

CSE 484/M584: Computer Security (and Privacy)

Spring 2025

David Kohlbrenner
dkohlbre@cs

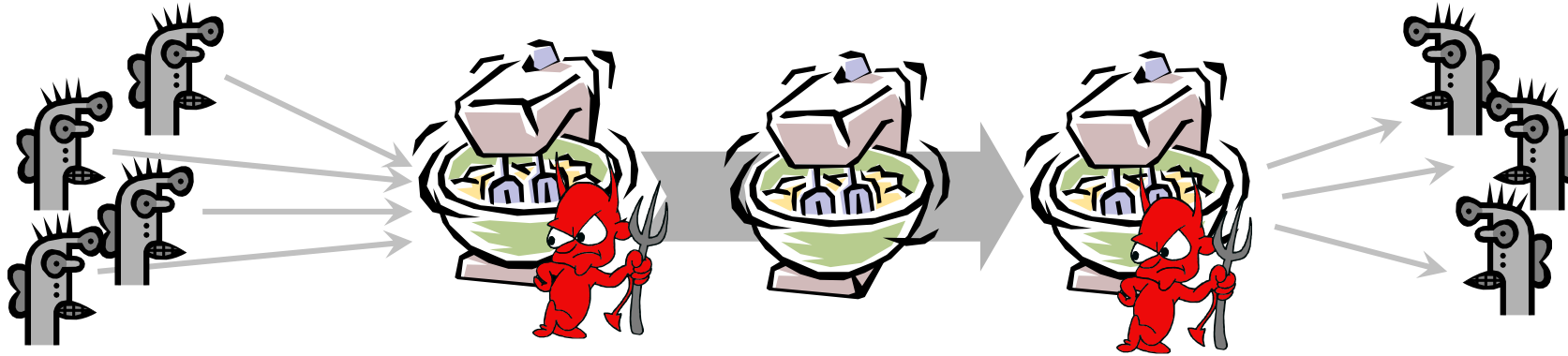
UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner, Nirvan Tyagi. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials

Admin

- Homework 2 due today
- Lab 4 next goes out tonight (or tomorrow morning)
 - Critical to start immediately on. Really.
 - Section with be very helpful for getting started. Attend.

Part 2: Anonymity in Communication

Mix Cascades and Mixnets

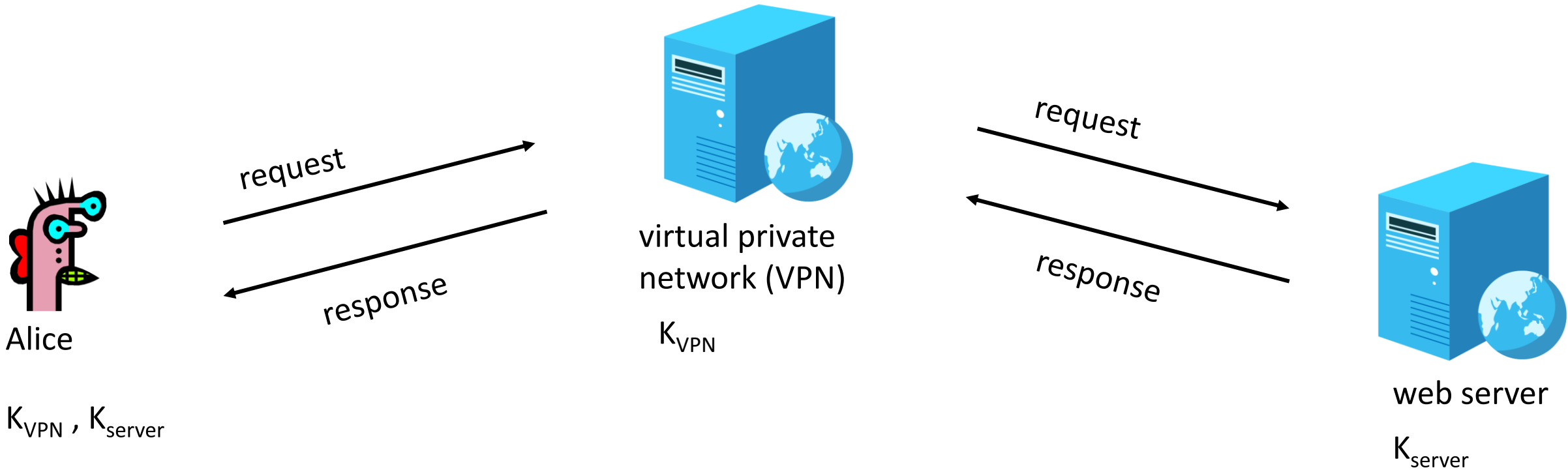


- Messages are sent through a **sequence of mixes**
 - Can also form an arbitrary network of mixes (“mixnet”)
- Some of the mixes may be controlled by attacker, but even a single good mix ensures anonymity
- Pad and buffer traffic to foil **correlation attacks**

Disadvantages of Basic Mixnets

- Public-key encryption and decryption at each mix are **computationally expensive**
- Basic mixnets have **high latency**
 - OK for email, not OK for anonymous Web browsing
- Challenge: **low-latency anonymity network**

