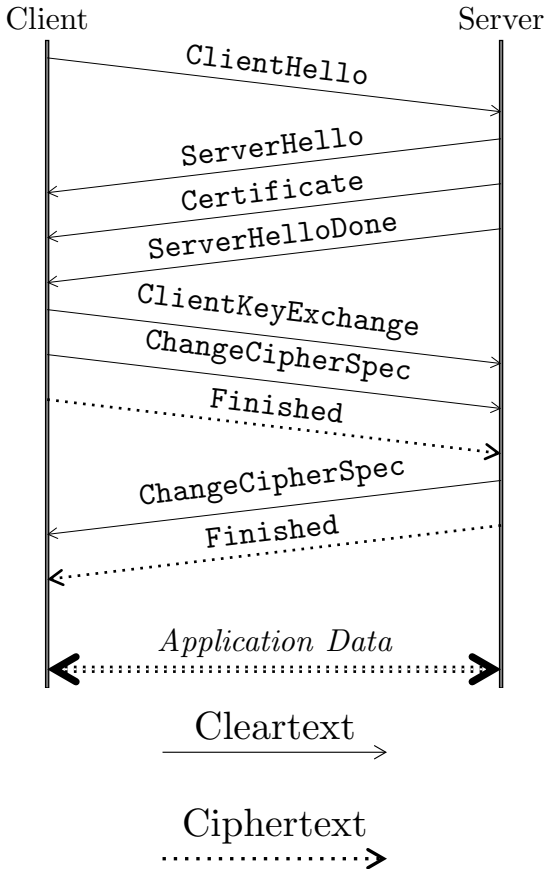
TLS Record Protocol: Security Analysis and Defense-in-depth Countermeasures for HTTPS

Olivier Levillain, Baptiste Gourdin, Hervé Debar

ANSSI, Sekoia, Télécom SudParis

ASIACCS 2015

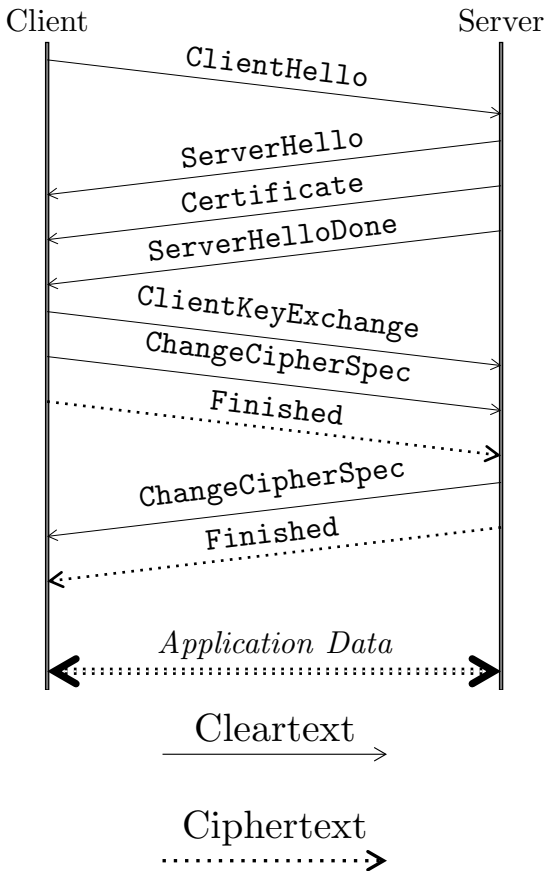
TLS in a nutshell



Two phases

- secure channel establishment
  - algorithm negotiation
  - server authentication
  - key exchange to obtain a shared secret
- application data exchanges using this channel

TLS in a nutshell



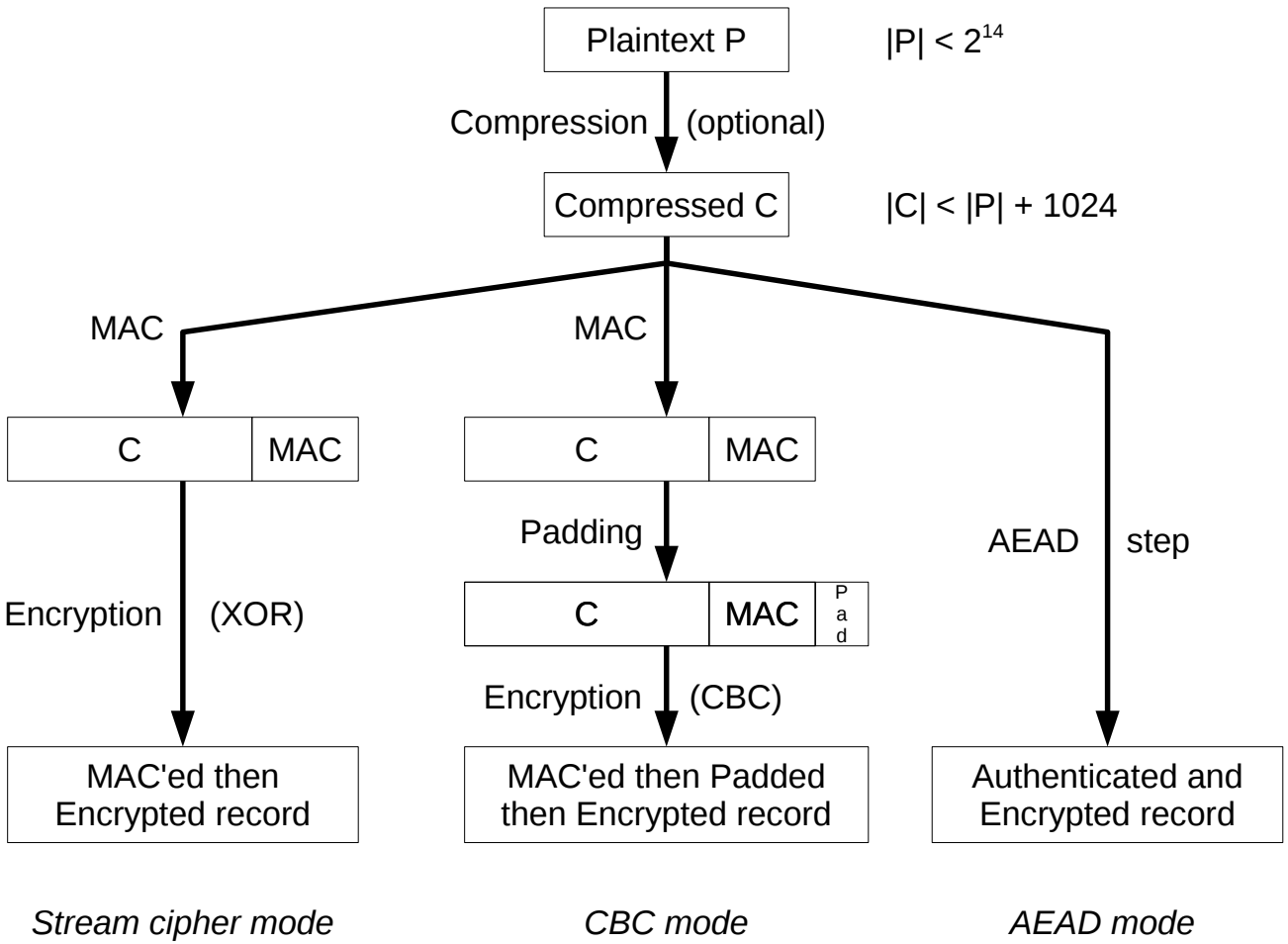
Two phases

- secure channel establishment
  - algorithm negotiation
  - server authentication
  - key exchange to obtain a shared secret
- application data exchanges using this channel

This talk focuses on the second phase, the Record Protocol

TLS Record Protocol

After the handshake, records can be protected using 3 different schemes:



