Time is on my Side

Exploiting Timing Side Channel Vulnerabilities on the Web

Sebastian Schinzel*
Friedrich-Alexander Universität Erlangen-Nürnberg
Lehrstuhl für Informatik 1
IT-Sicherheitsinfrastrukturen

Web 1.0: sebastian.schinzel@cs.fau.de
Web 2.0: https://twitter.com/seecurity

*Supported by Deutsche Forschungsgemeinschaft (DFG)
as part of SPP 1496 “Reliably Secure Software Systems”

Me

- PhD candidate at the Security Research Group within the Department of Computer Science in Erlangen (Prof. Felix Freiling)
  - Side Channel attacks & mitigations
  - Software security, Penetration Testing
- Professional work at Virtual Forge GmbH
  - SAP Security, focus on SAP's programming language ABAP
  - Static code analysis, Penetration Testing

Penetration Testing - Movies vs. Reality

Hollywood-style penetration testing:

http://en.wikipedia.org/wiki/Swordfish_(film)

"Gabriel pressures Stanley [...] to hack a government system in 60 seconds while simultaneously being held at gunpoint by Gabriel's bodyguard [...] and receiving fellatio from a young woman. [...]

Stanley succeeded in hacking the system."

Penetration Testing - Movies vs. Reality

A bit more structure when outside of Hollywood:

- Preparation
- Reconnaissance (gather information)
- Evaluation of gathered information
- Testing & Exploiting
- Reporting

What "Domino's Pizza" knows about US foreign affairs...

"And Bomb The Anchovies" - http://www.time.com/time/magazine/article/0,9171,970860,00.html

What "Domino's Pizza" knows about US foreign affairs...

"And Bomb The Anchovies" - http://www.time.com/time/magazine/article/0,9171,970860,00.html

Common Invasive Attacks vs. Side Channel Attacks

Invasive attacks (e.g. Buffer Overflows, SQL Injection, XSS, Format String Injection, ...)

➡ change original control flow

Side Channels (storage side channels, timing side channels)

➡ don't change original control flow

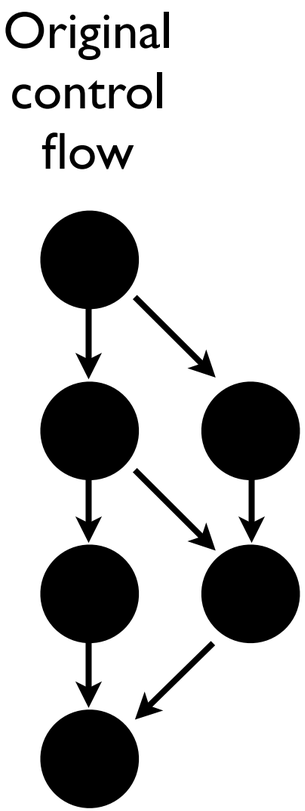
Common Invasive Attacks vs. Side Channel Attacks

Invasive attacks (e.g. Buffer Overflows, SQL Injection, XSS, Format String Injection, ...)

➡ change original control flow

Side Channels (storage side channels, timing side channels)

➡ don't change original control flow



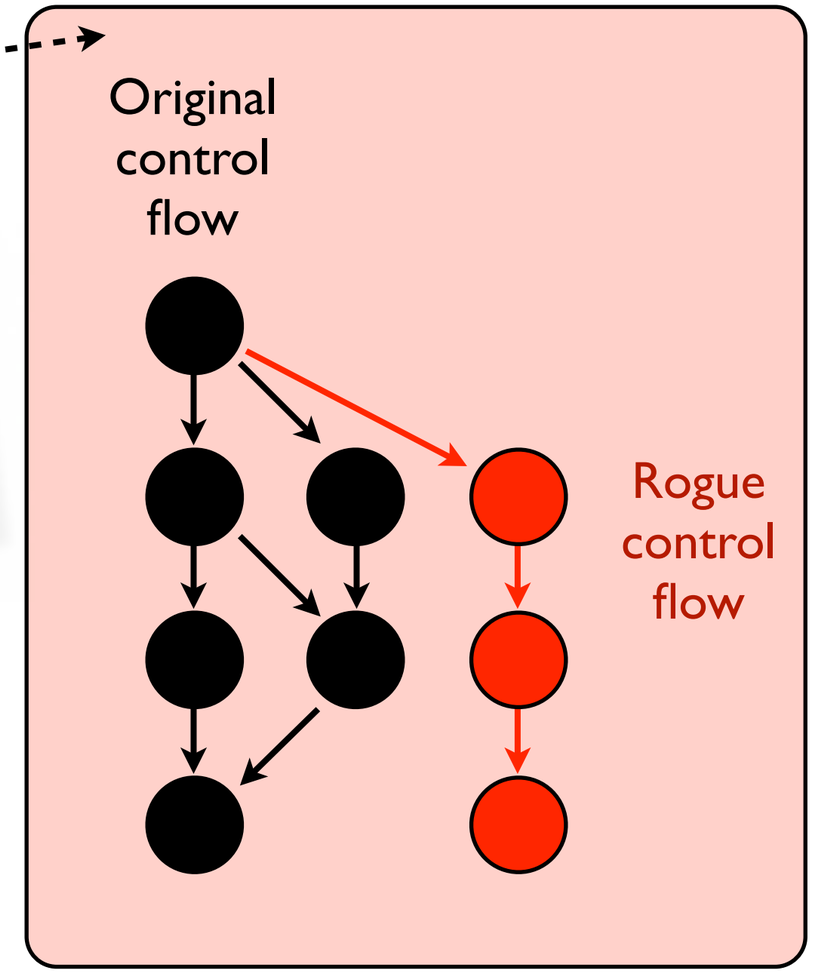
Common Invasive Attacks vs. Side Channel Attacks

Invasive attacks (e.g. Buffer Overflows, SQL Injection, XSS, Format String Injection, ...)

➡ change original control flow

Side Channels (storage side channels, timing side channels)

➡ don't change original control flow



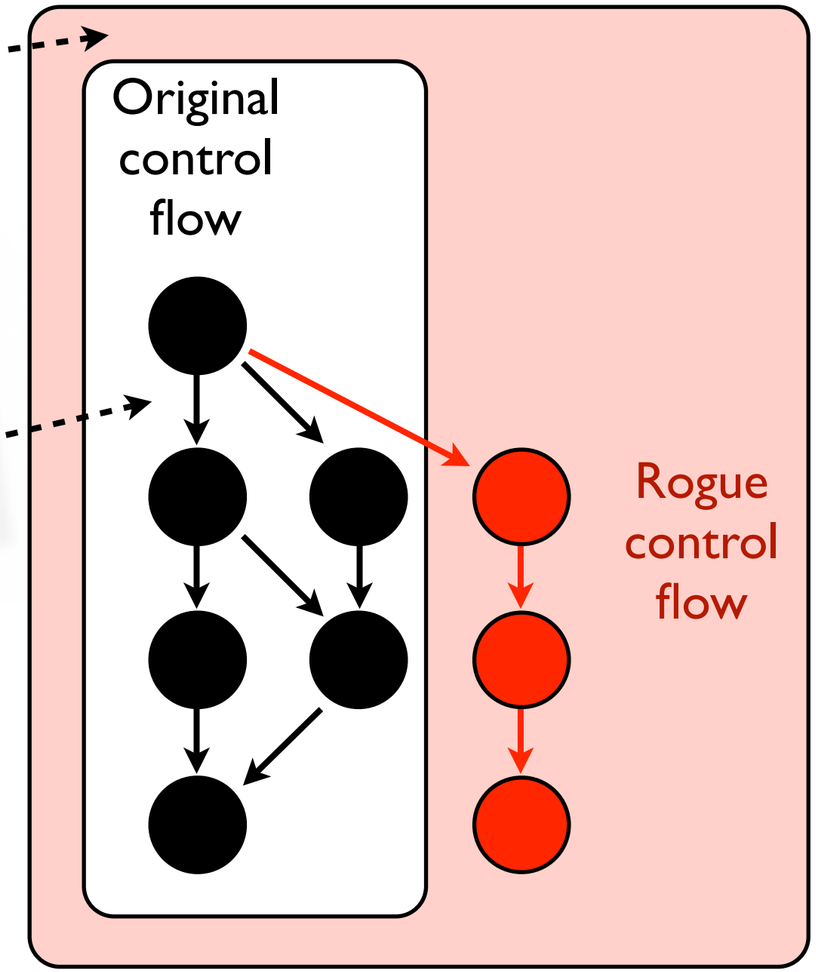
Common Invasive Attacks vs. Side Channel Attacks

Invasive attacks (e.g. Buffer Overflows, SQL Injection, XSS, Format String Injection, ...)

➡ change original control flow

Side Channels (storage side channels, timing side channels)

➡ don't change original control flow

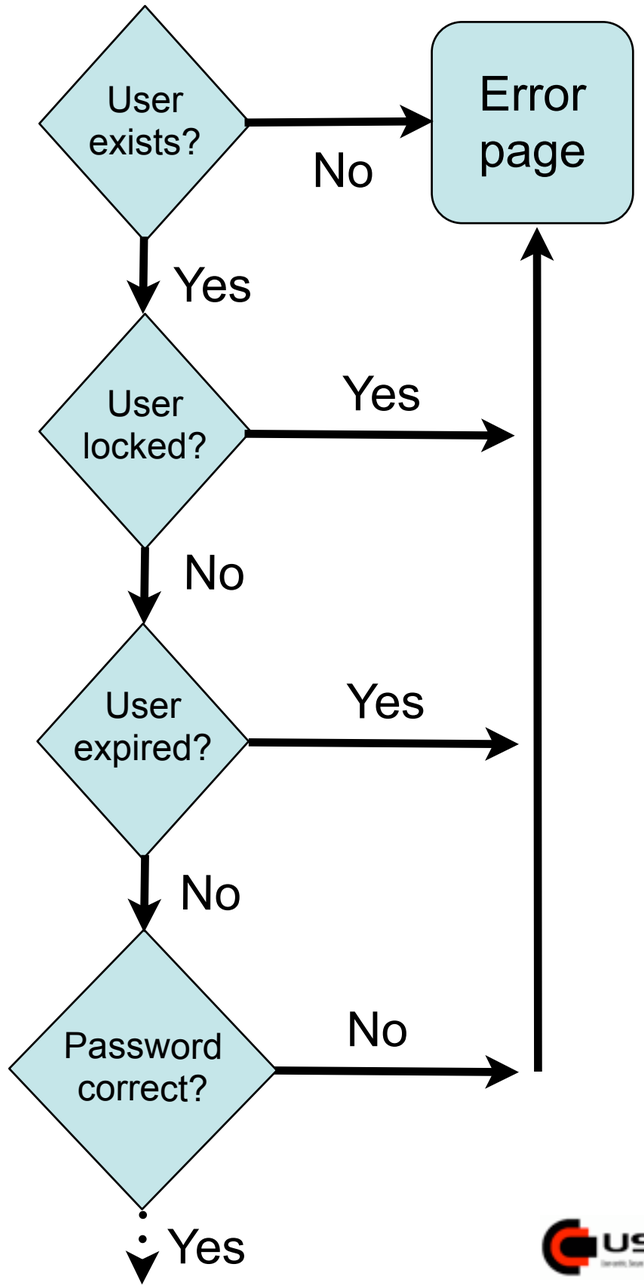


Benign differences on protocol level correlate with sensitive information [7] (here Typo3 backend)

Non-existent user name (s=0)	*Existing user name (s=1)*
HTTP/1.1 200 OK *Date: Mon, 25 Jan 2010 11:47:55 GMT* *Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny4 with Suhosin-Patch* *X-Powered-By: PHP/5.2.6-1+lenny4* *Expires:* ***Thu, 19 Nov 1981 08:52:00 GMT*** ***Last-Modified: Mon, 25 Jan 2010 11:47:55 GMT*** ***Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0*** ***Pragma: no-cache*** *Vary: Accept-Encoding* *Content-Type: text/html;charset=iso-8859-1* *Content-Length: 5472*	*HTTP/1.1 200 OK* *Date: Mon, 25 Jan 2010 11:47:45 GMT* *Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny4 with Suhosin-Patch* *X-Powered-By: PHP/5.2.6-1+lenny4* *Expires:* ***0*** ***Cache-Control: no-cache, must-revalidate*** ***Pragma: no-cache*** ***Last-Modified: Mon, 25 Jan 2010 11:47:45 GMT*** *Vary: Accept-Encoding* *Content-Type: text/html;charset=iso-8859-1* *Content-Length: 5472*

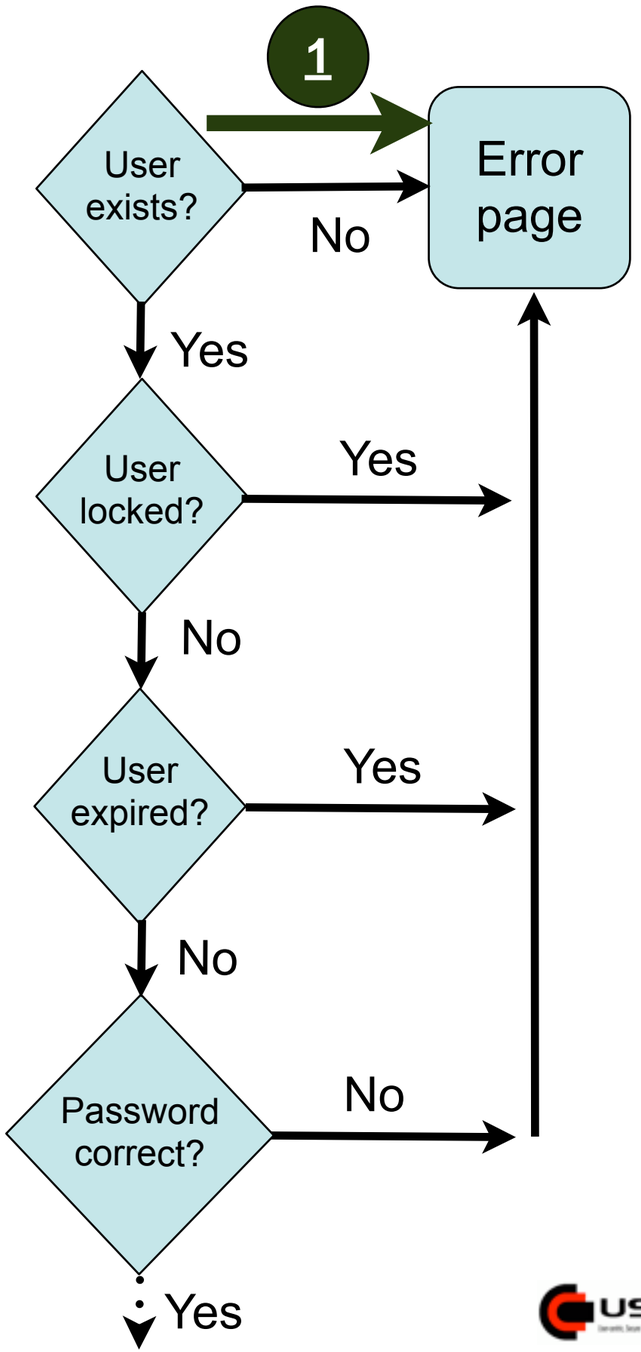
Timing Side Channels

Response time depends on secret information (Typo3 backend)



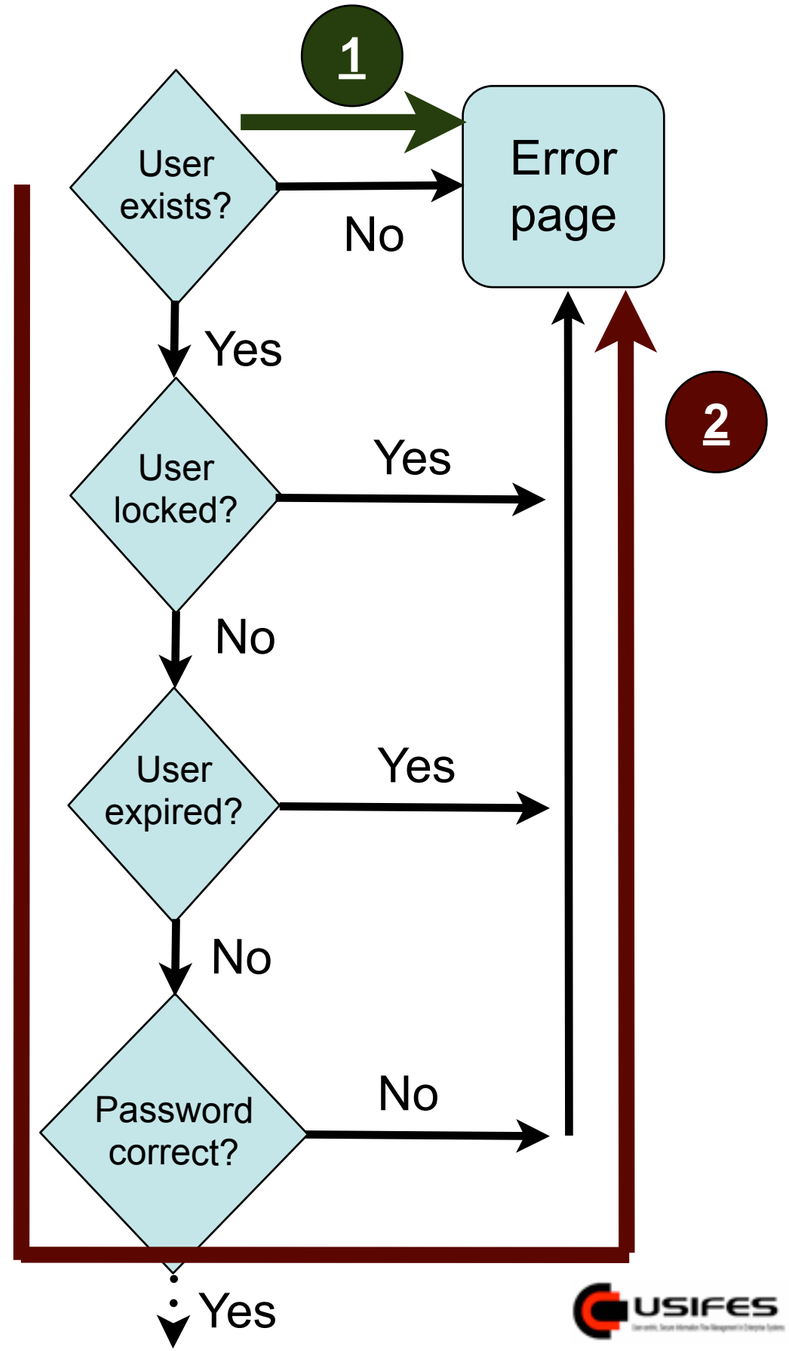
Timing Side Channels

Response time depends on secret information (Typo3 backend)



Timing Side Channels

Response time depends on secret information (Typo3 backend)

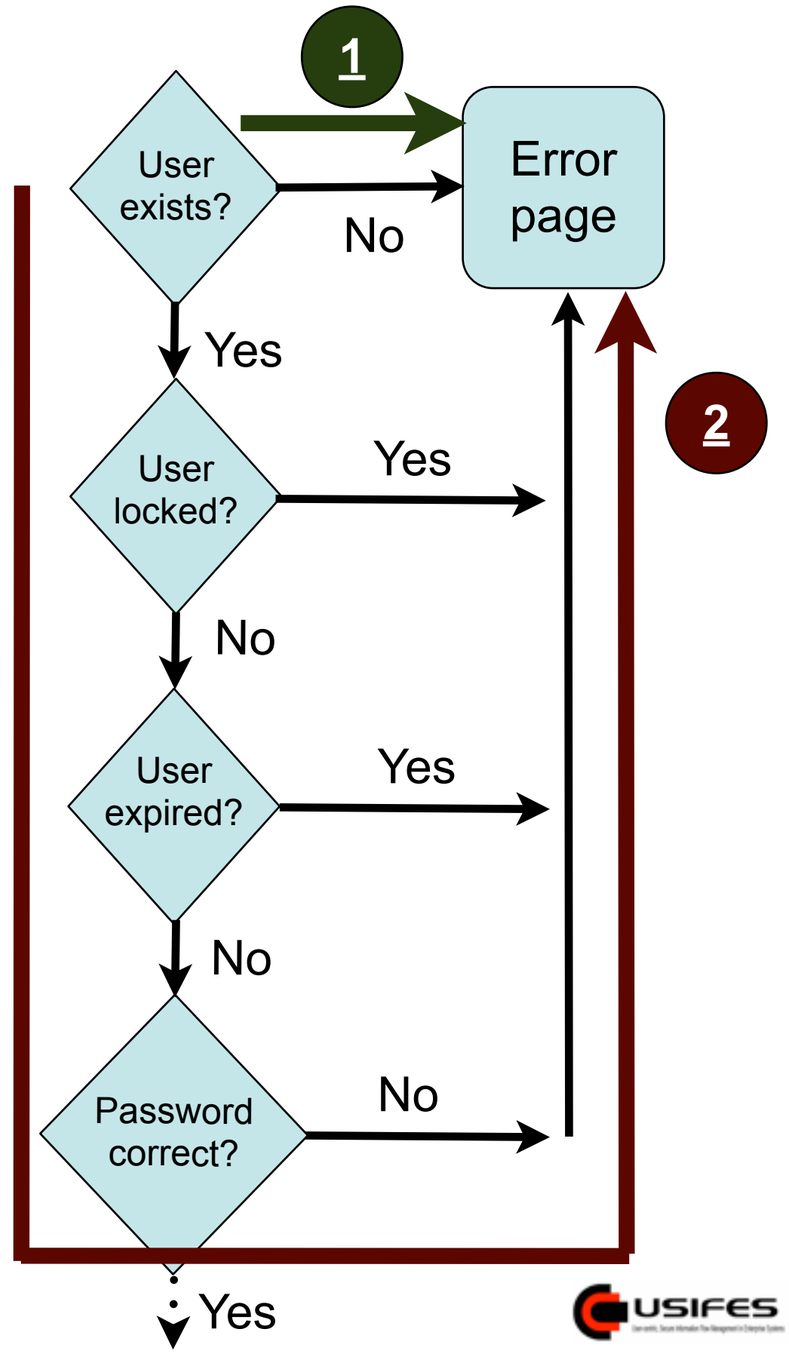


Timing Side Channels

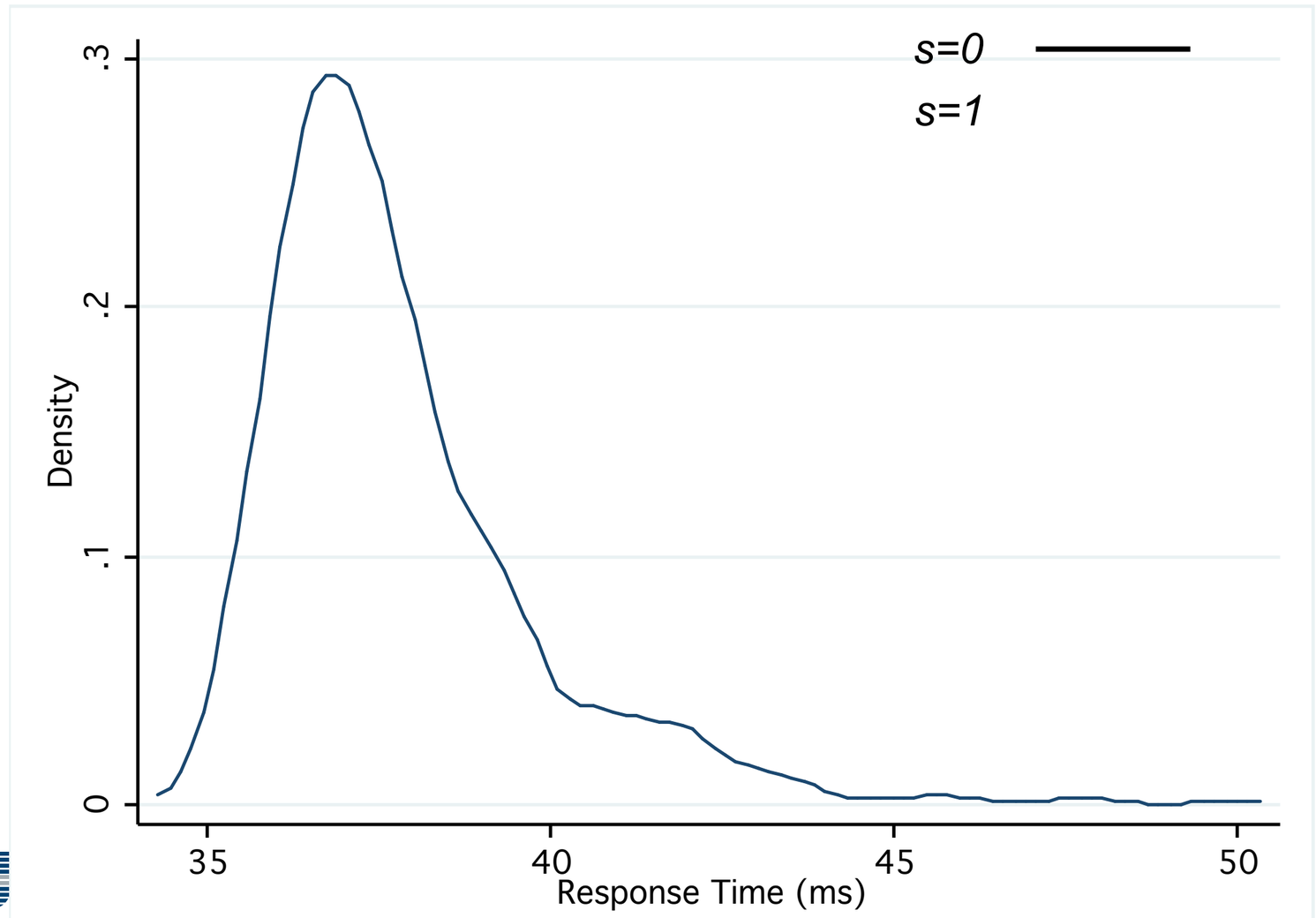
Response time depends on secret information (Typo3 backend)

Unfortunately, it's not that easy...