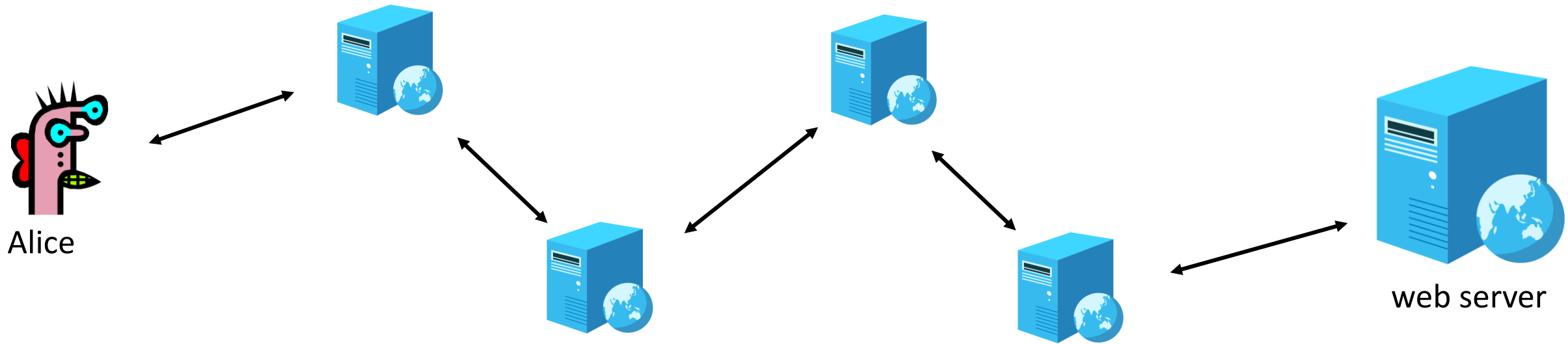
Using a Proxy Server



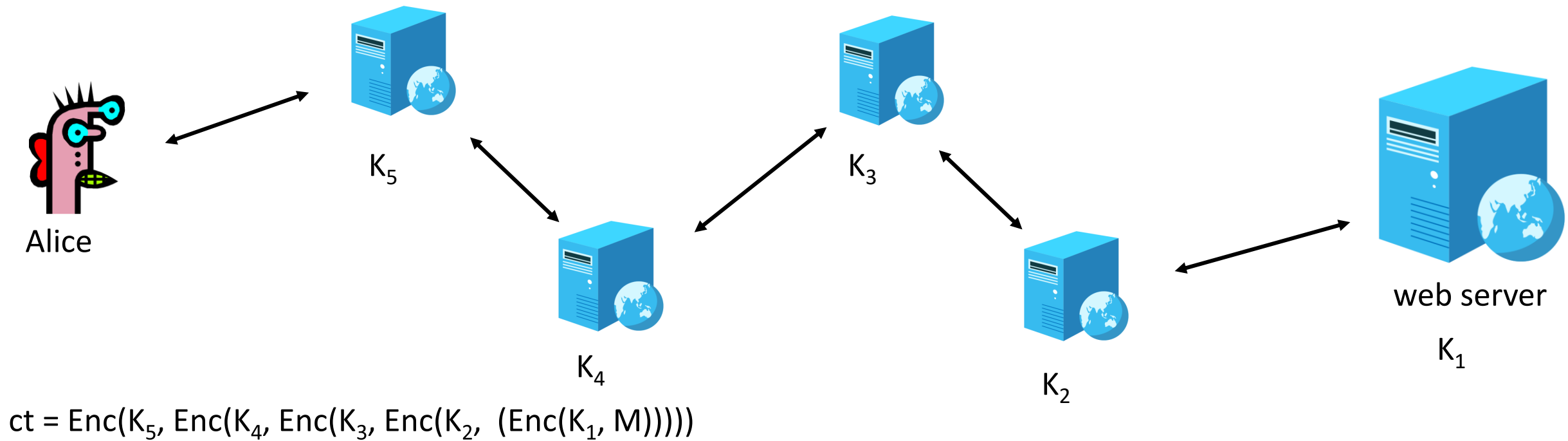
$$ct_{server} = \text{Enc}(K_{server}, M)$$

$$ct_{VPN} = \text{Enc}(K_{VPN}, ct_{server})$$

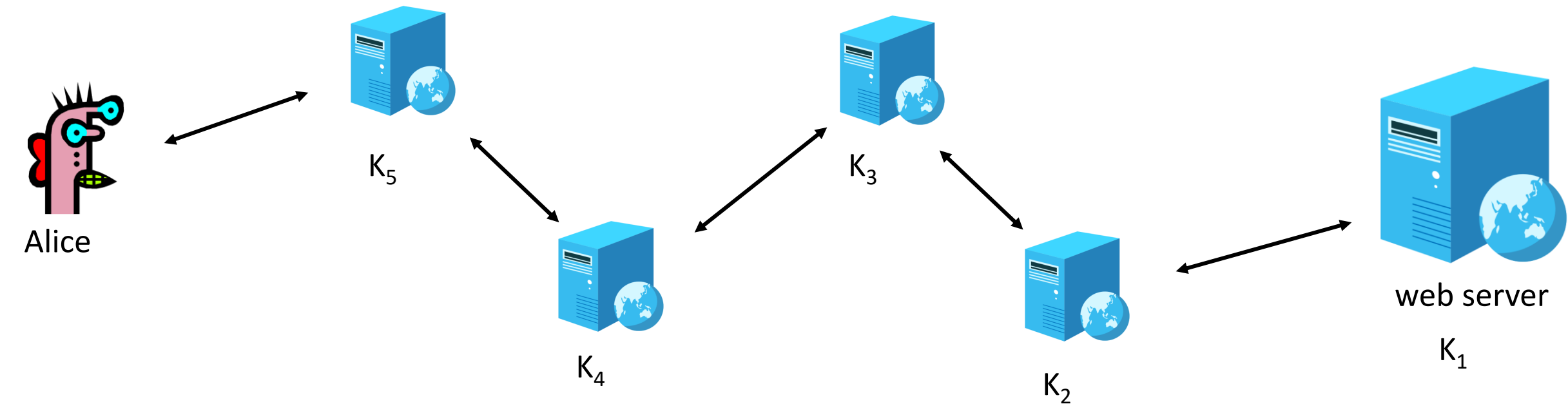
Onion Routing



Onion Routing



Onion Routing



$ct = \text{Enc}(K_5, \text{Enc}(K_4, \text{Enc}(K_3, \text{Enc}(K_2, (\text{Enc}(K_1, M))))))$

