CIS 551 / TCOM 401

Computer and Network Security

Spring 2009

Lecture 21

Announcements

- Plan for Today:
  - Human Authentication
  - Anonymity

- Project 4 is due 28 April 2009 at 11:59 pm
  - Available on the web

- Final exam has been scheduled:
  Friday, May 8, 2009
  9:00am – 11:00am, Moore 216

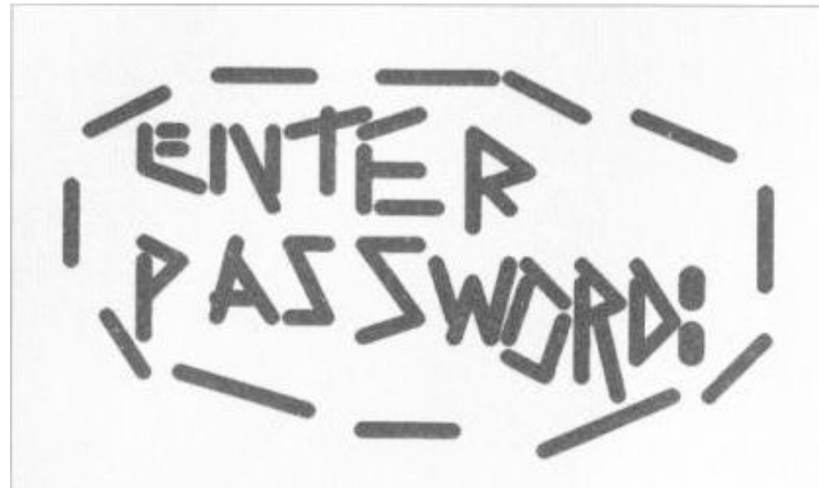
Identifying a particular human

- Human Authentication is based on one or more of the following:
- Something you know
  - password
- Something you have
  - driver's license, Penn Card
- Something inherent about you
  - Biometrics, location

Passwords

- Shared code/phrase
- Client sends to authenticate

- Simple, right?
- How do you…
  - Establish them to begin with?
  - Stop them from leaking?
  - Stop them from being guessed?



SOURCE: NASA

Prime Mover Problem

- Out of band
  - Physical mail
  - Email
  - Attached to the box
- Piggybacking
  - Swipe Penn Card to make PennKey
  - But where does the chain stop?
    - Penn Card -> drivers license -> birth certificate

Leaks & Challenges

- Social engineering
- Managing large numbers of passwords:
  - Writing the password down on paper
  - Storing it in an electronic "safe"
  - Using a web browsers 'remember this password' feature
- Legal and responsibility
  - Shared password == shared liability

Guessing

- The "no such user" mistake
  - Gives an attacker information about usernames
- The "here's who we are" mistake
  - Gives an attacker information about who might have an account
- Common words, phrases for passwords
- Null passwords, "password", username, backwards, etc.
- Dictionary attacks

- How bad is it?

1979 Survey of 3,289 Passwords

- With no constraints on choice of password, Morris and Thompson got the following results:
  - 15 were a single ASCII letter.
  - 72 were strings of two ASCII letters.
  - 464 were strings of three ASCII letters.
  - 47 were strings of four alphanumerics.
  - 706 were five letters, all upper-case or all lower-case.
  - 605 were six letters, all lower case.

1990s Surveys of 15K Passwords

- Klein (1990) and Spafford (1992)
  - 2.7% guessed in 15 minutes
  - 21% in a week
  - Sounds ok? Not if the passwords last 30 days
- Tricks
  - Letter substitutions, words backwards, common names, patterns, etc.
  - Anything you can think of off the top of your head, a hacker can think of too
- Lazy users!
  - Weakest link is always the way of the attack

More Recent Password Surveys

- 2009:
  - ~33% of users have one password for all web sites
  - ~48% of users have multiple passwords
  - ~19% of users have unique password for each site

- 2005 survey by RSA Inc:
  - ~28% IT employees must remember > 13 passwords
  - ~30% IT employees have 6 – 12 passwords

- 2003: Users will give away their password for a cheap gift.

Heuristics for Guessing Attacks

- The dictionary with the words spelled backwards
- A list of first names (best obtained from some mailing list). Last names, street names, and city names also work well.
- The above with initial upper-case letters.
- All valid license plate numbers in your state. (About 5 hours work in 1979 for New Jersey.)
- Room numbers, social security numbers, telephone numbers, and the like.
- Sports teams, etc.

What makes a good password?

- Password Length
  - 64 bits of randomness is hard to crack
  - 64 bits is roughly 20 “common” ASCII characters
  - But… People can’t remember random strings
  - Longer not necessarily better: people write the passwords down
- Pass phrases
  - English Text has roughly 1.3 random bits/char.
  - Thus about 50 letters of English text
  - Hard to type without making mistakes!
- In practice
  - Non-dictionary, mixed case, mixed alphanumeric
  - Not too short (or too long) 8 - 12 characters
  - Tools that check password strength
    - http://www.microsoft.com/protect/yourself/password/checker.mspx
    - http://www.fastcrack.com/pwcheck.html

Hacks on plaintext password file

- Is the password file readable by the OS?
  - Then if I break the OS
- Can privileged users see the file?
  - … and make copies
- Is the file backed up somewhere
  - … insecure?
- Is the file/password in plaintext somewhere in memory?
  - Core dump
- Fool the user
  - A program that masquerades as the authentication program

Counter-hacks

- Control-Alt-Del for logging in
  - Establishes a "trusted path" in hardware
  - Prevents trojan horses from intercepting passwords
- Slow down / restrict number of tries
  - Make guessing take too long
  - e.g. 3 tries and you're blocked for 30 seconds
- Encrypt the password file
  - “Salt" - to prevent duplicates
  - Use one way hashes or encryptions on the passwords

Add Salt

- "Salt" the passwords by adding random bits.
  - Decreases the likelihood that two identical passwords will appear as identical entries in the password file.
- 12 bit salt results in 4,096 versions of each password.
- Unix: /etc/passwd entry:

user_id	$salt_u$	$Hash(salt_u + passwd_u)$	…

- Modern implementations use so-called *shadow* password files /etc/shadow that aren't world readable.

One Time Passwords

- Shared lists.
- Sequentially updated.
- One-time password sequences based on a one-way (hash) function.

- "Dongles"
  - Small devices that generate a sequence of random numbers from a secret seed.
  - Synchronized with the remote location when the dongle is assigned to a user
  - Often requires a pin or other password for local authentication
  - Can be stolen or lost!

