CIS 551 / TCOM 401

Computer and Network Security

Spring 2009

Lecture 22

Announcements

- Plan for Today:
  - Anonymity / Onion Routing
  - Grab bag: Secret Sharing
- Project 4 is due 28 April 2009 at 11:59 pm
  - Available on the web
- Please Read "*Analysis of an Electronic Voting System*"
  - Khono, et al. 2004 available on the course web page
- Final exam has been scheduled:
  Friday, May 8, 2009
  9:00am – 11:00am, Moore 216

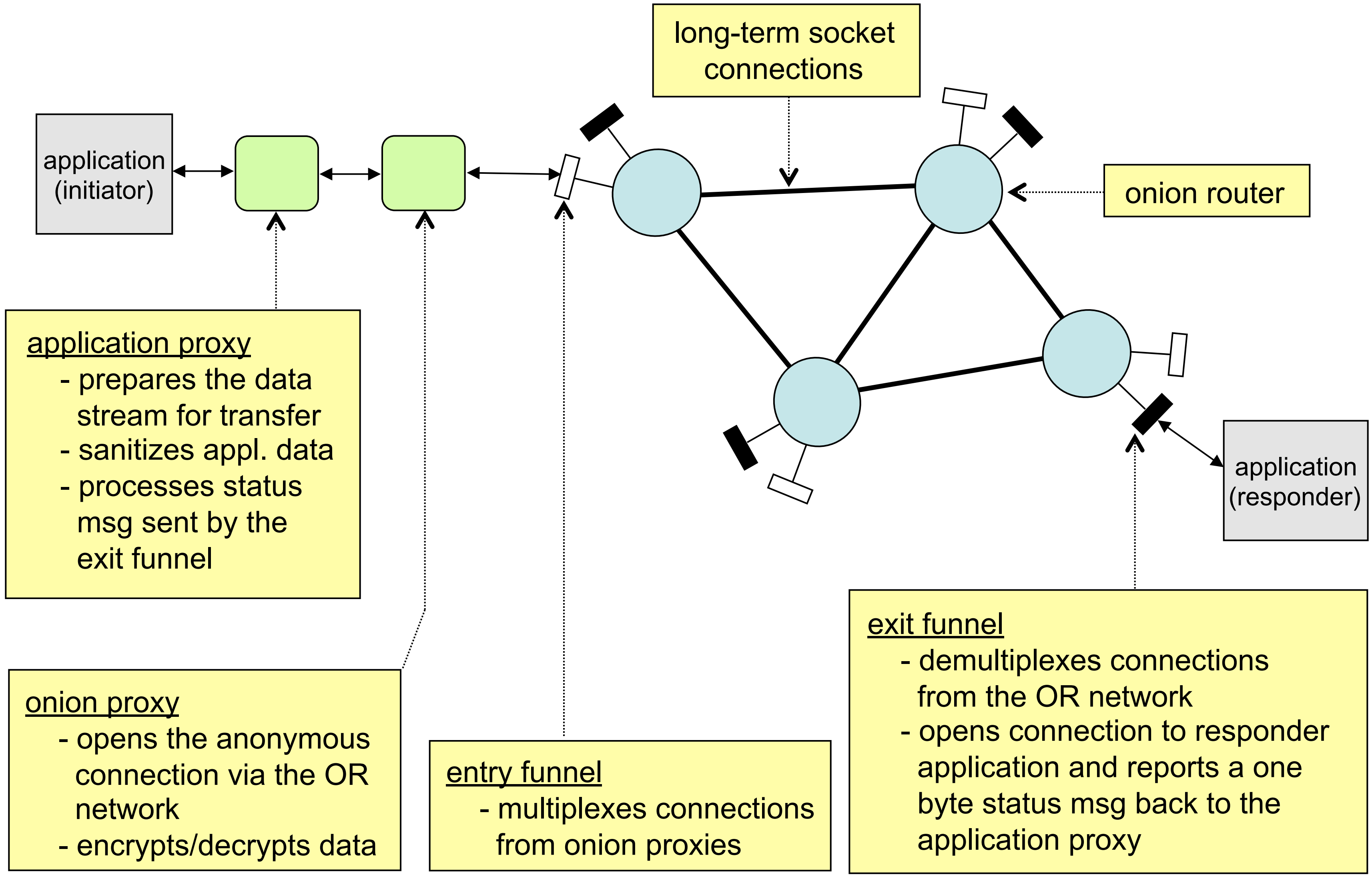
Mix Networks

- Original Chaumian decryption mix:
  - Implemented with set of servers
  - Input: list of encrypted values
    - Enc(Enc(Enc(…c…)))
  - Output: same list, decrypted
    - But order of list permuted
  - Each server in mix permutes list and removes one layer of encryption
- Civitas based on a re-encyprtion mix network
  - Input: List of encrypted messages
  - Output: Permuted list of re-encrypted messages
  - Re-encryption in El Gamal requires only the public key

A real-time MIX network – Onion routing

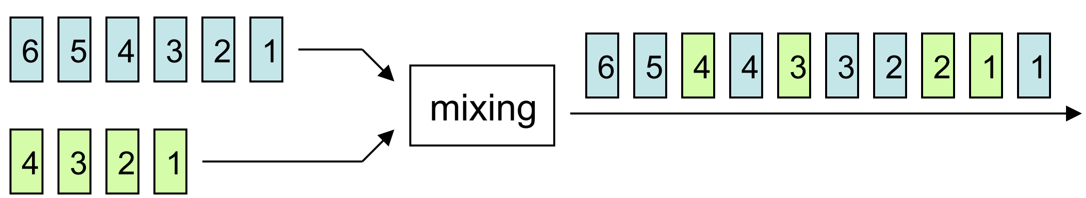
- general purpose infrastructure for anonymous communications over a public network (e.g., Internet)
- supports several types of applications (HTTP, FTP, SMTP, rlogin, telnet, …) through the use of application specific proxies
- operates over a (logical) network of onion routers
  - onion routers are 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



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 (output feedback) 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

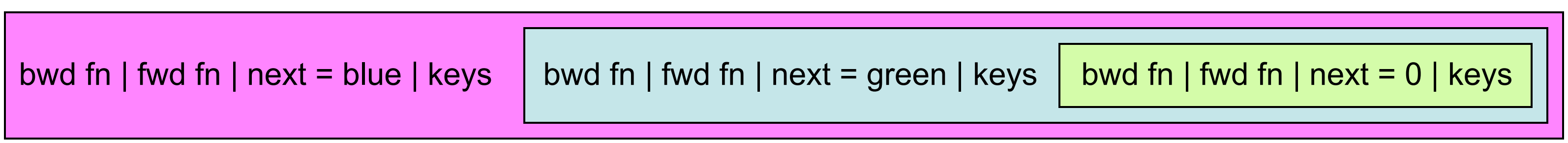
- the application is configured to connect to the application proxy instead of the real destination
- upon a new request, the application proxy
  - decides whether to accept the request
  - opens a socket connection to the onion proxy
  - passes a *standard structure* to the onion proxy
  - standard 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)