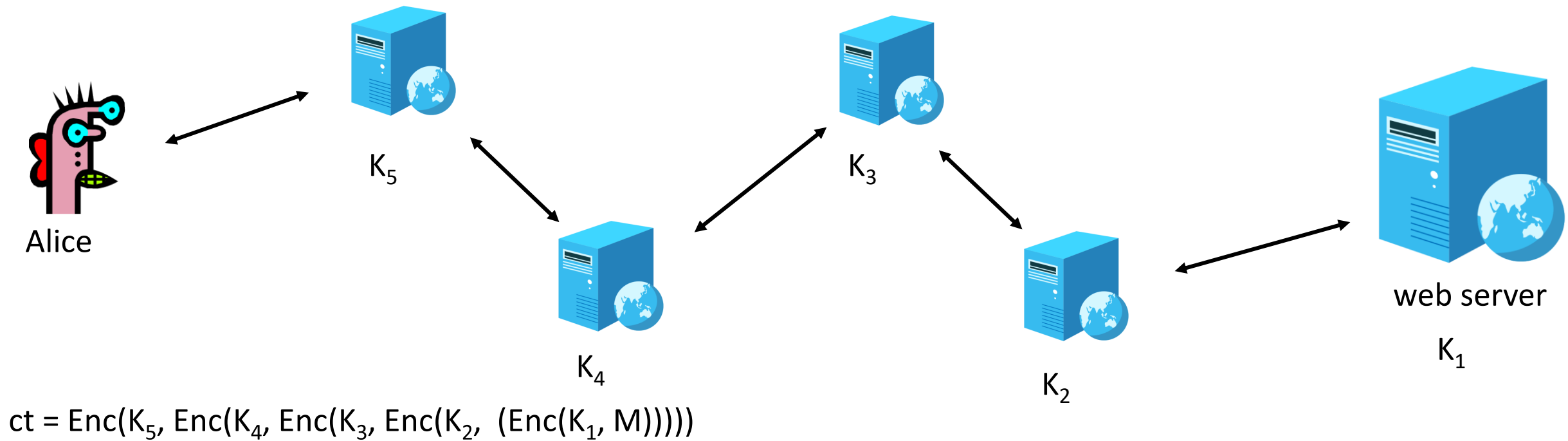
Gradescope:

Q: If K₄ is compromised, what does it learn about Alice's communications?

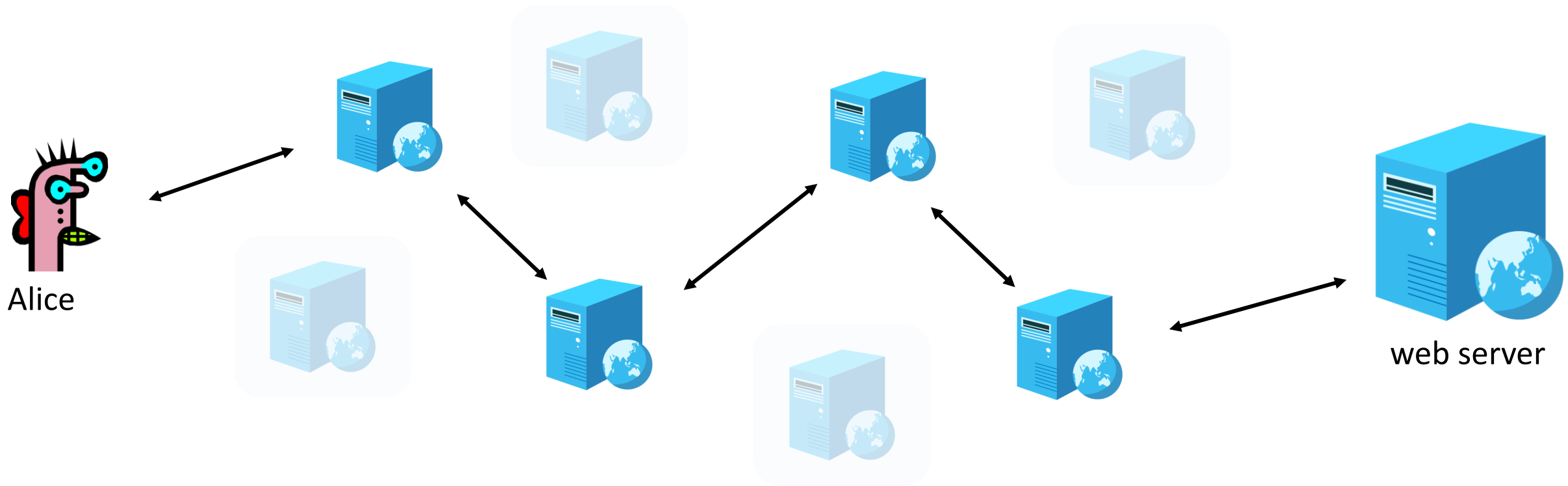
Q: If K₅ is compromised, what does it learn about Alice's communications?

Q: If K₂ is compromised, what does it learn about Alice's communications?