Hash-based 1-time Passwords

- Alice identifies herself to verifier Bart using a well-known one-way hash function H.

- One-time setup.
  - Alice chooses a secret w.
  - Fixes a constant t for the number of times the authentication can be done.
  - Alice securely transfers $H^t(w)$ to Bart

$$\underbrace{H(H(H\ldots(H(w))}_{t \text{ times}}\ldots))$$

Hash-based 1-time Passwords

- Protocol actions. For session i, claimant A does the following to identify itself:
  - A computes w' = $H^{(t-i)}(w)$ and transmits the value to B.
  - B checks that i is the correct session (i.e. that the previous session was i-1) and checks to see if H(v) = w' where v was the last value provided by A (as part of session i-1).
  - B saves w' and i for use in the next session.

- It's hard to compute x from H(x).
  - Even though attacker gets to see $H^{(t-i)}(x)$, they can't guess then next message $H^{(t-(i+1))}(x)$.

One-time passwords: i^{th} authentication

Alice

$\{A, i, H^{(t-i)}(w)\}$ →

Bart

W
H(-)

$H^{(t-i+1)}(w)$,
H(-)

- Alice does the following to identify herself:
  - A computes w' = $H^{(t-i)}(w)$ and transmits the value to B.
  - B checks that i is the correct session (i.e.. that the previous session was i-1) and checks to see if H(w') = v where v was the last value provided by A (as part of session i-1).
  - B saves w' and i for use in the next session.

S/Key Passwords

- Hash-based one-time authentication used in practice
  - RFC 1760 / 2289
- Internally, S/Key uses 64 bit numbers
- For human use, each 64 bit number is mapped to 6 short words:
  - Example: "ROY HURT SKI FAIL GRIM KNEE"
- Should be used in conjunction with other encryption to prevent man-in-the-middle attacks

Biometrics

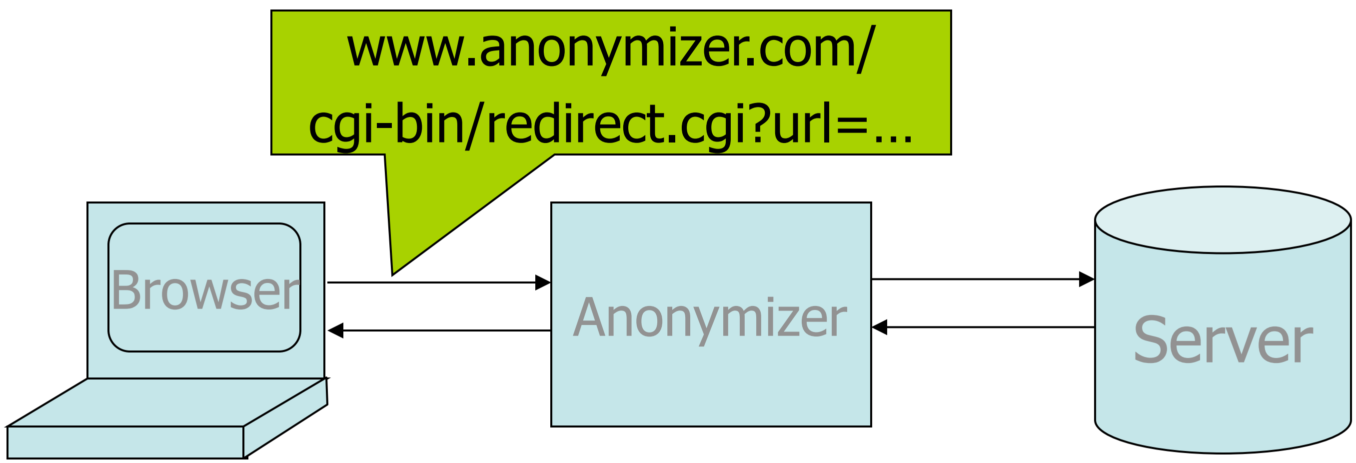
- Fingerprints:
  - Scanner gets geometry of identifiable features on the fingerprint
  - Used in laptops, some high-end PDAs
  - Requires clean hands
- Face recognition:
  - Identifies features like distance between eyes, nose width, etc. to generate a set of numbers
  - Can work even from a distance via a camera
- Retinal image:
  - Pattern of blood vessels at the back of the eye
  - Scanning takes ~15 seconds of looking into the scanner
  - Used in military and government installations
- Iris scan, voice analysis, signature, hand print

Anonymity?

- Sender anonymity:
  - The identity of the sender is hidden, while the receiver (and message) might not be
- Receiver anonymity:
  - The identity of the receiver is hidden (message and sender might not be)
- Unlinkability of sender and receiver:
  - Although the sender and receiver can be identified as participating in communication, they cannot be identified as communicating *with each other*.

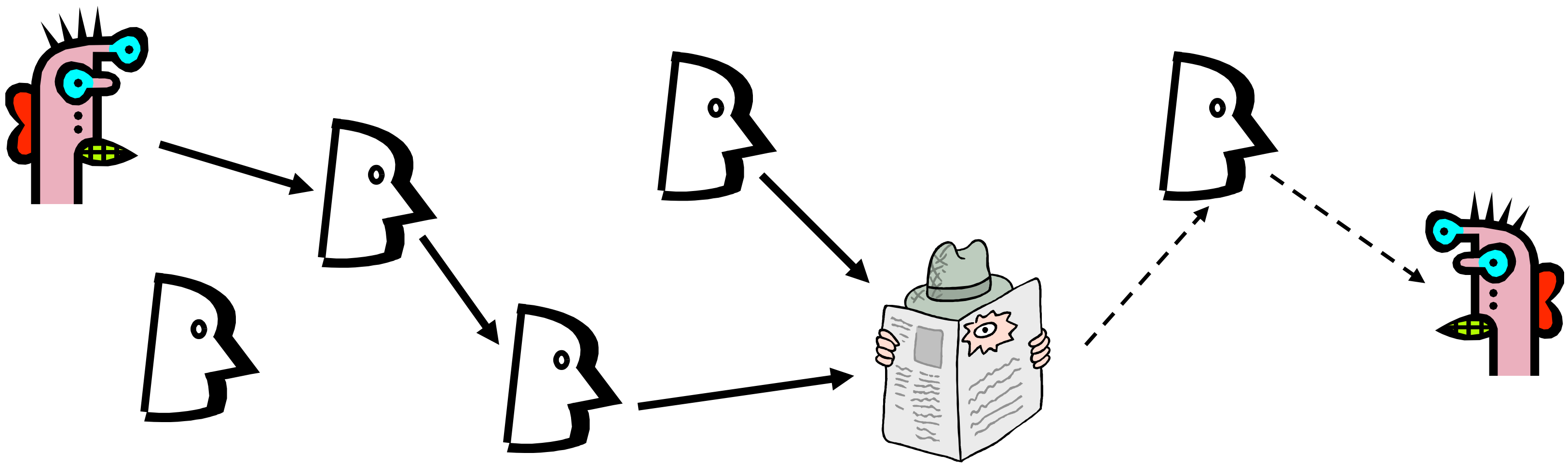
Browsing Anonymizers

- Anonymizer.com
- Web Anonymizer hides your IP address



- What does anonymizer.com know about you?