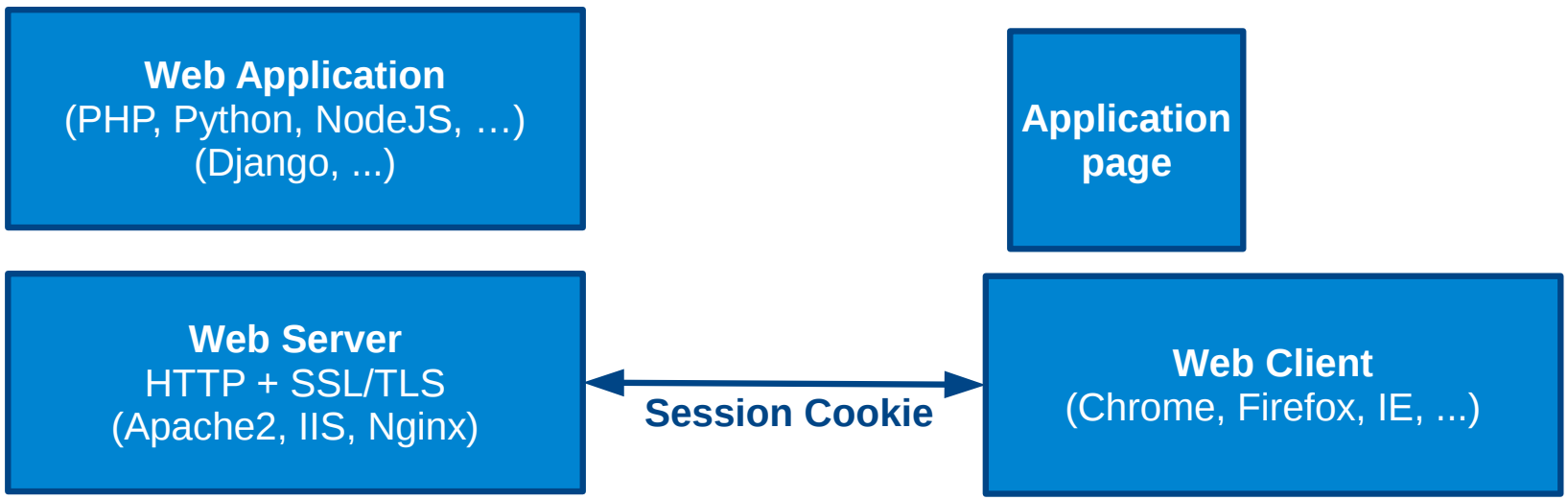
Well, all started... in 2011

- 2011 : BEAST
  - CBC mode with implicit IV
- 2012 : CRIME (followed by TIME and BREACH)
  - Compression attacks
- 2013 - 2014 : Lucky13 (followed by POODLE)
  - CBC Padding Attacks
- 2014 : RC4 biases (no real name)
  - RC4 statistical biases

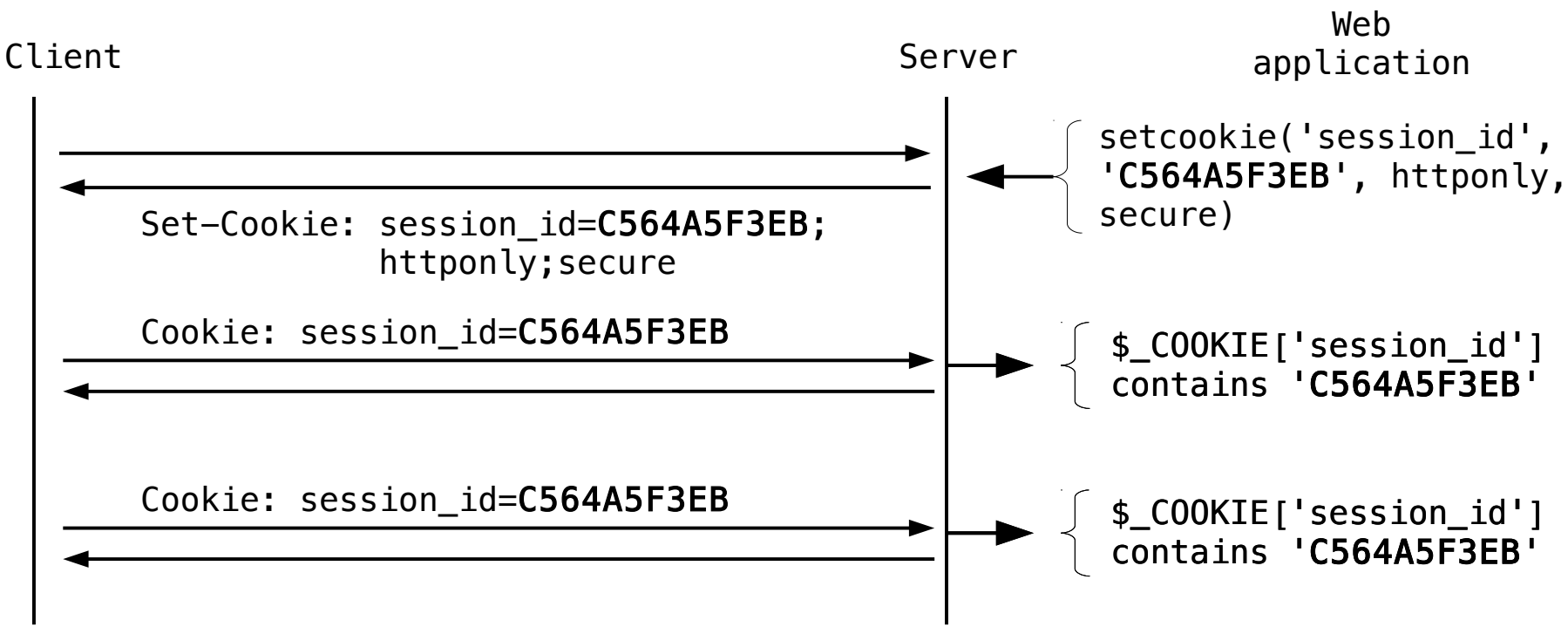
The cookie monsters

- BEAST, TIME, CRIME, BREACH, Lucky13, POODLE, RC4 biases, ...
  - all the PoCs went after cookies
  - all relies on having the cookie repeated inside the TLS channel

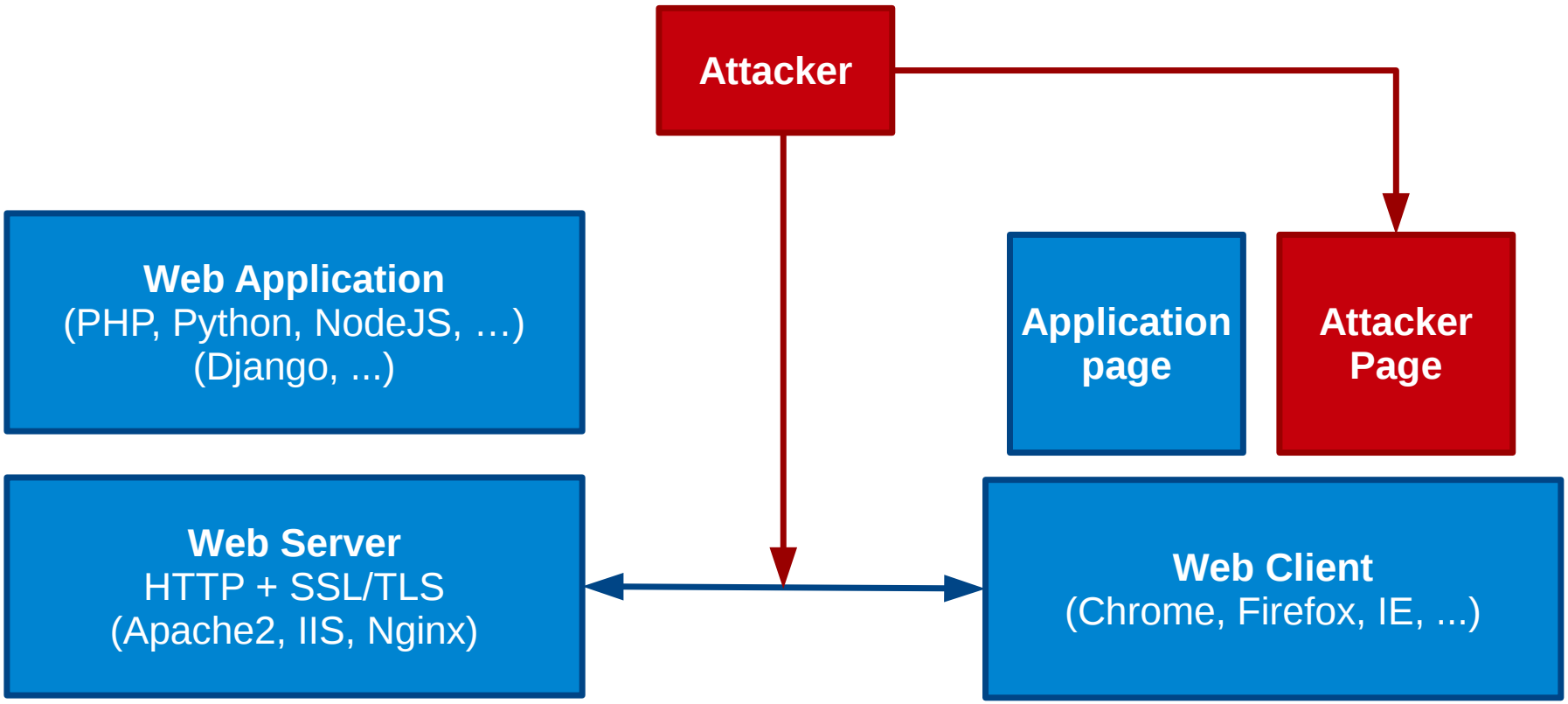
Model


Web Application
(PHP, Python, NodeJS, …)
(Django, ...)
Application page
Web Server
HTTP + SSL/TLS
(Apache2, IIS, Nginx)
Session Cookie
Web Client
(Chrome, Firefox, IE, ...)