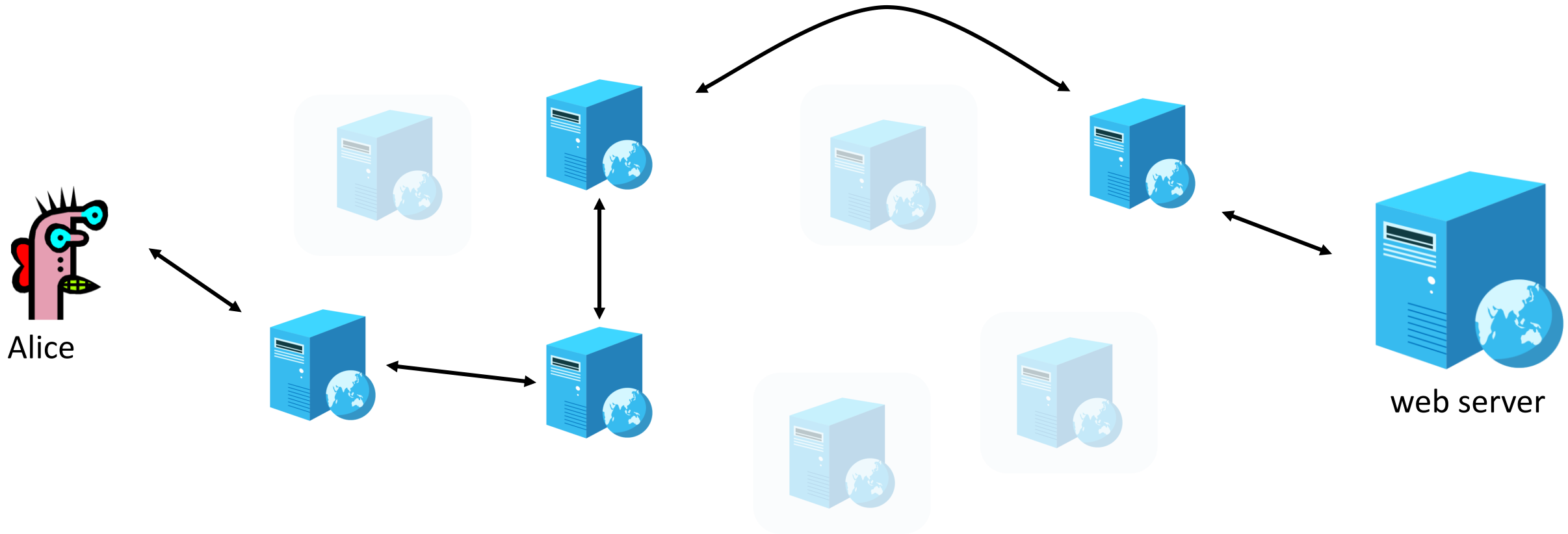
Onion Routing



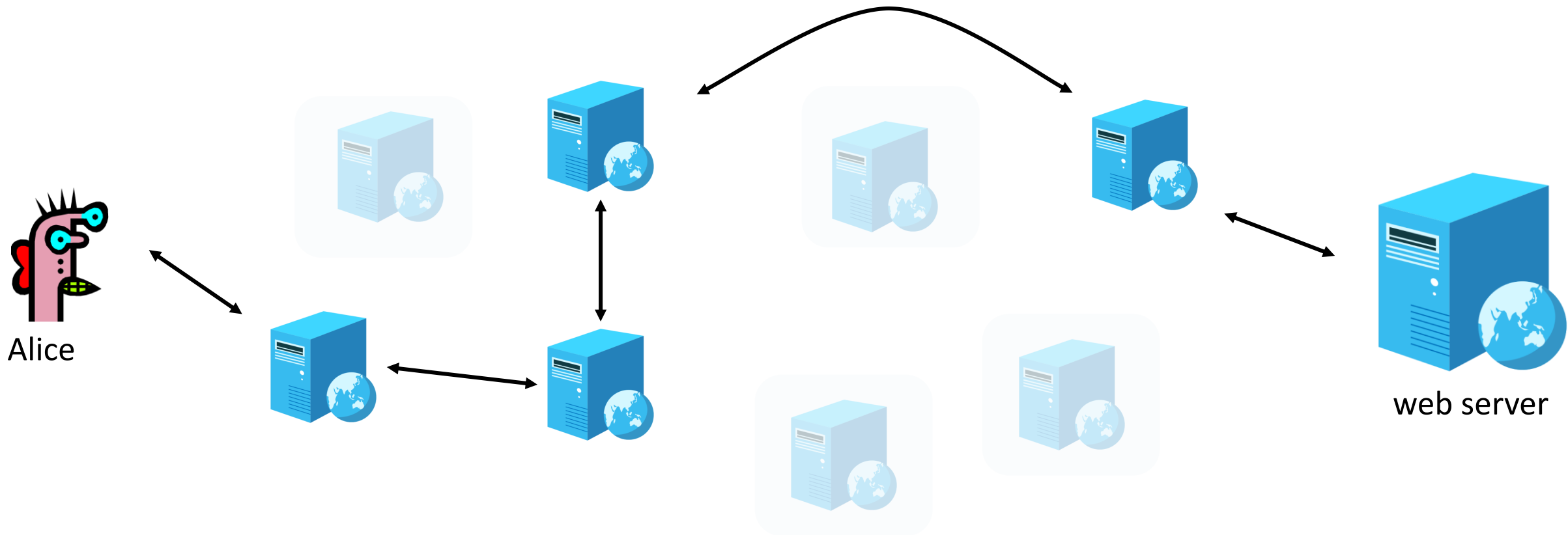
Tor: Randomized Onion Routing



Tor: Randomized Onion Routing

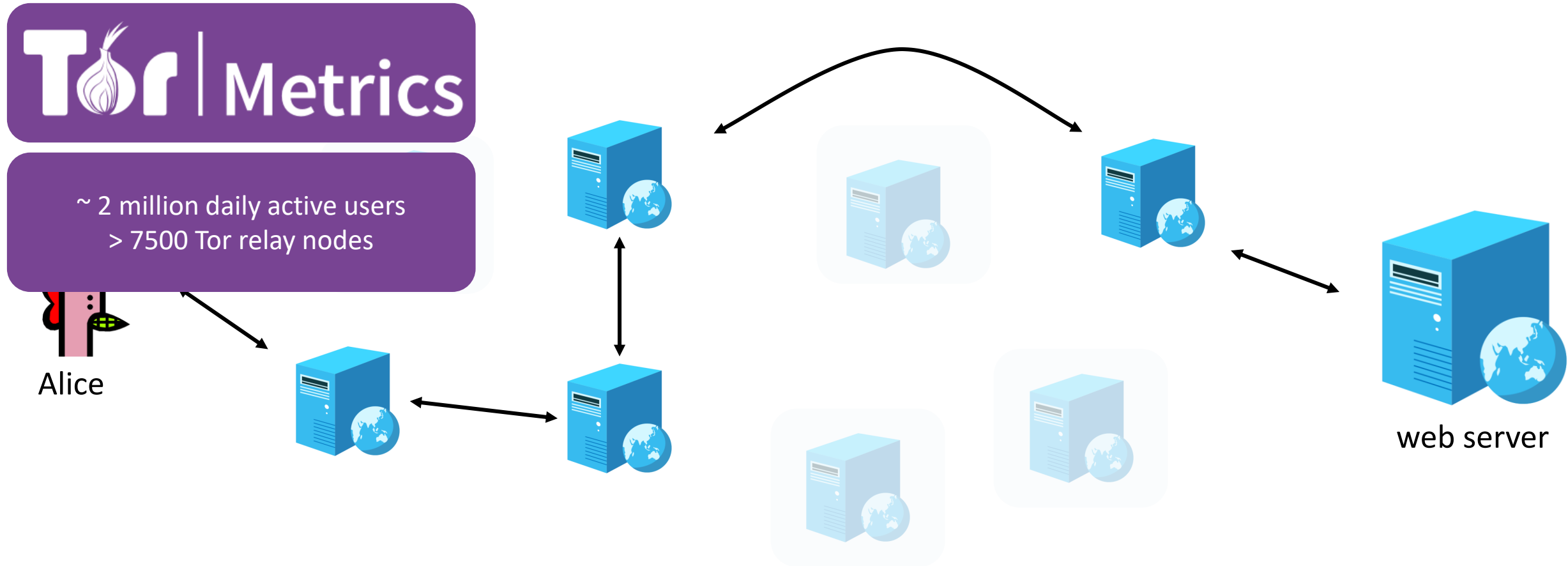