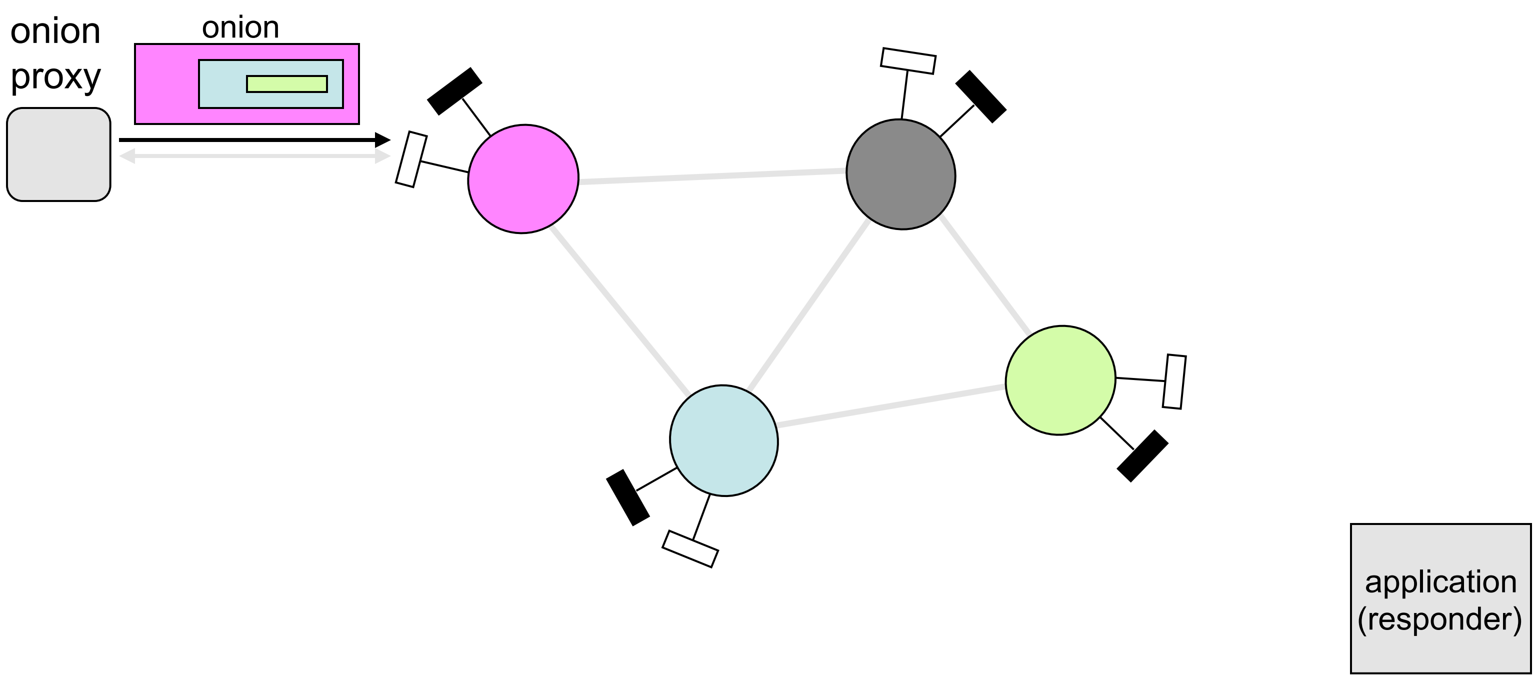
- upon reception of the standard structure, the onion proxy
  - decides whether to accept the request
  - establishes an anonymous connection through some randomly selected onion routers by constructing and passing along an *onion*
  - sends the standard structure to the exit funnel of the connection
  - 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

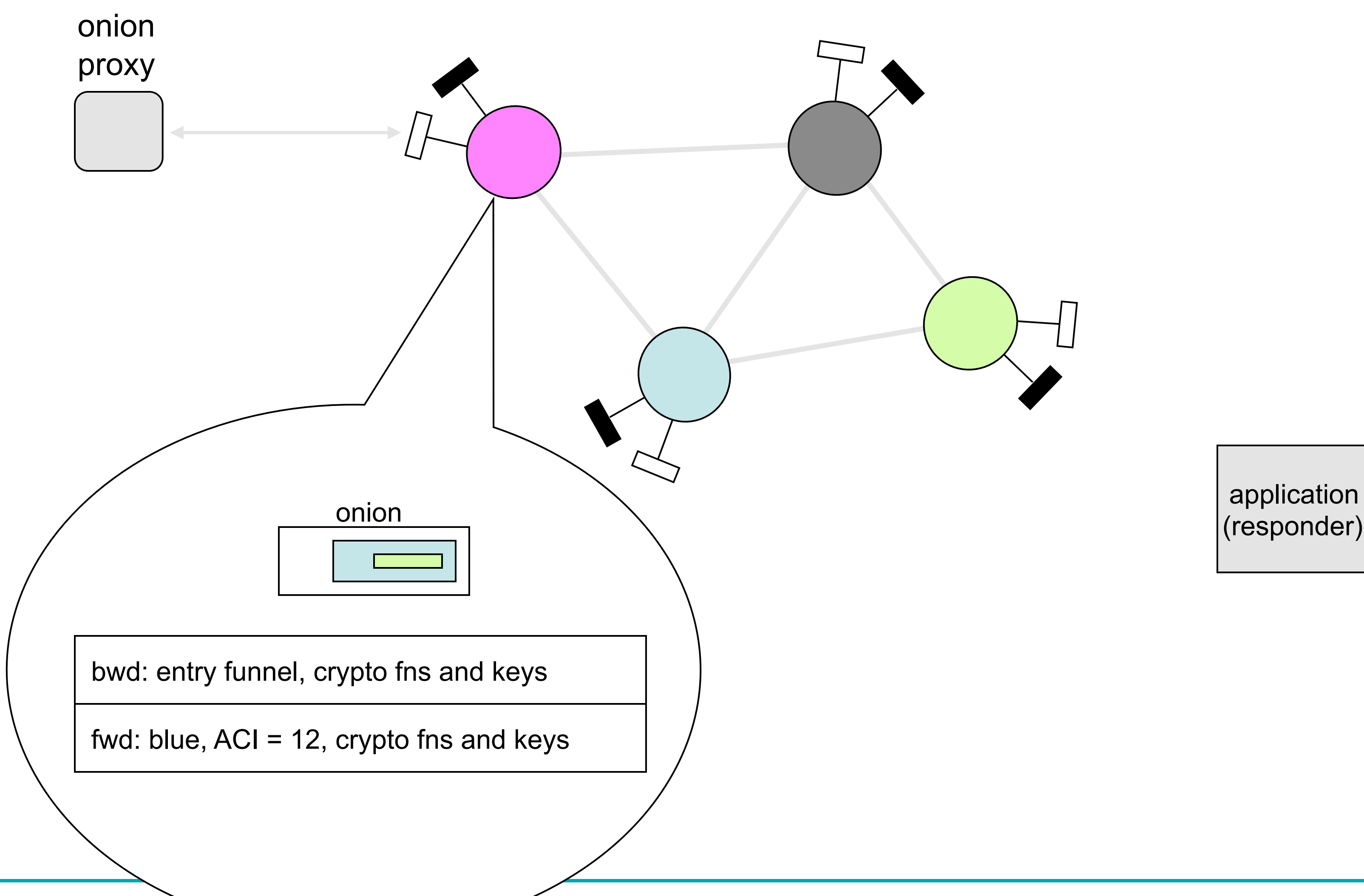
- an onion is a multi-layered data structure
- it encapsulates the route of the anonymous connection within the OR network
- each layer contains
  - backward crypto function (DES-OFB, RC4)
  - forward crypto function (DES-OFB, RC4)
  - IP address and port number of the next onion router
  - expiration 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