- Problem: random like delays (jitter) makes measuring response times difficult
  - You cannot directly measure response time *t*, but only *t* + *jitter*
- Analysing timing channels can be quite challenging...



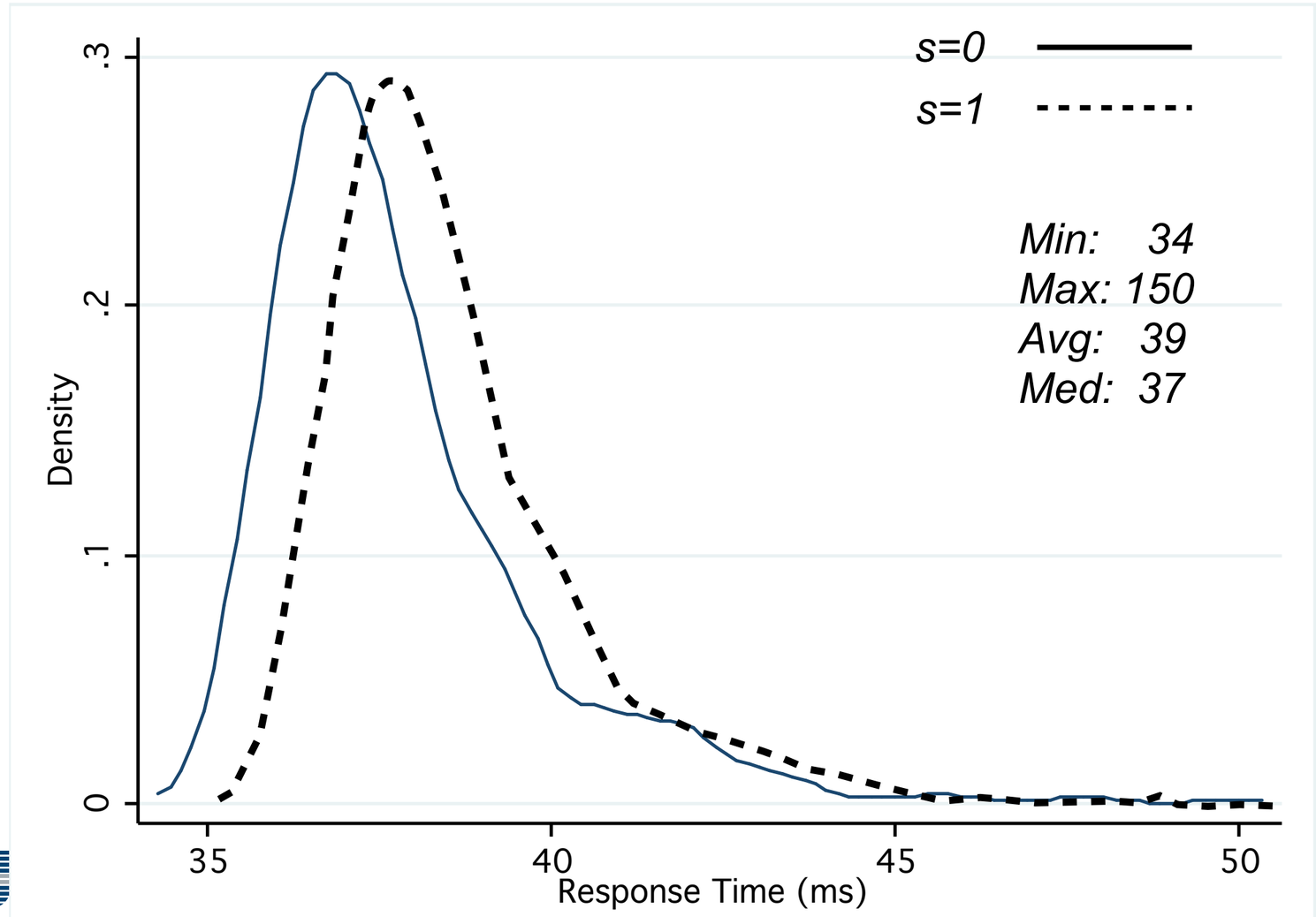
- Example: Two different values measured many times



Analysing Timing Measurements

Difficulty of Timing Measurements

- Example: Two different values measured many times



Dos and Don’ts of Timing Measurements

Timing precision

- Measurement precision over network down to single digit microseconds
  - ... and even hundreds of nanoseconds in certain scenarios [1]

- A few tips of how to do timing measurements over networks:

Use fine-grained timer (`rdtsc` assembly instruction)

```
unsigned long long ret;
unsigned long minor;
unsigned long mayor;

asm volatile(
     "cpuid \n"    // Prevent out of order execution
       "rdtsc"
    : "=a"(minor), // lower 32bit of result
       "=d"(mayor)  // lower 32bit of result
    : "a" (0)
    : "%ebx", "%ecx"
);
ret = ((((ticks) mayor) << 32) | ((ticks) minor));

// Result: 64 bit value with clock ticks
// since CPU initialisation
```

Dos and Don'ts of Timing Measurements

Fine-grained timer

Use fine-grained timer (`rdtsc` assembly instruction)

- PRO: tied to the CPU clock speed
  - 1 rdtsc tick = 1/clock speed
  - 1 rdtsc tick on 2GHz CPU = 0.5 nanoseconds

Use fine-grained timer (`rdtsc` assembly instruction)

- PRO: tied to the CPU clock speed
  - 1 rdtsc tick = 1/clock speed
  - 1 rdtsc tick on 2GHz CPU = 0.5 nanoseconds
- CON: tied to the CPU clock speed
  - Clock speed fluctuations because of power management
  - Convertion from rdtsc tickt to second? No platform-independent way to get CPU clock speed

Parallelise measurements

Naïve timing measurement approach:

1. measure $A = \langle A_1, A_2, A_3, A_4, ..., A_n \rangle$ at time t_a
2. measure $B = \langle B_1, B_2, B_3, B_4, ..., B_n \rangle$ at time t_b
3. compare the sets of timings A and B for significant timing differences

Problem: The jitter at time t_a was probably different than at time t_b

Parallelise measurements

Naïve timing measurement approach:

1. measure $A = \langle A_1, A_2, A_3, A_4, ..., A_n \rangle$ at time t_a
2. measure $B = \langle B_1, B_2, B_3, B_4, ..., B_n \rangle$ at time t_b
3. compare the sets of timings A and B for significant timing differences

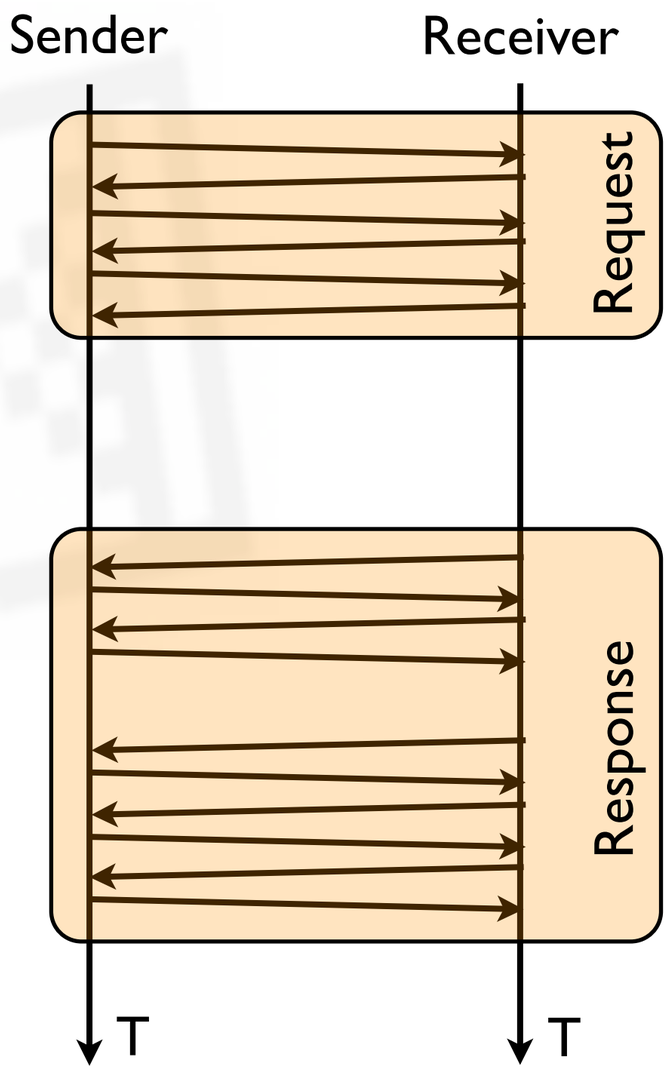
Problem: The jitter at time t_a was probably different than at time t_b

Better timing measurement approach:

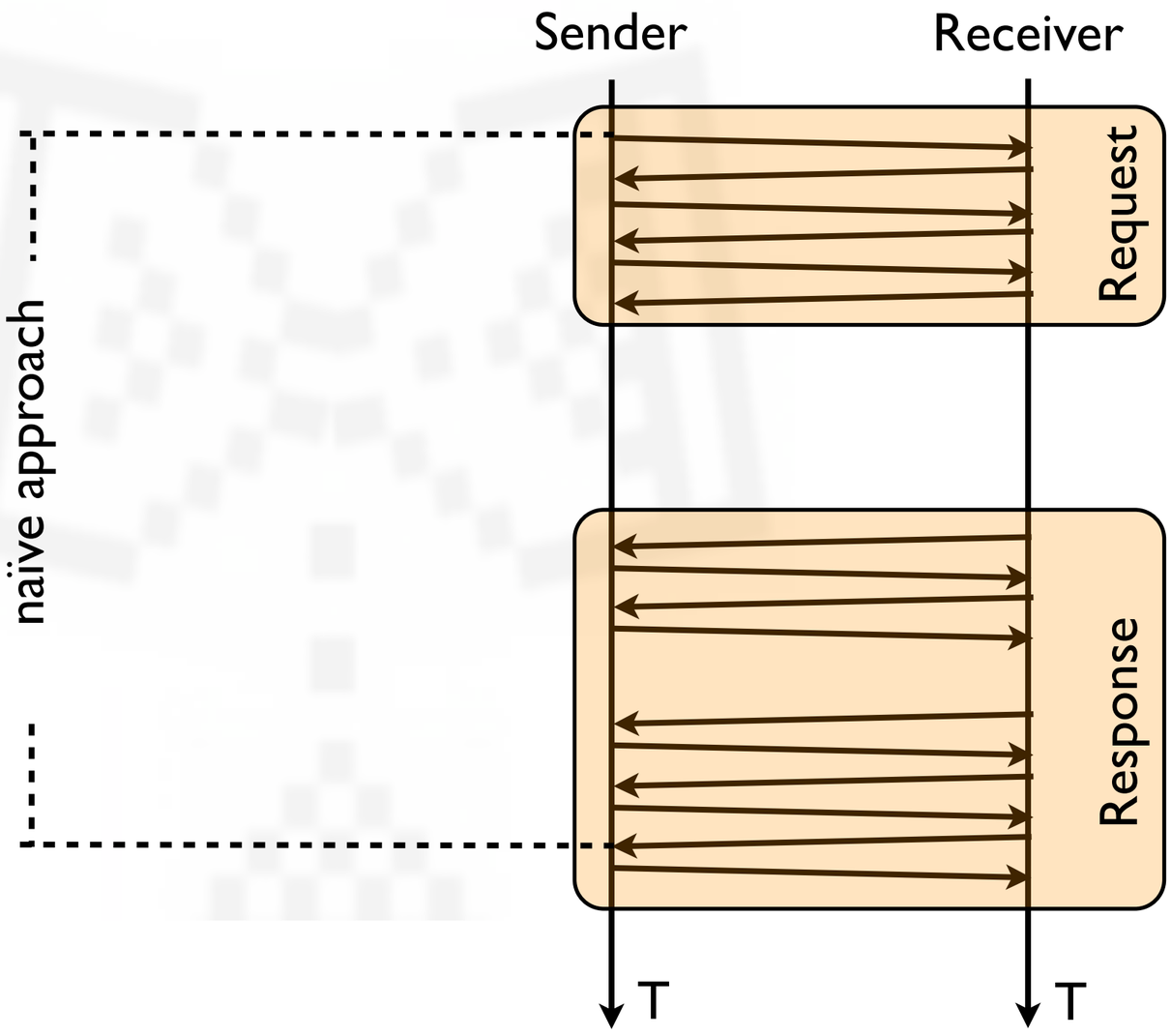
1. measure alternatingly $A_1, B_1, A_2, B_2, A_3, B_3, A_4, B_4, ..., A_n B_n$
2. separate A and B
3. compare the sets of timings A and B