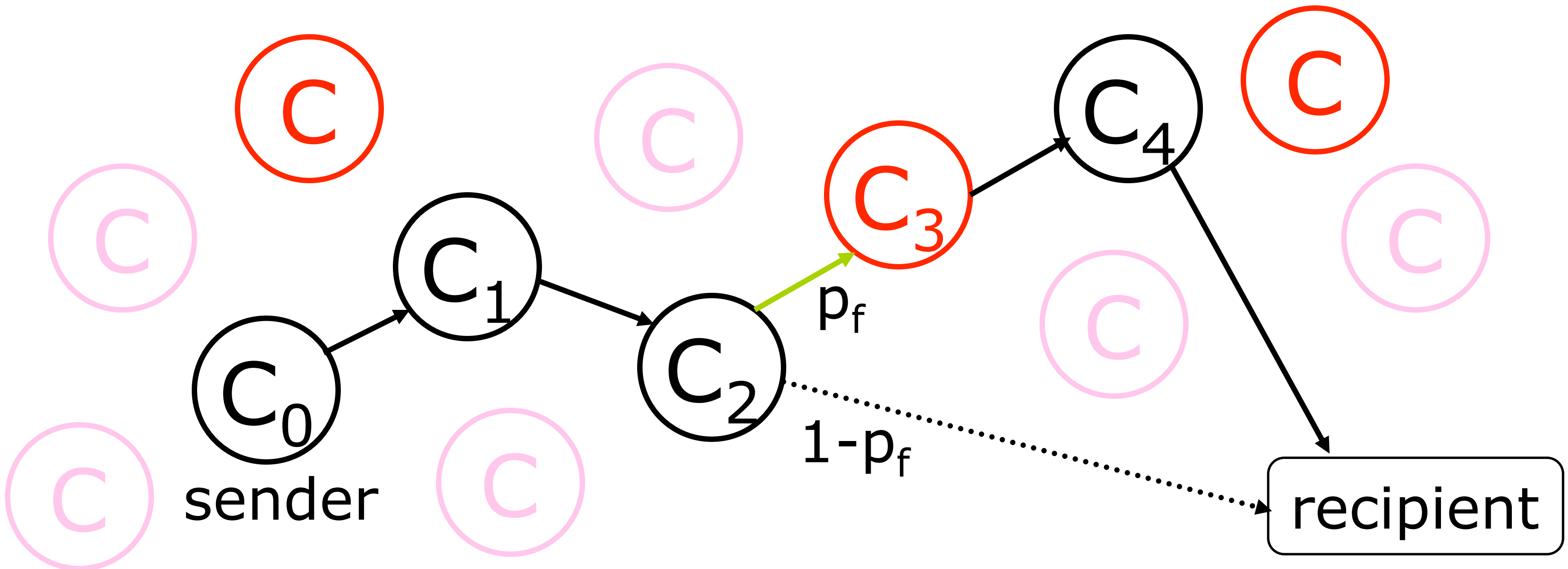
Related approach to anonymity



- Hide source of messages by routing them randomly
- Routers don't know for sure if the apparent source of the message is the actual sender or simply another router
  - Only secure against local attackers!
- Existing systems: Freenet, Crowds, etc.

Crowds

http://avirubin.com/crowds.pdf [Reiter,Rubin '98]



- Sender randomly chooses a path through the crowd
- Some routers are honest, some corrupt
- After receiving a message, honest router flips a coin
  - With probability P_f routes to the next member on the path
  - With probability $1- P_f$ sends directly to the recipient

What Does Anonymity Mean?

- Degree of anonymity:
  - Ranges from absolute privacy to provably exposed
- Beyond suspicion
  - The observed source of the message is no more likely to be the actual sender than anybody else
- Probable innocence
  - Probability <50% that the observed source of the message is the actual sender

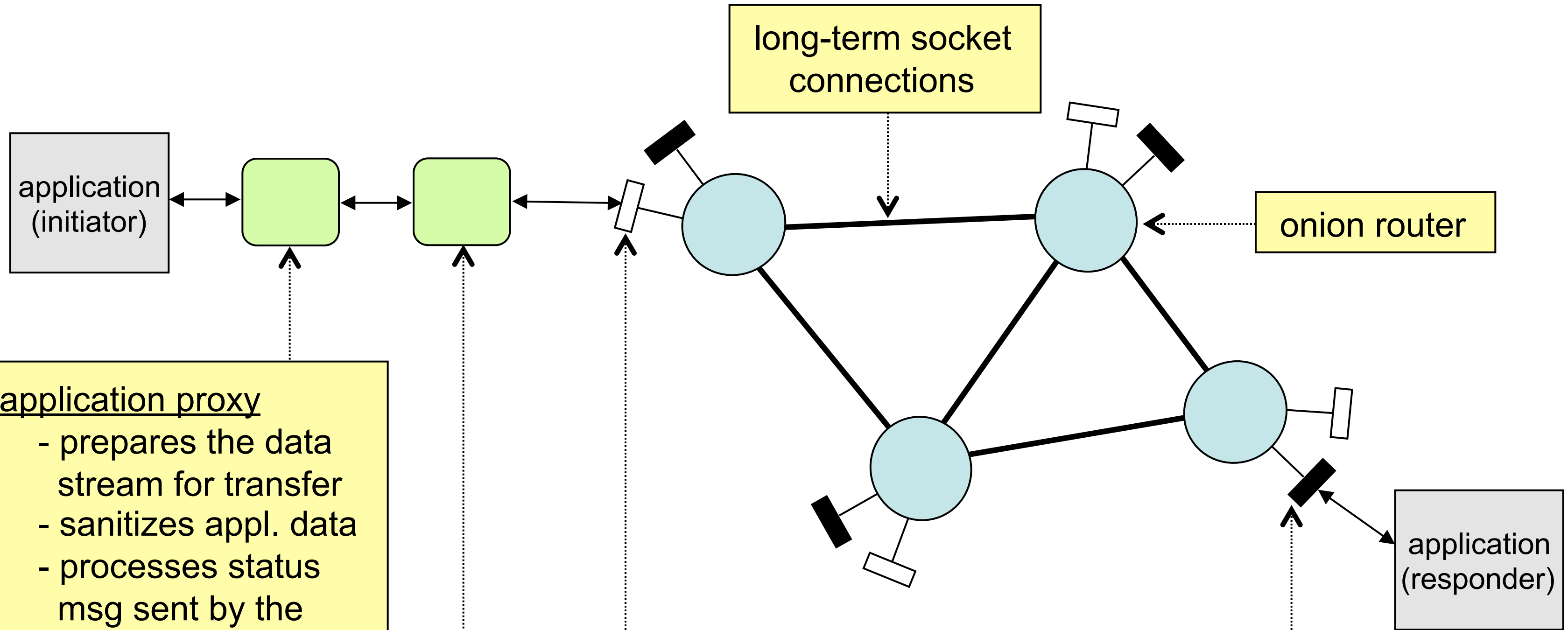
Guaranteed by Crowds if there are sufficiently few corrupt routers