RFC6265: HTTP State Management Mechanism



Attacker Model



Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary of the proposed countermeasures

Countermeasures	Beast	L 13	RC4	*IME	POODLE
Structural changes to TLS					
Use TLS 1.0					+
Use TLS 1.1	+				+
Encrypt-then-MAC		+			
Changes related to TLS ciphersuites or compression methods					
Use CBC mode			+		
Use RC4	+	+			+
Use a new stream cipher	+	+	+		+
Use AEAD (TLS 1.2)	+	+	+		+
No TLS compression				+	
Changes related to TLS implementations					
$1/n-1$ split	+				
Constant-time CBC		+			
Anti poodle splitting					+

Summary

- Since 2011, seven attacks affecting the Record Protocol

Summary

- Since 2011, seven attacks affecting the Record Protocol
- Generally, each attack has been thwarted using a **specific fix**

Summary

- Since 2011, seven attacks affecting the Record Protocol
- Generally, each attack has been thwarted using a **specific fix**
- TLS 1.2 with AEAD suites offer a clean fix for most attacks
  - **But ...** TLS 1.2 is not implemented everywhere
  - **But ...** Older versions still supported

Summary

- Since 2011, seven attacks affecting the Record Protocol
- Generally, each attack has been thwarted using a **specific fix**
- TLS 1.2 with AEAD suites offer a clean fix for most attacks
  - **But ...** TLS 1.2 is not implemented everywhere
  - **But ...** Older versions still supported
- A common denominator: all PoCs target **repeated secrets**
  - cookies or anti-CSRF tokens

Summary

- Since 2011, seven attacks affecting the Record Protocol
- Generally, each attack has been thwarted using a **specific fix**
- TLS 1.2 with AEAD suites offer a clean fix for most attacks
  - **But ...** TLS 1.2 is not implemented everywhere
  - **But ...** Older versions still supported
- A common denominator: all PoCs target **repeated secrets**
  - cookies or anti-CSRF tokens
- What if we could **avoid this repetition** ?

First-order attacks

Considered attacks: for each encrypted record, the attacker can retrieve some information about κ consecutive bytes of plaintext

- Typically, $\kappa = 1$ and the attacker can check whether a cleartext byte is equal to a guessed value (e.g. Lucky13)
- Sometimes, the attacker must aggregate information resulting from several records (e.g. RC4 biases)
- Even if it can be raised, κ is at most 4 in realistic scenarios

- Such attacks can be called **first-order attacks**

The Masking Principle

As for the term *first-order attacks*, we borrow from the side-channel attacks litterature the *masking principle*.

- Each time a secret s of κ bytes must be transmitted
- Pick a random value m (the mask) of the same length
- Send the pair $(m, m \oplus s)$

- Thus, the value can trivially be recomputed
- But the representation on the wire is different for every message

The Masking Principle

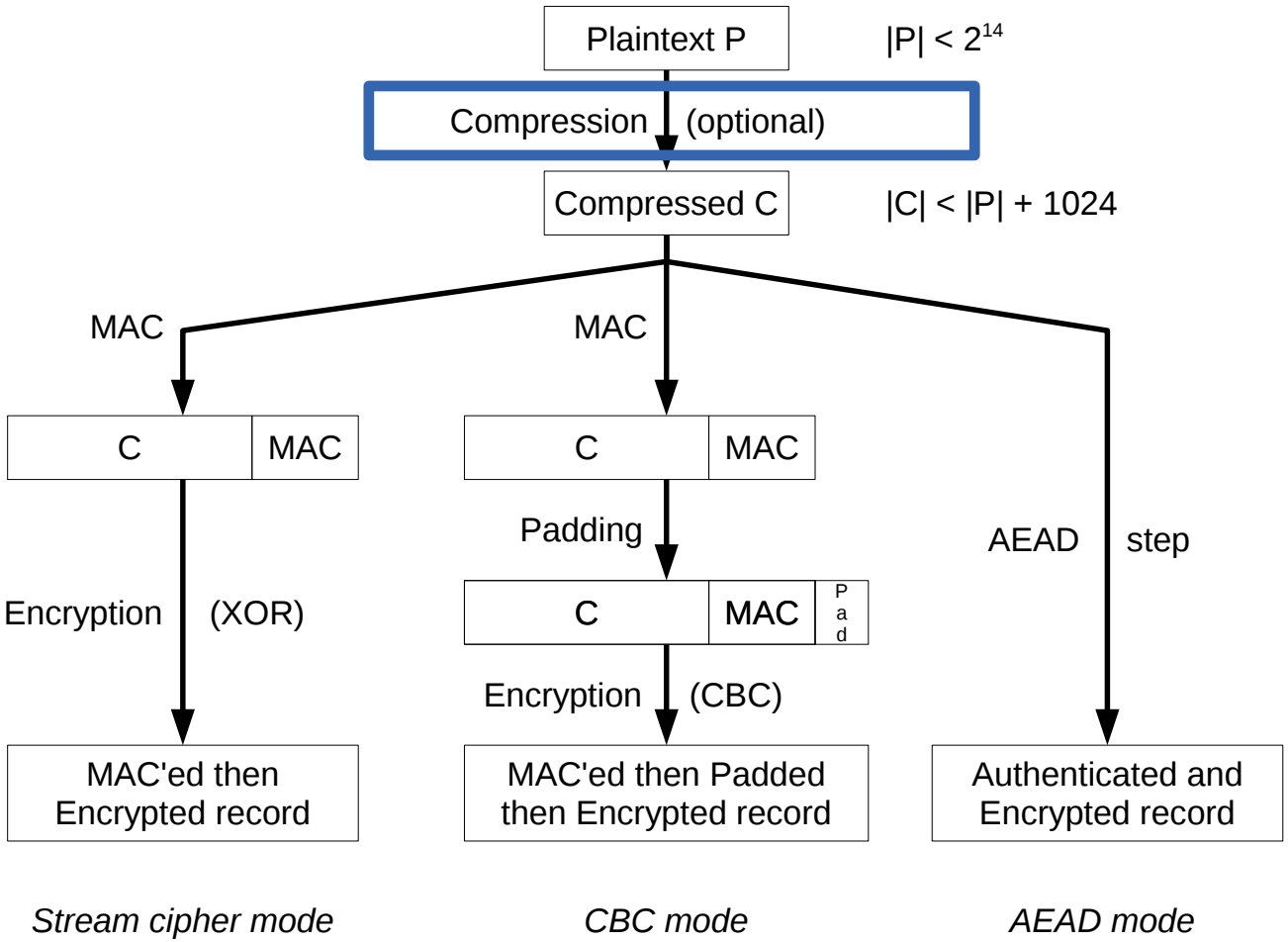
As for the term *first-order attacks*, we borrow from the side-channel attacks litterature the *masking principle*.

