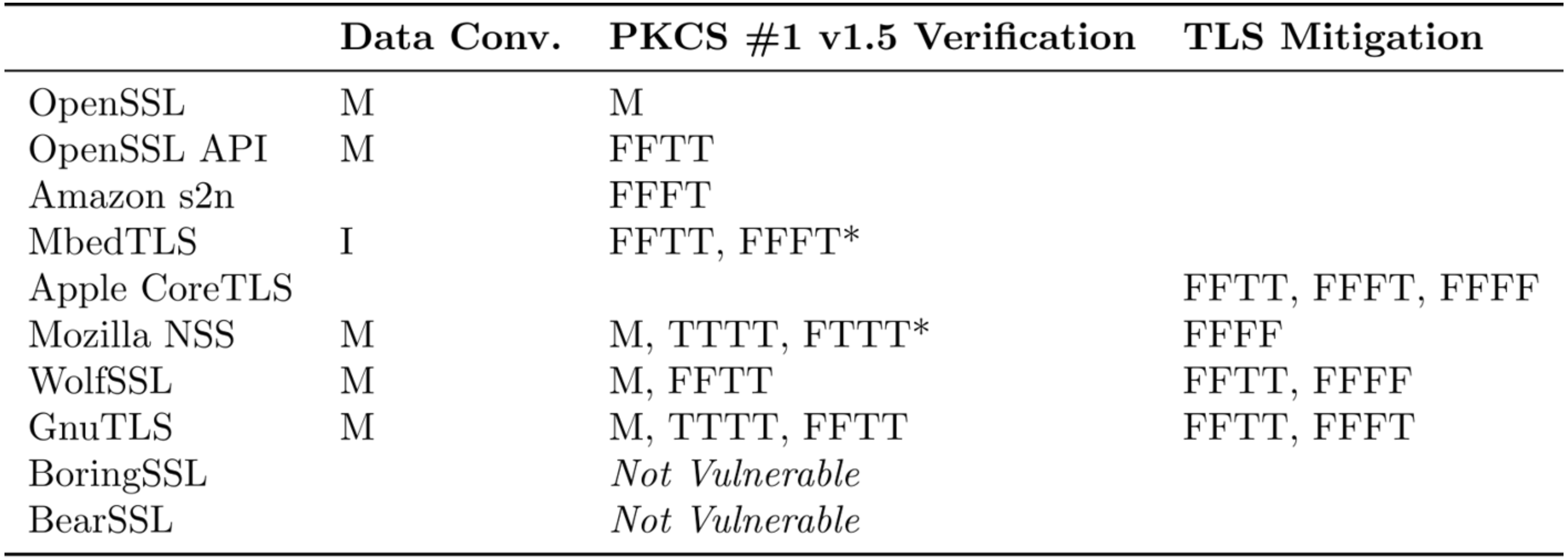
- Hard to reduce time variability in RSA KX

  But *most* implementation managed
- Very hard to implement RSA KX in constant time
- Pseudo-constant time is only pseudo-secure

Why is fixing Bleichenbacher so hard?

- Hard to reduce time variability in RSA KX

  But *most* implementation managed
- Very hard to implement RSA KX in constant time
- Pseudo-constant time is only pseudo-secure



	Data Conv.	PKCS #1 v1.5 Verification	TLS Mitigation
OpenSSL	M	M	
OpenSSL API	M	FFTT	
Amazon s2n		FFFT	
MbedTLS	I	FFTT, FFFT*	
Apple CoreTLS			FFTT, FFFT, FFFF
Mozilla NSS	M	M, TTTT, FTTT*	FFFF
WolfSSL	M	M, FFTT	FFTT, FFFF
GnuTLS	M	M, TTTT, FFTT	FFTT, FFFT
BoringSSL		*Not Vulnerable*	
BearSSL		*Not Vulnerable*	

Data Conversion

- RSA decryption works with big integer numbers

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes
- Converting from number to bytes in constant time is hard

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes
- Converting from number to bytes in constant time is hard
- We found:

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes
- Converting from number to bytes in constant time is hard
- We found:
  - Conditional padding with zeros for small numbers

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes
- Converting from number to bytes in constant time is hard
- We found:
  - Conditional padding with zeros for small numbers
  - Conditional branching on size of padding

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes
- Converting from number to bytes in constant time is hard
- We found:
  - Conditional padding with zeros for small numbers
  - Conditional branching on size of padding