Solution: Jitter in A and B are approximately the same

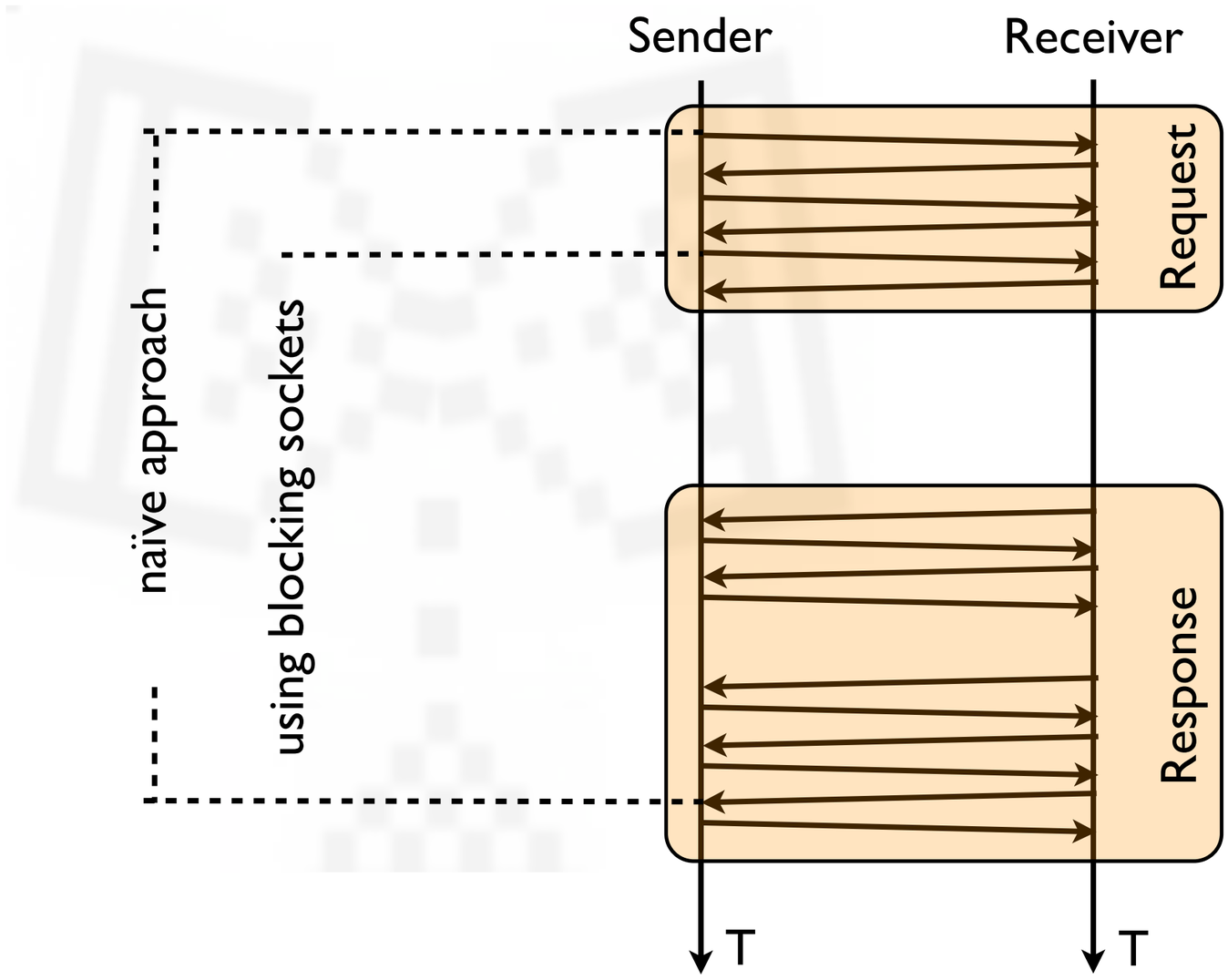
Starting & end point for measurements



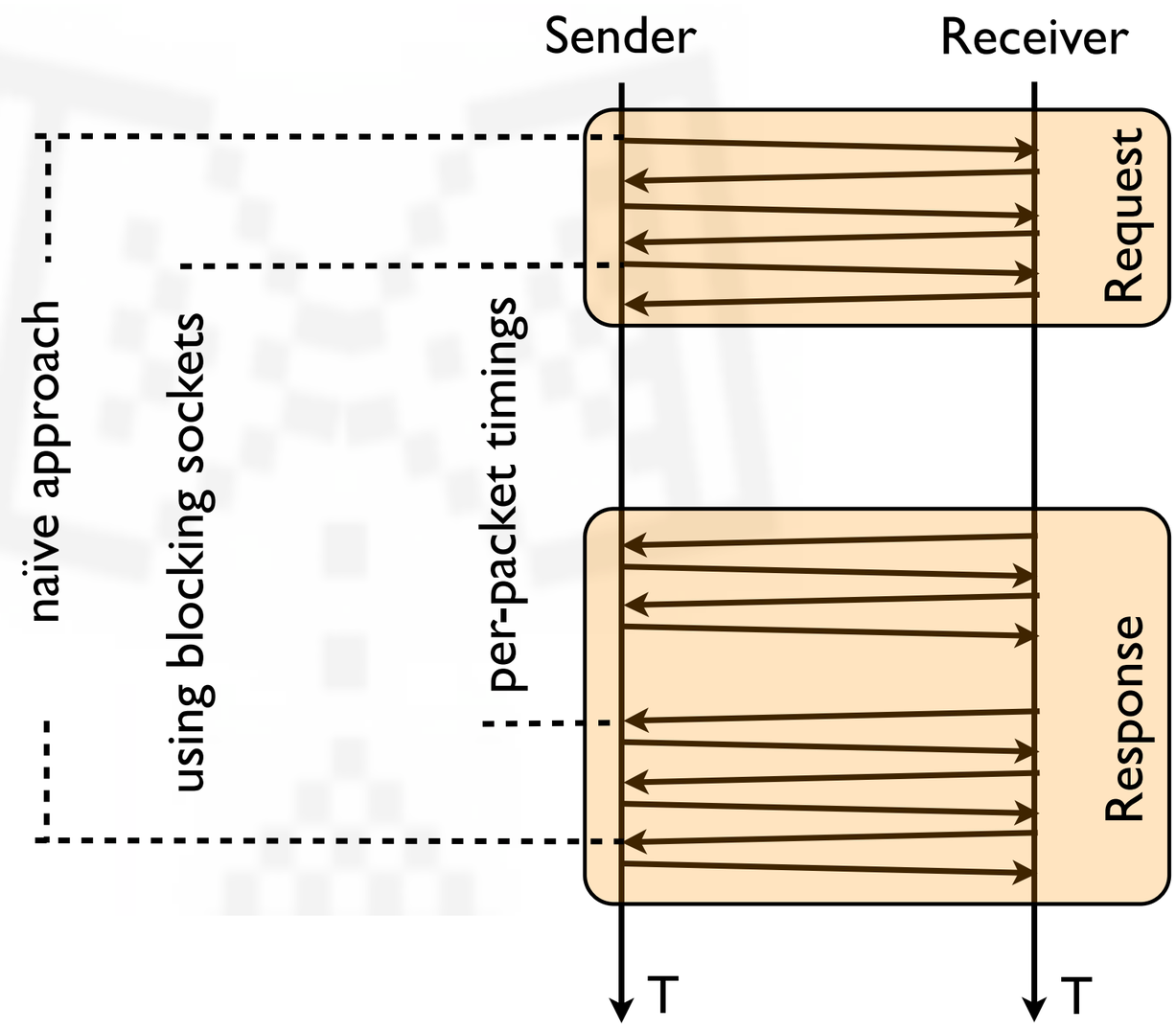
Starting & end point for measurements



Starting & end point for measurements



Starting & end point for measurements

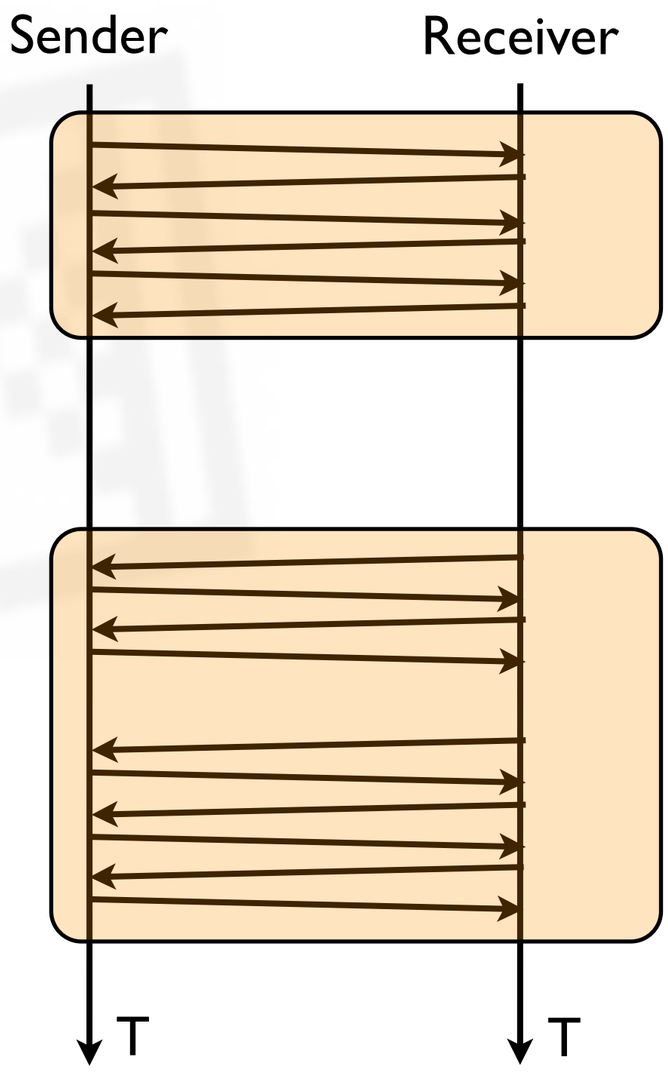


Dos and Don'ts of Timing Measurements

Choose starting and end point for measurements

Starting & end point for measurements

1. ⧖ start timer
2. send request
3. receive request
4. ⧖ stop timer

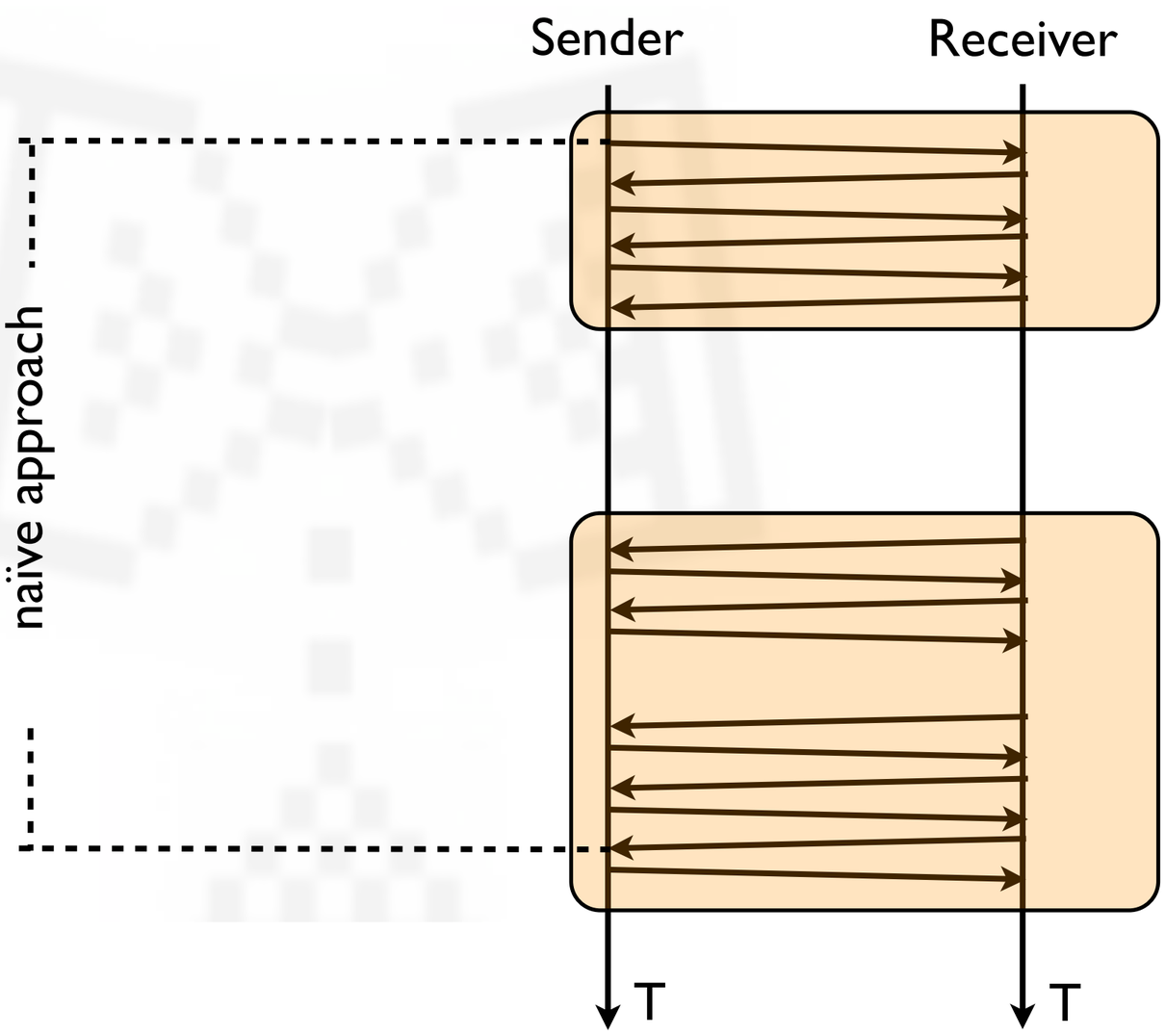


Dos and Don'ts of Timing Measurements

Choose starting and end point for measurements

Starting & end point for measurements

1. ⧖ start timer
2. send request
3. receive request
4. ⧖ stop timer

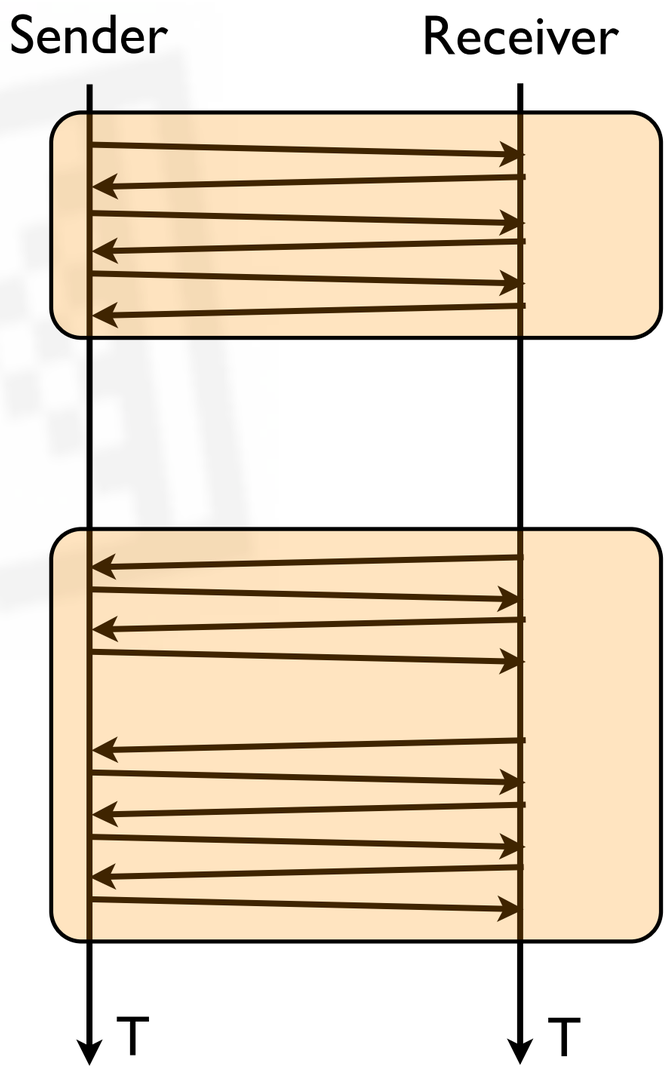


Dos and Don'ts of Timing Measurements

Choose starting and end point for measurements

Starting & end point for measurements

1. send n-1 bytes of request
2. ⧖ start timer
3. send last byte of request
4. receive response
5. ⧖ stop timer

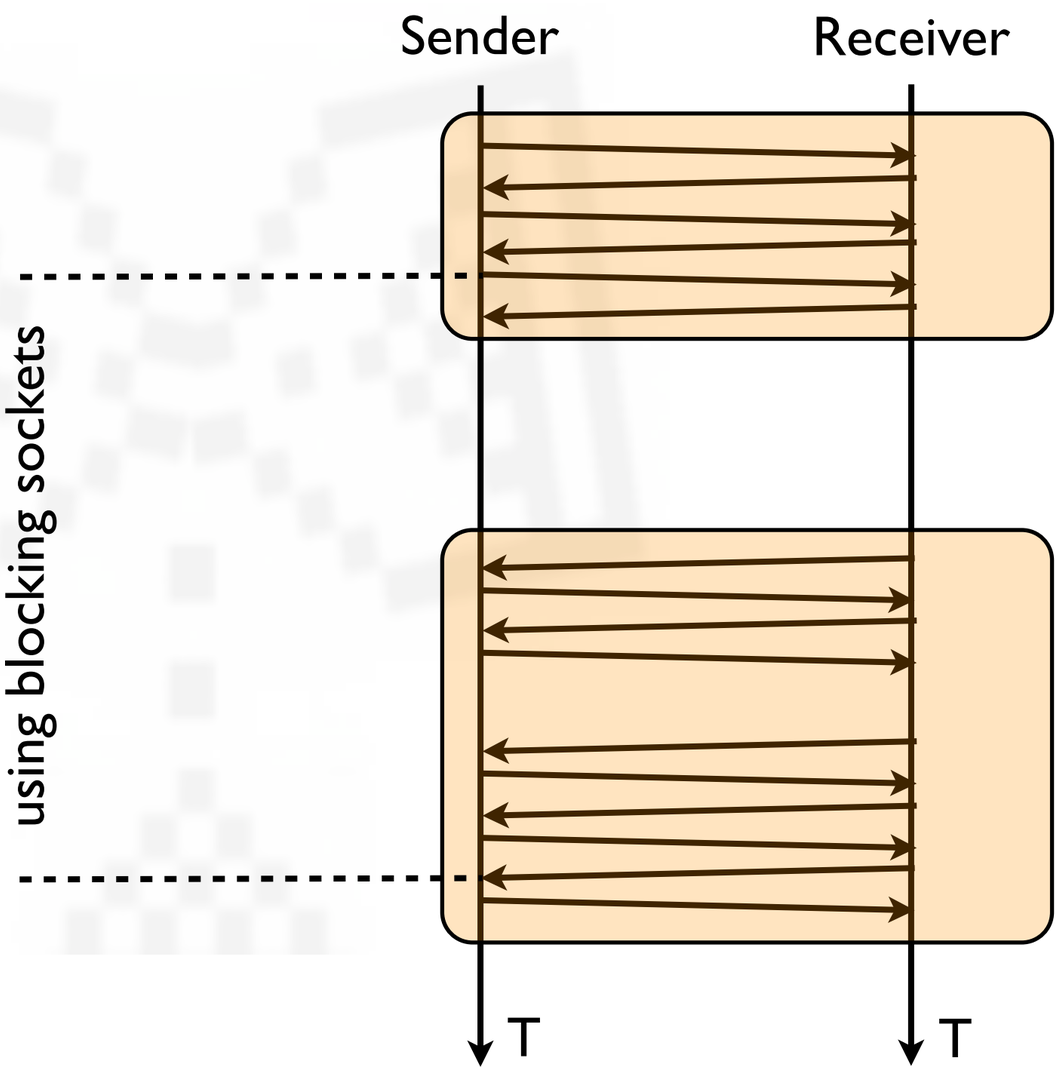


Dos and Don'ts of Timing Measurements

Choose starting and end point for measurements

Starting & end point for measurements

1. send n-1 bytes of request
2. ⧖ start timer
3. send last byte of request
4. receive response
5. ⧖ stop timer

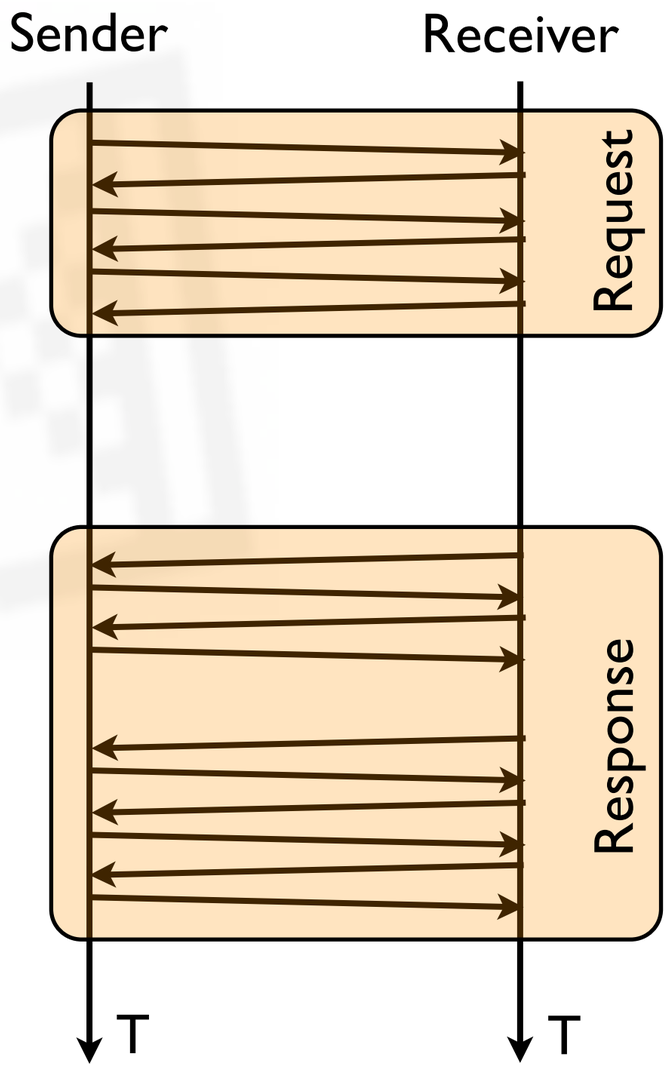


Dos and Don'ts of Timing Measurements

Choose starting and end point for measurements

Starting & end point for measurements

1. send n-1 bytes of request
2. ⌛ start timer
3. send last byte of request
4. wait for receival of n^{th} byte of response
5. ⌛ stop timer

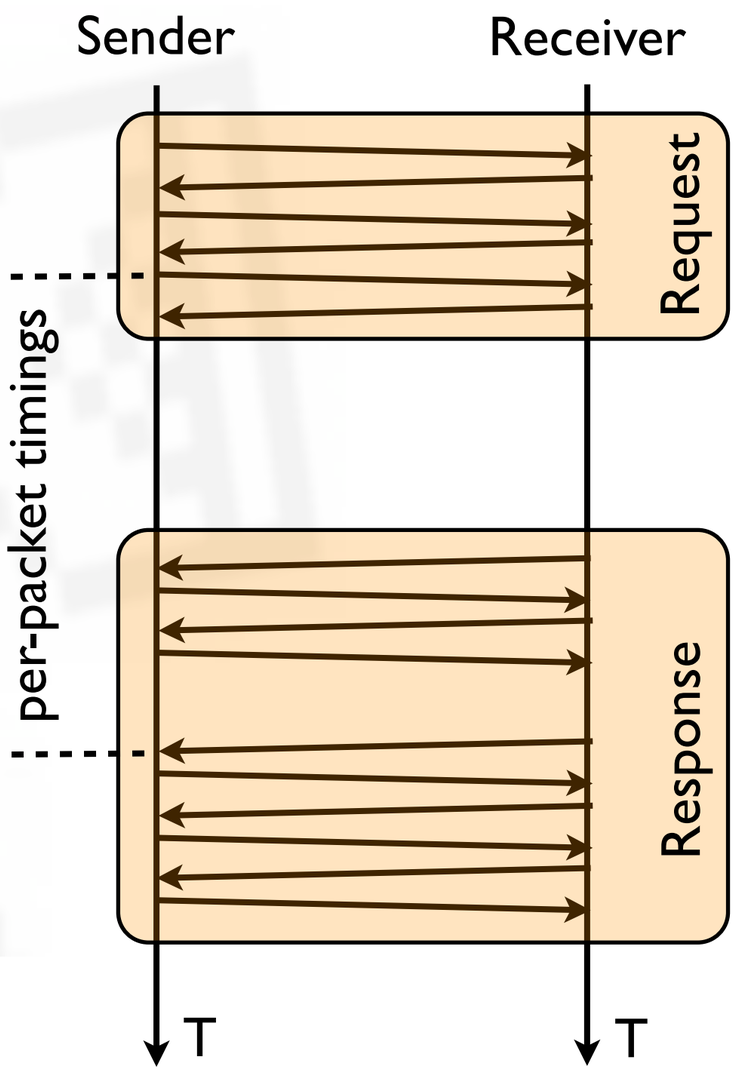


Dos and Don'ts of Timing Measurements

Choose starting and end point for measurements

Starting & end point for measurements

1. send n-1 bytes of request
2. ⧖ start timer
3. send last byte of request
4. wait for receival of n^{th} byte of response
5. ⧖ stop timer

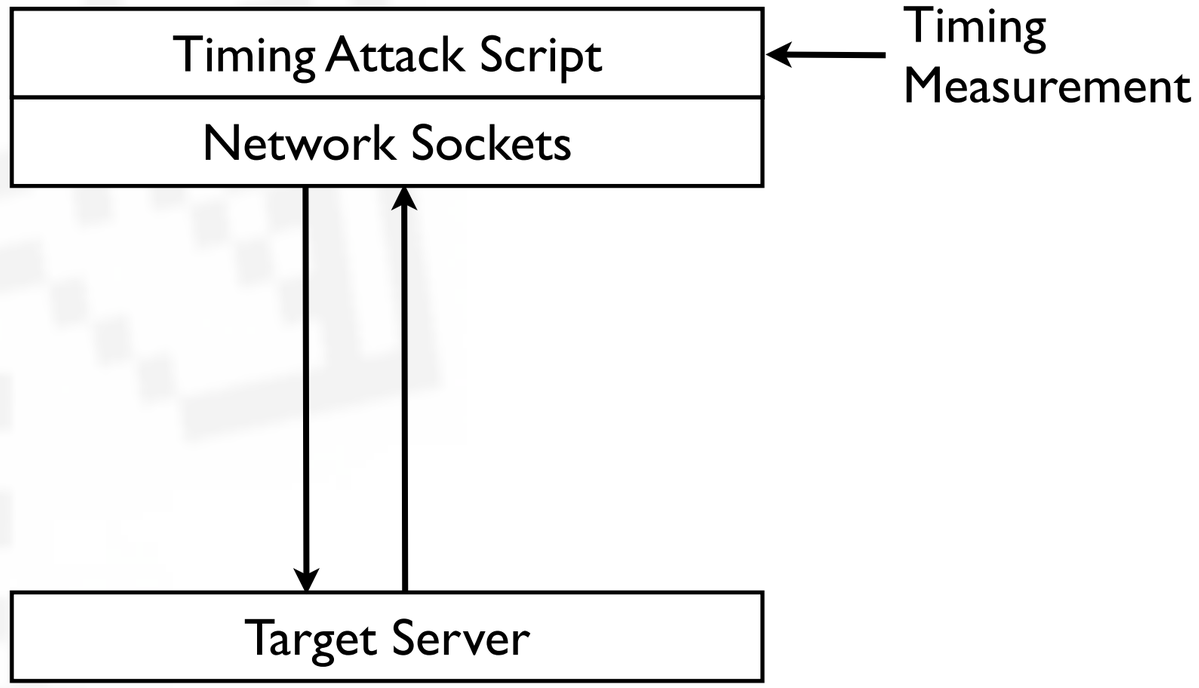


Miscellaneous tips for timing measurements

- Disable power management (e.g. SpeedStep)
- Measure over the wire (no WI-FI)
- Disable periodic tasks on your local maschine
- Keep your part of the network idle (in other words, don’t do it from hacker conferences...)
- Skip the first few dozen measurements (jitter because of cache warm-up)

Presenting FAU Timer

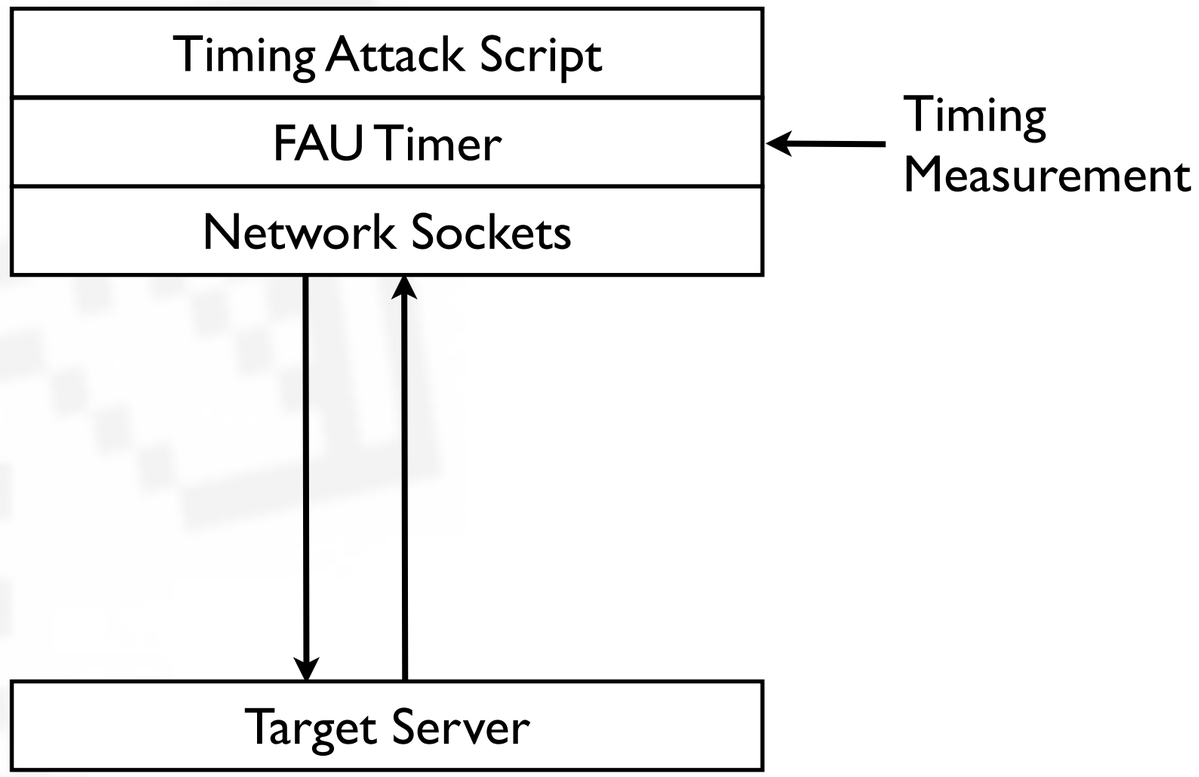
- Request/response handling within a compact C library
- Ported to Python (others are planned)
- Encapsulates logic for timing measurement



- Just send your requests, the lib does all the measurements

Presenting FAU Timer

- Request/response handling within a compact C library
- Ported to Python (others are planned)
- Encapsulates logic for timing measurement



- Just send your requests, the lib does all the measurements

Demo

Lots of help from Isabell Schmitt and Niels Iciek

Planned release in January, will be announced on Twitter: @seecurity

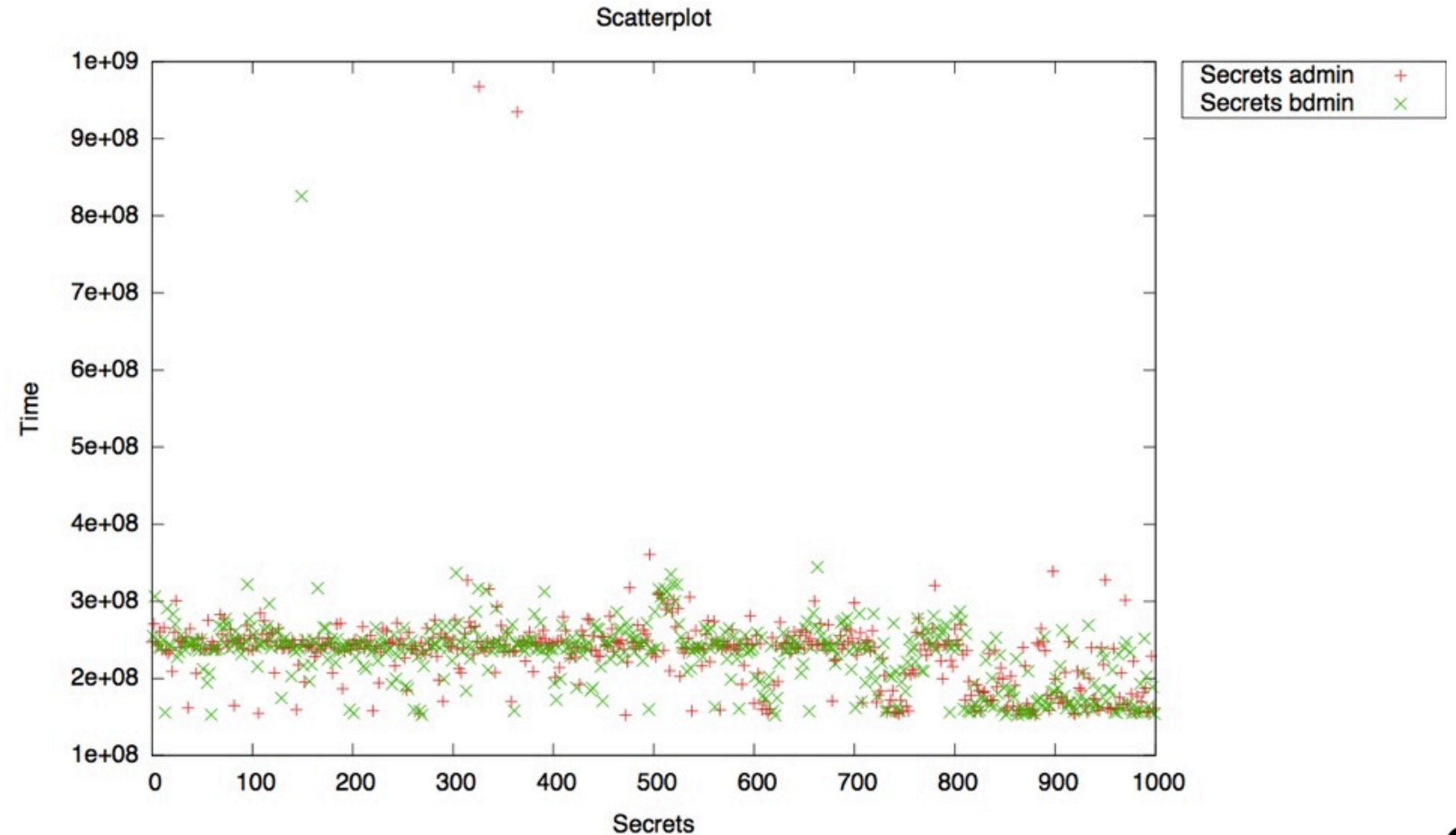
Analysing Timing Measurements

Analyse measurements graphically

- Tools
  - Your favorite spreadsheet
  - Gnuplot, Mathlab, Stata, R, ...
- Display data in various plot types
  - Scatter-Plot (detect temporal disturbances, overall quality of measurements)
  - Box-Plot (compare median, min, max, lower & upper quartile)
  - Histogram, Cumulative Distribution Function (CDF) (compare distributions of data sets)