Tor: Randomized Onion Routing



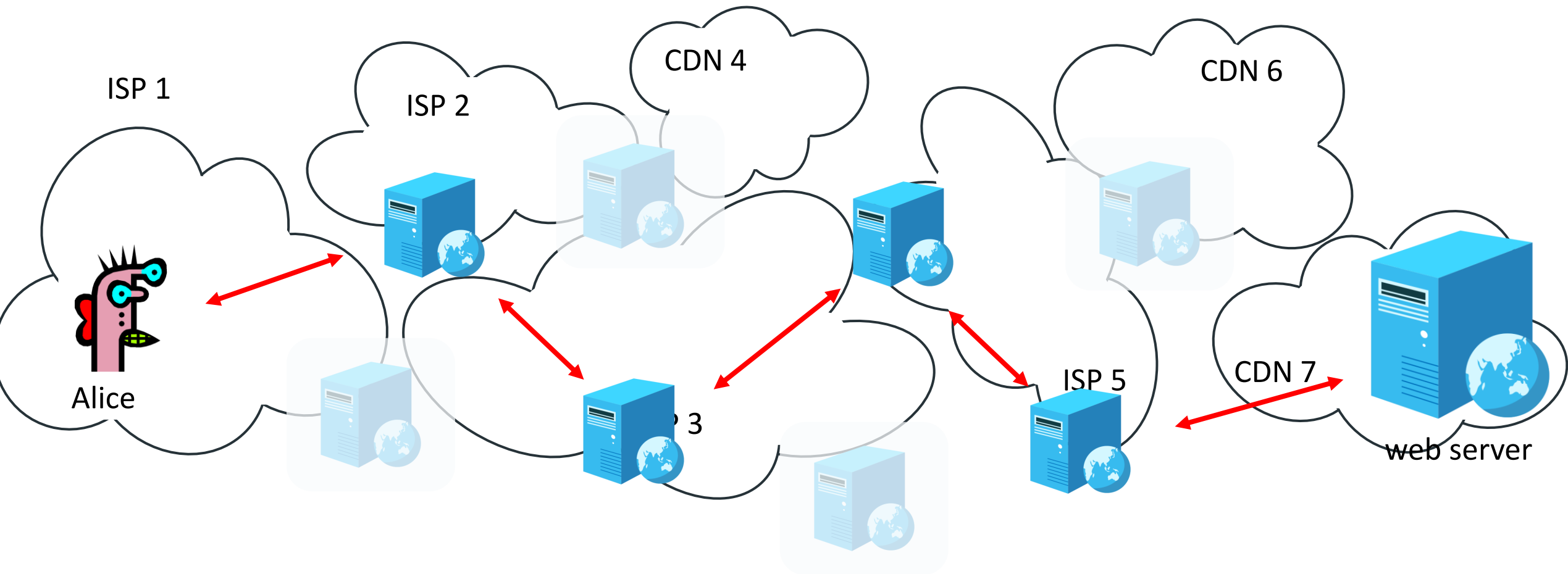
- Random paths prevent a single Tor exit node from being associated with a user's browsing behavior