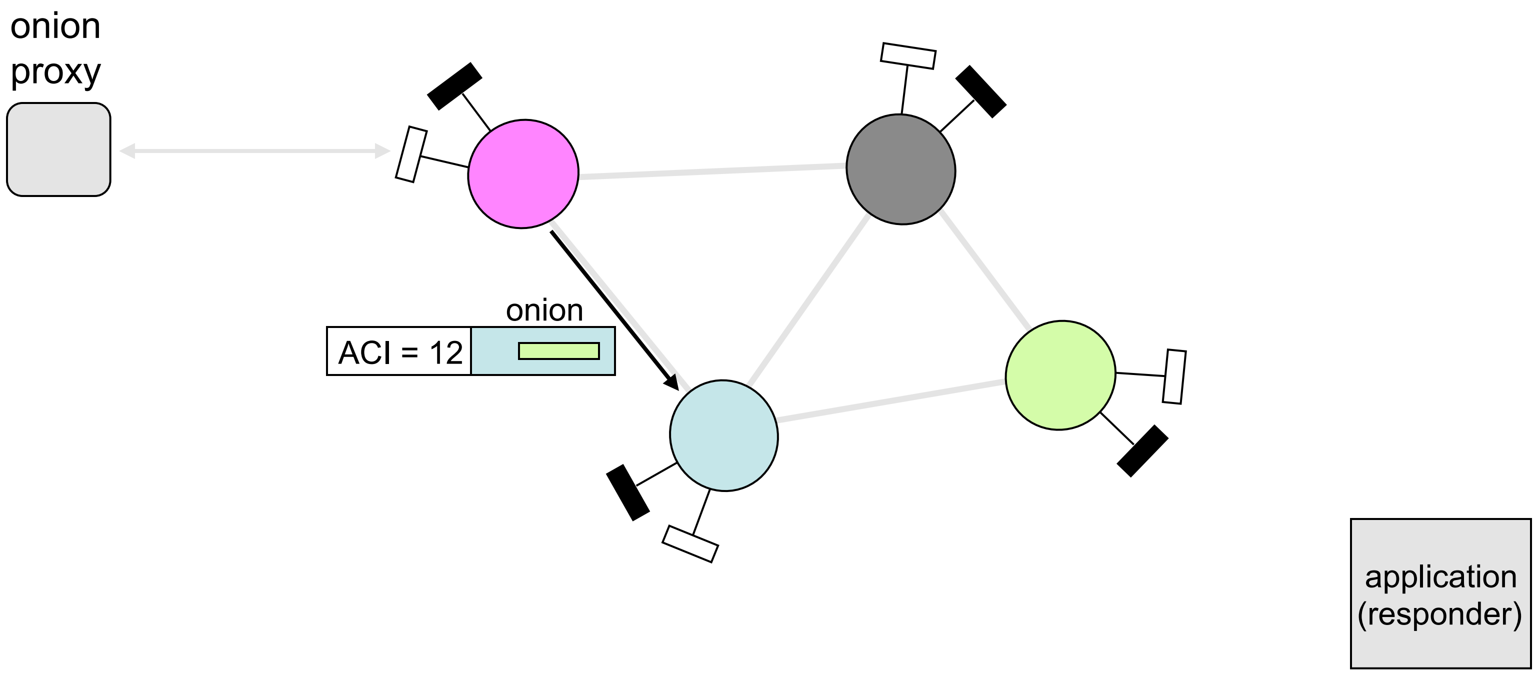
Anonymous connection setup



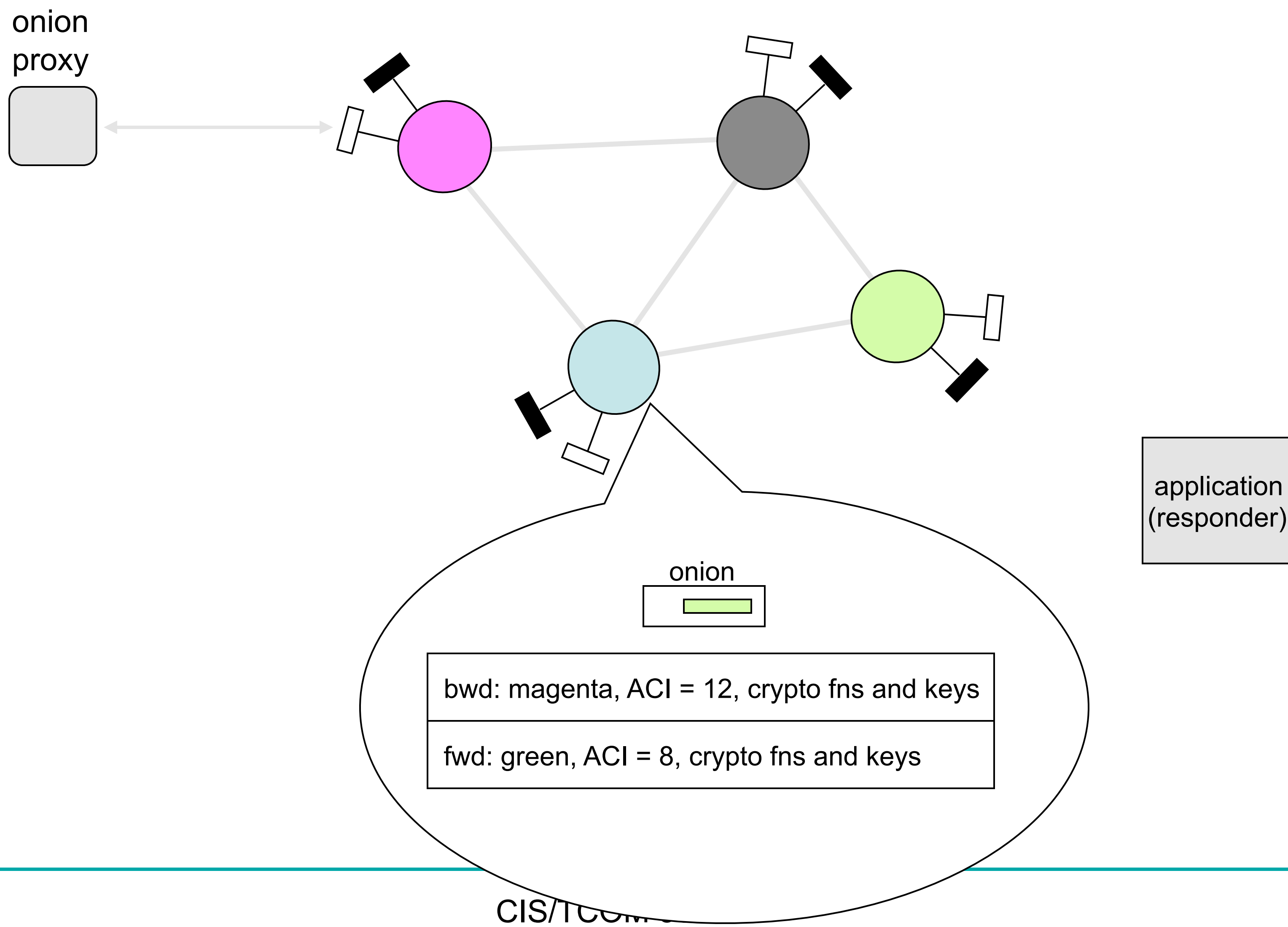
Anonymous connection setup



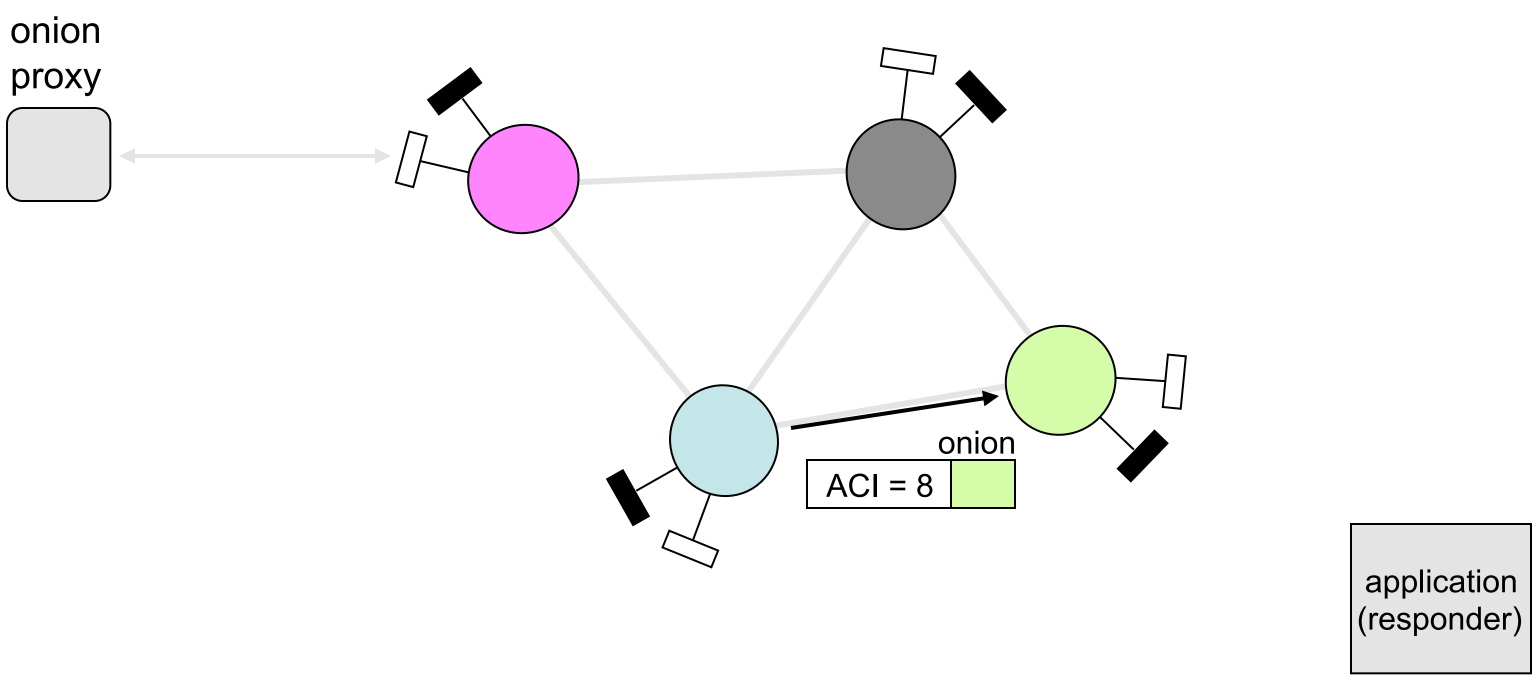
Anonymous connection setup



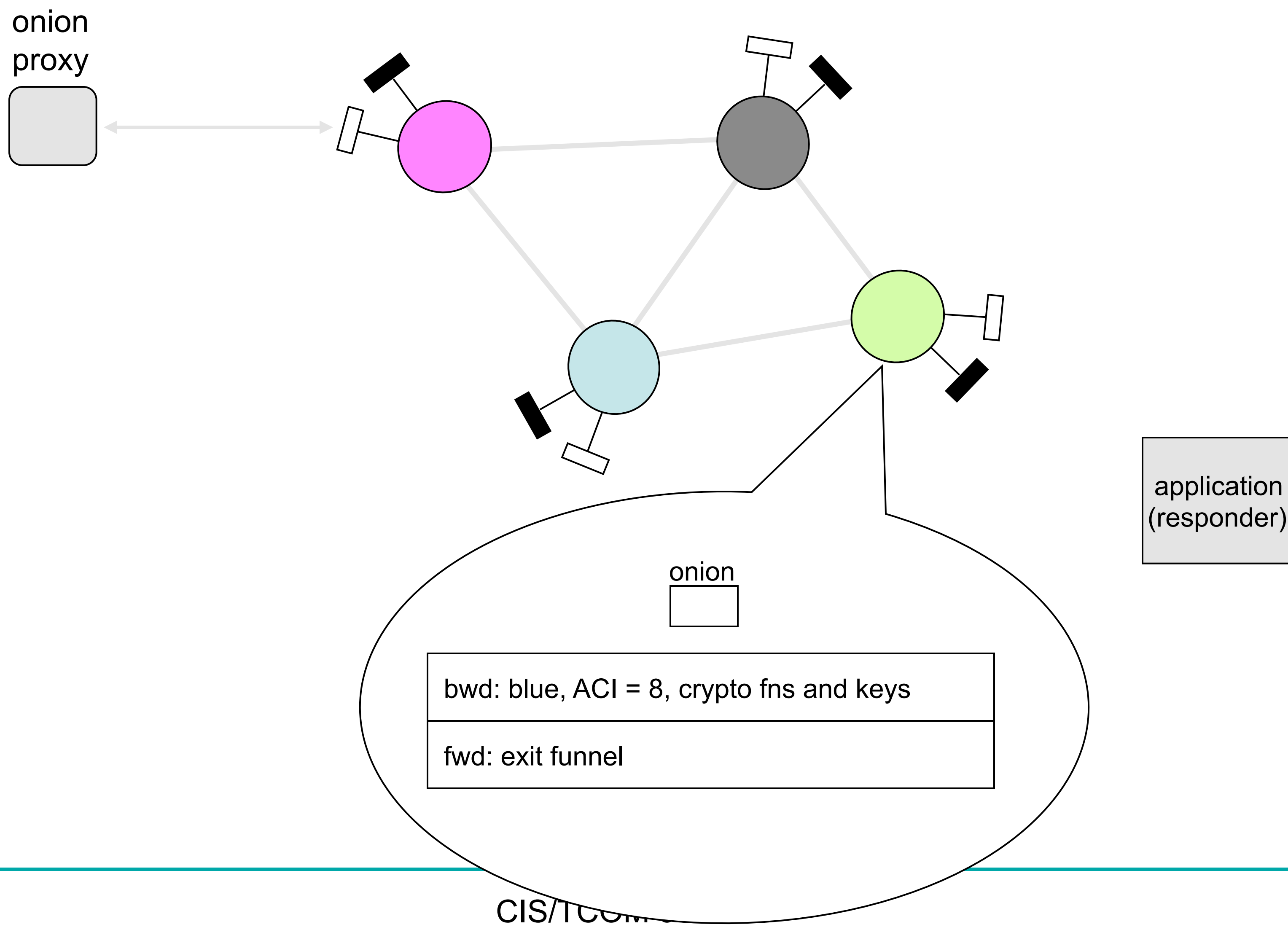
Anonymous connection setup



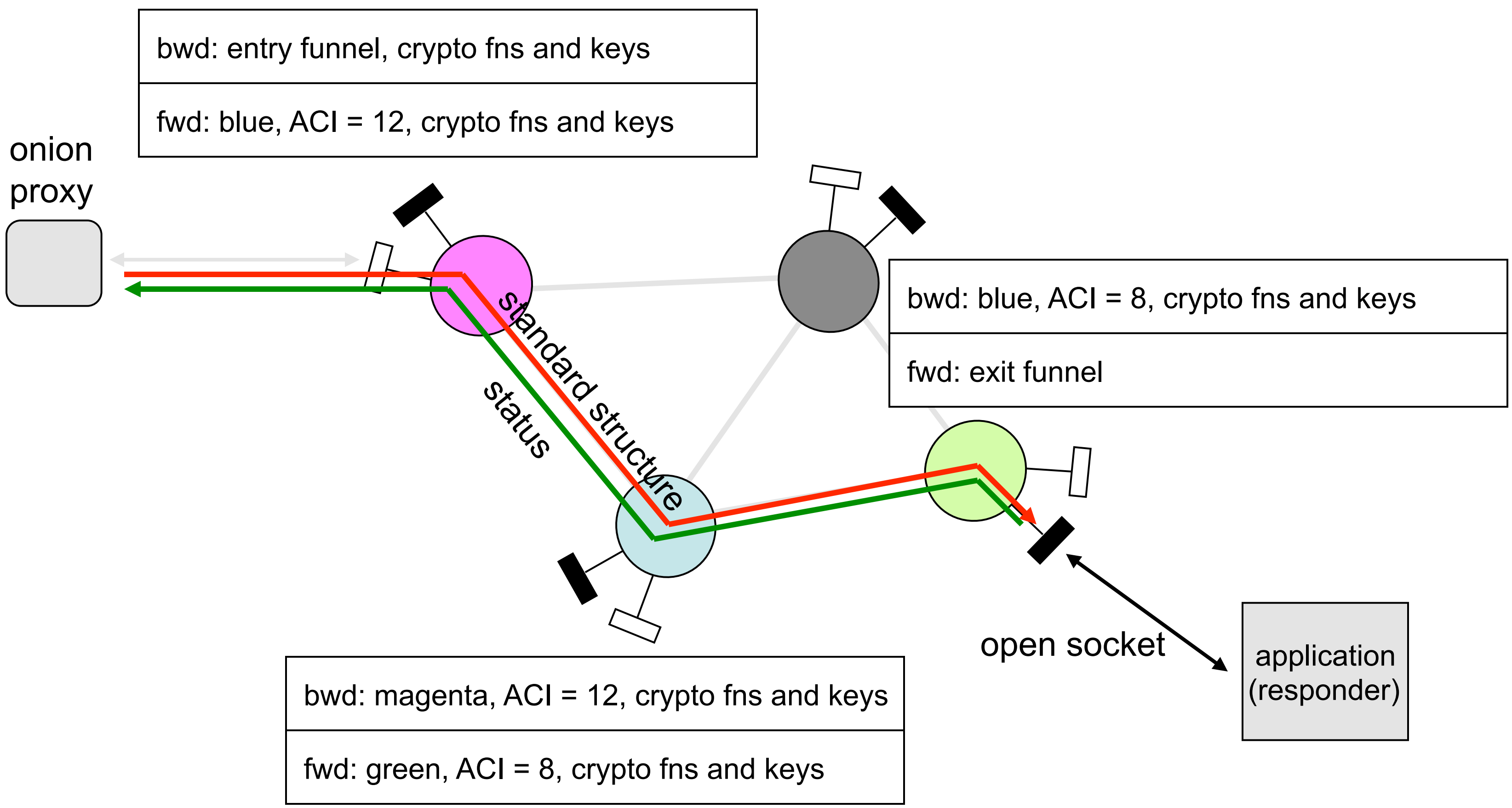
Anonymous connection setup



Anonymous connection setup



Anonymous connection setup

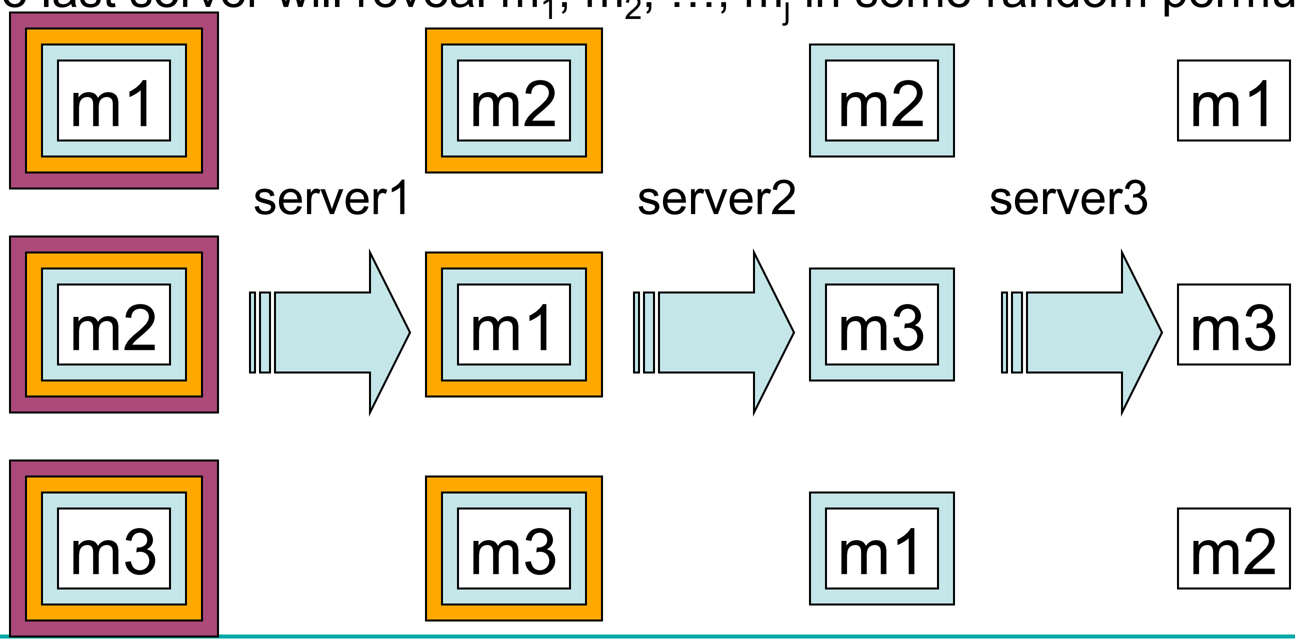


Data movement

- 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

Mix networks: Anonymity

- Chaum 1981: Basic Mix network
- Suppose that there are N servers with public keys $K_1 \ldots K_N$.
- A mix message M_a looks like: $K_1\{K_2\{\ldots K_N\{m_a\}\ldots\}\}$
- To anonymize a set of messages $M_1, M_2, \ldots, M_j$:
  - Server i decrypts the messages, permutes them, and forwards them to server i+1