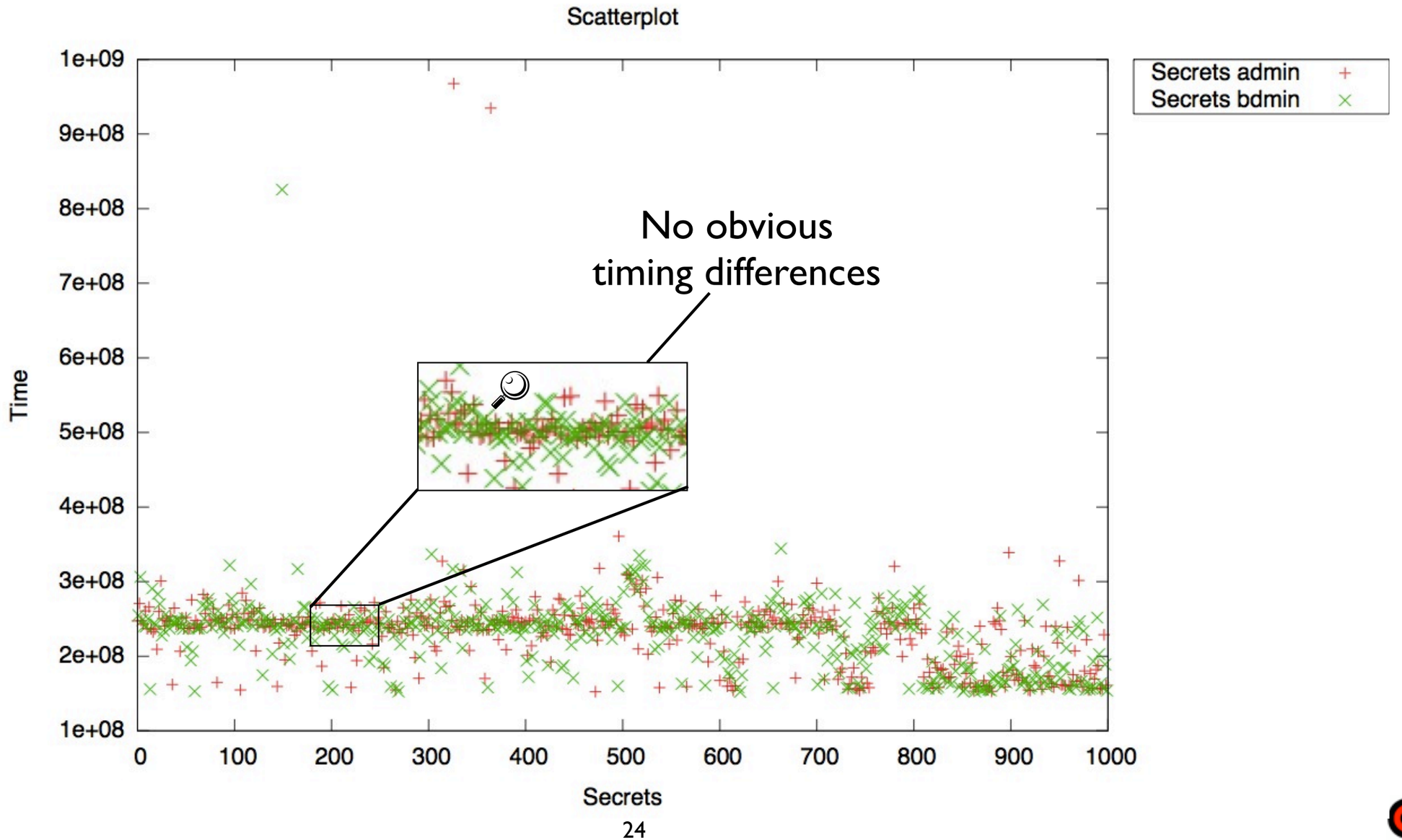
Analysing Timing Measurements

Analyse measurements graphically - Scatterplot



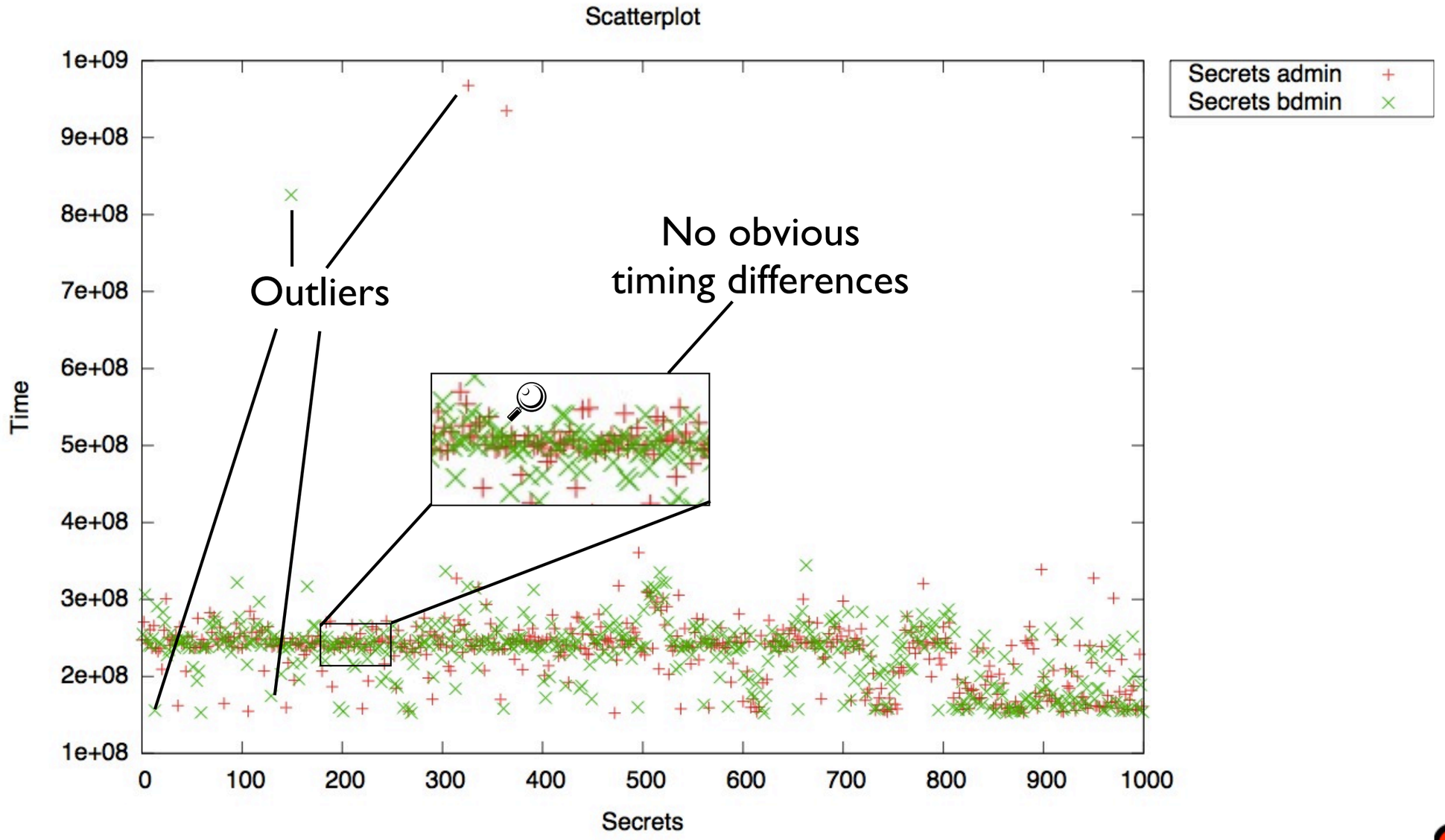
Analysing Timing Measurements

Analyse measurements graphically - Scatterplot



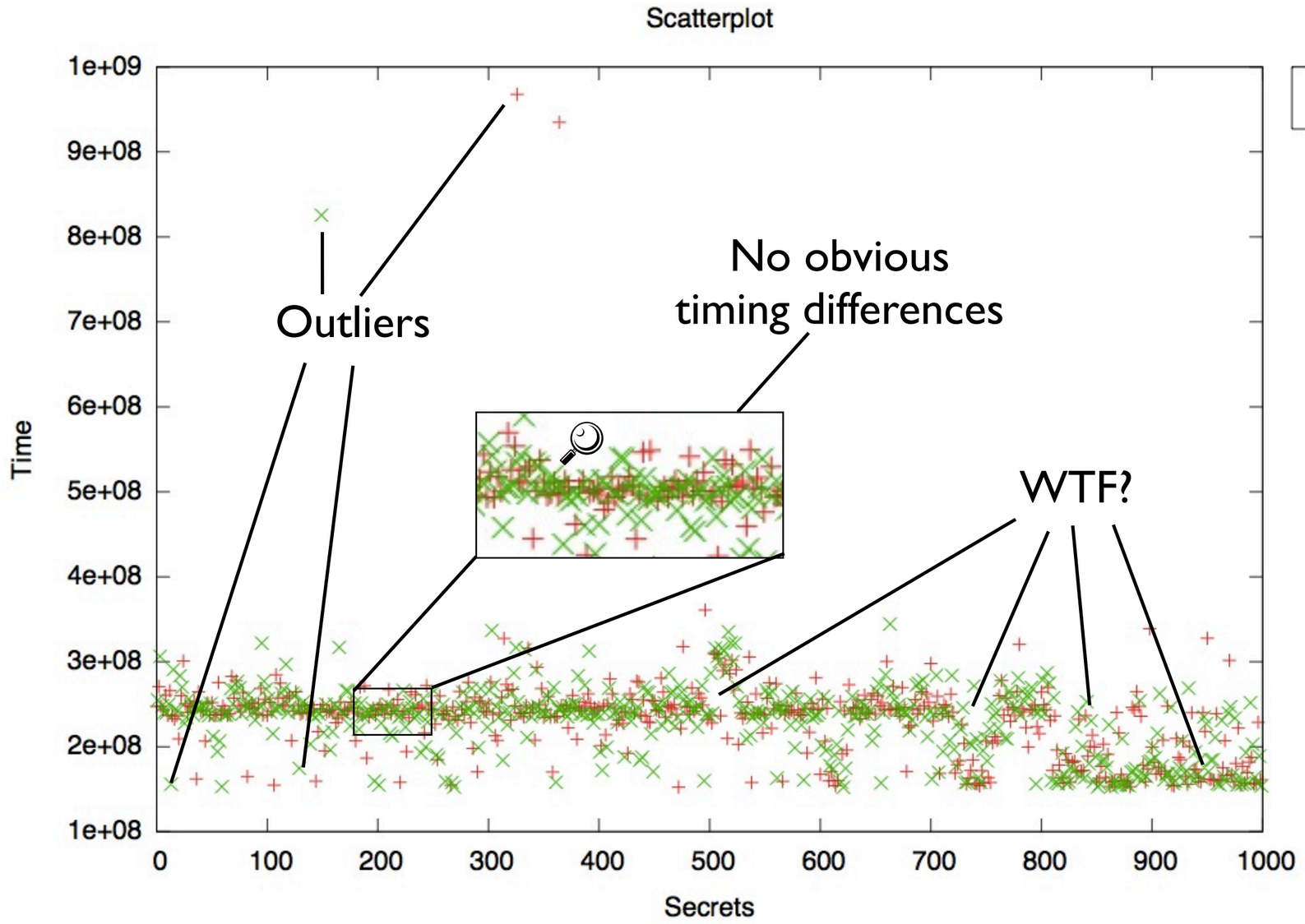
Analysing Timing Measurements

Analyse measurements graphically - Scatterplot



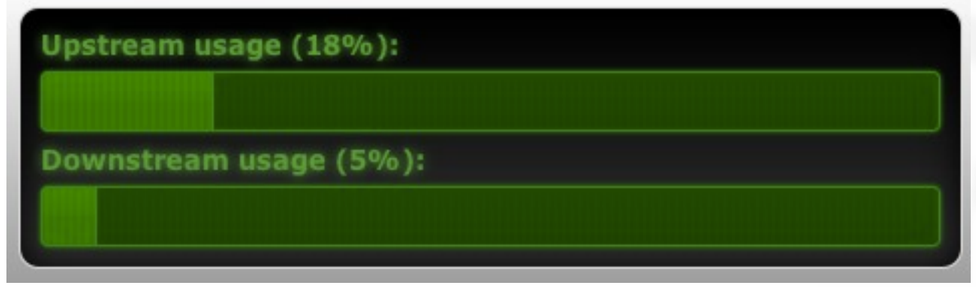
Analysing Timing Measurements

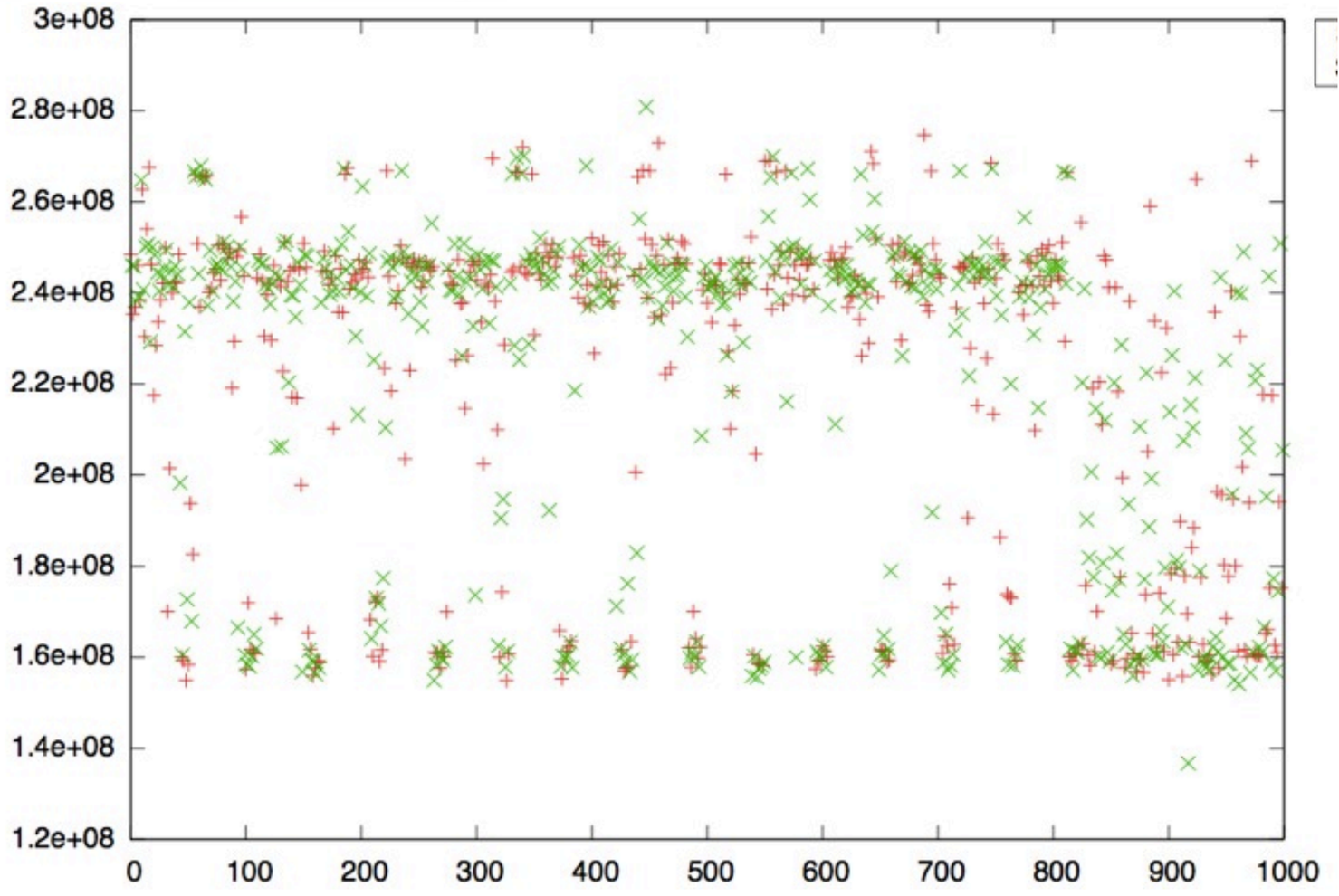
Analyse measurements graphically - Scatterplot



Time: 16:30 in the speakers' room:

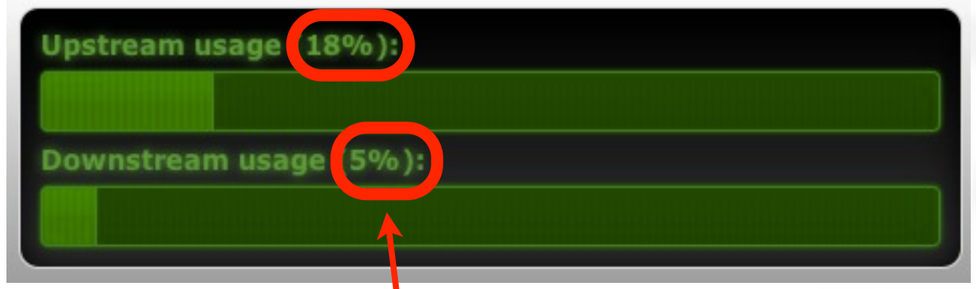
<u>28c3 in action...</u>



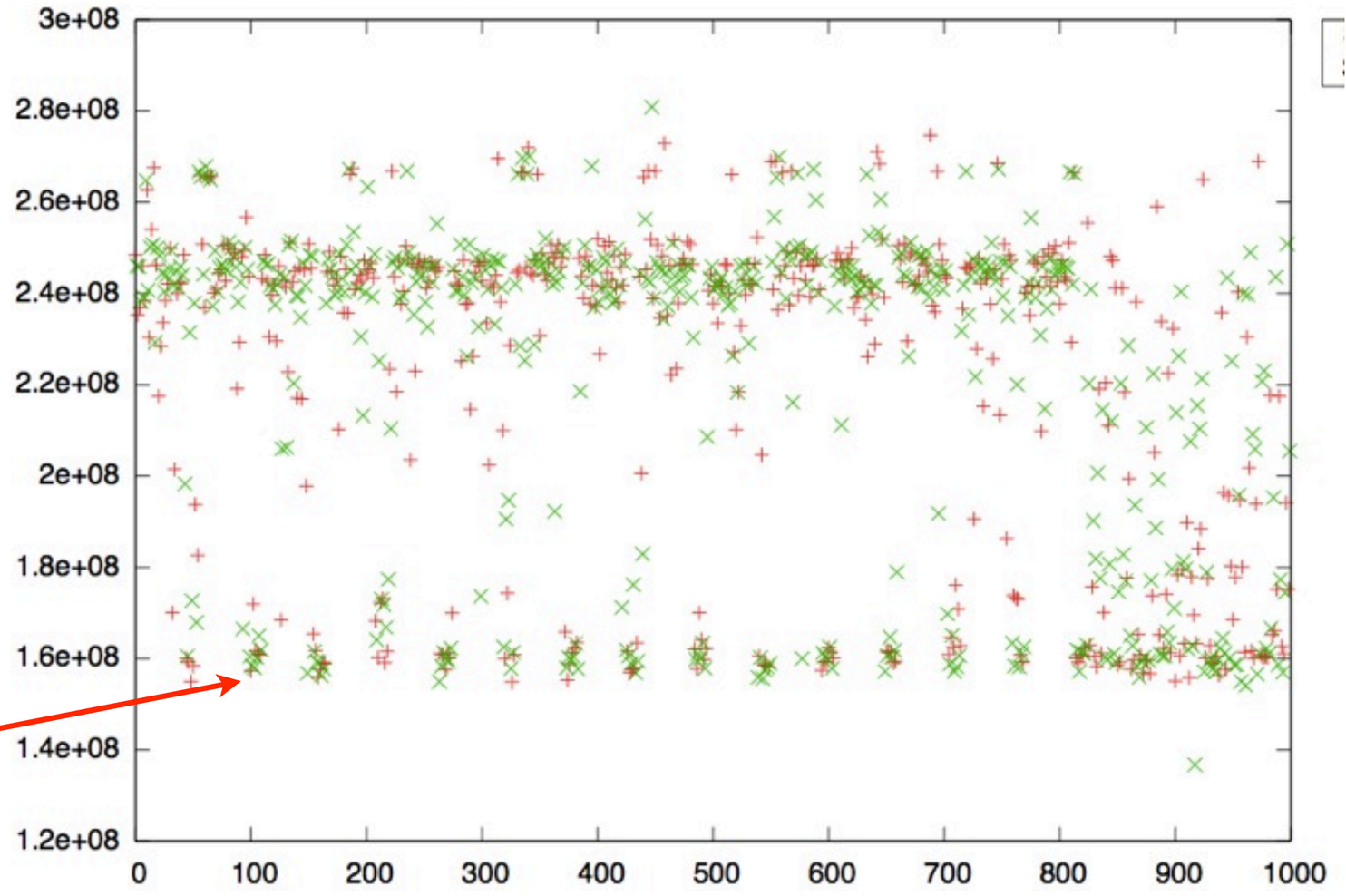


Time: 16:30 in the speakers' room:

<u>28c3 in action...</u>

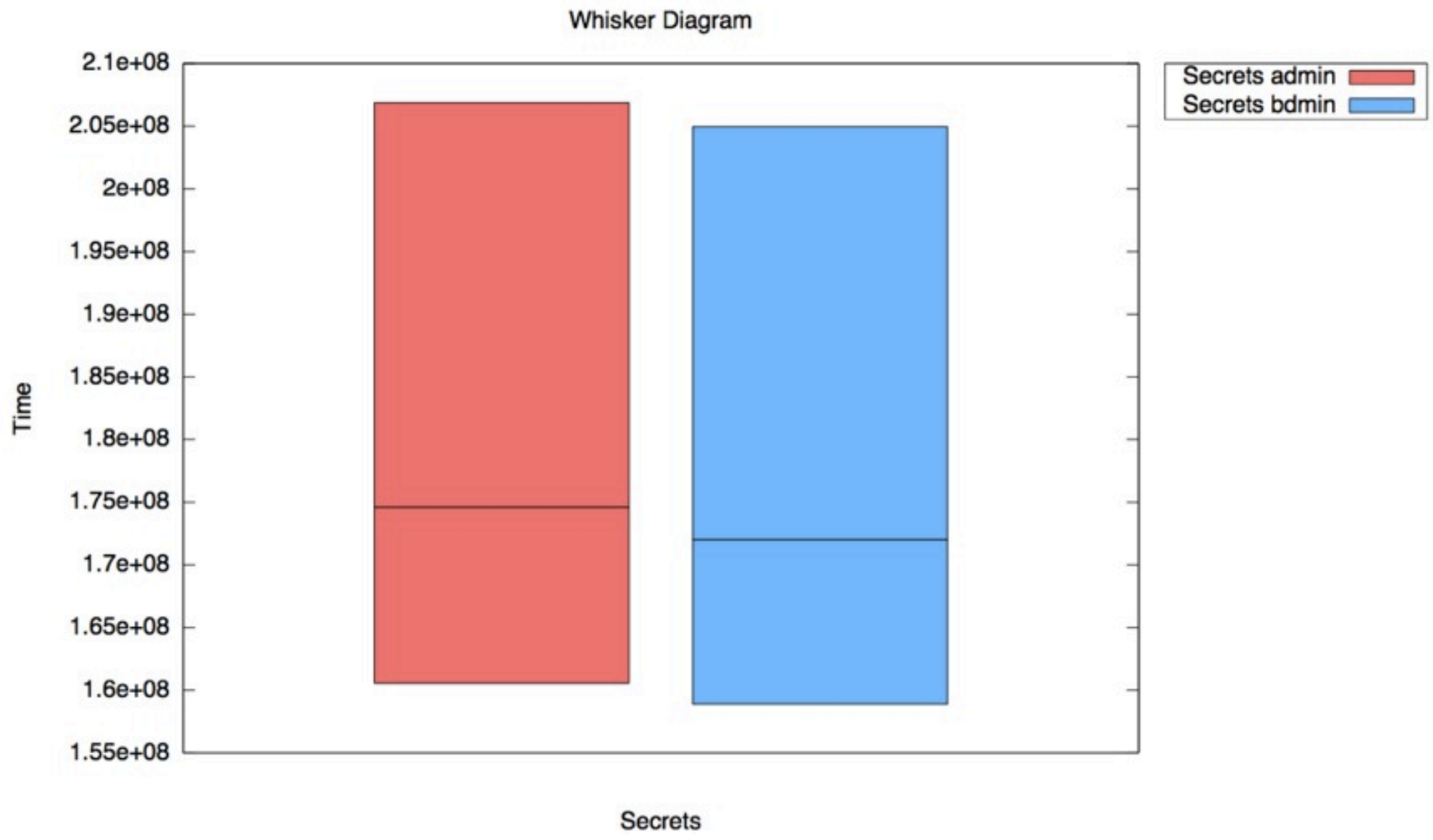


Not so sure about that...