- Each time a secret s of κ bytes must be transmitted
- Pick a random value m (the mask) of the same length
- Send the pair $(m, m \oplus s)$

- Thus, the value can trivially be recomputed
- But the representation on the wire is different for every message

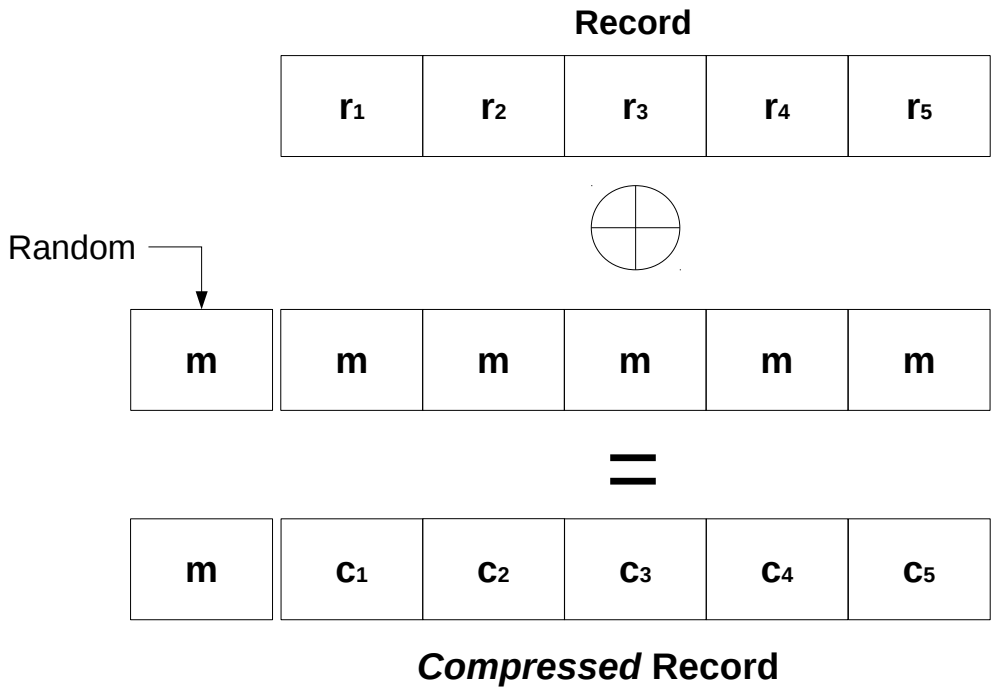
- Since the attacker can only recover information about κ consecutive bytes for each record, she only obtains random data

Masking the TLS layer



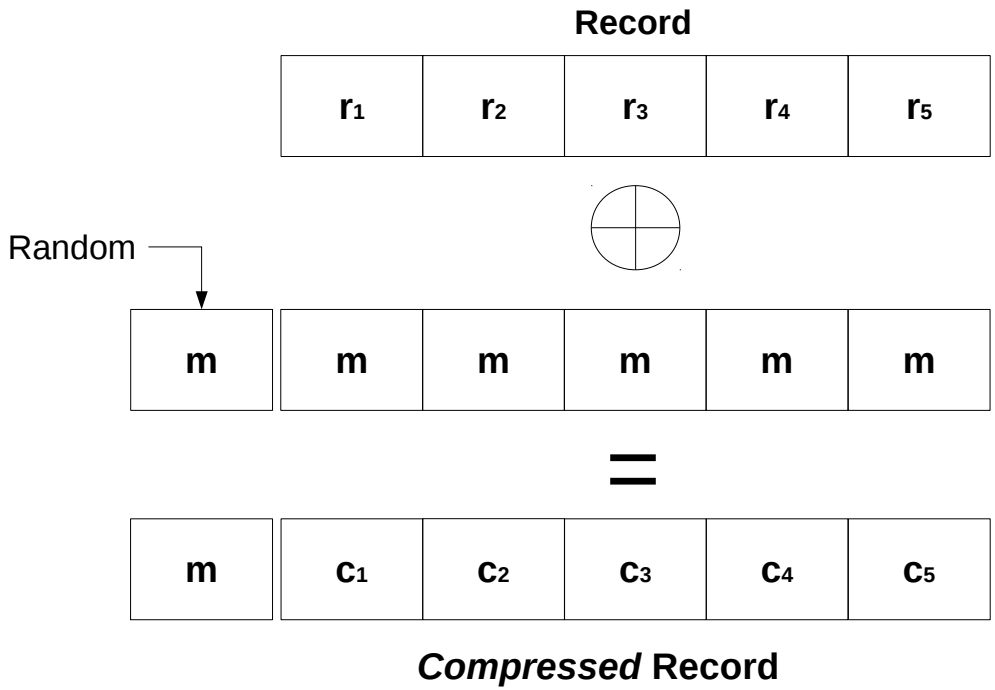
Masking the TLS layer

- In practice, TLS Compression layer allows almost *any* reversible transformation of the plaintext



Masking the TLS layer

- In practice, TLS Compression layer allows almost *any* reversible transformation of the plaintext



- This toy implementation does *not* follow the principle edicted before, since the whole record is masked, not just the secret

Implementation

OpenSSL implementation

- New compression method : `scramble`
- 75-line patch to add the scramble method
- Mask length is set to 8
- Some minor patches needed to add `scramble` support into `s_client` and `s_server`

- CPU and network bandwidth are negligible
- In practice, compression is now obsolete in TLS, so deploying a new compression method is irrelevant

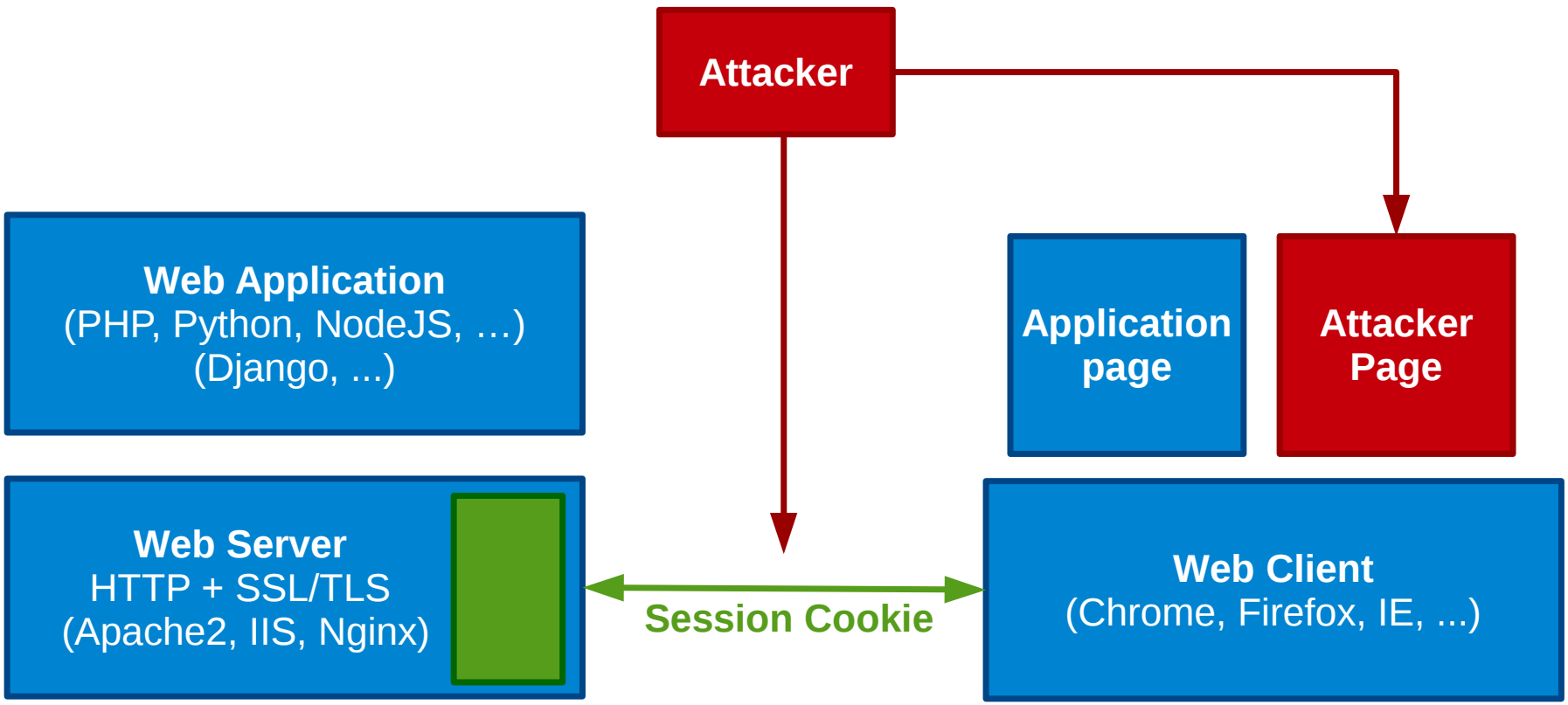
Security analysis

- The first BEAST proof of concept used WebSockets
- It was thwarted when 4-byte masking was introduced to avoid some confusion attacks