Tor: Randomized Onion Routing

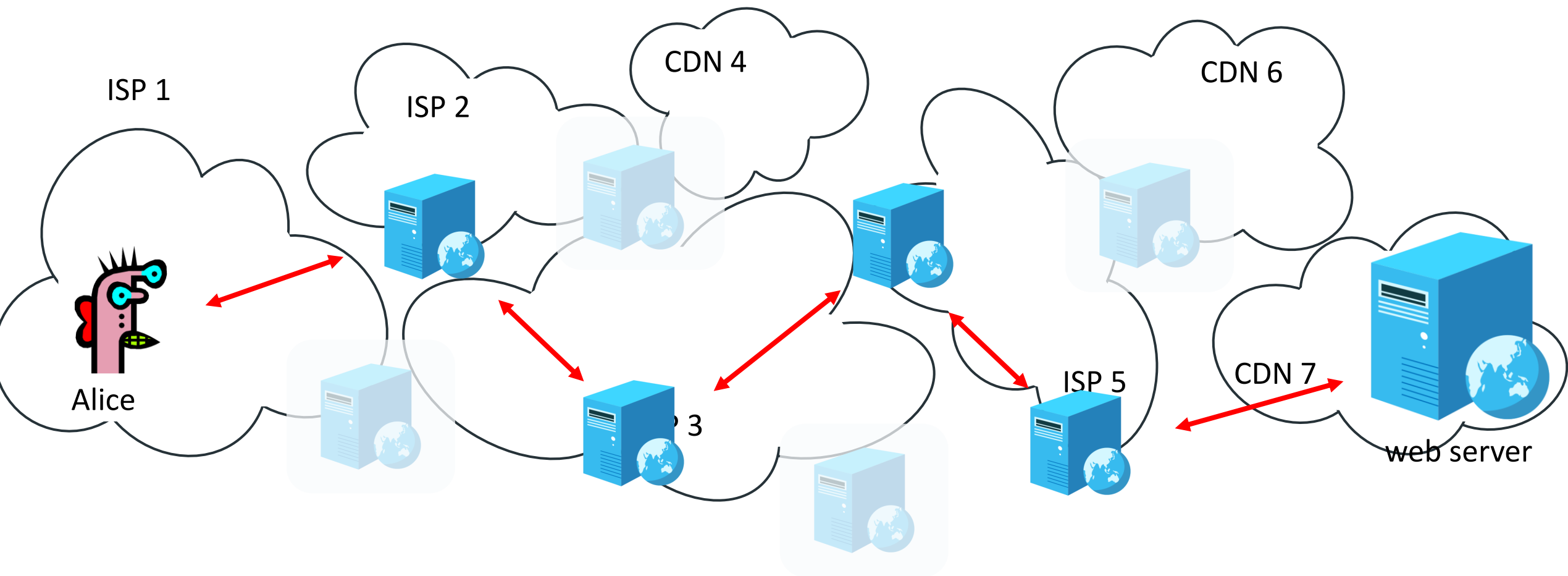


- Random paths prevent a single Tor exit node from being associated with a user's browsing behavior

Network Observing Adversaries

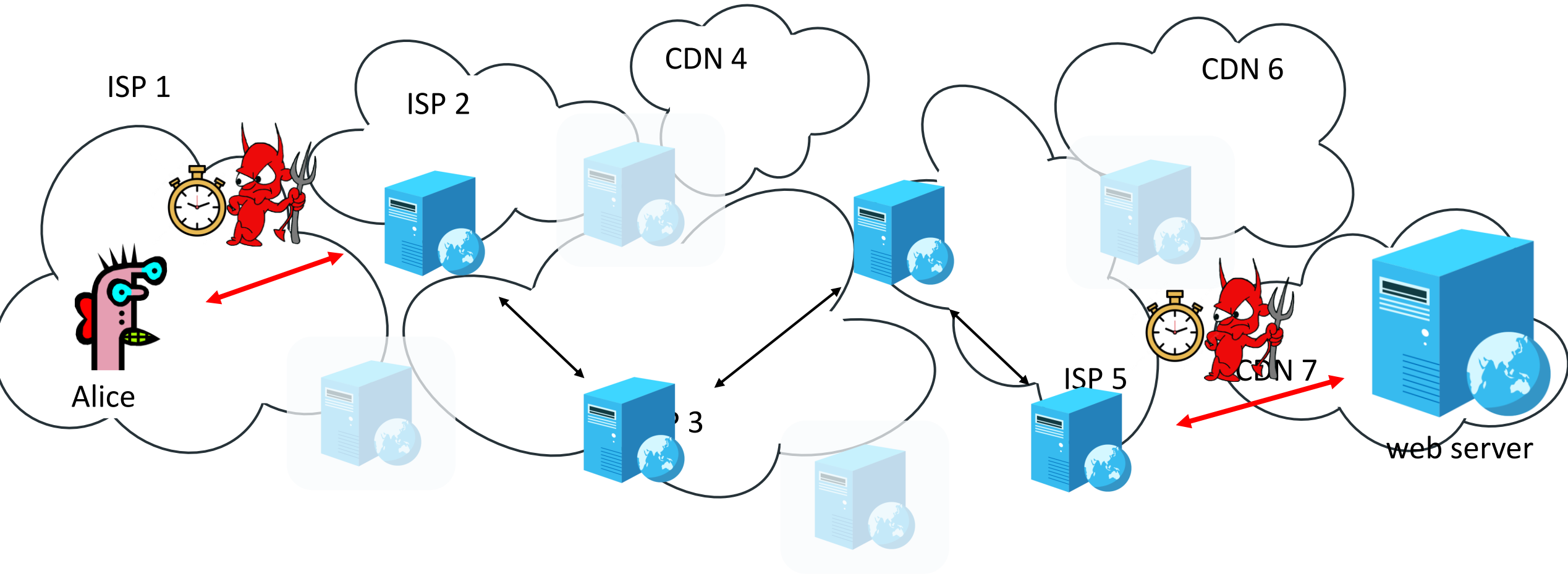


Network Observing Adversaries



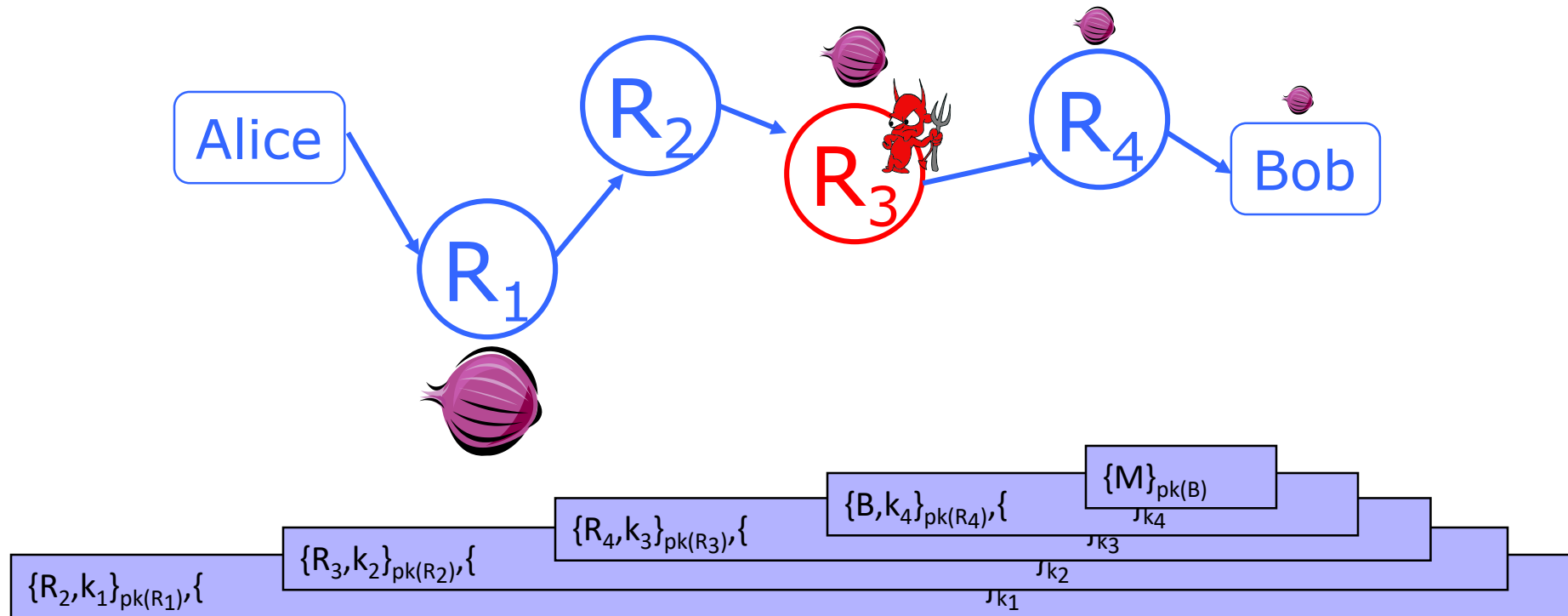
- Adversary that observes network can trivially trace user requests to server