- Possible innocence
  - Non-trivial probability that the observed source of the message is not the actual sender

A real-time MIX network – Onion routing

- general purpose infrastructure for anonymous communications over a public network (e.g., Internet)
- supports several types of applications (HTTP, FTP, SMTP, rlogin, telnet, …) through the use of application specific proxies
- operates over a (logical) network of onion routers
  - onion routers are real-time Chaum MIXes (messages are passed on nearly in real-time → this may limit mixing and weaken the protection!)
  - onion routers are under the control of different administrative domains → makes collusion less probable
- anonymous connections through onion routers are built dynamically to carry application data
- distributed, fault tolerant, and secure

Overview of OR architecture



<u>application proxy</u>
- prepares the data stream for transfer
- sanitizes appl. data
- processes status msg sent by the exit funnel

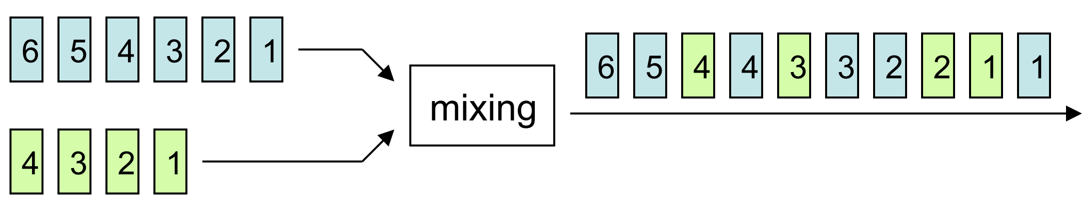
<u>onion proxy</u>
- opens the anonymous connection via the OR network
- encrypts/decrypts data

<u>entry funnel</u>
- multiplexes connections from onion proxies

<u>exit funnel</u>
- demultiplexes connections from the OR network
- opens connection to responder application and reports a one byte status msg back to the application proxy

OR network setup and operation

- long-term socket connections between "neighboring" onion routers are established → links
- neighbors on a link setup two DES keys using the Station-to-Station protocol (one key in each direction)
- several anonymous connections are multiplexed on a link
  - connections are identified by a connection ID (ACI)
  - an ACI is unique on a link, but not globally
- every message is fragmented into fixed size *cells* (48 bytes)
- cells are encrypted with DES in OFB mode (null IV)
  - optimization: if the payload of a cell is already encrypted (e.g., it carries (part of) an onion) then only the cell header is encrypted
- cells of different connections are mixed, but order of cells of each connection is preserved



Anonymous connection setup

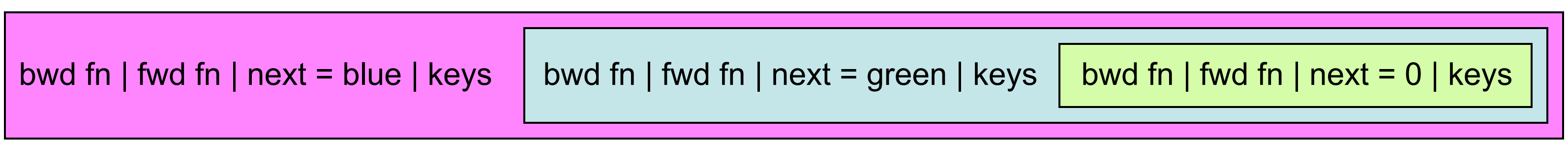
- the application is configured to connect to the application proxy instead of the real destination
- upon a new request, the application proxy
  - decides whether to accept the request
  - opens a socket connection to the onion proxy
  - passes a *standard structure* to the onion proxy
  - standard structure contains
    - application type (e.g., HTTP, FTP, SMTP, …)
    - retry count (number of times the exit funnel should retry connecting to the destination)
    - format of address that follows (e.g., NULL terminated ASCII string)
    - address of the destination (IP address and port number)
  - waits response from the exit funnel before sending application data

Anonymous connection setup (2)