  - The last server will reveal $m_1, m_2, \ldots, m_j$ in some random permutation:



Cryptographic Techniques

- Zero-knowledge (ZK) proofs
  - Vote proofs, tabulation proofs
- Secret Sharing
- Homomorphic Encryption
  - Blind signatures

- Motivating Application: electronic voting

Civitas: Secure Remote Voting

- A secure, remote voting system implemented at Cornell by Michael Clarkson, Steve Chong, and Andrew Myers
  - `http://www.cs.cornell.edu/projects/civitas`
- Based on an earlier voting scheme proposed by
  - Ari Juels, Dario Catalano, and Markus Jakobsson (WPES 2005)
- Terminology:
  - *Voting system:* (software) implementation
  - *Voting scheme:* cryptographic construction
  - *Voting method:* algorithm for choosing between candidates

Remote Voting

- *Remote*: no supervision of voting or voters
  - Generalizes Internet voting
- The authors argue that this is the right problem to solve:
  - Does not assume supervision
  - Internet voting is common (Debian, ACM, SERVE)
  - Absentee ballots
  - Some states moving entirely to remote voting (Oregon, Washington)

Integrity

- The final tally cannot be changed to be different from a public counting of the votes

Verifiability:

The final tally is verifiably correct

- Including:
  - Voters can check their own vote is included (*voter verifiability*)
  - Anyone can check that only authorized votes are counted, and no votes are changed during tallying (*universal verifiability*)
  - No ballot stuffing (i.e. a voter can't submit multiple votes)
- Sorely lacking in real-world systems

Confidentiality

- No adversary can learn any more about votes than is revealed by the final tally

Anonymity:

The adversary cannot learn how voters vote, unless voters collude with the adversary.