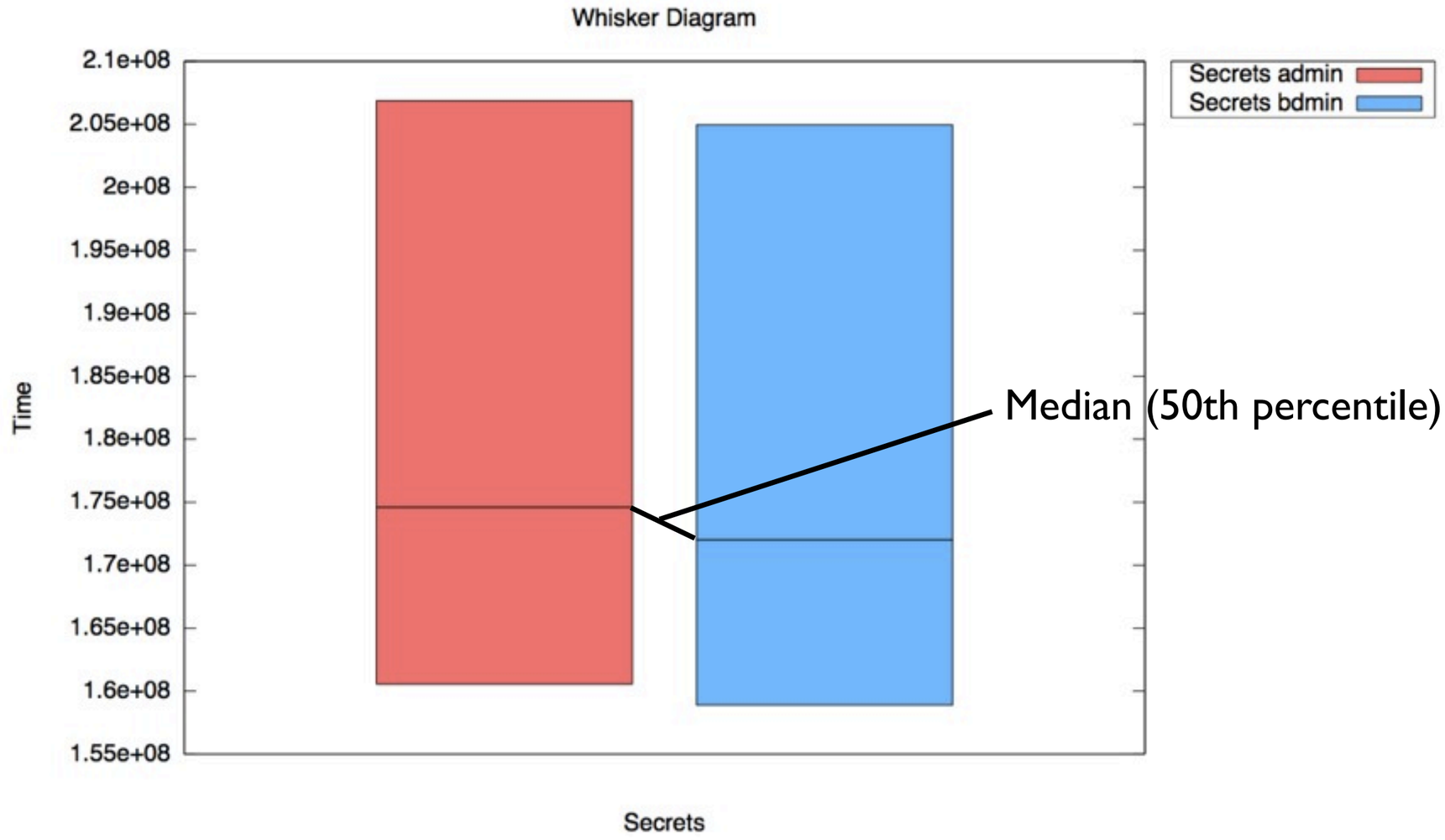
Analysing Timing Measurements

Analyse measurements graphically - Box Plot (also Whisker Plot)



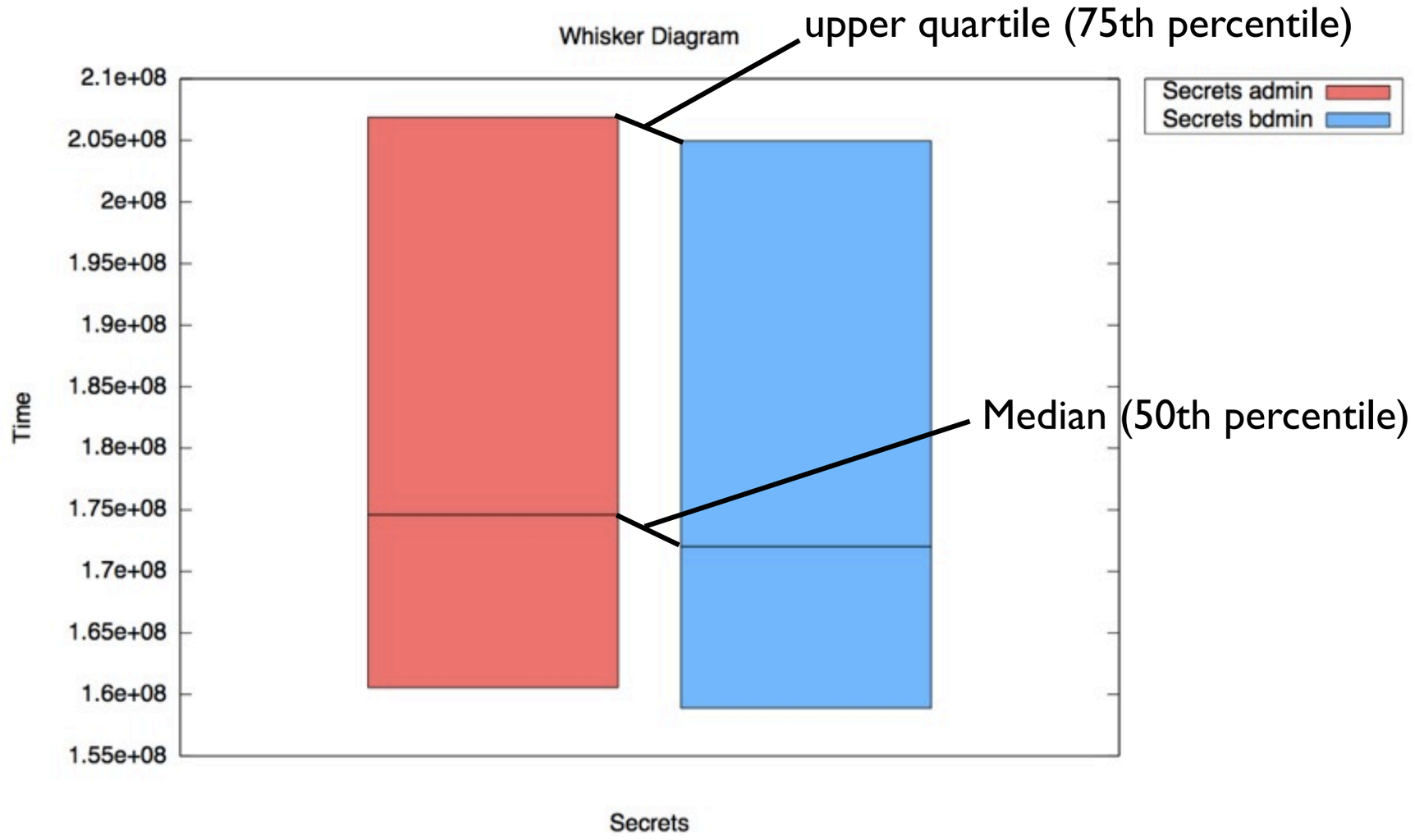
Analysing Timing Measurements

Analyse measurements graphically - Box Plot (also Whisker Plot)



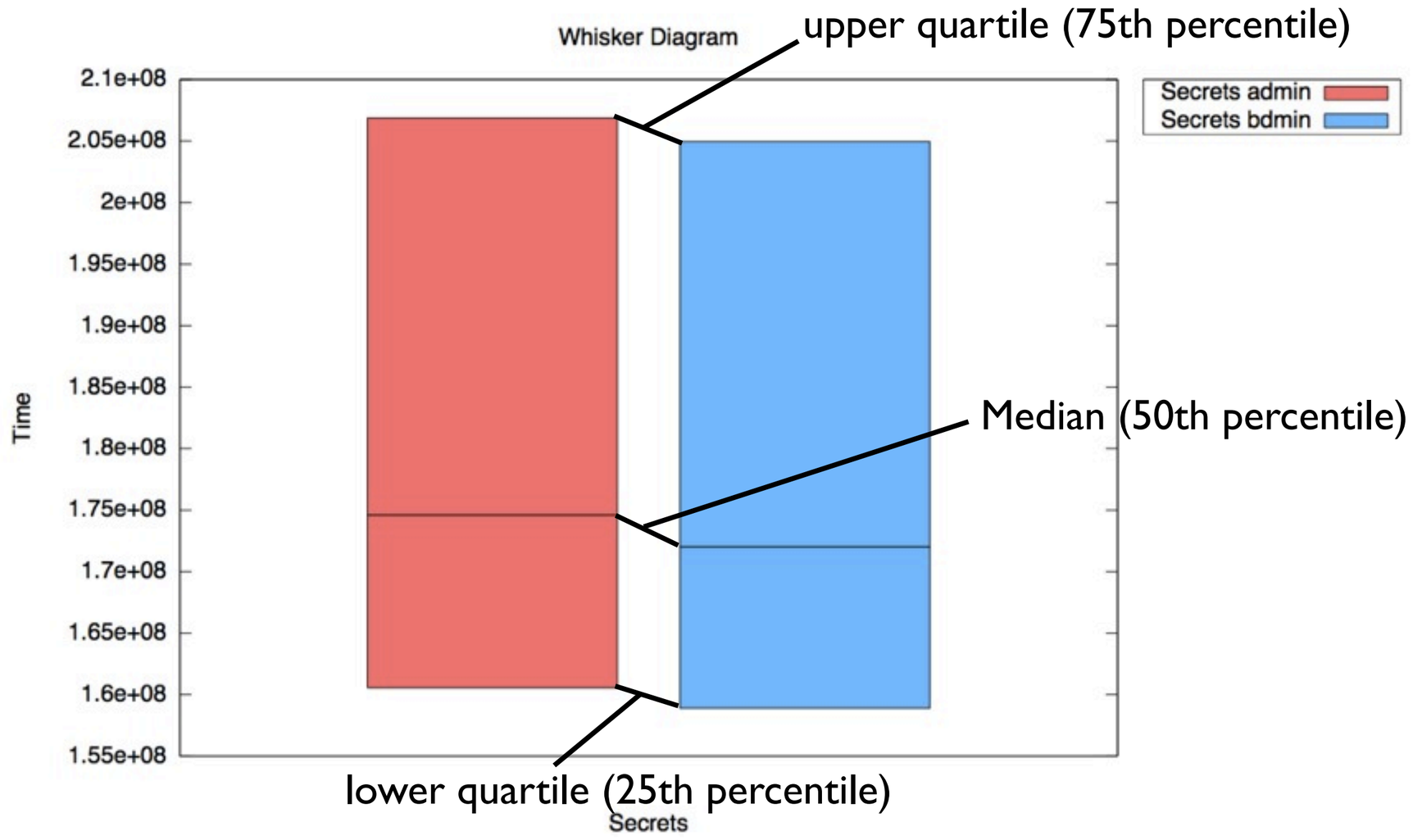
Analysing Timing Measurements

Analyse measurements graphically - Box Plot (also Whisker Plot)



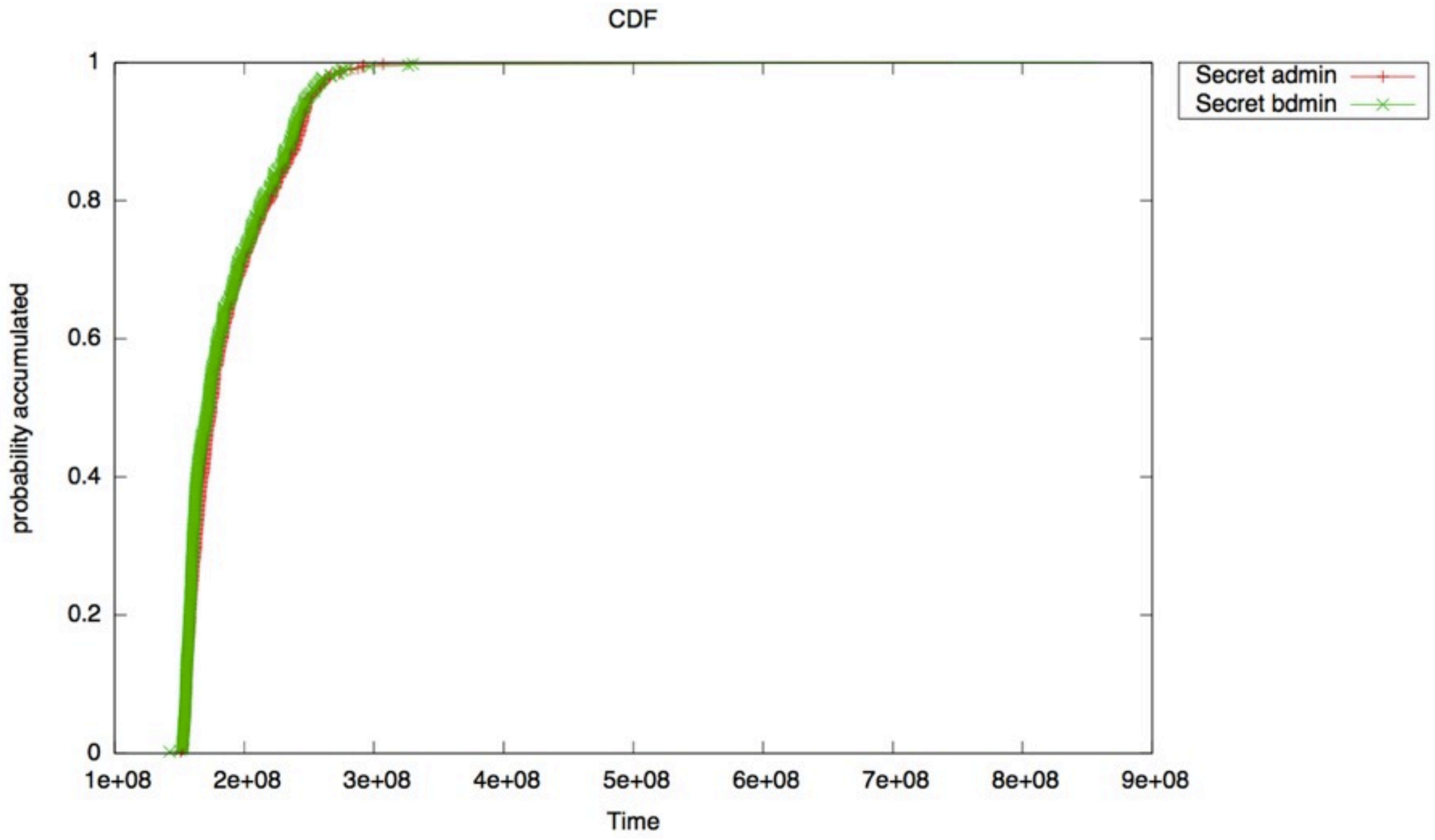
Analysing Timing Measurements

Analyse measurements graphically - Box Plot (also Whisker Plot)



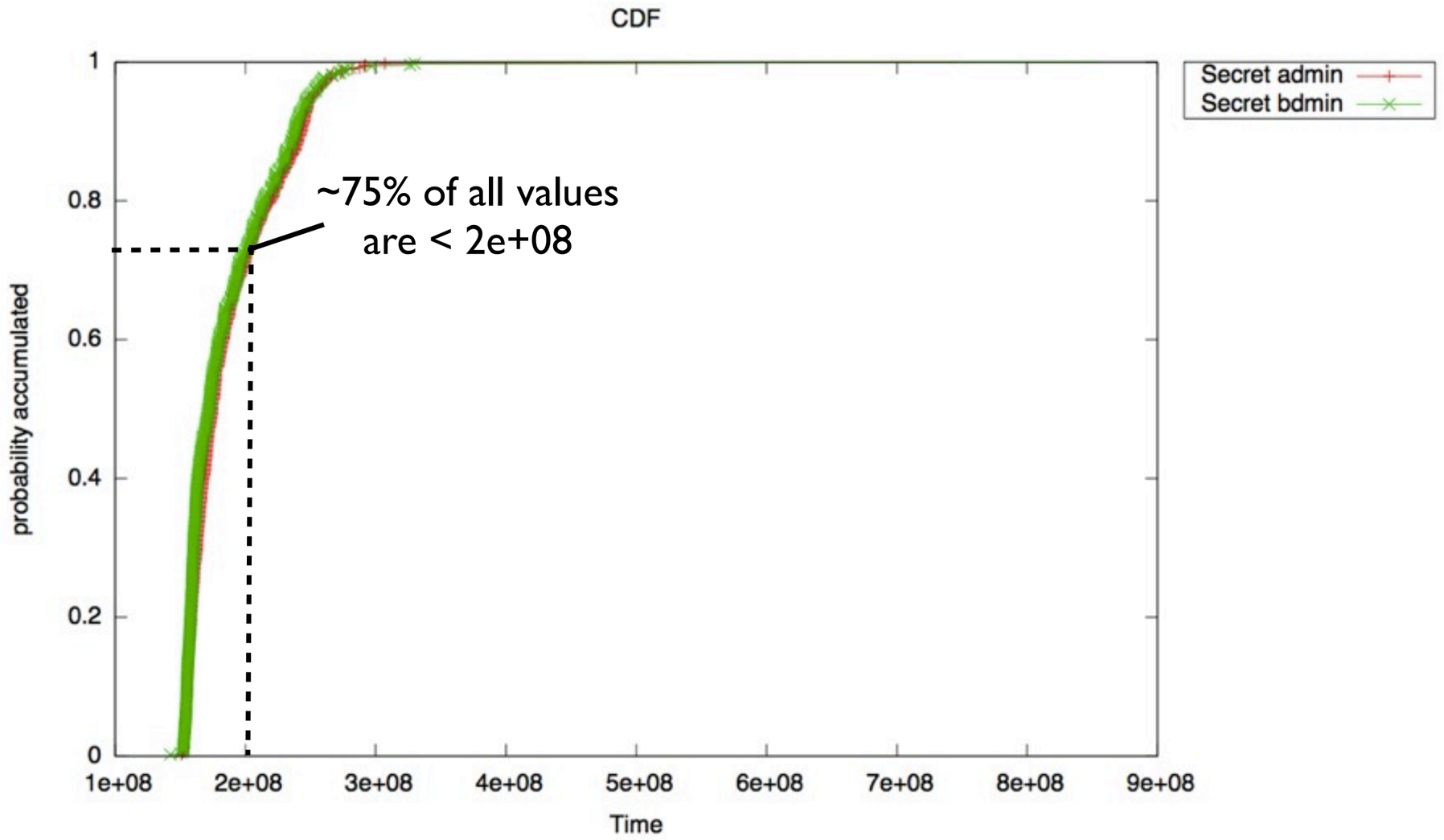
Analysing Timing Measurements

Analyse measurements graphically - Cumulative Distribution Function (CDF)



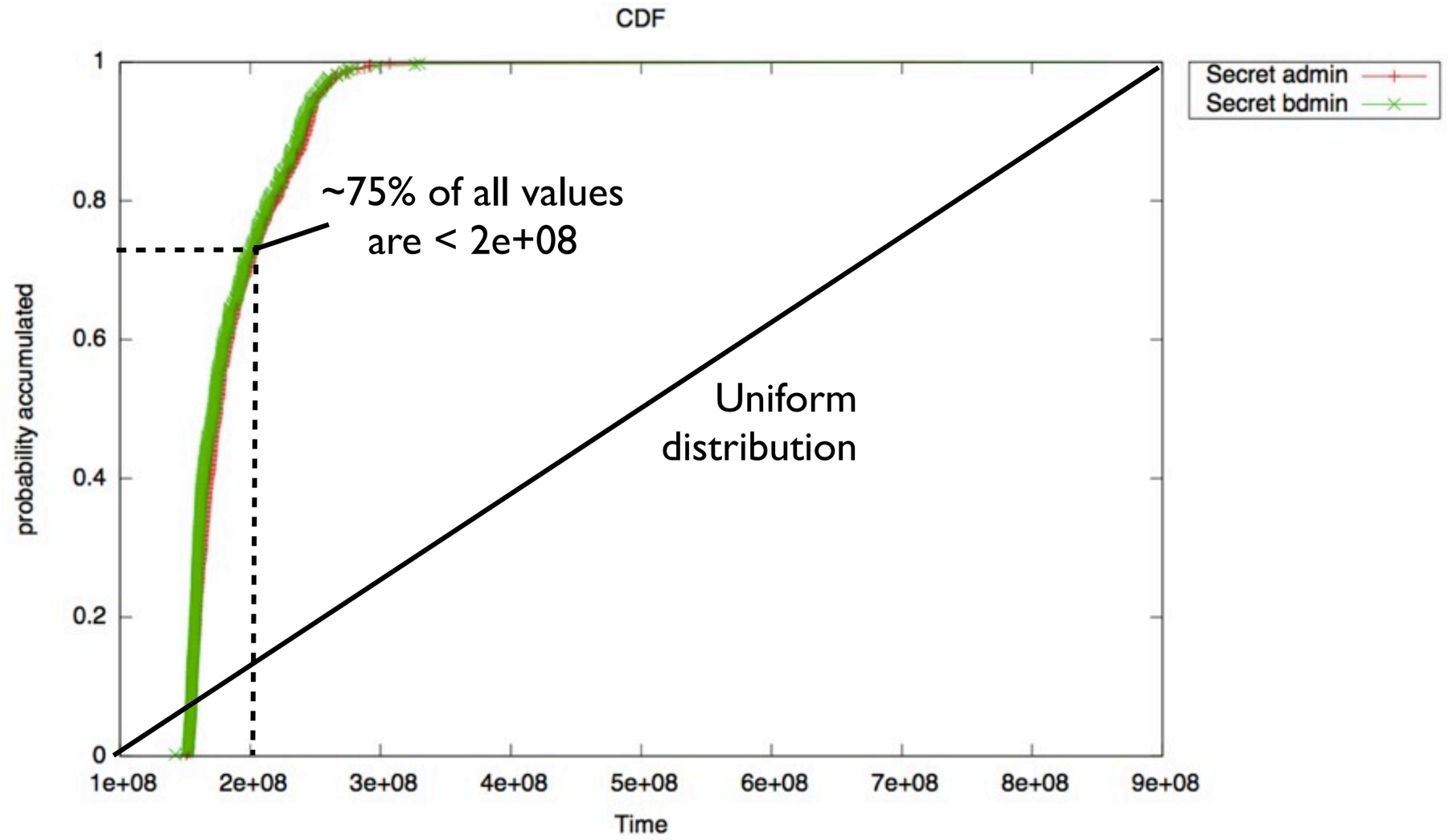
Analysing Timing Measurements

Analyse measurements graphically - Cumulative Distribution Function (CDF)