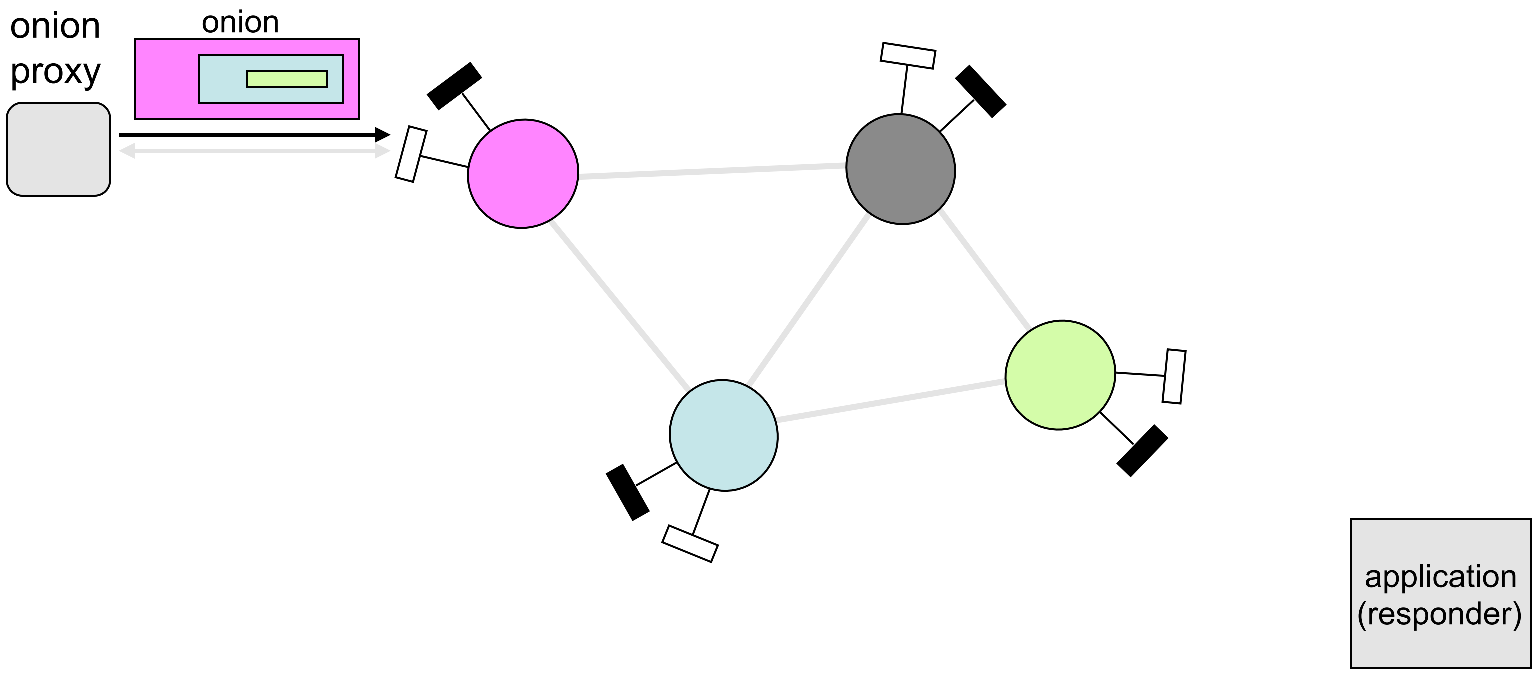
- upon reception of the standard structure, the onion proxy
  - decides whether to accept the request
  - establishes an anonymous connection through some randomly selected onion routers by constructing and passing along an *onion*
  - sends the standard structure to the exit funnel of the connection
  - after that, it relays data back and forth between the application proxy and the connection
- upon reception of the standard structure, the exit funnel
  - tries to open a socket connection to the destination
  - it sends back a one byte status message to the application proxy through the anonymous connection (in backward direction)
  - if the connection to the destination cannot be opened, then the anonymous connection is closed
  - otherwise, the application proxy starts sending application data through the onion proxy, entry funnel, anonymous connection, and exit funnel to the destination

Onions

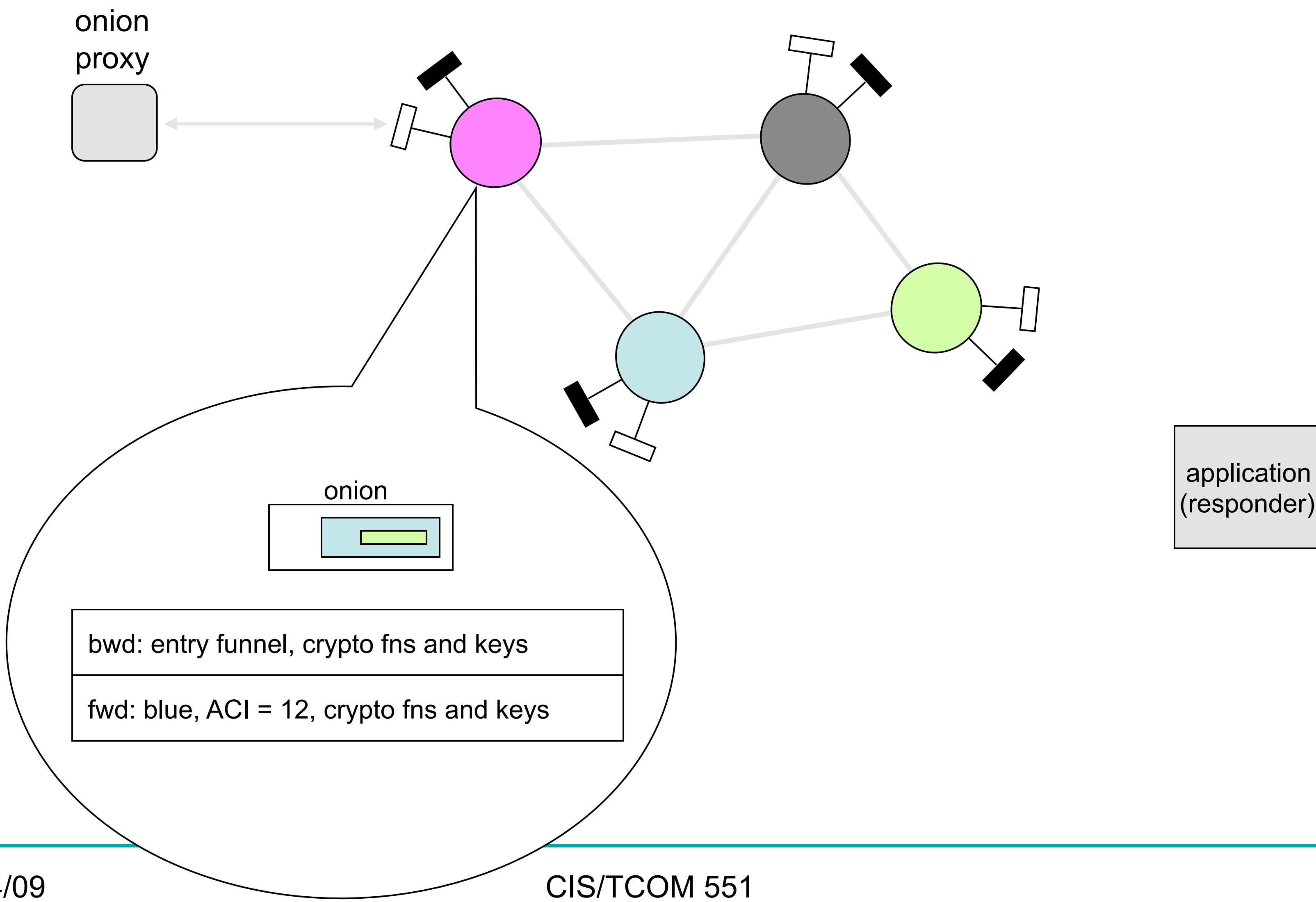
- an onion is a multi-layered data structure
- it encapsulates the route of the anonymous connection within the OR network
- each layer contains
  - backward crypto function (DES-OFB, RC4)
  - forward crypto function (DES-OFB, RC4)
  - IP address and port number of the next onion router
  - expiration time
  - key seed material
    - used to generate the keys for the backward and forward crypto functions
- each layer is encrypted with the public key of the onion router for which data in that layer is intended