- Anonymity is too weak
  - Allows vote selling and coercion of voters

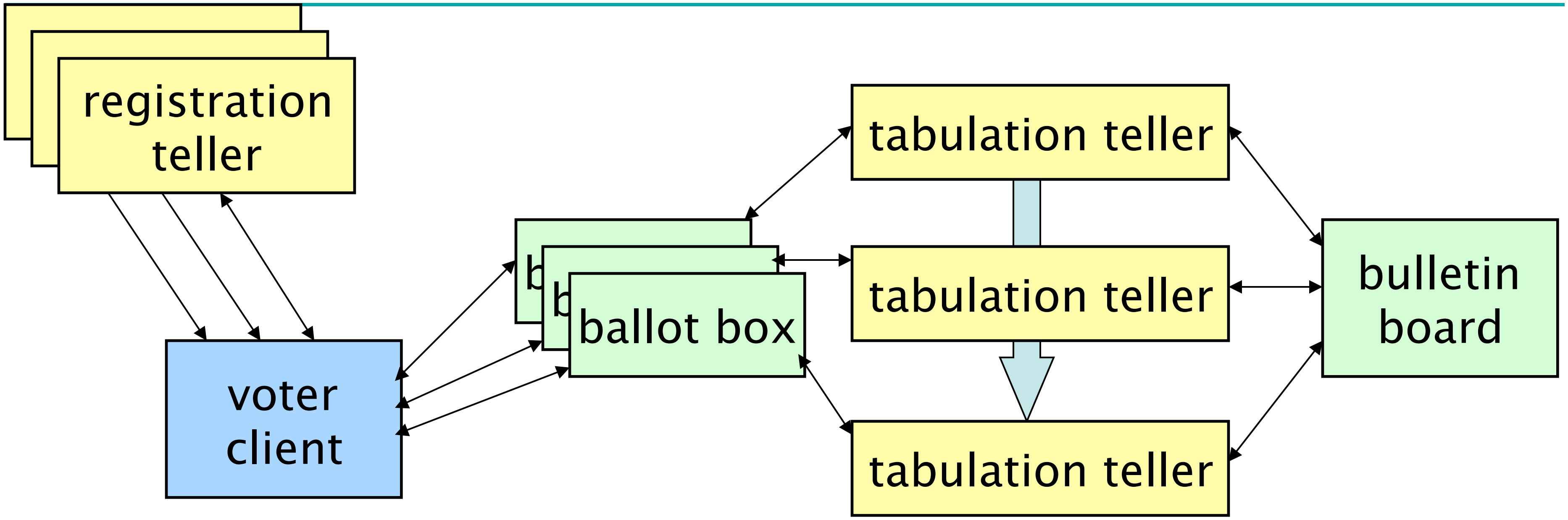
Confidentiality [2]

Coercion resistance:

Voters cannot prove whether or how they voted, even if they can interact with the adversary while voting.

- Required by Civitas
- Stronger than anonymity (or *receipt-freeness*)
  - Adversary could be your employer, spouse, …
  - Must defend against forced abstention or randomization

Civitas Architecture



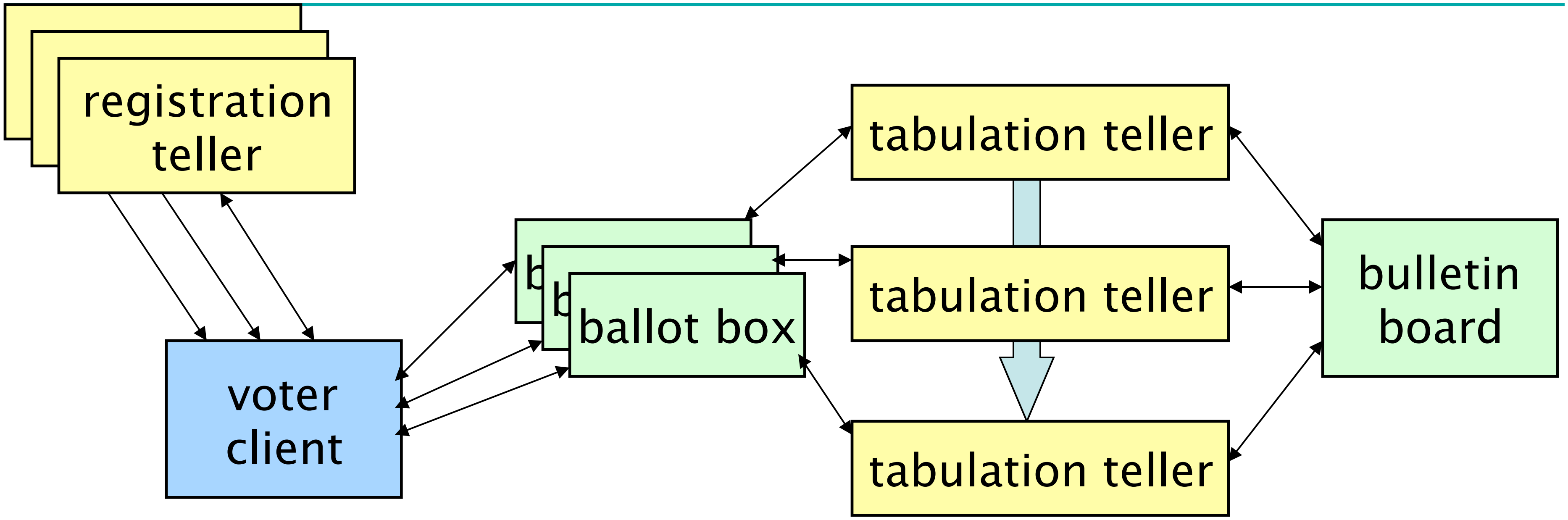
Civitas scheme

What makes this secure? Why do we believe it is?

Key Idea of Scheme

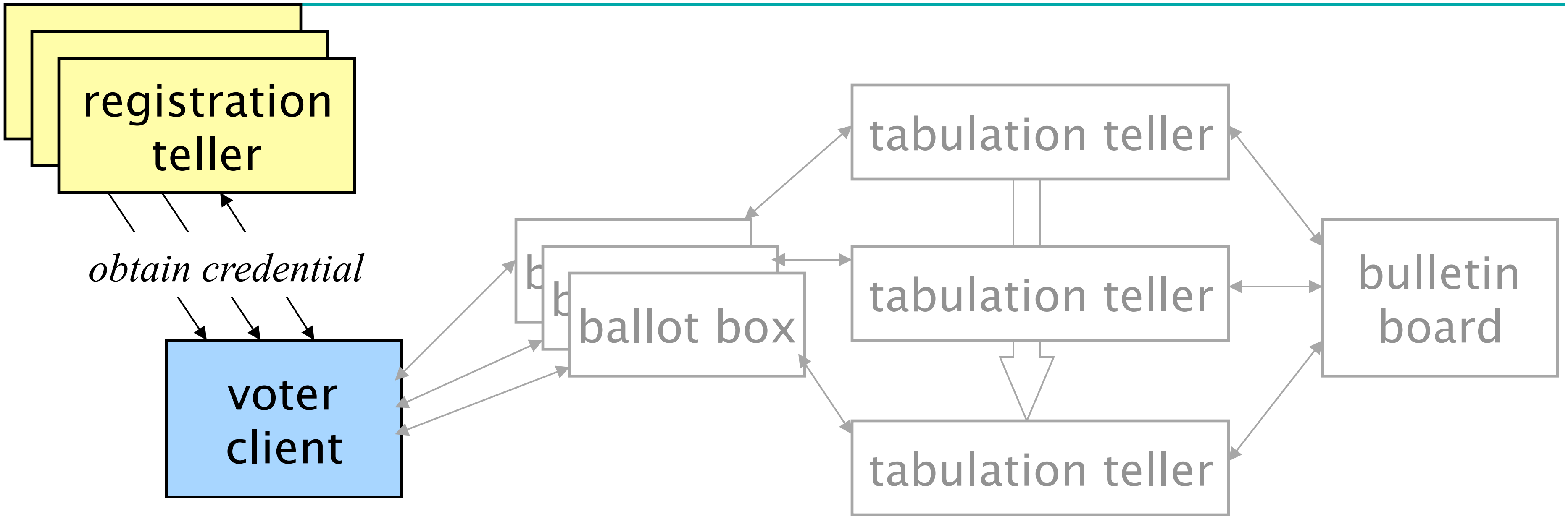
- Voter encrypts vote and “signs” with a *credential*
  - Votes are posted anonymously to ballot boxes
- Invalid credentials and votes are eliminated without revealing which were invalid
  - Tabulation tellers post ZK ("Zero Knowledge") proofs
- Anyone can verify election by checking ZK proofs
- Voter resists coercion by inventing fake credential and using it to behave exactly as coercer demands
  - Voter needs some time not under coercer’s control to use his real credential