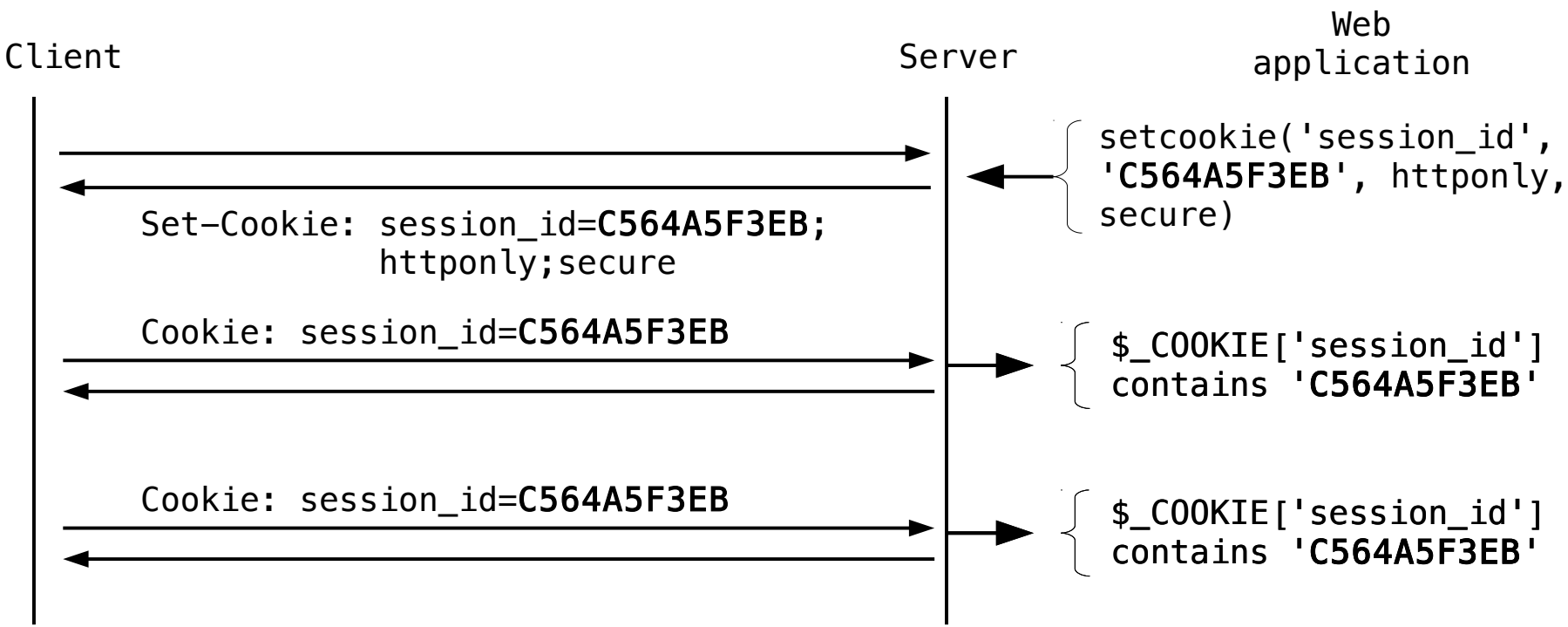
- TLS scrambling would thus thwart BEAST
- It should also work against Lucky 13, RC4 single-byte biases and POODLE

- Yet, it does not only mask the secrets, so some attacks still work (e.g.: application-level compression)
- It should only be considered as a toy implementation

Masking at the HTTP Layer



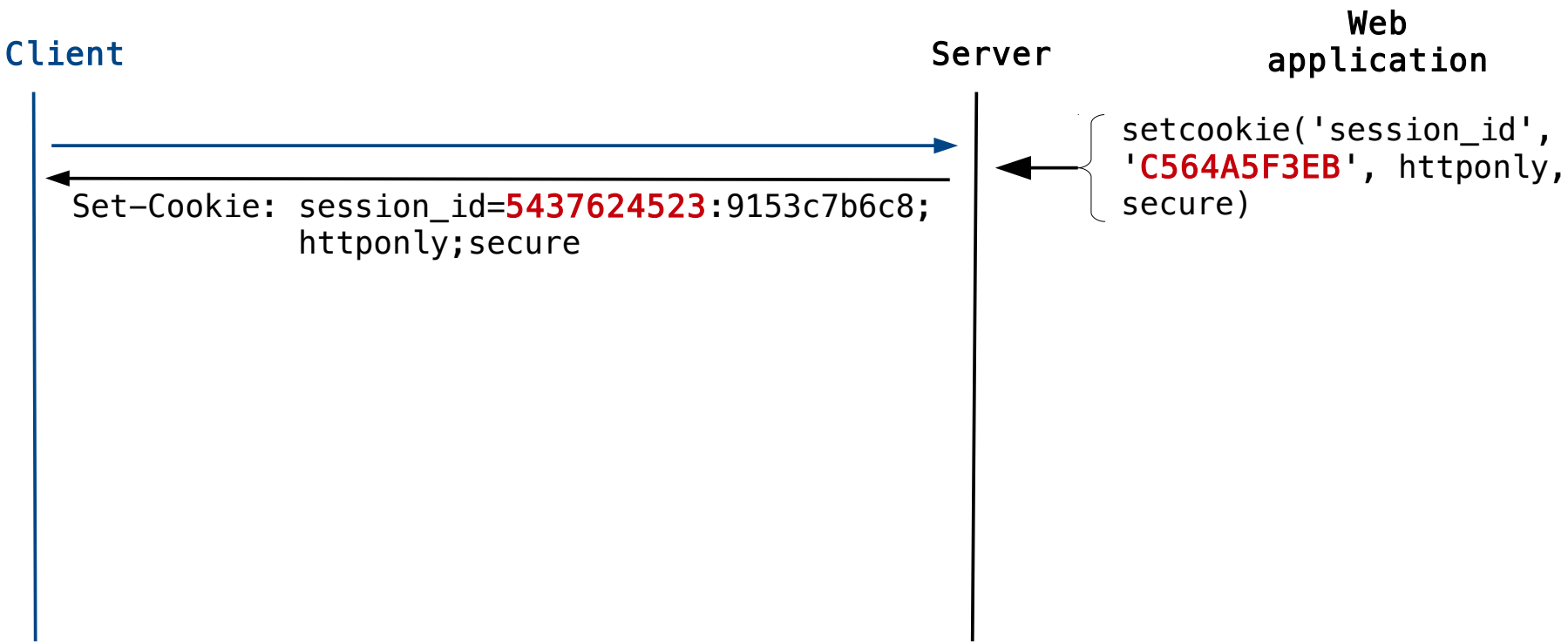
RFC6265: HTTP State Management Mechanism



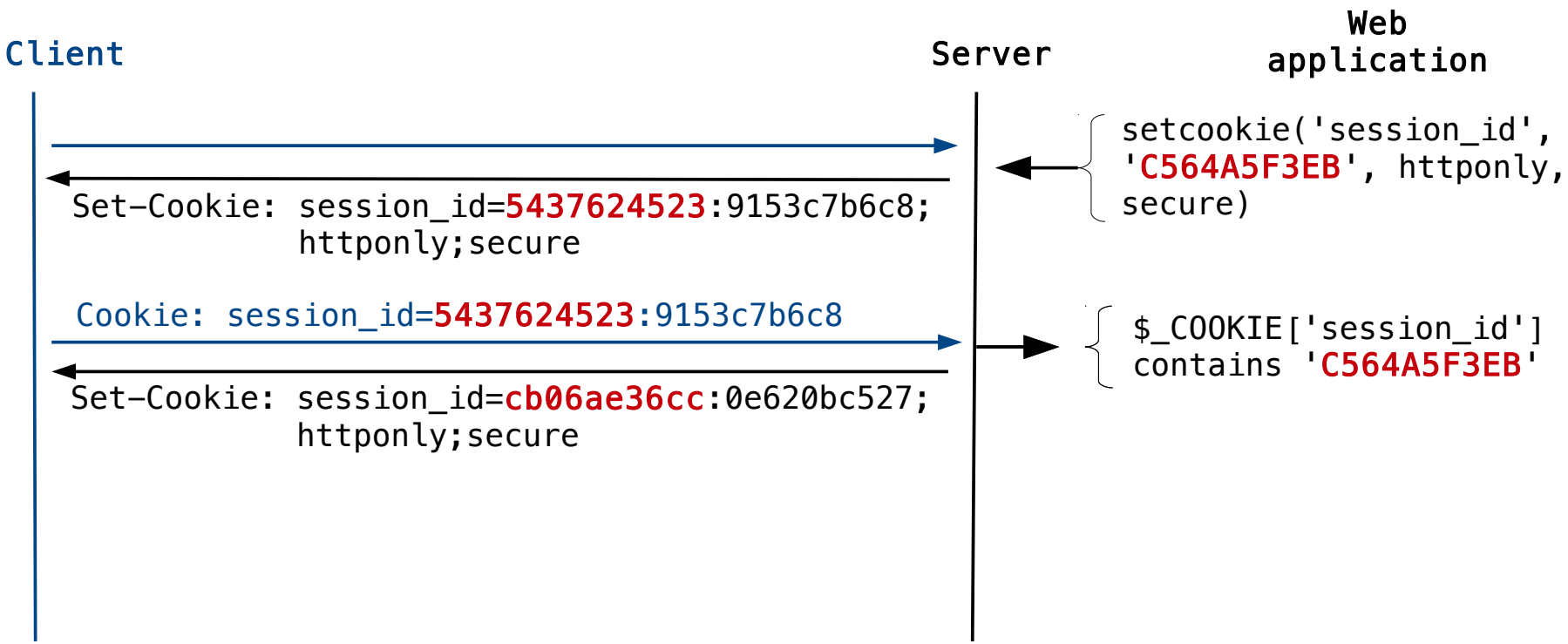
MCookies

- Objective: use a random mask for each sent cookie
- Targets: secure & httpOnly cookies
- How: for each server response, send a fresh pair $(m, m \oplus c)$