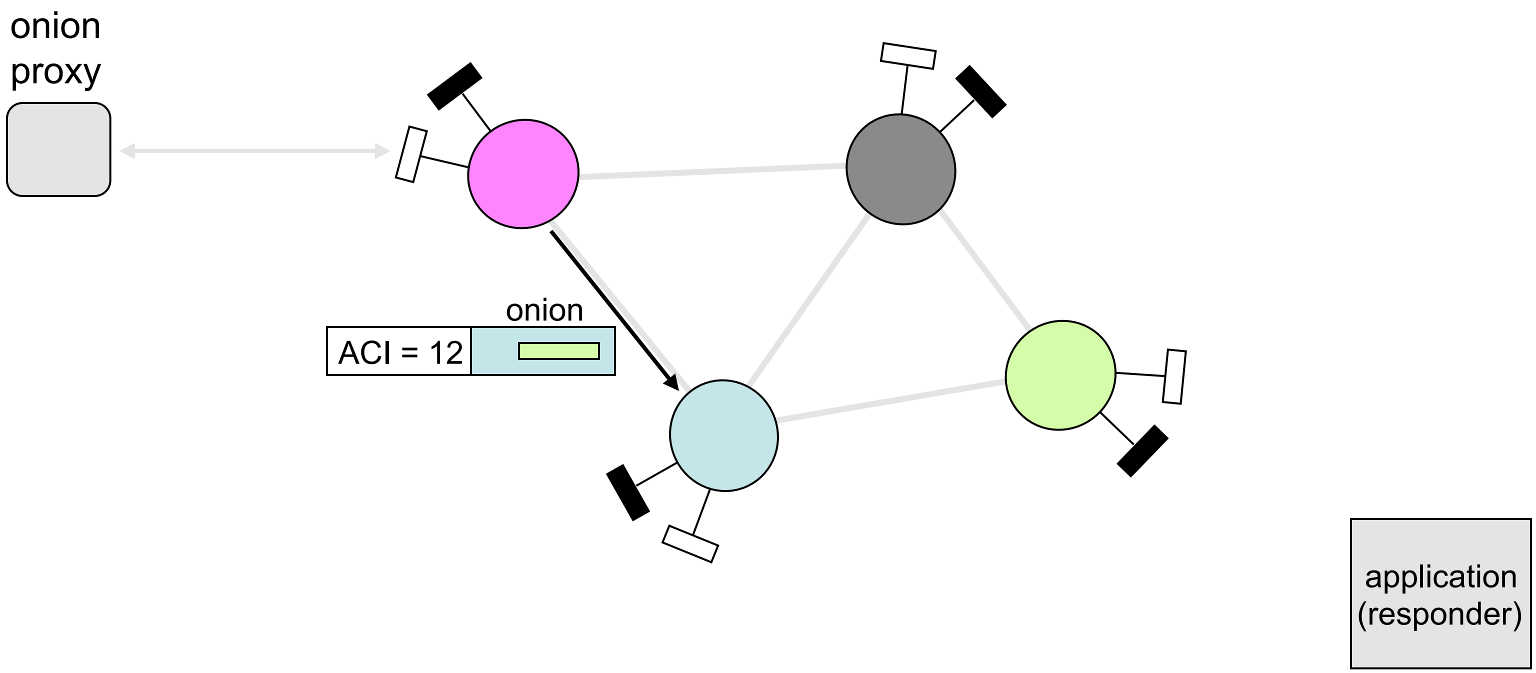
Anonymous connection setup



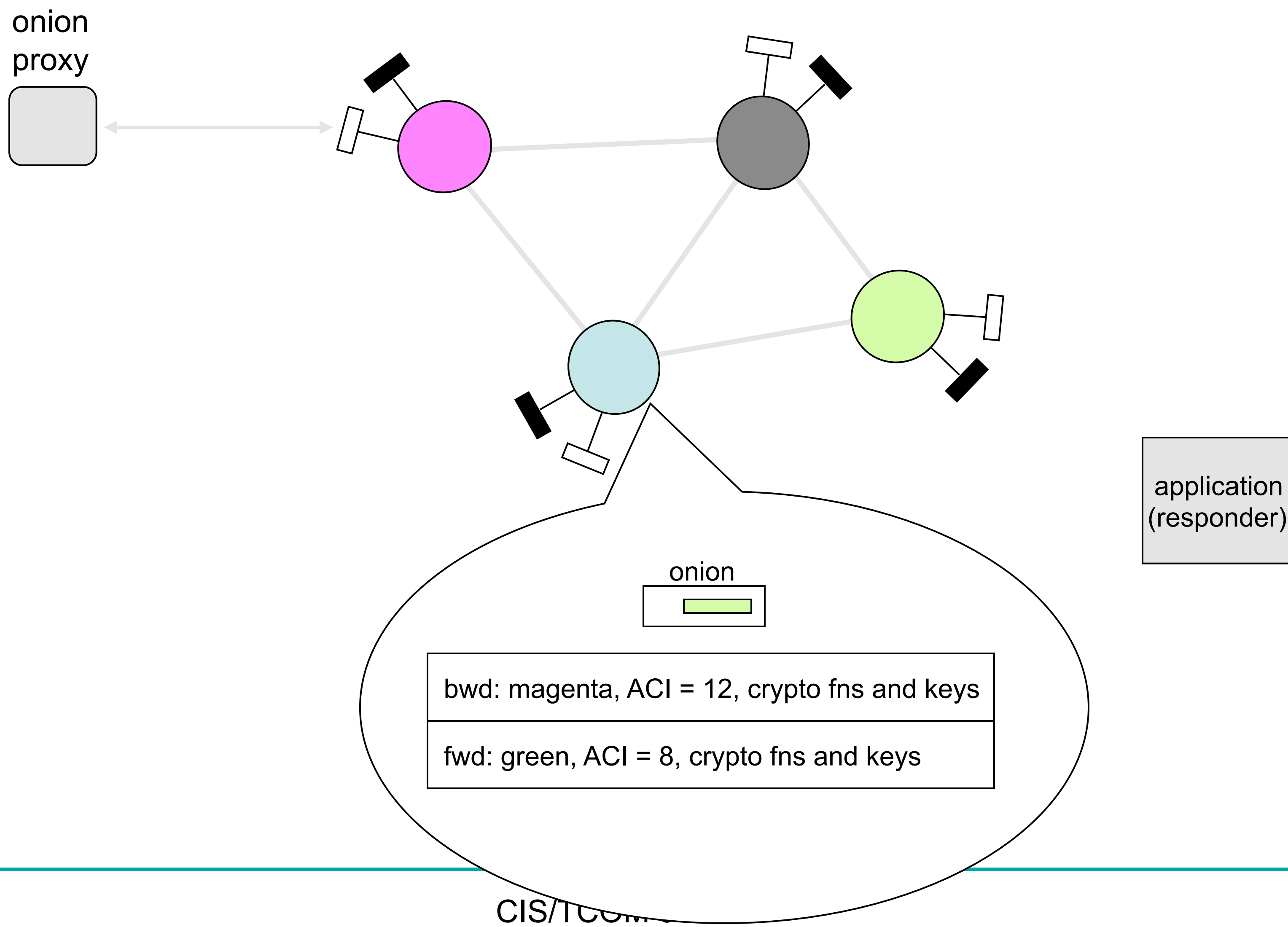
Anonymous connection setup



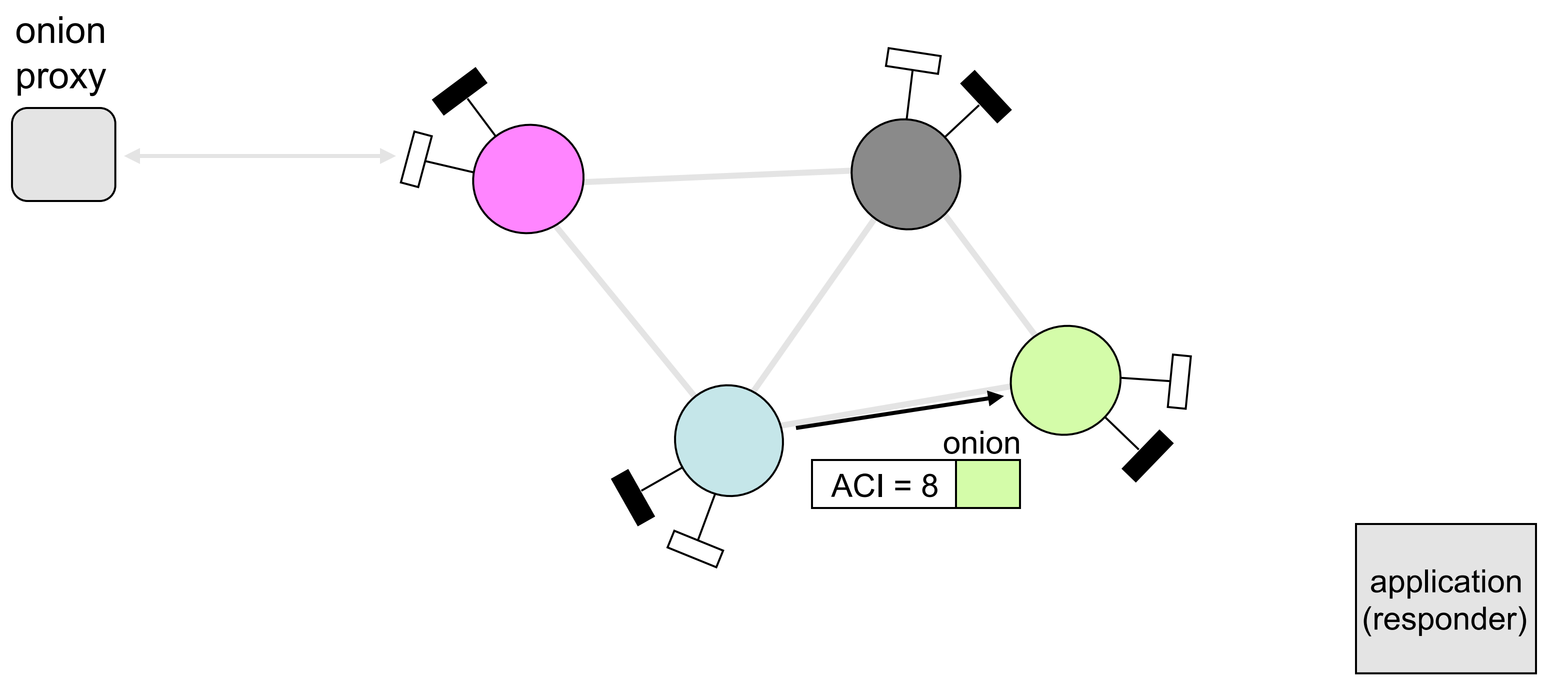
Anonymous connection setup



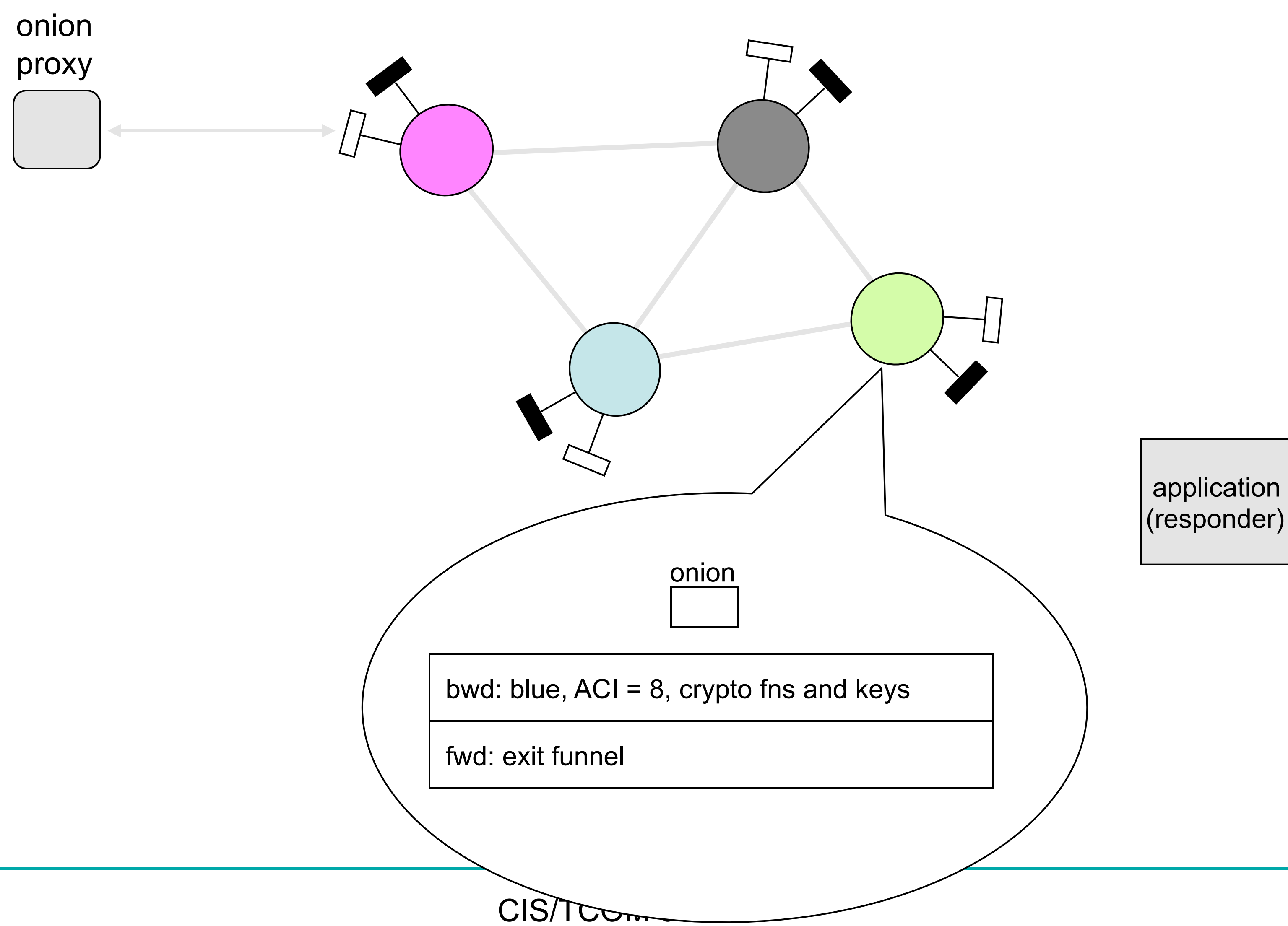
Anonymous connection setup



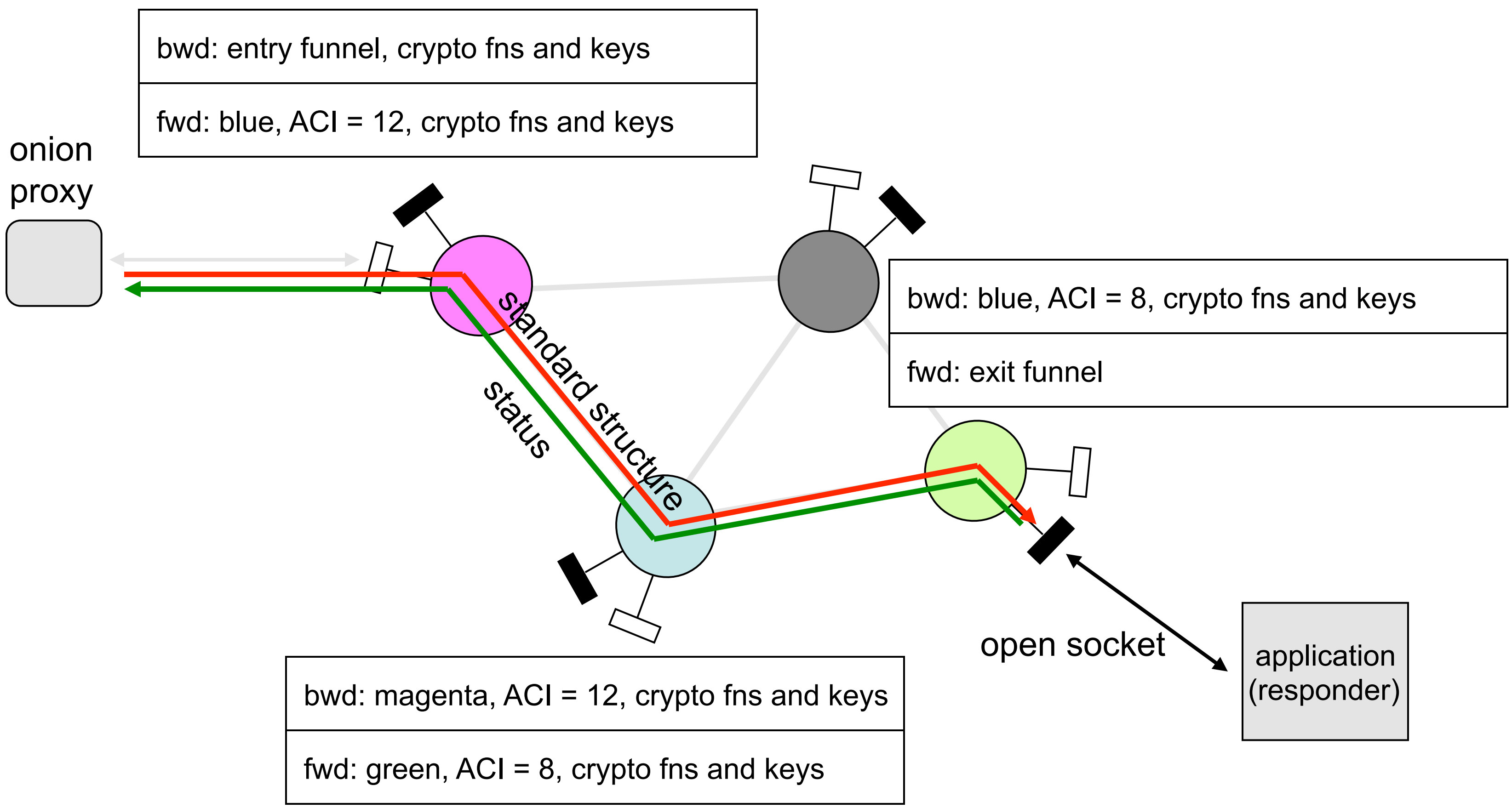
Anonymous connection setup



Anonymous connection setup



Anonymous connection setup



Data movement

- forward direction
  - the onion proxy adds all layers of encryption as defined by the anonymous connection
  - each onion router on the route removes one layer of encryption
  - responder application receives plaintext data
- backward direction
  - the responder application sends plaintext data to the last onion router of the connection (due to sender anonymity it doesn’t even know who is the real initiator application)
  - each onion router adds one layer of encryption
  - the onion proxy removes all layers of encryption