Cryptography



Assumption 1. Decision Diffie-Hellman, RSA, SHA-256 random oracle model.

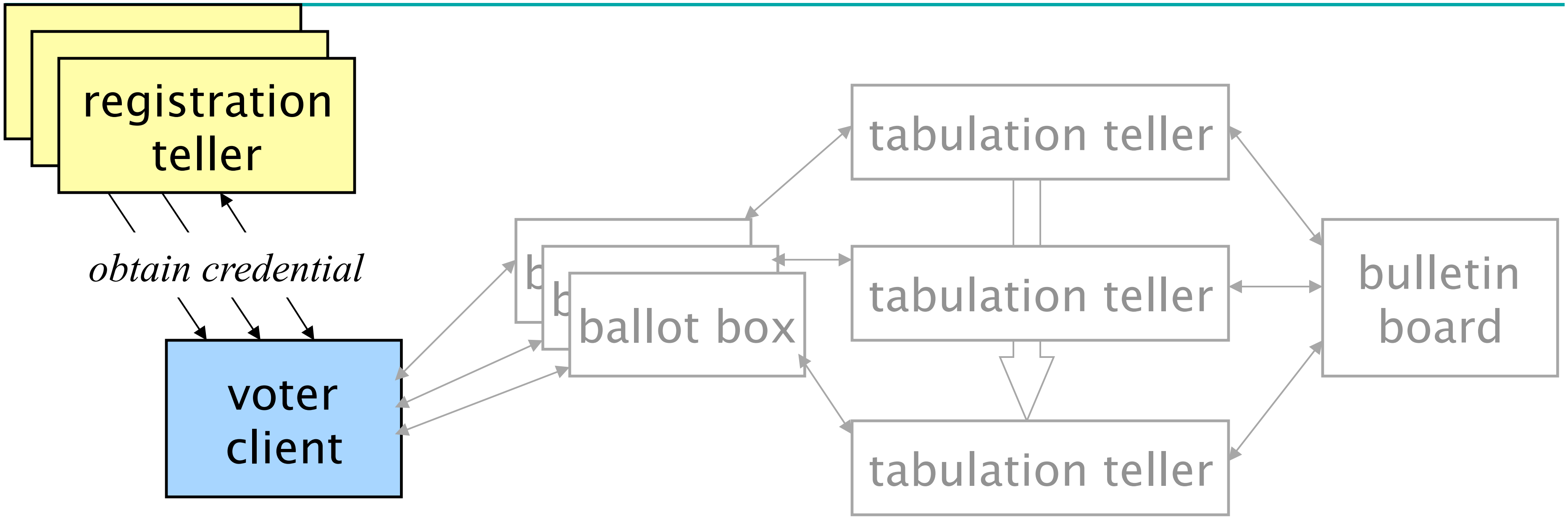
Registration



Assumption 2. The adversary cannot masquerade as voter during registration.

Implement with: strong authentication, non-transferable secrets.

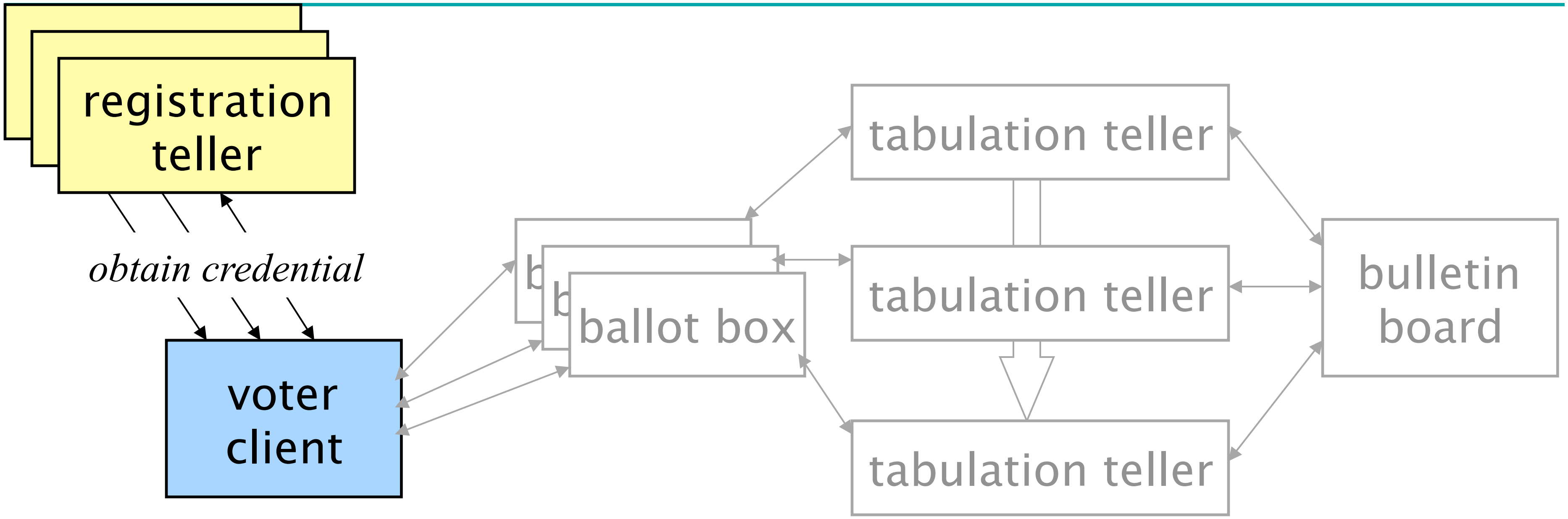
Registration



Assumption 3. Each voter trusts at least one registration teller and has an untappable channel to that teller.