- Timing difference is negligible but easy to detect with cache attacks (e.g., a conditional call to memset)

Data Conversion

- RSA decryption works with big integer numbers
- PKCS #1 v1.5 padding scheme works with bytes
- Converting from number to bytes in constant time is hard
- We found:
  - Conditional padding with zeros for small numbers
  - Conditional branching on size of padding

- Timing difference is negligible but easy to detect with cache attacks (e.g., a conditional call to memset)
- Vulnerabilities arise from low-level serialization functions

PKCS #1 v1.5 Verification

- Requires multiple validity checks

PKCS #1 v1.5 Verification

- Requires multiple validity checks

- We found:

PKCS #1 v1.5 Verification

- Requires multiple validity checks

- We found:
  - Conditional calls to memcpy

PKCS #1 v1.5 Verification

- Requires multiple validity checks

- We found:
  - Conditional calls to memcpy
  - Conditional writes to error log

PKCS #1 v1.5 Verification

- Requires multiple validity checks

- We found:
  - Conditional calls to memcpy
  - Conditional writes to error log
  - Conditional branching on validity checks

PKCS #1 v1.5 Verification

- Requires multiple validity checks

- We found:
  - Conditional calls to memcpy
  - Conditional writes to error log
  - Conditional branching on validity checks

- Again - Timing difference is negligible but easy to detect with cache attacks

TLS mitigation

- Goal – same behavior if verification succeeds or fails
    - Use random key if verification fails

TLS mitigation

- Goal – same behavior if verification succeeds or fails
  - Use random key if verification fails
- We found:

TLS mitigation

- Goal – same behavior if verification succeeds or fails
    - Use random key if verification fails

- We found:
    - Conditional branching on verification results

TLS mitigation

- Goal – same behavior if verification succeeds or fails
    - Use random key if verification fails

- We found:
    - Conditional branching on verification results
    - Conditional memory accesses

TLS mitigation

- Goal – same behavior if verification succeeds or fails
    - Use random key if verification fails
- We found:
    - Conditional branching on verification results
    - Conditional memory accesses
    - Conditional calls to random key generation

TLS mitigation

- Goal – same behavior if verification succeeds or fails
  - Use random key if verification fails

- We found:
  - Conditional branching on verification results
  - Conditional memory accesses
  - Conditional calls to random key generation

- Again - Timing difference is negligible but easy to detect with cache attacks

Our results

Our results

- New Techniques for Microarchitectural Padding Oracle Attacks, vulnerabilities in 7 out 9 implementations
  - PoC for Manger and Bleichenbacher attacks

Our results

- New Techniques for Microarchitectural Padding Oracle Attacks, vulnerabilities in 7 out 9 implementations
    - PoC for Manger and Bleichenbacher attacks

- Boosting attack efficacy using BEAST

Our results

- New Techniques for Microarchitectural Padding Oracle Attacks, vulnerabilities in 7 out 9 implementations
  - PoC for Manger and Bleichenbacher attacks
- Boosting attack efficacy using BEAST
- Parallelization for downgrade attack
  - PoC for Manger parallelization using LLL

99% OF THE WORLDS COOKIES ARE CONSUMED BY 1% OF THE MONSTERS
LIFE
# OCCUPY SESAME STREET

Disclosure

- We disclosed to:
  - OpenSSL, Mozzila’s NSS, Amazon’s s2n, Apple’s CoreTLS, mbed TLS, wolfSSL, GnuTLS
- All have patched their code, with various levels of success
- Lots of stories…

Recommendation

- Many recommendations for several layers of mitigations in the paper
  - Bottom line **Don’t use RSA KX**
  - It has failed us too many times

Recommendation

- Many recommendations for several layers of mitigations in the paper
  - Bottom line **Don’t use RSA KX**
  - It has failed us too many times
- If you really really really must
  - Separate your certificates!
  - …

Recommendation

- Many recommendations for several layers of mitigations in the paper
  - Bottom line **Don’t use RSA KX**
  - It has failed us too many times
- If you really really really must
  - Separate your certificates!
  - …
- But please just

Re

- Ma
  mit
  - B
  - It
- If y
  - S
  - ..
- But

Conclusion

- Mitigating padding attacks on RSA is not impossible (but very close to it)

- Paper website
  https://cat.eyalro.net

Conclusion

- Mitigating padding attacks on RSA is not impossible (but very close to it)

- Paper website
  https://cat.eyalro.net

- Any questions?