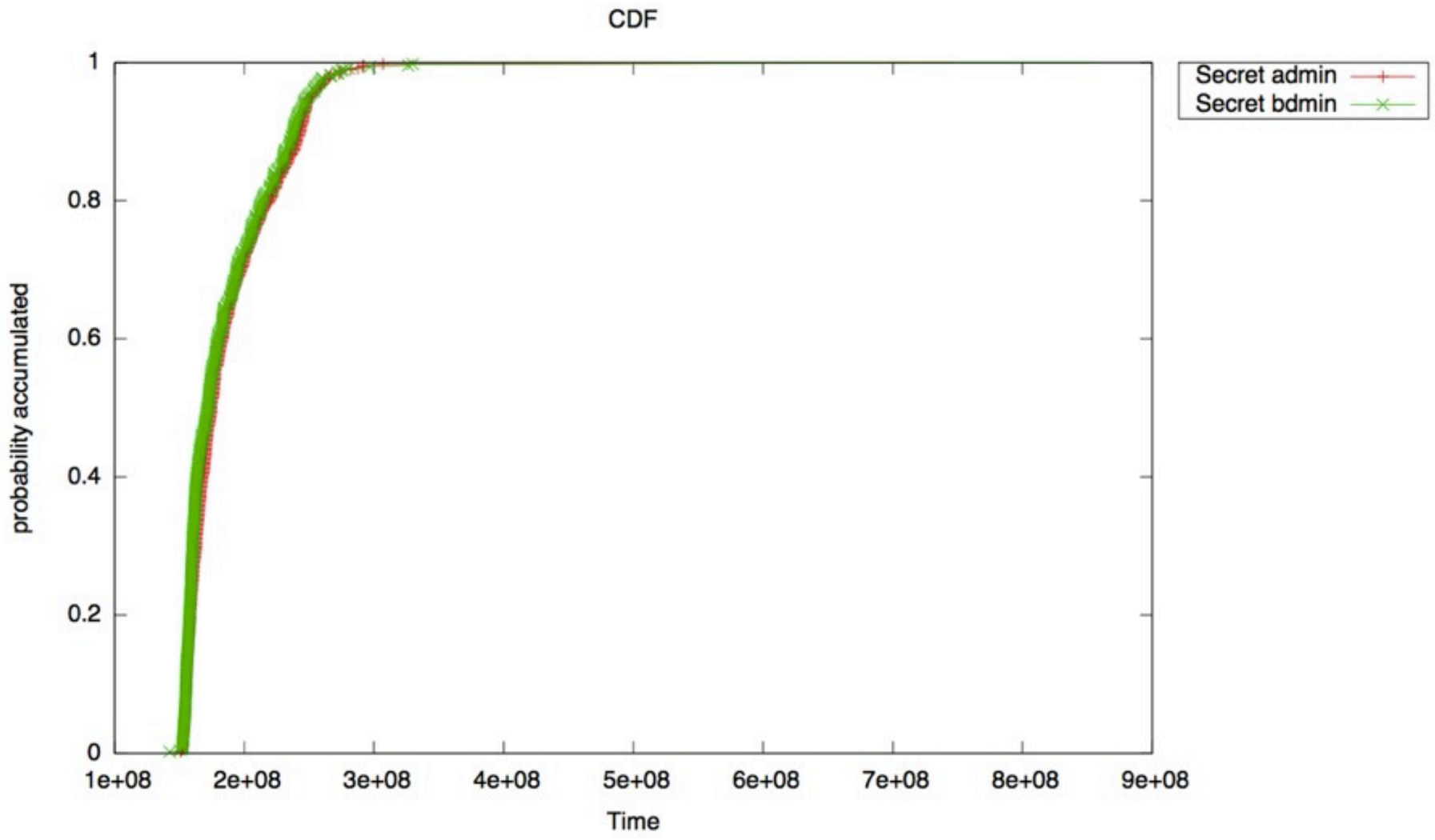
Analysing Timing Measurements

Analyse measurements graphically - Cumulative Distribution Function (CDF)



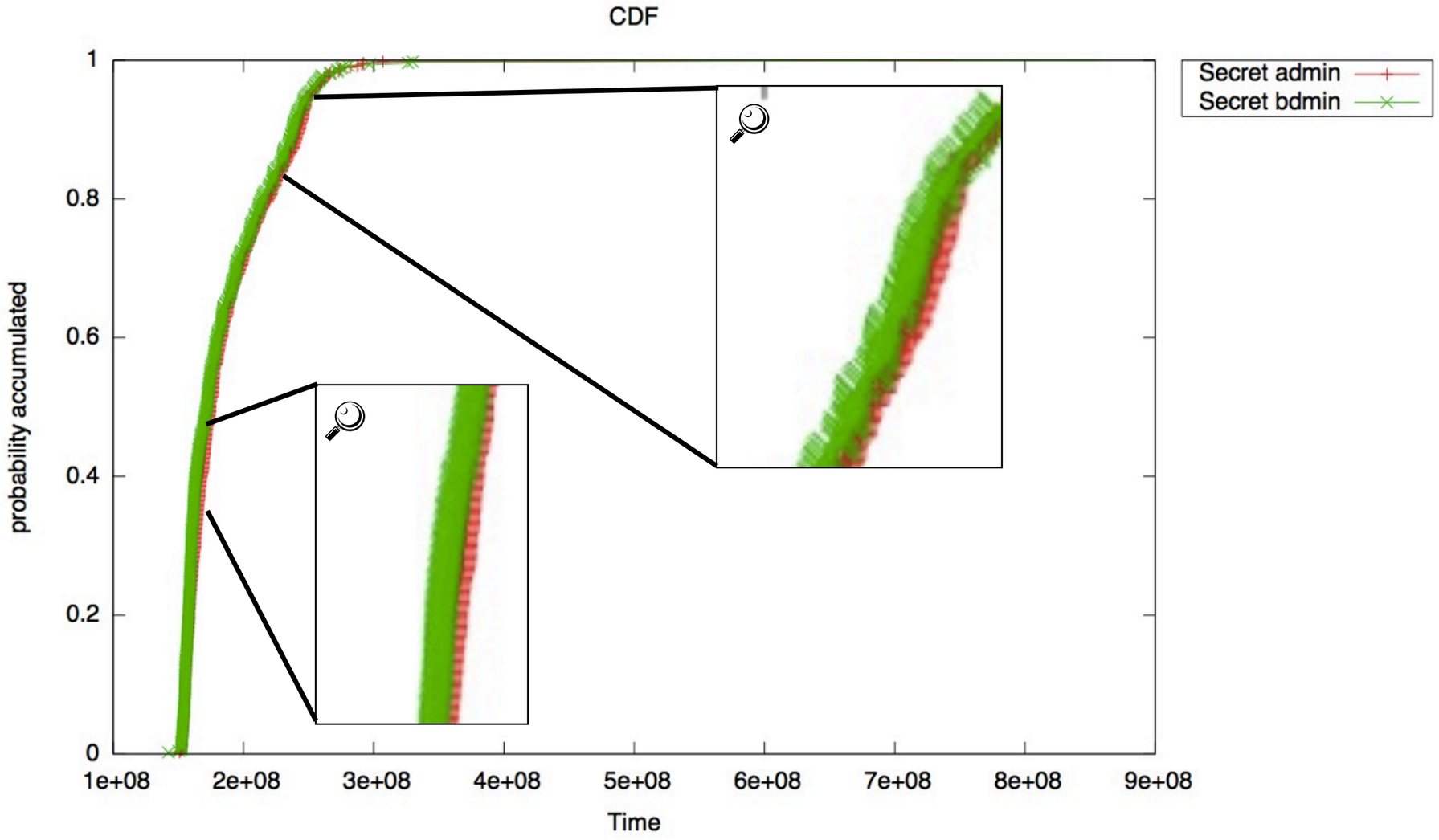
Analysing Timing Measurements

Analyse measurements graphically - Cumulative Distribution Function (CDF)



Analysing Timing Measurements

Analyse measurements graphically - Cumulative Distribution Function (CDF)



Analyse measurements with algorithms

- Standard algorithms
  - Student's t-test (requires normally distributed data → not applicable)
  - Wilcoxon-Test (applicable, but performs poor)
- Crosby's "Box Test" [1] seems to work best
  - Filter: only use measurements between 5th and 10th percentile
- Stay tuned for demo...

Examples for attacks

Examples for attacks
User Name Guessing at Typo3 Backend

Guessing administrative user names at Typo3 backend (similar to [3])

- Attacker chooses non-existing user name, e.g. ‘31337’
  - the user name in question: ‘admin’

- Hypothesis: login requests with ‘admin’ take measurably longer than those with ‘31137’

Examples for attacks

User Name Guessing at Typo3 Backend

Demo

Lots of help from Isabell Schmitt and Niels Iciek

Planned release in January, will be announced on Twitter: @seecurity

Examples for attacks
Amount of Private Pictures in Gallery

Guessing amount of hidden pictures in Gallery [3]

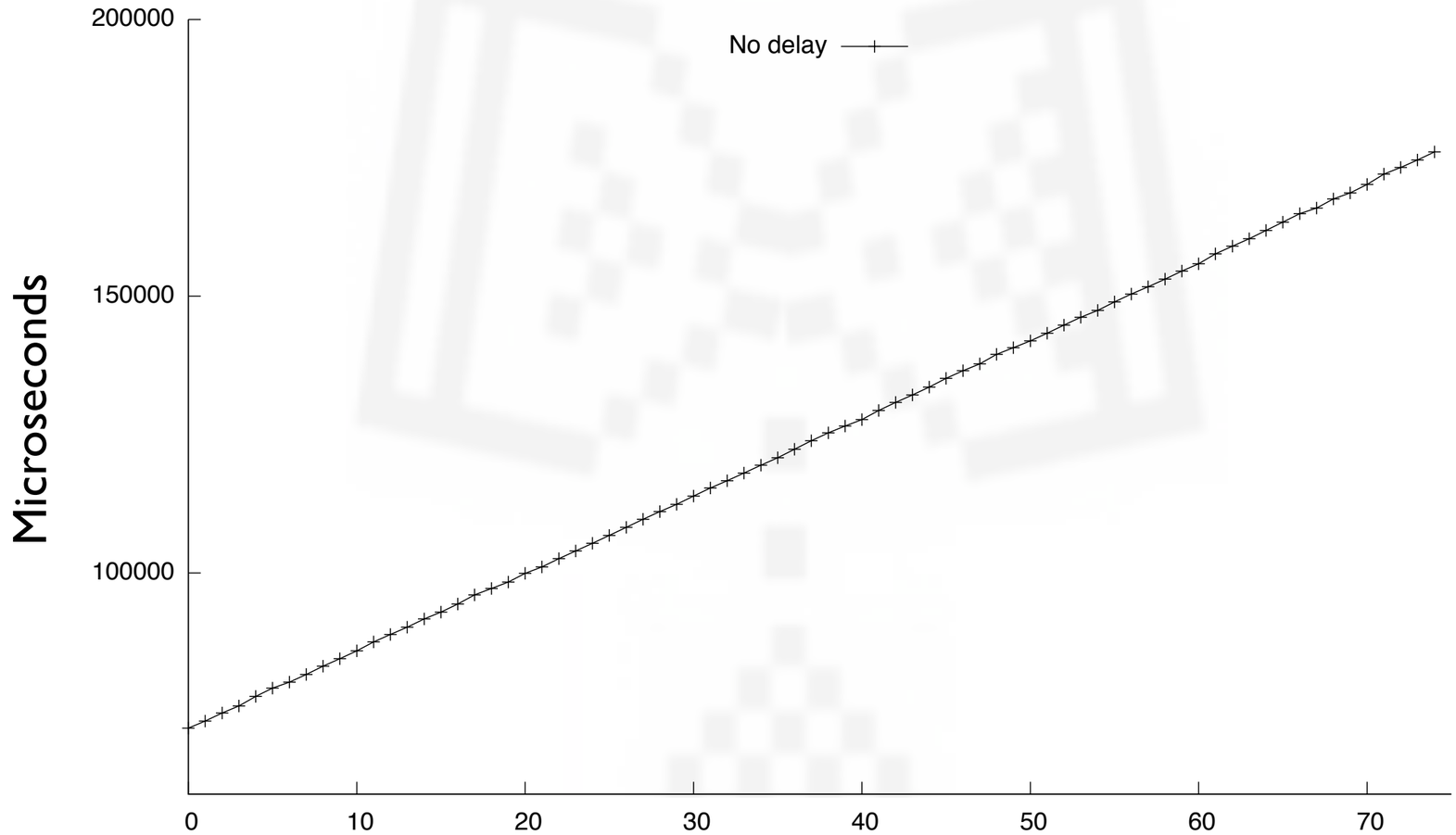
- Attacker wants to view private pictures in Gallery
- Question: which album contains many private pictures?

- Hypothesis: response time of displaying album depends on the absolute amount of pictures
  - response_time ~= #public pictures + #private pictures

Examples for attacks

Amount of Private Pictures in Gallery

All albums show only a single picture to anonymous users



FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Breaking XML Encryption

- Joint work with Juraj Somorovsky and Tibor Jager from Ruhr-Uni-Bochum

What is XML Encrpytion?

- ⇒ encrypt subtrees of XML doc
- session key is RSA-encrypted (hybrid encryption)
- subtree AES-encrypted with session key

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId">
    <EncryptionMethod Algorithm="...xmlenc#rsa-1_5"/>
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==</CipherValue>
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncDataId-2"/>
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId-2">
   <EncryptionMethod Algorithm="...xmlenc#aes128-cbc"/>
   <CipherData>
    <CipherValue>3bP...Zx0=</CipherValue>
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

c_{key}

c_{data}

Breaking XML Encryption

- Joint work with Juraj Somorovsky and Tibor Jager from Ruhr-Uni-Bochum

What is XML Encrpytion?

- ⇒ encrypt subtrees of XML doc
- session key is RSA-encrypted (hybrid encryption)
- subtree AES-encrypted with session key

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId">
    <EncryptionMethod Algorithm="...xmlenc#rsa-1_5"/>
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==</CipherValue>
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncDataId-2"/>
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId-2">
   <EncryptionMethod Algorithm="...xmlenc#aes128-cbc"/>
   <CipherData>
    <CipherValue>3bP...Zx0=</CipherValue>
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

c_{key}

c_{data}

Examples for attacks

Breaking XML Encryption - What is XML Encryption?

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId">
    <EncryptionMethod Algorithm="...xmlenc#rsa-1_5"/>
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==</CipherValue>
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncDataId-2"/>
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId-2">
   <EncryptionMethod Algorithm="...xmlenc#aes128-cbc"/>
   <CipherData>
    <CipherValue>3bP...Zx0=</CipherValue>
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

c_{key}

c_{data}

Examples for attacks

Breaking XML Encryption - What is XML Encryption?

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId"
    <EncryptionMethod Algorith
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncD
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId
   <EncryptionMethod Algorithm
   <CipherData>
    <CipherValue>3bP...Zx0=</C
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

Examples for attacks

Breaking XML Encryption - What is XML Encryption?

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId"
    <EncryptionMethod Algorith
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncD
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId
   <EncryptionMethod Algorithm
   <CipherData>
    <CipherValue>3bP...Zx0=</C
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

Decrypting XML Encryption messages

1. Decrypt session key $m = dec_{rsa}(c_{key})$
2. Return error if m does not comply with PKCS#1, else:
3. Decrypt c_{data} (results in XML subtree)
4. Copy subtree in XML doc
5. Parse XML doc
6. Return error if XML doc is invalid

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId"
    <EncryptionMethod Algorith
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncD
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId
   <EncryptionMethod Algorithm
   <CipherData>
    <CipherValue>3bP...Zx0=</C
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

Decrypting XML Encryption messages

1. Decrypt session key $m = dec_{rsa}(c_{key})$
2. Return error if m does not comply with PKCS#1, else:
3. Decrypt c_{data} (results in XML subtree)
4. Copy subtree in XML doc
5. Parse XML doc
6. Return error if XML doc is invalid

timing
differerence?

→ Determine PKCS#1 compliance through response time

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId"
    <EncryptionMethod Algorith
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncD
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId
   <EncryptionMethod Algorithm
   <CipherData>
    <CipherValue>3bP...Zx0=</C
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

“Bleichenbacher attack” [9]

- breaks RSA within ~1 million requests

requires Oracle *O*:

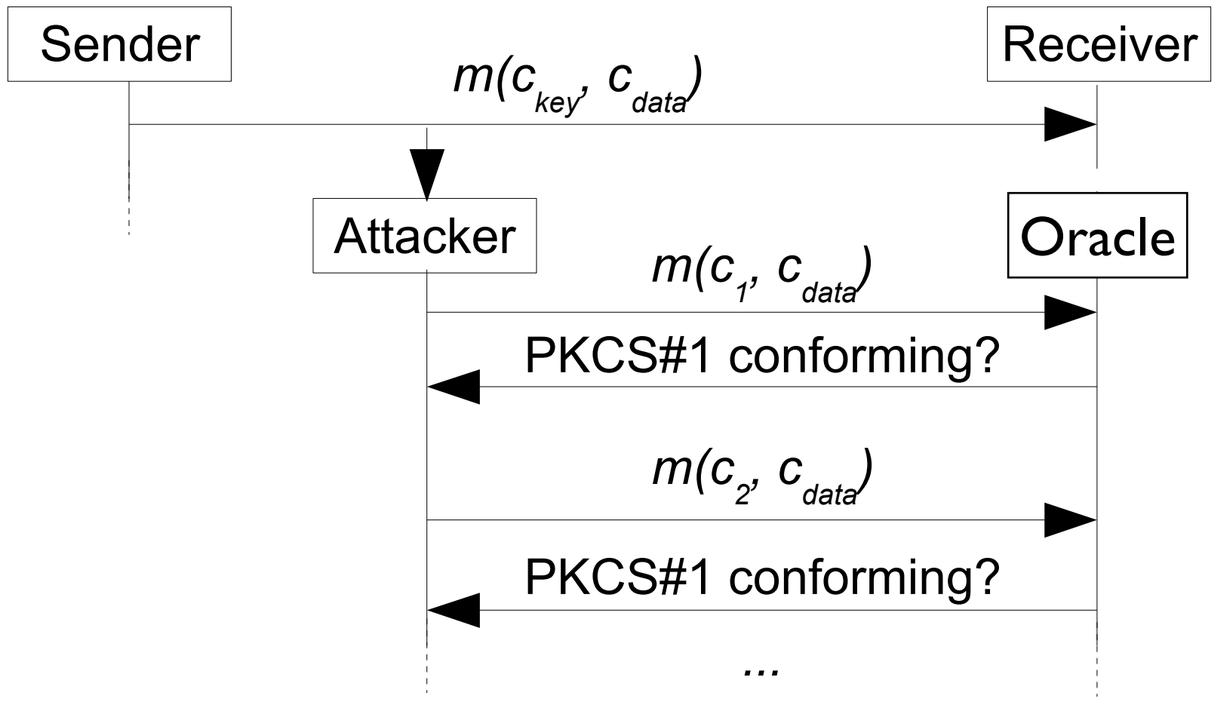
- *O* tells if a chosen ciphertext decrypts to PKCS#1-compliant encoding

"Bleichenbacher attack" [9]

- breaks RSA within ~1 million requests

requires Oracle *O*:

- *O* tells if a chosen ciphertext decrypts to PKCS#1-compliant encoding

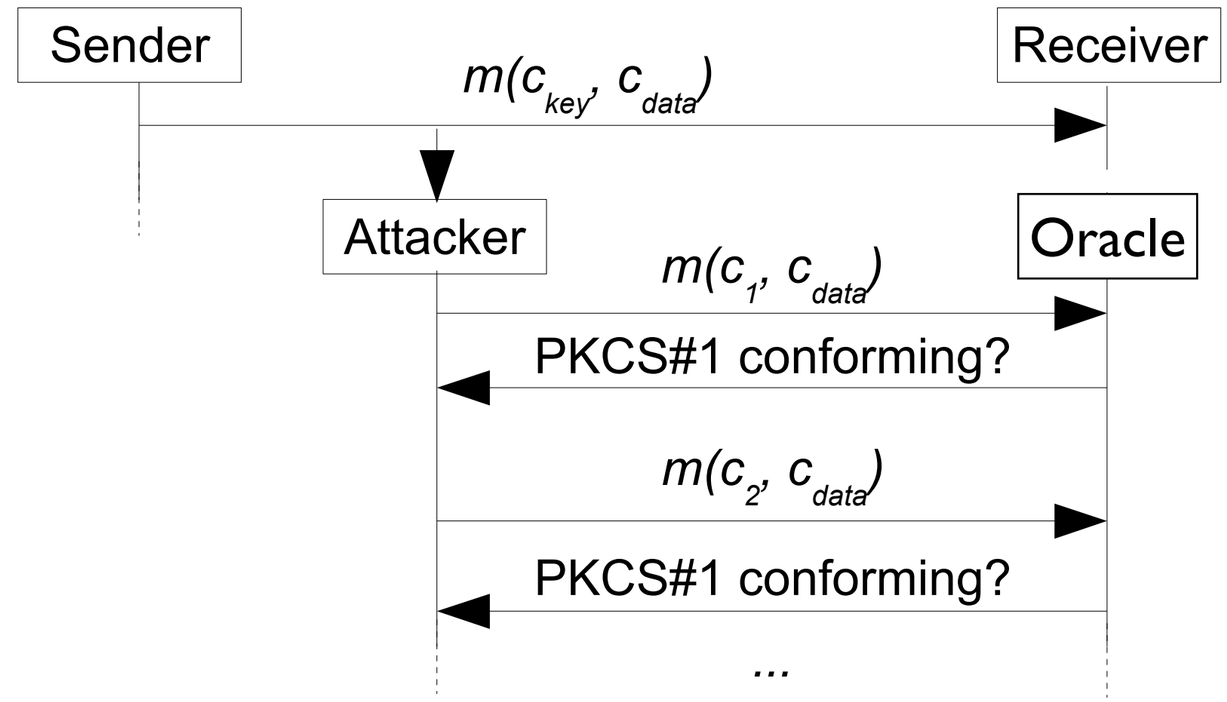


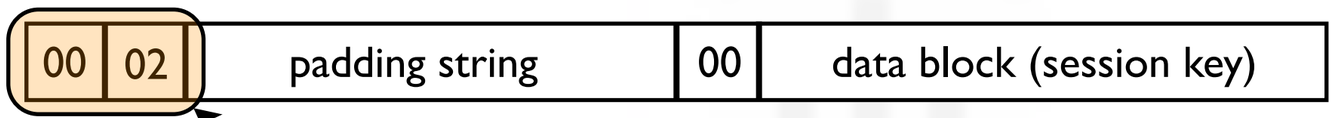
"Bleichenbacher attack" [9]

- breaks RSA within ~1 million requests

requires Oracle *O*:

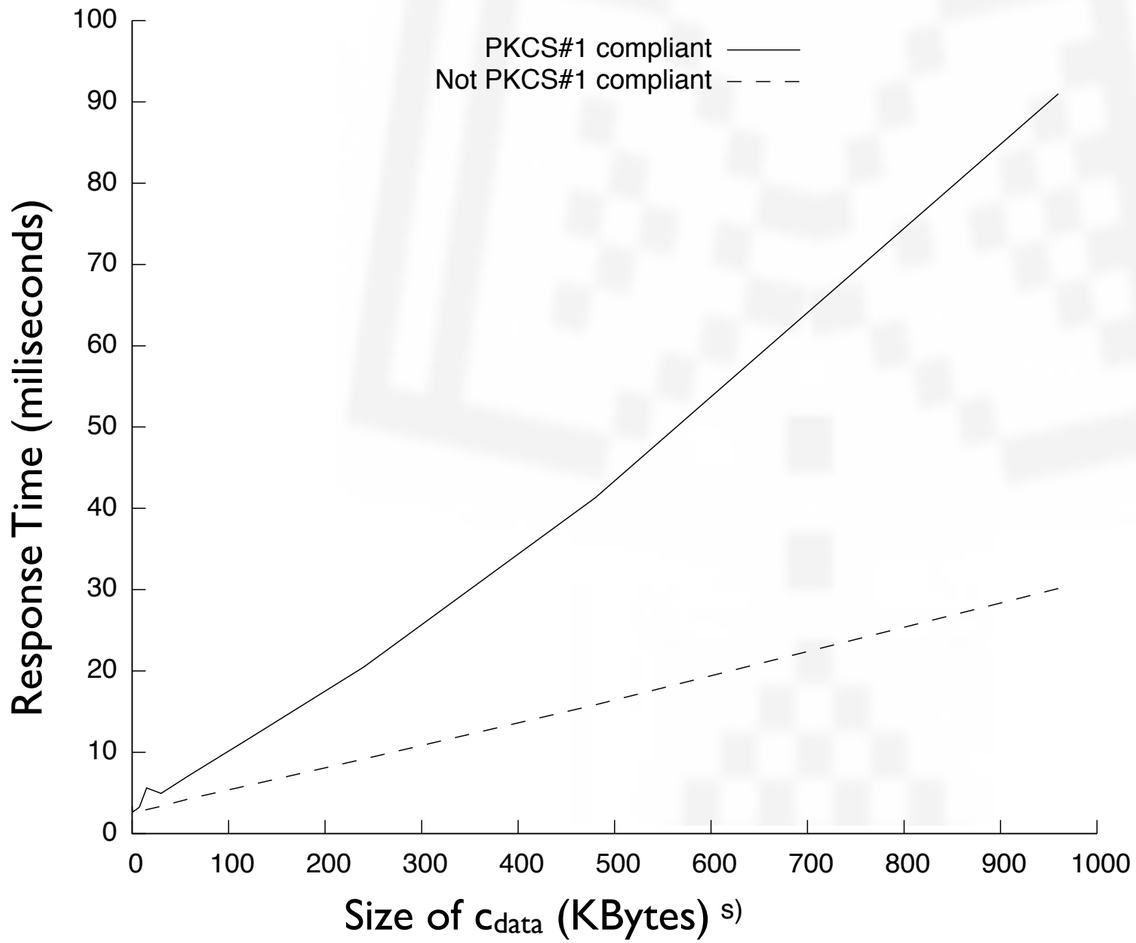
- *O* tells if a chosen ciphertext decrypts to PKCS#1-compliant encoding