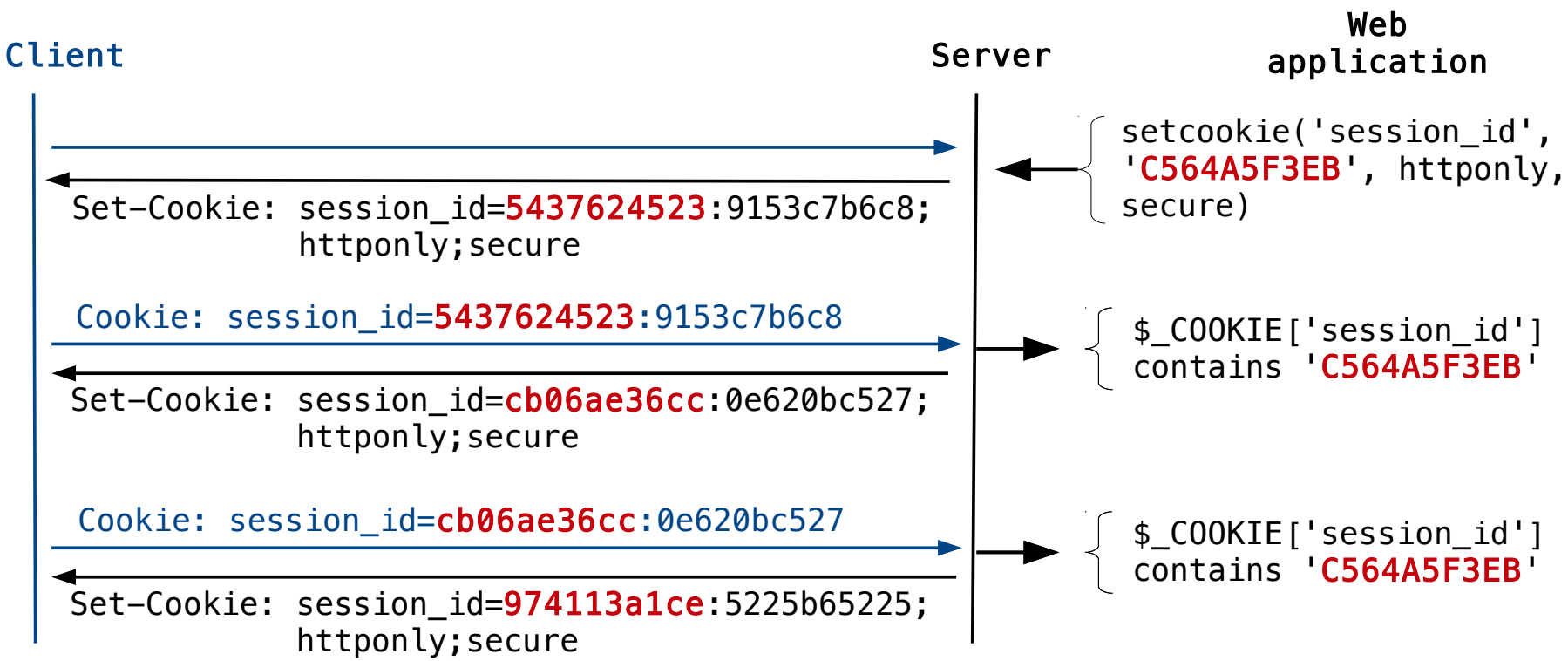
MCookies


Client
Server
Web application
setcookie('session_id', 'C564A5F3EB', httponly, secure)
Set-Cookie: session_id=5437624523:9153c7b6c8; httponly;secure

MCookies


Client
Server
Web application
setcookie('session_id', 'C564A5F3EB', httponly, secure)
Set-Cookie: session_id=5437624523:9153c7b6c8; httponly;secure
Cookie: session_id=5437624523:9153c7b6c8
$_COOKIE['session_id'] contains 'C564A5F3EB'
Set-Cookie: session_id=cb06ae36cc:0e620bc527; httponly;secure

MCookies


Client
Server
Web application
setcookie('session_id', 'C564A5F3EB', httponly, secure)
Set-Cookie: session_id=5437624523:9153c7b6c8; httponly;secure
Cookie: session_id=5437624523:9153c7b6c8
$_COOKIE['session_id'] contains 'C564A5F3EB'
Set-Cookie: session_id=cb06ae36cc:0e620bc527; httponly;secure
Cookie: session_id=cb06ae36cc:0e620bc527
$_COOKIE['session_id'] contains 'C564A5F3EB'
Set-Cookie: session_id=974113a1ce:5225b65225; httponly;secure

Implementation

- Implemented as a simple Apache2 module (500 loc)
  - `a2enmod mcookies` is enough
  - Works with sequential requests
  - Works with parallel requests
- However some cookies attributes are lost in the process
  - (`Expires`, `Max-Age`, `Domain`, `Path`)

Implementation

- Expiration attribute is client side
- Session expiration should always be done server side.
- Fix 1: Add theses attributes to the MCookie
  - Server response: $(m : m \oplus v : a)$
  - Client request: $(m : m \oplus v : a)$
- Fix 2: Configure the Apache module
  - `vim /etc/apache2/mods-enabled/...`