Timing Attacks Against Tor



- Adversary that observes network can trivially trace user requests to server
- Adversary that observes part of network can also perform timing attacks to infer requests

Onion Routing



- Routing info for each link encrypted with router's public key
- Each router learns only the identity of the next router

Issues and Notes of Caution

- The simple act of using Tor could make one a **target for additional surveillance**
- Hosting an exit node could result in **illegal activity coming from your machine**

RCA Lab

Security Engineering

- At some point, the vendor finds out about the bug
 - Publicly or privately revealed by finder
 - Internally found by code auditing
 - Found being used in-the-wild

Upon Receiving a Vulnerability Report

- Suppose you work on the security team at a company
- You receive a report of a vulnerability in the wild, including a working proof-of-concept exploit

Root-Cause Analysis (RCA)

- **Consider:**
 - What is different between a “normal” interaction and the exploit?
 - What part(s) of the program are relevant to that interaction?
 - Add logging/debugging here! (But consider that it might affect the exploit...)
 - Develop theories about what is happening
 - Test your theories!

The Goals for RCAs

- Identify what the exploit accomplishes
- Identify the major steps the exploit takes

The Goals for RCAs

- Identify what the exploit accomplishes
- Identify the major steps the exploit takes
- Find the specific code components (if any exist) that are responsible
 - Aka: the vulnerability
 - Consider that an exploit might leverage *missing* features!

The Goals for RCAs

- Identify what the exploit accomplishes
- Identify the major steps the exploit takes
- Find the specific code components (if any exist) that are responsible
 - Aka: the vulnerability
 - Consider that an exploit might leverage *missing* features!
- Find “nearby” bugs
 - i.e., if you fix the most-responsible line of code, is it still vulnerable?

The Goals for RCAs

- Identify what the exploit accomplishes
- Identify the major steps the exploit takes
- Find the specific code components (if any exist) that are responsible
 - Aka: the vulnerability
 - Consider that an exploit might leverage *missing* features!
- Find “nearby” bugs
 - i.e., if you fix the most-responsible line of code, is it still vulnerable?
- Plan out a patch
 - Minimize breaking explicit features of the program
 - Minimize breaking implicit features of the program

Terminology

- “Zero Days” – 0 days (aka “o-days”)
 - Refers to a bug that is made publicly known at the same time as the vendor is told
 - The vendor has had ‘0 days’ of lead time to fix it

Terminology

- “Zero Days” – 0 days (aka “o-days”)
 - Refers to a bug that is made publicly known at the same time as the vendor is told
 - The vendor has had ‘0 days’ of lead time to fix it
- CVE Number
 - Common Vulnerabilities and Exposures
 - E.g. CVE-2022-4135

Terminology

- “Zero Days” – 0 days (aka “o-days”)
 - Refers to a bug that is made publicly known at the same time as the vendor is told
 - The vendor has had ‘0 days’ of lead time to fix it
- CVE Number
 - Common Vulnerabilities and Exposures
 - E.g. CVE-2022-4135
- CWE
 - Common Weakness Enumeration
 - Standardized list of bug types

Terminology

- “Zero Days” – 0 days (aka “o-days”)
 - Refers to a bug that is made publicly known at the same time as the vendor is told
 - The vendor has had ‘0 days’ of lead time to fix it
- CVE Number
 - Common Vulnerabilities and Exposures
 - E.g. CVE-2022-4135
- CWE
 - Common Weakness Enumeration
 - Standardized list of bug types
- CVSS
 - Common Vulnerability Scoring System
 - Very limited utility, scores barely correlated with impact

Project 0 (p0) Root Cause Analyses (RCAs)

- Google Project Zero (aka p0) is the premiere publicly-disclosing bug hunting team
- They produce detailed writeups of many bugs, and work with Google's Threat Analysis Group (aka TAG) to produce RCAs of in-the-wild bugs.
- You should go read some p0 RCAs!
<https://googleprojectzero.github.io/0days-in-the-wild/rca.html>

Sample RCA

CVE-2021-26411: Internet Explorer MSHTML Double-Free

Maddie Stone

The Basics

Disclosure or Patch Date: 9 March 2021

Product: Microsoft Internet Explorer

Advisory: <https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-26411>

Affected Versions: [KB4601319](#) and previous

First Patched Version: [KB5000802](#)

Issue/Bug Report: N/A

Patch CL: N/A

Bug-Introducing CL: N/A

Reporter(s): yangkang(@dnpushme) & huangyi(@C0rk1_H) & Enki

Sample RCA

The Code

Proof-of-concept:

```
<!-- saved from url=(0014)about:internet -->
<script>

String.prototype.repeat = function (size) { return new Array(size + 1).join(this); }

var ele = document.createElement('element')
var attr1 = document.createAttribute('attribute')
attr1.nodeValue = {
  valueOf: function() {
    alert('callback')
    alert(ele.attributes.length)
    ele.clearAttributes()
    alert(ele.attributes.length)
  }
}

ele.setAttributeNode(attr1)
ele.setAttribute('attr2', 'AAAAAAA')
ele.removeAttributeNode(attr1)

</script>
```

Exploit sample: https://enki.co.kr/blog/2021/02/04/ie_0day.html

Did you have access to the exploit sample when doing the analysis? Yes

Sample RCA

The Vulnerability

Bug class: Use-after-free

Vulnerability details:

A value of an attribute is able to be freed twice.