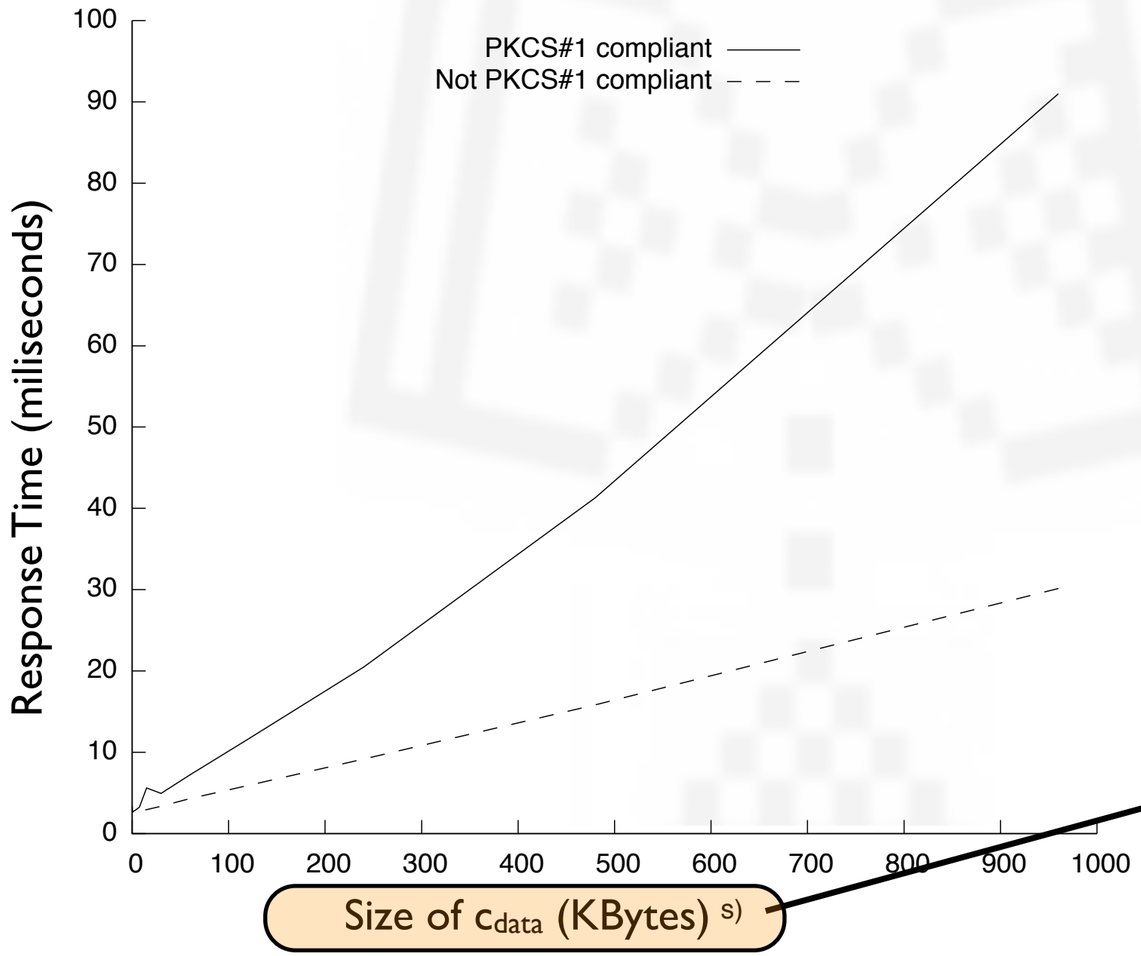
PKCS#1 conformance check

Bleichenbacher attack + XML Encryption?



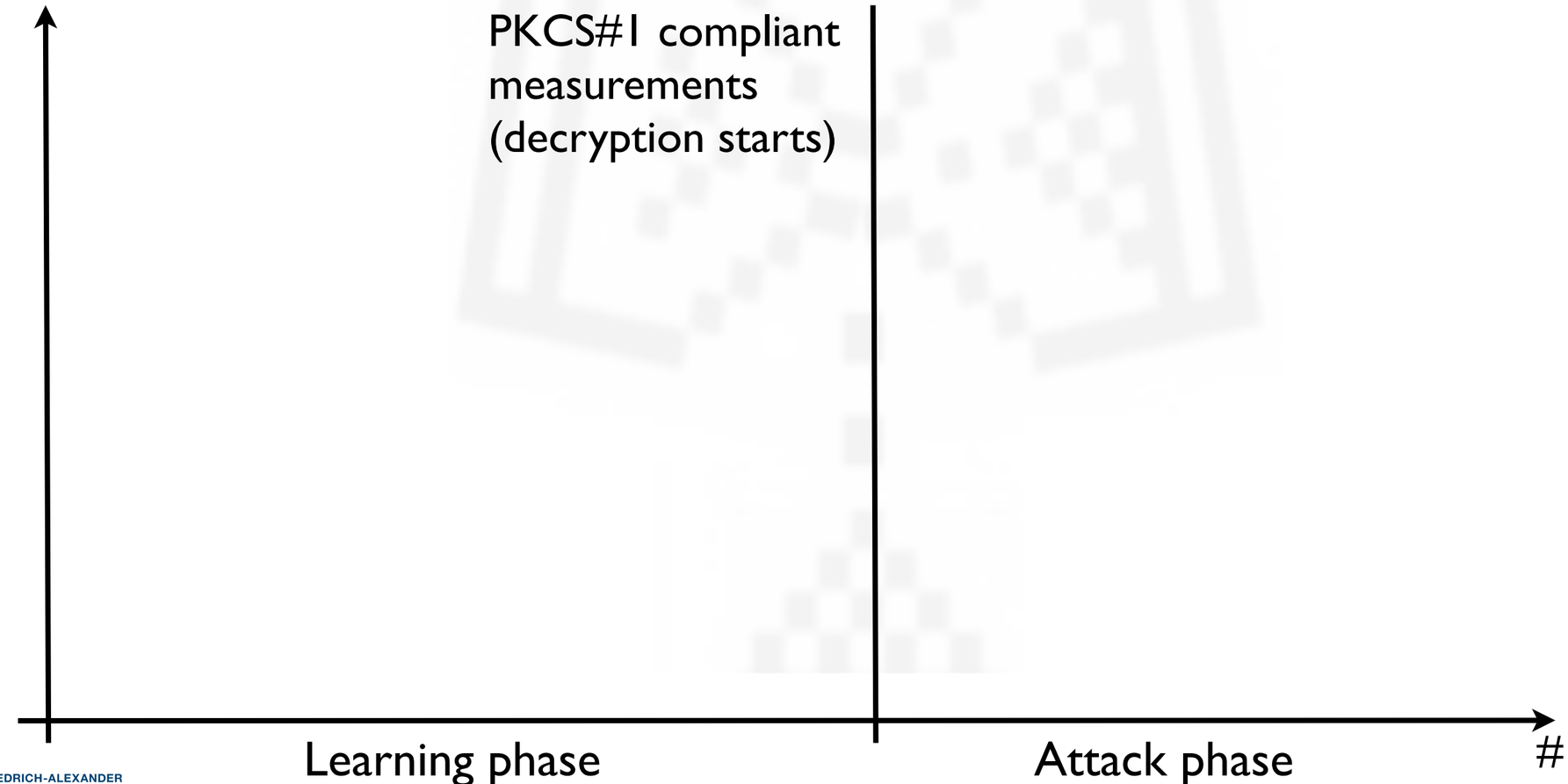

```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId"
    <EncryptionMethod Algorith
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncD
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId
   <EncryptionMethod Algorithm
   <CipherData>
    <CipherValue>3bP...Zx0=</C
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

Bleichenbacher attack + XML Encryption?

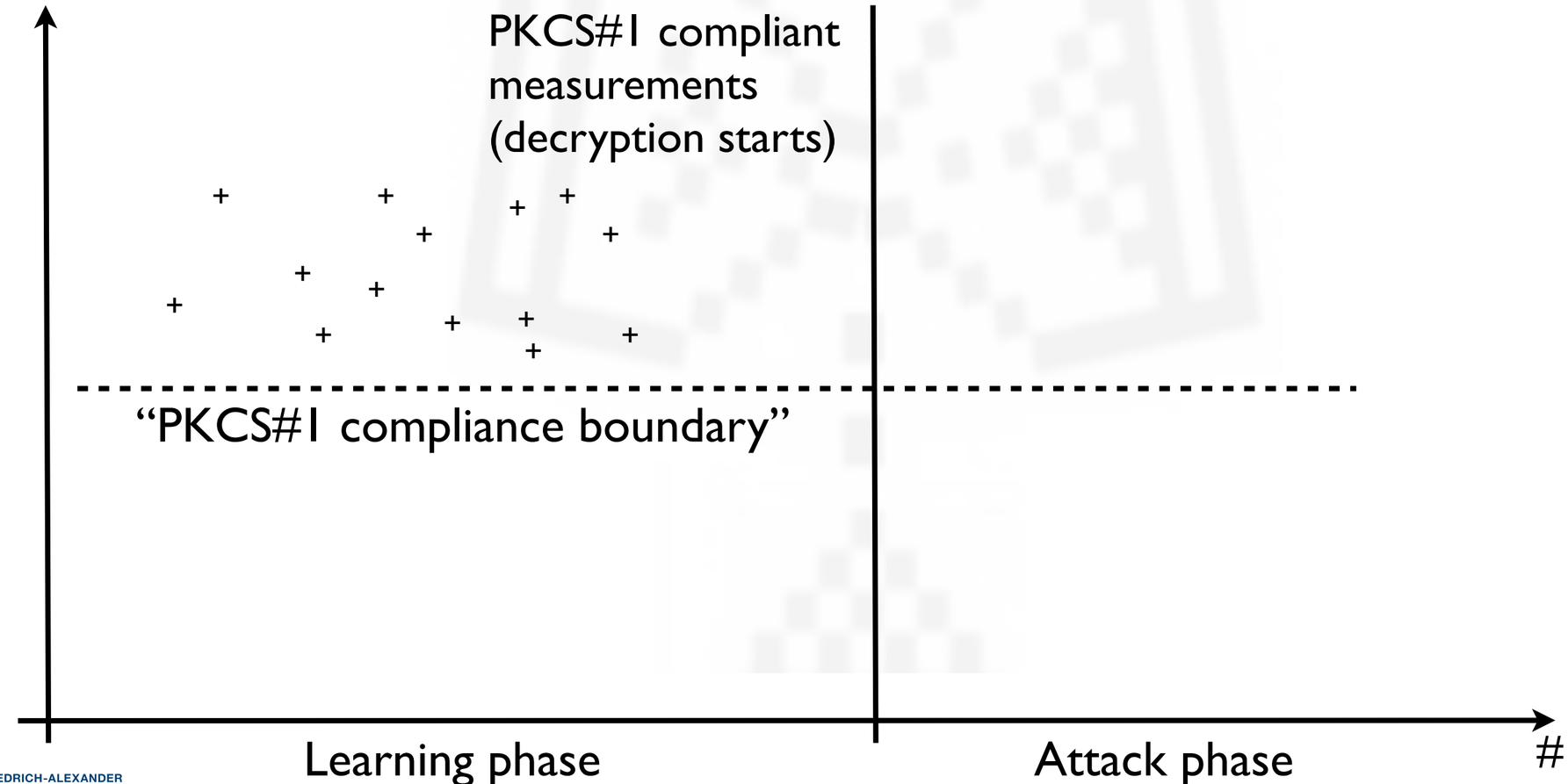