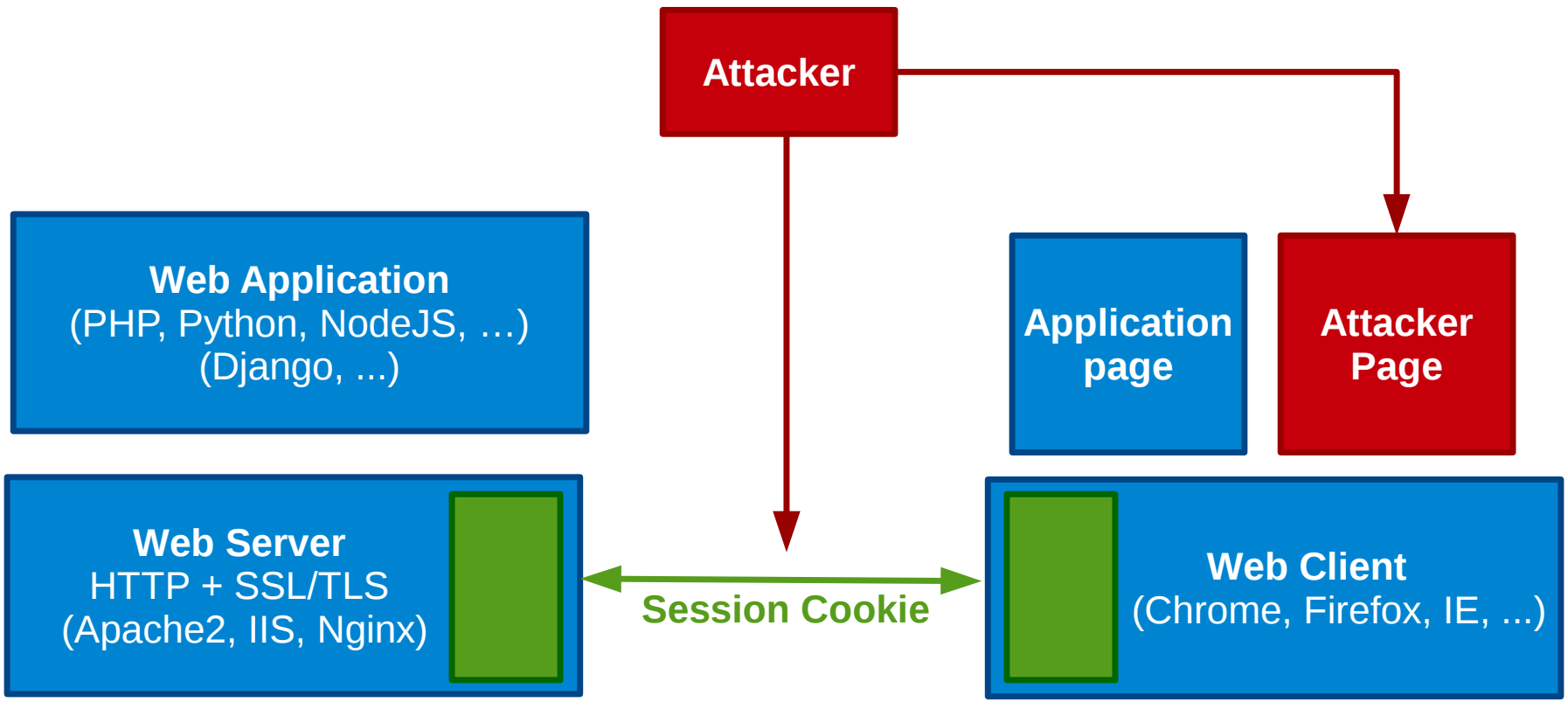
Experiment

- Result: it works but it forces the server to re-emit a cookie each time
- Overhead ?
- Experiment scenario
  - Simulate an active user browsing internet services
  - Dump the HTTP traffic
  - Emulate the same traffic using MCookies for each Secure+httpOnly cookie

Traffic type	**Raw volume**	**Overhead**	**Overh. optim.**
Sensitive	24 MB	+20.1 %	+14.9 %
Overall	122 MB	+4.1 %	+3.0 %

Table: Experiment result

What if the browser could handle the masking ?



Masked-Cookies

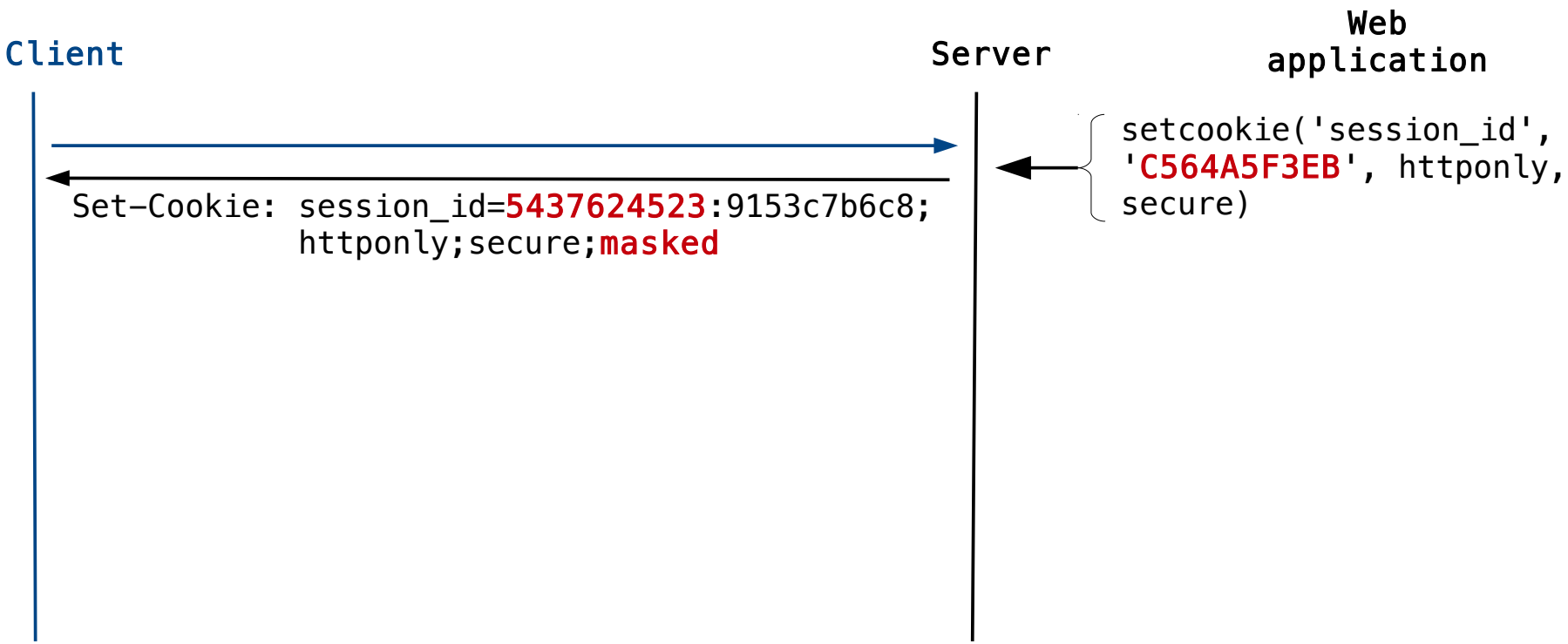
Idea:

- The server can specify which cookies to protect
- The browser now sends a fresh $(m, m \oplus v)$ with each request.
- No more overhead.

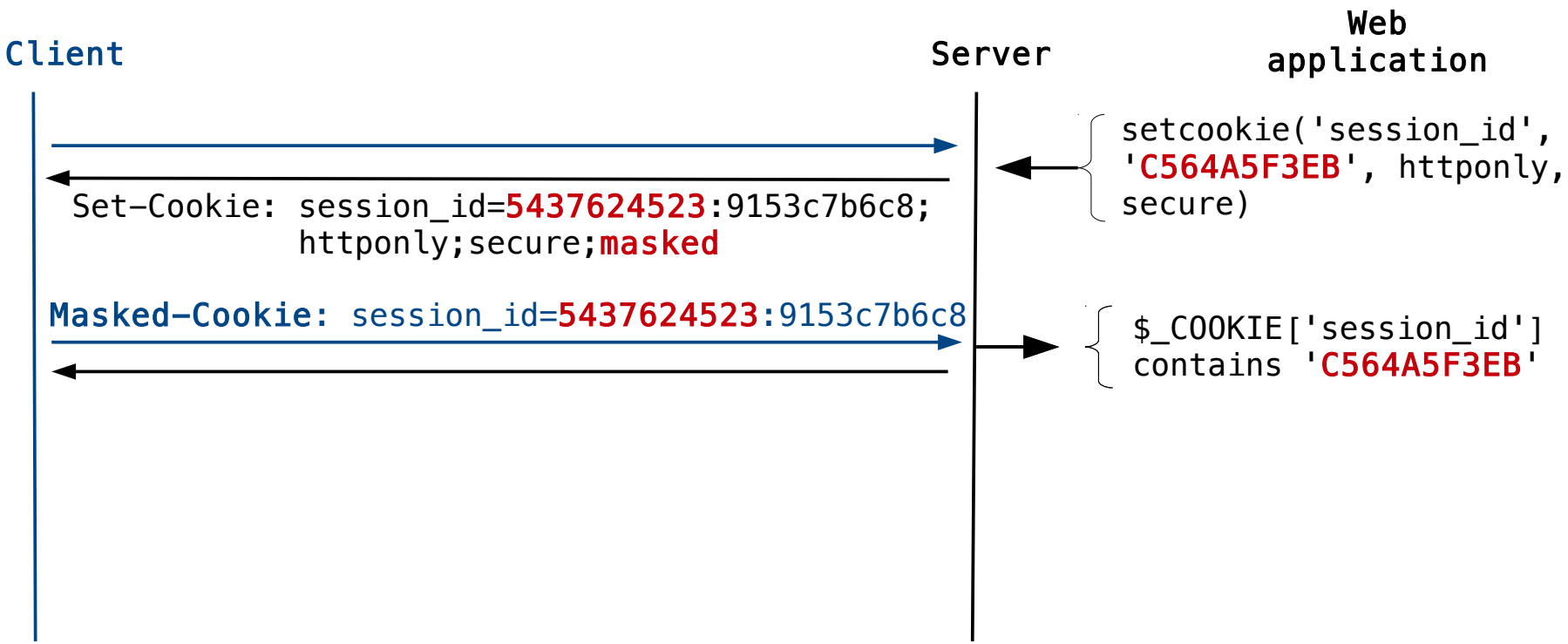
Proposal:

- `masked` attribute:
  - Set-Cookie: cookie=val;secure;httponly;**masked**

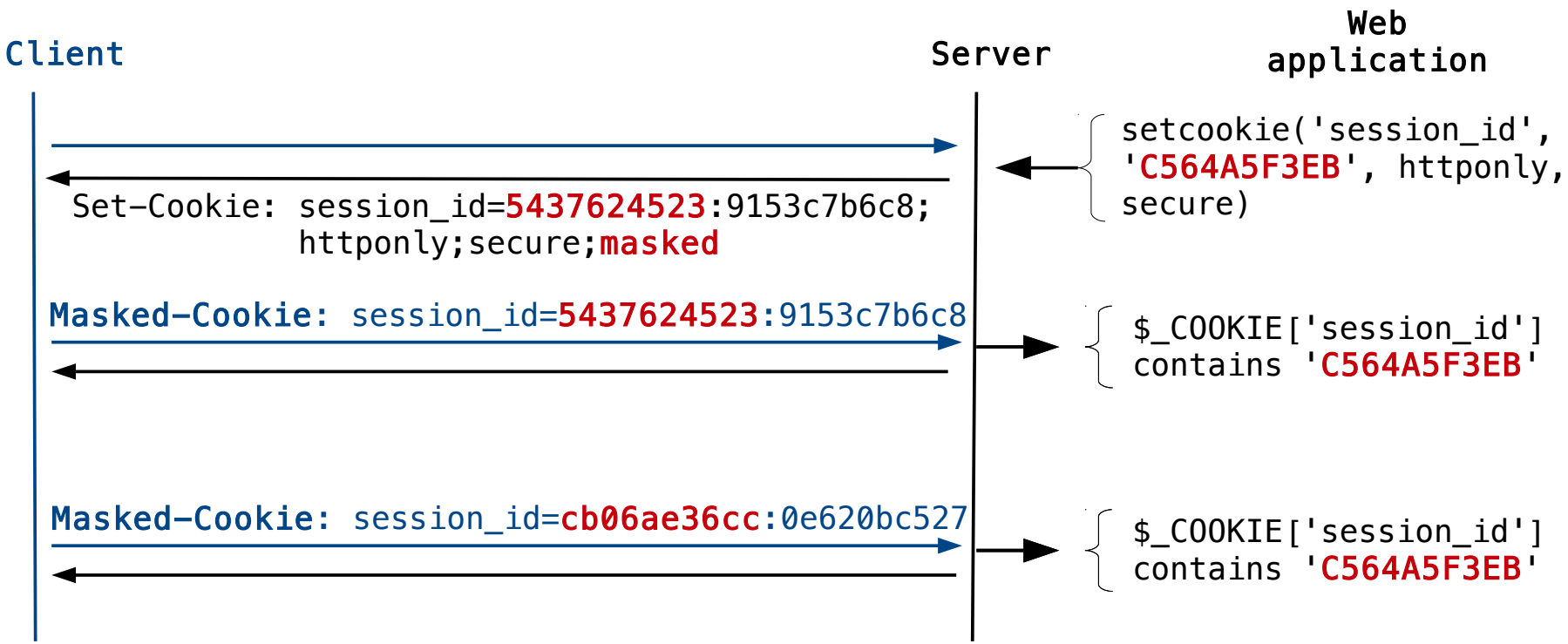
Masked-Cookies



Masked-Cookies



Masked-Cookies



Implementation

- Implemented as a simple Apache2 module (500 loc)
- and a patch for chromium. (200 loc)
  - Works with sequential requests
  - Works with parallel requests

MCookies Vs Masked-Cookies

- Same experiment scenario

Traffic type	**Raw traffic volume**	**Extra bandwidth**		
		w/o UA support		with UA support
		naive	optim.	
Sensitive	24 MB	+20.1 %	+14.9 %	+10.8 %
Overall	122 MB	+4.1 %	+3.0 %	+2.2 %

Table: Network overhead evalulation

MCookies Vs Masked-Cookies

- **CPU Overhead ?**

	Vanilla server	**MCookies enabled**	
		w/o UA support	with UA support
Static page	384	318 (-17 %)	382
Wordpress page	221	212 (-4 %)	220

Table: Performance results (transactions/second)

Summary

- Recent attacks on TLS Record Protocol rely on a **repeated secret**

Summary

- Recent attacks on TLS Record Protocol rely on a **repeated secret**
- Our proposal to implement defense-in-depth: **break this repetition**

Summary

- Recent attacks on TLS Record Protocol rely on a **repeated secret**
- Our proposal to implement defense-in-depth: **break this repetition**
- **MCookies and Masked Cookies** can be implemented and work

Summary

- Recent attacks on TLS Record Protocol rely on a **repeated secret**
- Our proposal to implement defense-in-depth: **break this repetition**
- **MCookies and Masked Cookies** can be implemented and work
- POODLE validated our approach

Summary

- Recent attacks on TLS Record Protocol rely on a **repeated secret**
- Our proposal to implement defense-in-depth: **break this repetition**
- **MCookies and Masked Cookies** can be implemented and work
- POODLE validated our approach

- Yet, secret randomization is a palliative countermeasure, **not the ultimate fix**

Summary

- Recent attacks on TLS Record Protocol rely on a **repeated secret**
- Our proposal to implement defense-in-depth: **break this repetition**
- **MCookies and Masked Cookies** can be implemented and work
- POODLE validated our approach

- Yet, secret randomization is a palliative countermeasure, **not the ultimate fix**
- MCookies should be useful as a **defense-in-depth countermeasure**, to get some time to patch

Questions?

Thank you for your attention

Some history

- 1994: Netscape publishes SSLv2 (`https://` is born)
- 1995: Netscape publishes SSLv3, which fixes major flaws
- 1999: TLS 1.0 (aka SSLv3.1) is standardised by the IETF
- 2006: TLS 1.1 fixes bugs in CBC mode and updates ciphersuites
- 2008: TLS 1.2 introduces modern cryptographic modes
- 2015 (?): TLS 1.3 is coming

Some history

- ~~1994: Netscape publishes SSLv2 (`https://` is born)~~
- 1995: Netscape publishes SSLv3, which fixes major flaws
- 1999: TLS 1.0 (aka SSLv3.1) is standardised by the IETF
- 2006: TLS 1.1 fixes bugs in CBC mode and updates ciphersuites
- 2008: TLS 1.2 introduces modern cryptographic modes
- 2015 (?): TLS 1.3 is coming

SSLv2 hopefully is history: this talk is about SSLv3 - TLS

CBC mode with implicit IV

- **Attack name**: BEAST
- **Authors**: Rogaway (theoretic), Duong and Rizzo (practical attack)
- **Date**: 1995 (theoretic), 2011 (practical attack)
- **Hypotheses and prerequisites**:
  - the TLS connection uses CBC with an implicit IV
  - the ciphertext is observable
  - the plaintext is partially controlled, adaptively
  - *the same secret is repeated in different connections*
- **Ideal fix**: use TLS 1.1 (explicit IV)
- **Common fix**: split records to randomize IV in practice

Compression attacks

- **Attack name**: CRIME (followed by TIME and BREACH)
- **Authors**: Duong and Rizzo
- **Date**: 2012 (practical attack)
- **Hypotheses and prerequisites**:
  - a form of compression is enabled (TLS or HTTP)
  - the ciphertext length is observable (packet size or timing difference)
  - plaintext can be loosely controlled
  - *the same secret is repeated in different connections*
- **Ideal fix**: avoid mixing attacker-controlled data and secrets
- **Common fix**: disable compression

CBC Padding Attacks

- **Attack name**: Lucky13 (followed by POODLE)
- **Authors**: Vaudenay (theoretic), AlFardan et al. (Lucky13) and Moeller et al. (POODLE)
- **Date**: 2002 (theoretic), 2013-2014 (practical attack)
- **Hypotheses and prerequisites**:
  - the connection uses CBC
  - the decryption process leaks information (Lucky13), or
  - the decryption uses SSLv3-style padding (POODLE)
  - the attacker can intercept and modify network packets
  - *the same secret is repeated in different connections*
- **Ideal fix**: use Encrypt-then-Mac or proper authenticated encryption
- **Common fix (Lucky13)**: implement constant-time CBC decryption
- **Common fix (POODLE)**: get rid of SSLv3

RC4 statistical biases

- **Attack name**: RC4 biases (no real name)
- **Authors**: AlFardan et al. (practical attack)
- **Date**: 2014
- **Hypotheses and prerequisites**:
  - RC4 is used to encrypt data
  - the ciphertext is observable
  - *the same secret is repeated in different connections*
- **Ideal fix**: ban RC4
- **Common fix**: ban RC4 when possible