When calling `removeAttributeNode`, `mshtml!CElement::ie9_removeAttributeNodeInternal` is called. `ie9_removeAttributeNodeInternal` first finds the 2 indices for the node entry in the attribute array for the element. (Attribute nodes have 2 entries in the attribute array whereas non-node attributes only have one.) The use-after-free occurs because there is a user-controlled callback between the calculating the indices and when they are used. The backing store buffer can be changed during this callback and the code doesn't verify that the index is still valid.

In the case of the above crash POC, the backing store buffer of the array is cleared using `clearAttributes` in the callback. `mshtml!CElement::clearAttributes` clears the attributes backing array by deleting/freeing the first element in the array and memmoving the following entries into that space. This means that when `clearAttributes` is finished and the code path returns to `removeAttributeNodeInternal`, we have now filled the freed space with whatever attribute was at the end of the attribute array and that attribute has also been freed. This can be a pointer to the user-controlled string object. In this case the "use" is a double free on the String object that holds the value for the second attribute added to the element.

Sample RCA

The Exploit

(The terms *exploit primitive*, *exploit strategy*, *exploit technique*, and *exploit flow* are [defined here](#).)

Exploit strategy (or strategies):

The double free is used to have two objects, one array of Dictionary items and one BSTR, allocated to the same block of memory on the heap. These two objects are able to be allocated to the same spot because a BSTR has the structure of: first 4 bytes is the length and the rest is the string data. The array of dictionary items is a VARIANT struct so it has the type in the first four bytes. The type is (VT_ARRAY | VT_VARIANT) = 0x200C. The BSTR also has a length of 0x200C, making them both valid objects.

The exploit uses this type confusion to leak the addresses of objects stored in the Dictionary by reading the bytes at the equivalent indices in the BSTR. It then uses the Dictionary to read and write to different memory values using a trick described in the "Exploitation, Part 1: From Arbitrary Write to Arbitrary Read" section of [this blog post](#) by Simon Zuckerbraun, including creating an ArrayBuffer whose address begins at 0x00000000 and its length is 0xFFFFFFFF, yielding an arbitrary read-write primitive.

The exploit bypasses Control Flow Guard (CFG) by overwriting the export address table entry in rpcrt4.dll for `__guard_check_icall_fptr` with `KiFastSystemCallRet`.

Exploit flow:

Two objects are allocated to the same block of memory thanks to the double free. This is used to first leak the address of objects using the VByteArray objects for dereferencing. Then the exploit overwrites the length and starting address of an ArrayBuffer to have an arbitrary read-write primitive from 0x00000000 - 0xFFFFFFFF.

Sample RCA

The Next Steps

Variant analysis

Areas/approach for variant analysis (and why):

- Audit anywhere that a callback occurs between where an index is calculated and when it's used.
- Fuzz HTML attributes, potentially with a tool like [Domato](#)

Found variants: N/A

Structural improvements

What are structural improvements such as ways to kill the bug class, prevent the introduction of this vulnerability, mitigate the exploit flow, make this type of vulnerability harder to exploit, etc.?

Ideas to kill the bug class:

- Verify the state of the objects being operated on after the callback. In this case it could be simplified to don't use indices that were calculated prior to a callback.
- Internet Explorer is now considered "legacy" software. Removing it from being accessible by default in the Windows operating system would reduce the attack surface.

Ideas to mitigate the exploit flow:

- Remove Internet Explorer from being enabled by default.

Lab 4 Learning Goals

- Combine lessons/skills from the quarter:
 - Identifying and understanding (and fixing) vulnerabilities
 - Debugging and execution tracing (e.g., with gdb)
 - Software and web security concepts
 - Clear technical communication
- You'll gain experience in:
 - **Root-causing** a security bug similar to ones seen in class
 - **Writing patches** for security bugs similar to ones seen in class
 - **Making sense of a moderate codebase** (~1500 or fewer loc)
- This lab is a more realistic depiction of what working in security industry on the defender side in “real life” might be like

tinyserv – a tiny, bad, HTTP server

- ~1500 lines of C code
- Moderately well commented
- Quite buggy
- You can interact with it via command line tools or a web browser

tinyserv – a tiny, bad, HTTP server

- ~1500 lines of C code
- Moderately well commented
- Quite buggy
- You can interact with it via command line tools or a web browser
- 4 exploits
 - Exploit 1: Given RCA, create patch (Part A)
 - Exploit 2: TAs will cover in Section!
 - Exploit 3 & 4: Given exploit, create RCA (Part B), create patch (Part C)

Lab 4 Components

- Part A
 - We will give you the RCA for an exploit, and **you have to write the patch**
- Part B
 - We will give you an exploit, and **you have to write the RCA**
 - (You can choose one of two exploits. You may do the other for extra credit.)
- Part C
 - **You have to write the patch** for Part B's exploit

Major Features

- “admin” login
 - Sets a randomized password on server start
 - Successful login sets a cookie that lets admins access admin.txt
 - admin.txt contains a log of requests received so far
 - (Our exploits work by demonstrating they can access admin.txt)
- Dynamic content fills
 - Some pages have dynamic content (notably 404s) that gets filled at request
- Response caching
 - Pages are cached in a hashtable on first send
 - Future responses will check the hashtable first

RCA Strategies

- Read through the server code (see tinyserv.c to start)
 - You don't have to understand everything!
- Read through the exploit inputs and try to guess which parts of the tinyserv code might be related; start debugging there!
- Use gdb for debugging and execution tracing
 - `gdb --args ./tinyserv ./files`
 - `break [function name or line number]`
 - `run`
 - From another terminal window, you can now run the exploits
- (Maybe:) Modify tinyserv.c to test things or add print statements