```
<Envelope>
 <Header>
  <Security>
   <EncryptedKey Id="EncKeyId"
    <EncryptionMethod Algorith
    <KeyInfo>...</KeyInfo>
    <CipherData>
     <CipherValue>Y2bh...fPw==
    </CipherData>
    <ReferenceList>
     <DataReference URI="#EncD
    </ReferenceList>
   </EncryptedKey>
  </Security>
 </Header>
 <Body>
  <EncryptedData Id="EncDataId
   <EncryptionMethod Algorithm
   <CipherData>
    <CipherValue>3bP...Zx0=</C
   </CipherData>
  </EncryptedData>
 </Body>
</Envelope>
```

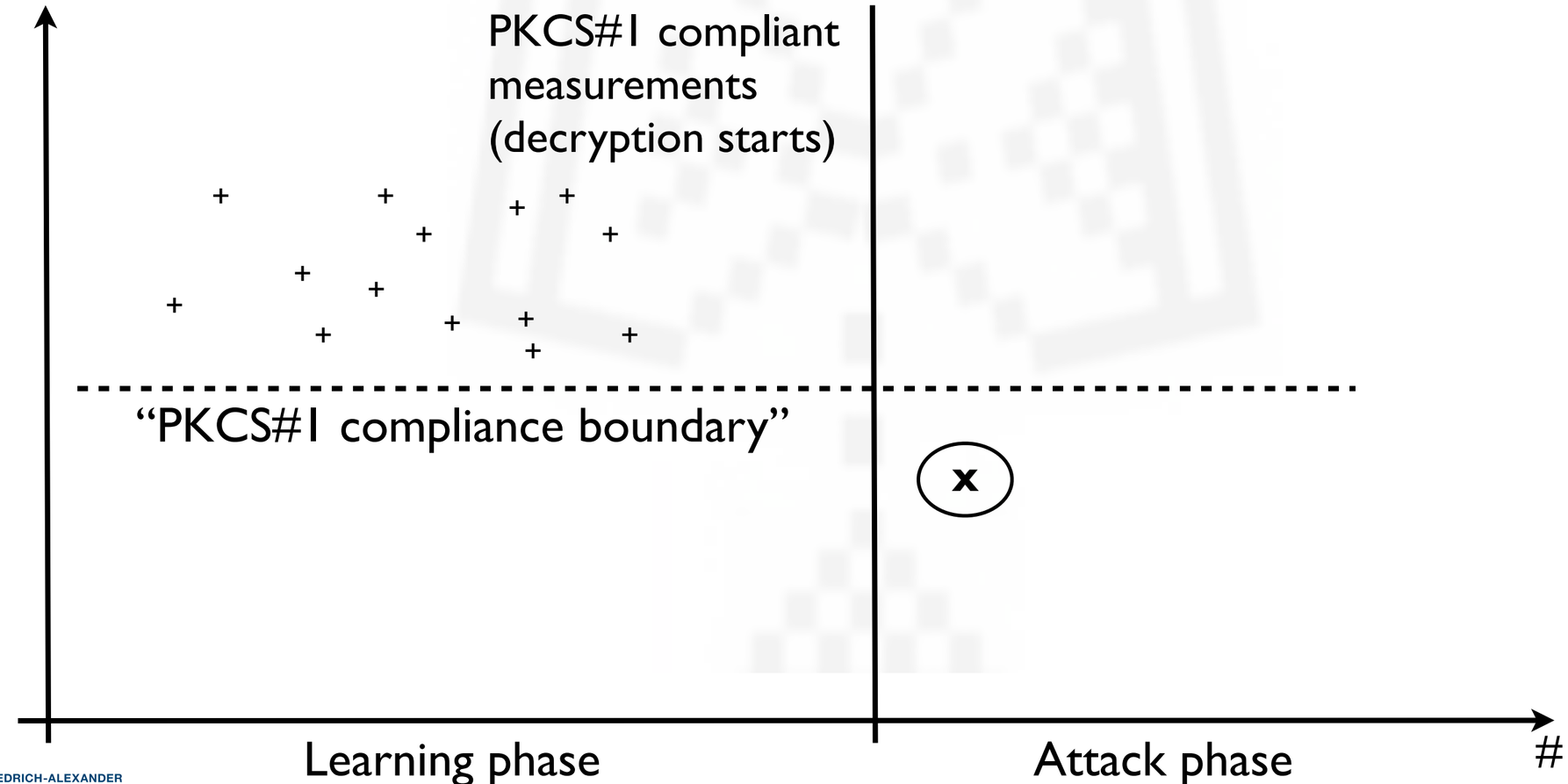
Possibilistic timing side channel attack



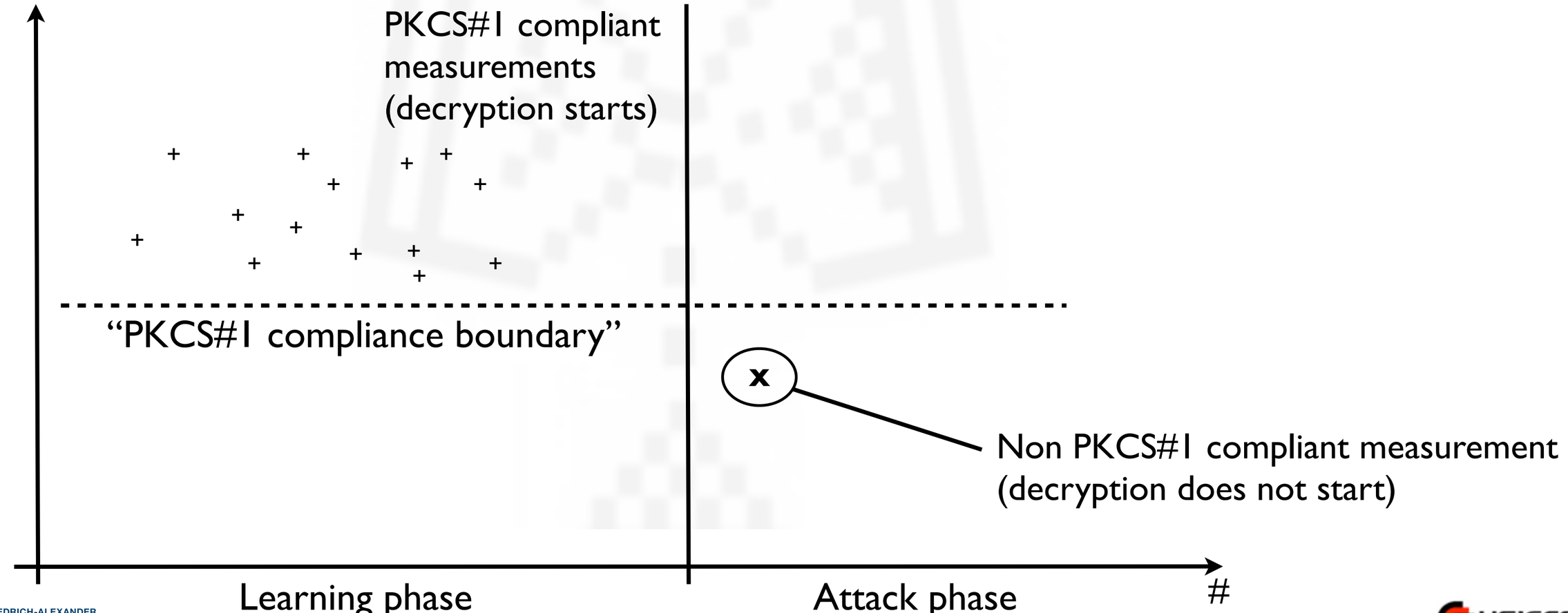
Possibilistic timing side channel attack



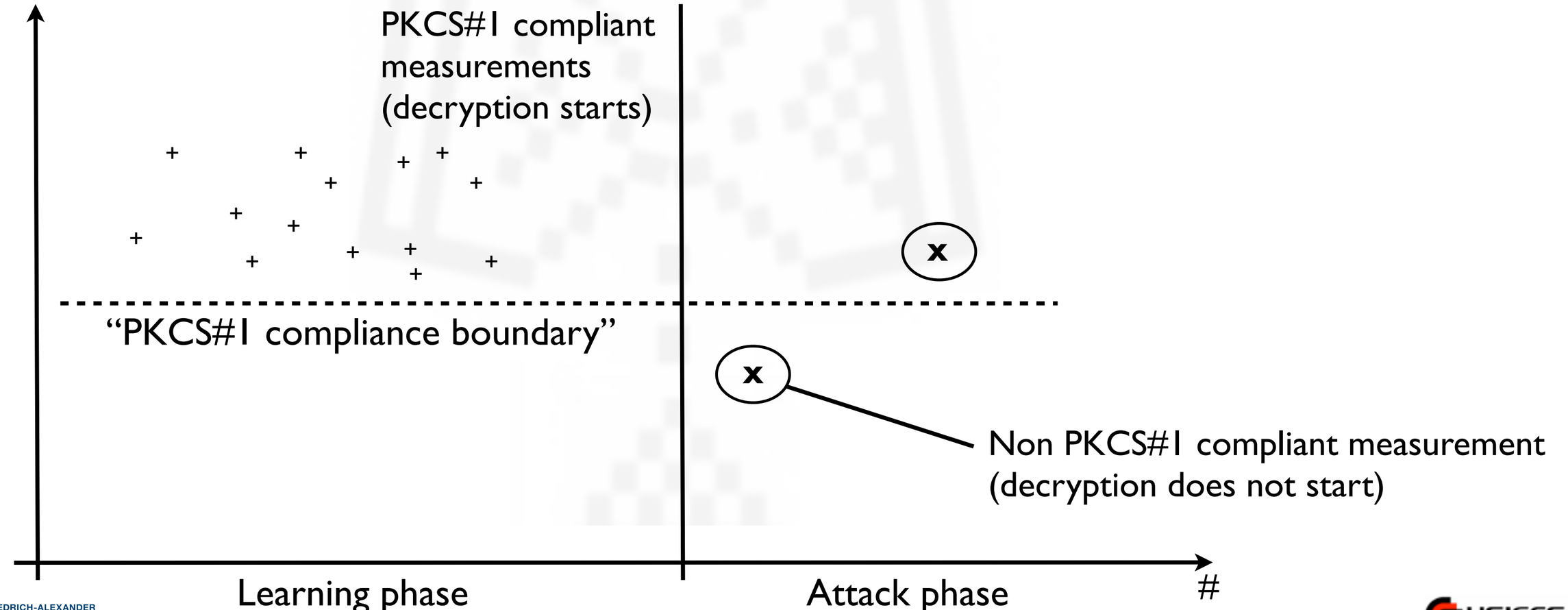
Possibilistic timing side channel attack



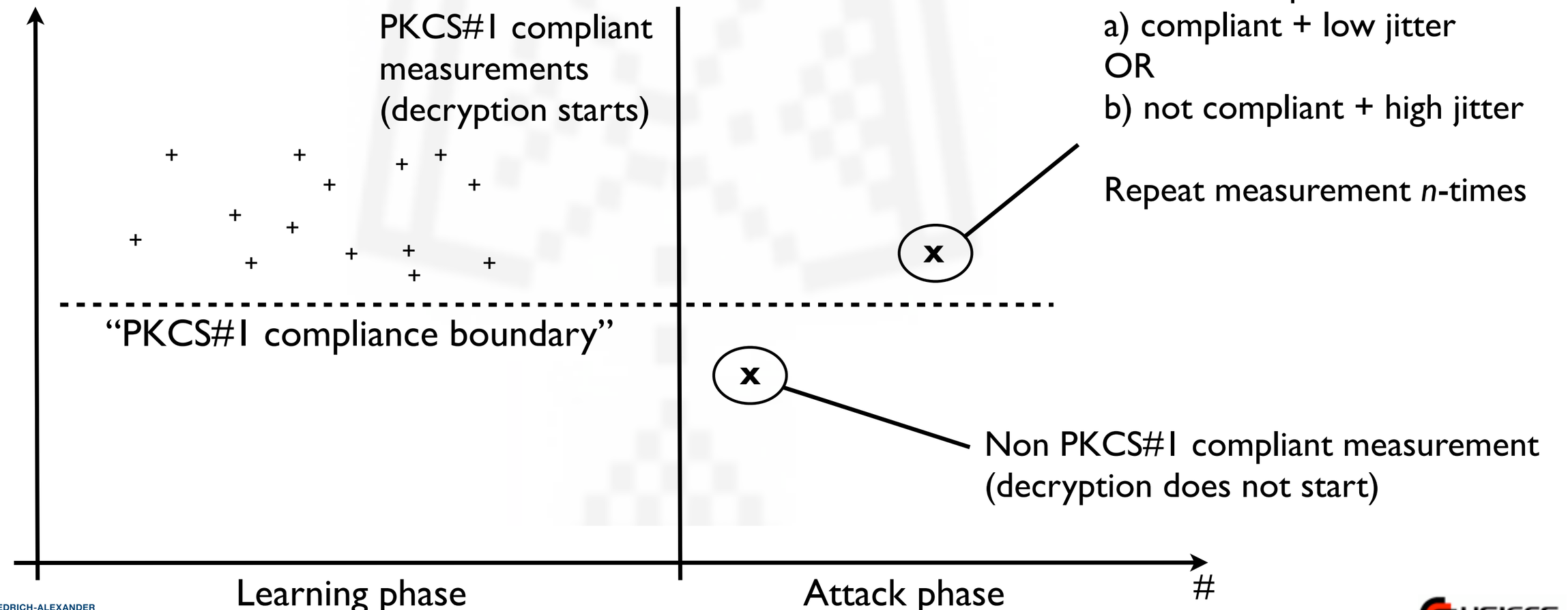
Possibilistic timing side channel attack



Possibilistic timing side channel attack



Possibilistic timing side channel attack

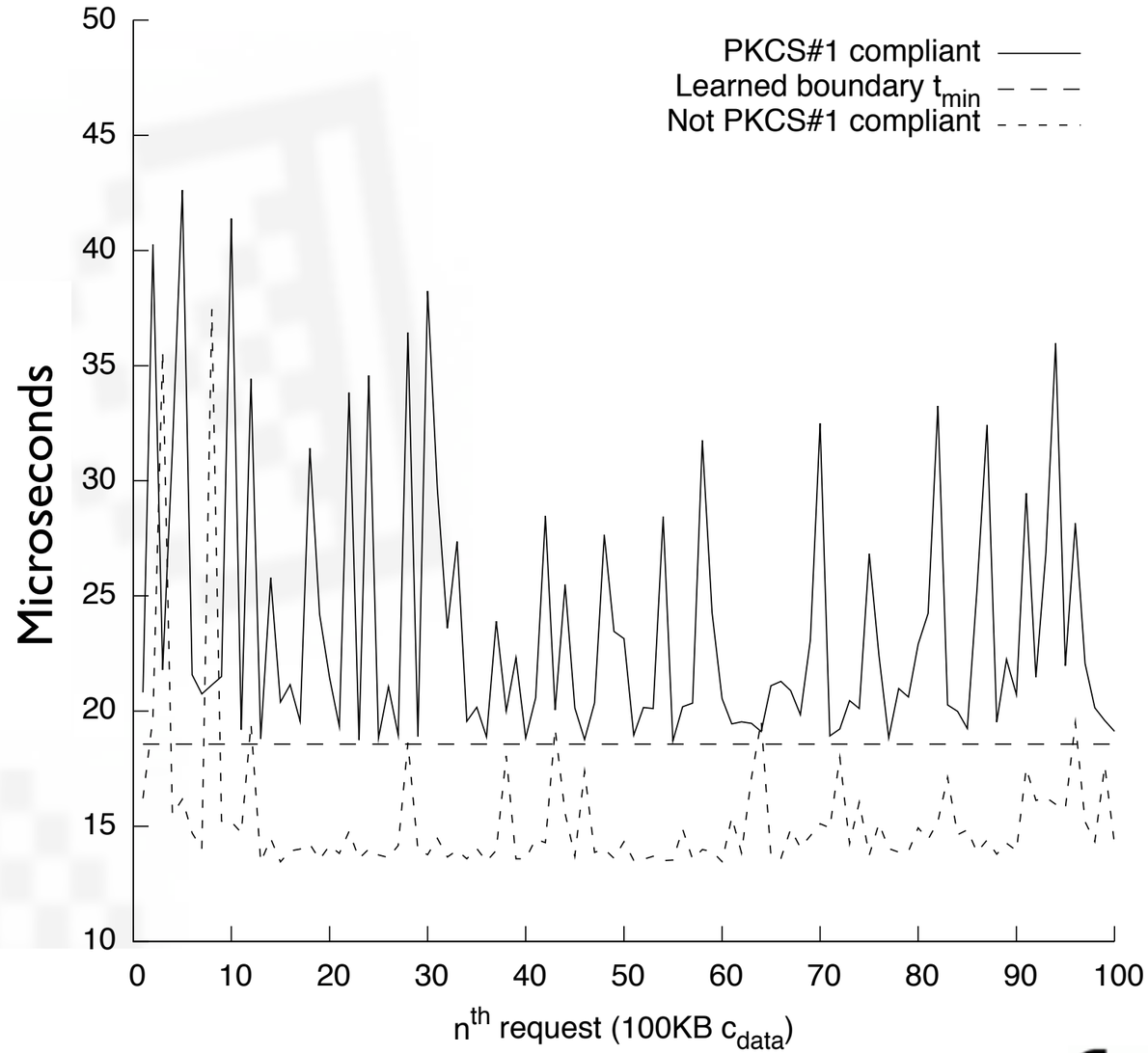


Examples for attacks

Breaking XML Encryption - Timing attack against local server

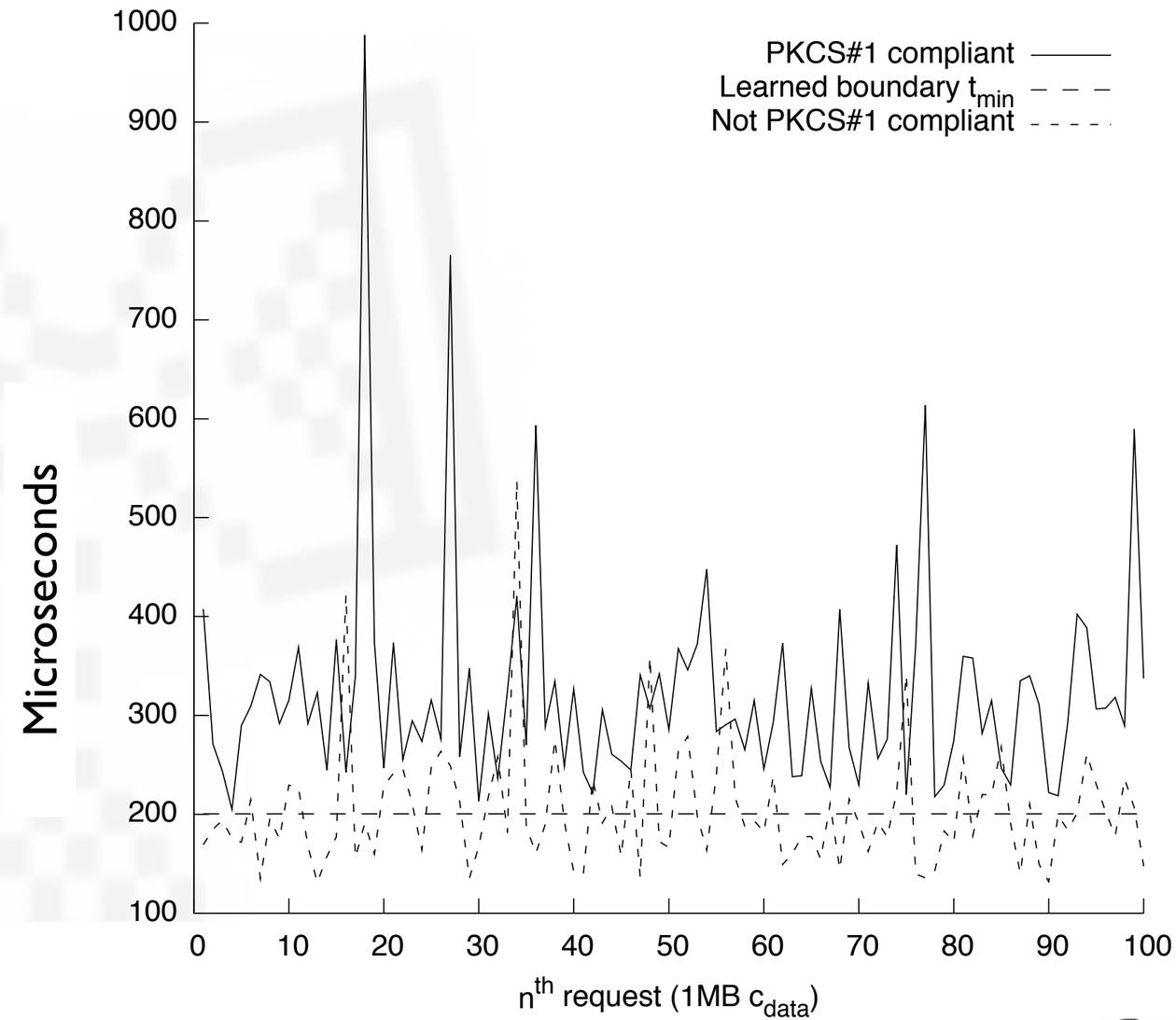
Attack against local server

- Decrypt ciphertext in ~3 hours
- Size of c_{data} was 100KB
- 321.870 oracle queries
- 398.123 actual requests (1.24 actual requests per oracle query)



Attack against Internet server (Planetlab)

- Decrypt ciphertext in < 1 week
- Size of c_{data} was 1MB
- 2000 requests per hour
- 1.2 requests per oracle query



Call for participation

Call for participation:

- Many open problems left, e.g.
  - integrate statistical hypothesis tests in FAU Analyser
  - Make interpacket timings available in FAU Timing
  - Come up with new and creative timing attacks
  - Test many, many applications... :-)

- Great topics for pentesters, researchers, and student theses!

Literature & further reading

[1] Scott A. Crosby and Dan S. Wallach and Rudolf H. Riedi, *Opportunities and Limits of Remote Timing Attacks,* ACM Trans. Inf. Syst. Secur, 12(3), 2009.

[2] Tibor Jager and Juraj Somorovsky, *How to break XML encryption,* Proceedings of the18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011

[3] Edward W. Felten and Michael A. Schneider. *Timing attacks on web privacy*. In SIGSAC: 7th ACM Conference on Computer and Communications Security. ACM SIGSAC, 2000.

[4] Andrew Bortz and Dan Boneh. Exposing private information by timing web applications. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, WWW, pages 621–628. ACM, 2007.

[5] Andrew Bortz and Dan Boneh. *Exposing private information by timing web applications*. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, WWW, pages 621–628. ACM, 2007.

[7] Sebastian Schinzel, *An Efficient Mitigation Method for Timing Side Channels on the Web,* Proceedings of IFIP/SEC 2011.
http://sebastian-schinzel.de/_download/cosade-2011-extended-abstract.pdf

[8] Felix C. Freiling and Sebastian Schinzel, *Detecting Hidden Storage Side Channel Vulnerabilities in Networked Applications,* Proceedings of IFIP/SEC 2011.
http://sebastian-schinzel.de/_download/ifip-sec2011.pdf

[9] Daniel Bleichenbacher. *Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1*. In Advances in Cryptology – CRYPTO 1998, pages 1–12, 1998.

Thanks for your attention!

Discussion...

Web 1.0: sebastian.schinzel@cs.fau.de
Web 2.0: https://twitter.com/seecurity

Backup

Preventing Timing Side Channel Attacks

Preventing Bleichenbacher attack against XML Encryption

Vulnerable:

1. Decrypt session key $m = dec_{rsa}(c_{key})$
2. **Return error** if *m* does not comply with PKCS#1 and **stop here**
3. Decrypt c_{data} (results in XML subtree)
4. Copy subtree in XML doc
5. Parse XML doc
6. Return error if XML doc is invalid

Examples for attacks

Breaking XML Encryption - Preventive Measures

Preventing Bleichenbacher attack against XML Encryption

Vulnerable:

1. Decrypt session key $m = dec_{rsa}(c_{key})$
2. **Return error** if *m* does not comply with PKCS#1 and **stop here**
3. Decrypt c_{data} (results in XML subtree)
4. Copy subtree in XML doc
5. Parse XML doc
6. Return error if XML doc is invalid

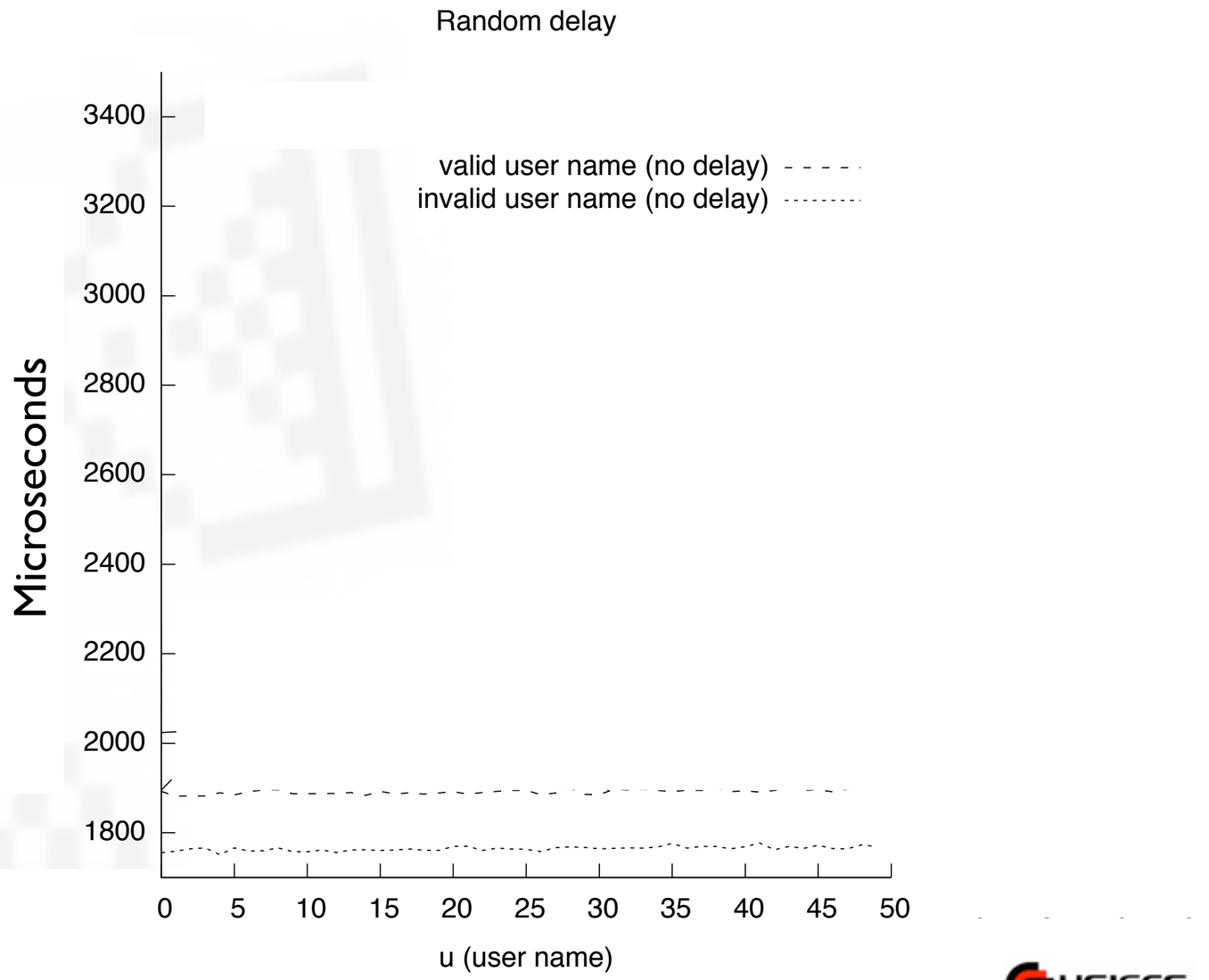
Fixed:

1. Decrypt session key $m = dec_{rsa}(c_{key})$
2. **Generate random session key *m'*** if *m* does not comply with PKCS#1 **and continue**
3. Decrypt c_{data} (results in XML subtree)
4. Copy subtree in XML doc
5. Parse XML doc
6. Return error if XML doc is invalid

Random delay padding

1. r = random() % 200 µs
2. usleep(r)

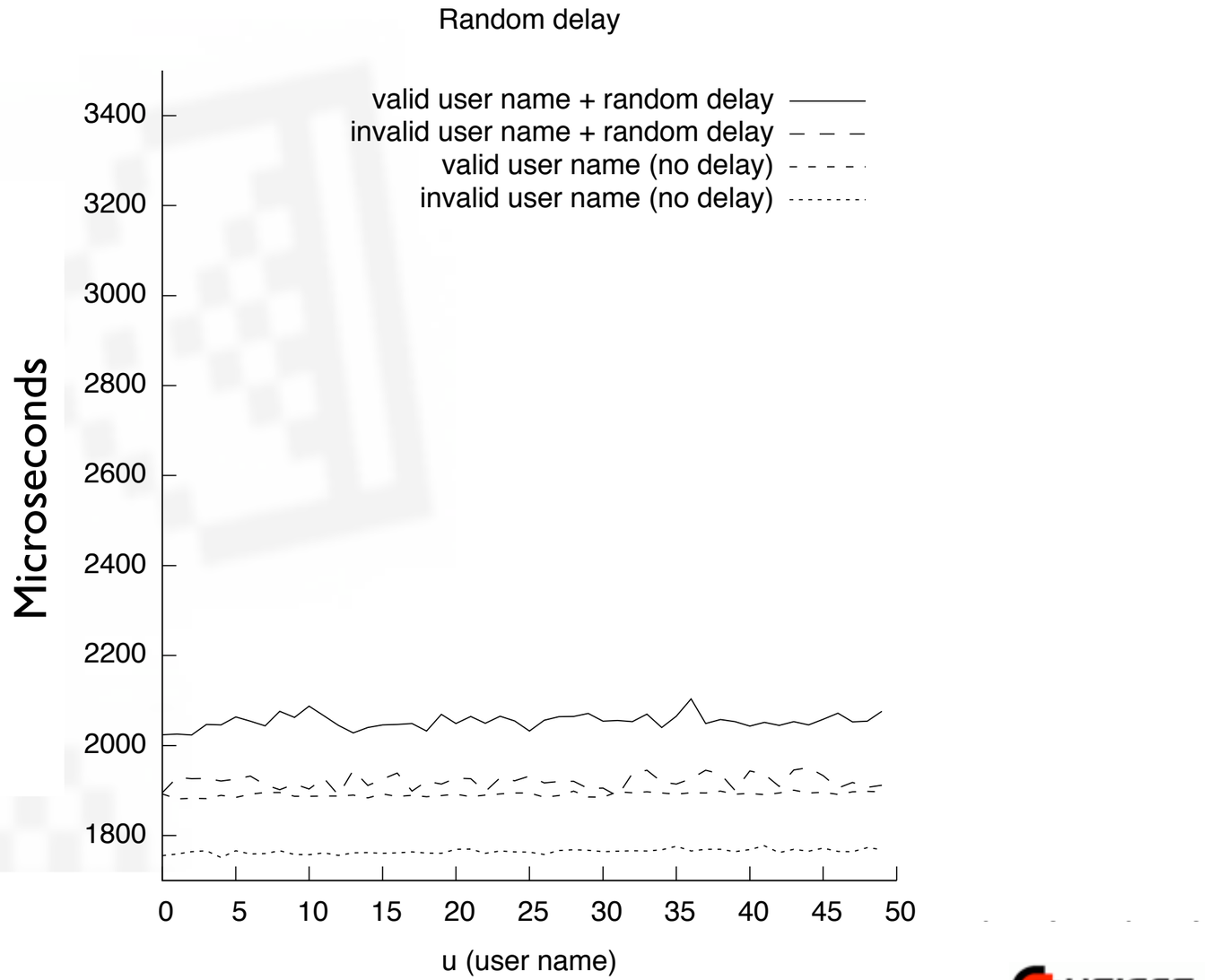
- PRO: increases effort for timing analysis (attacker needs to measure more often)
- CON: proper filtering will remove the random delay



Random delay padding

1. r = random() % 200 µs
2. usleep(r)

- PRO: increases effort for timing analysis (attacker needs to measure more often)
- CON: proper filtering will remove the random delay

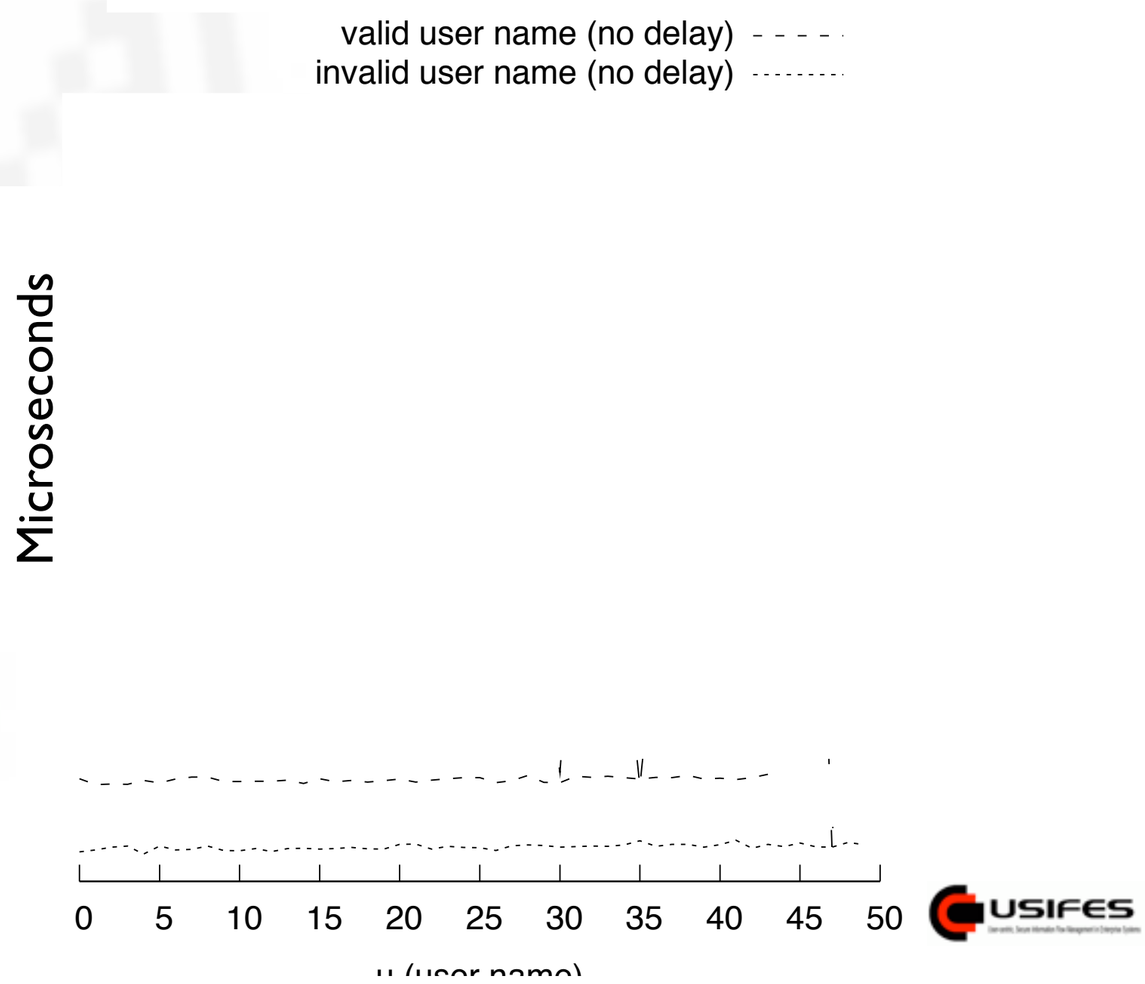


Deterministic and Unpredictable Delay

1. r = md5(input + secret) % 200 µs

2. usleep(r)

- PRO: offers security guarantees that are independent of the amount of measurements



Deterministic and Unpredictable Delay

1. r = md5(input + secret) % 200 µs

2. usleep(r)

- PRO: offers security guarantees that are independent of the amount of measurements