Why: weakest known assumption for coercion resistance Implement with: advance, in person registration; information-theoretic encryption

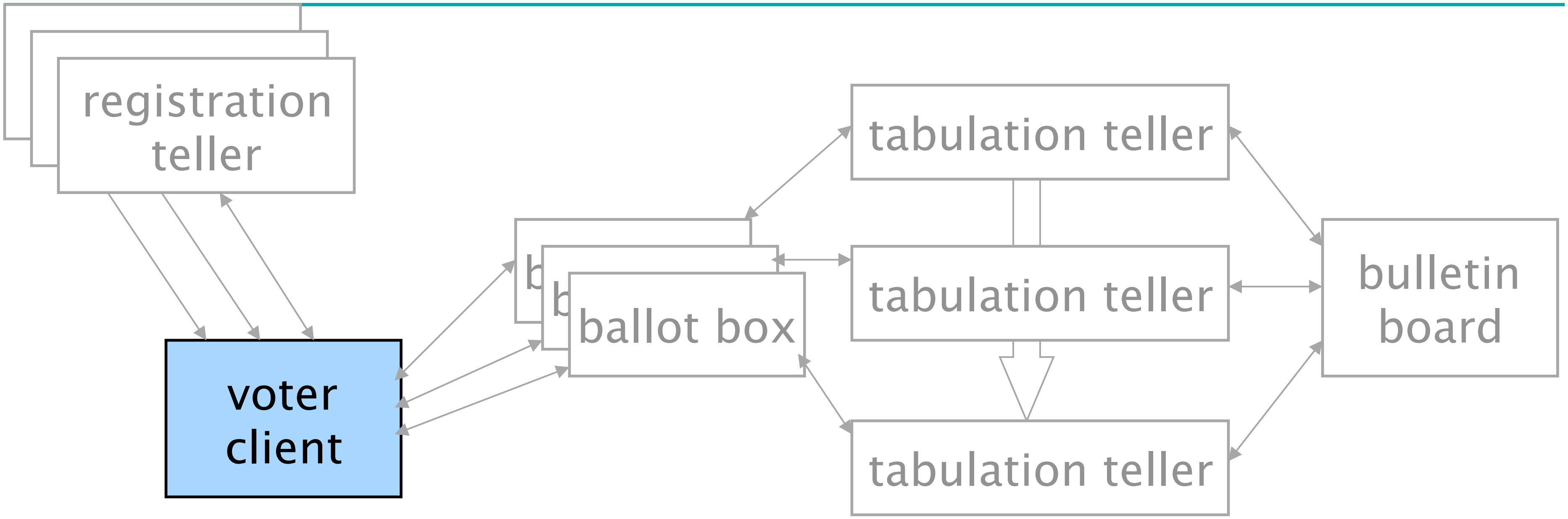
Registration



Assumption 3. Each voter trusts at least one registration teller and has an untappable channel to that teller.

Failure implies not coercion-resistant;
doesn't impact verifiability.

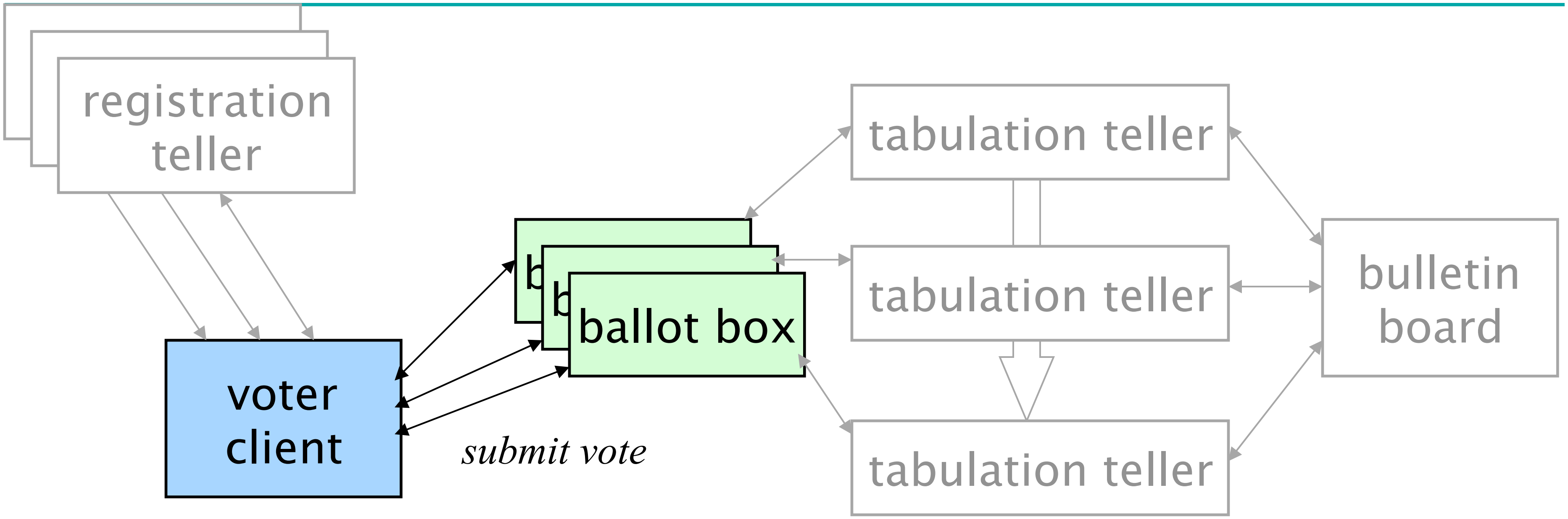
Voting



Assumption 4. Voters trust their voting client.

Reasonable: voter can choose client.

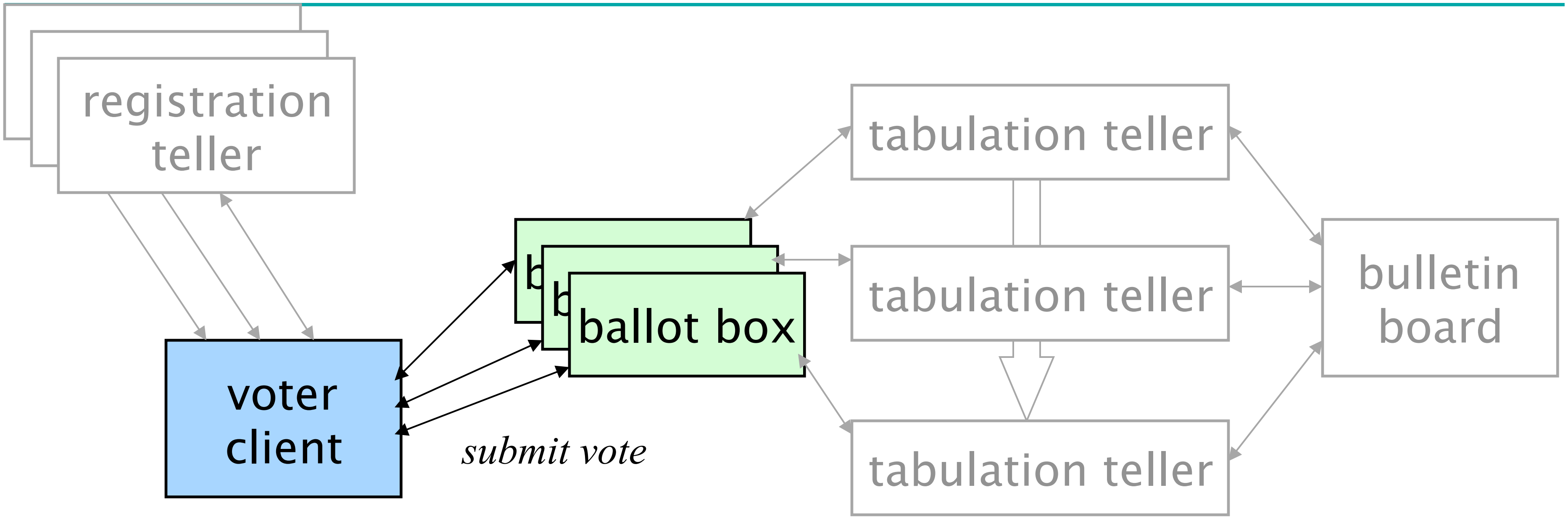
Voting



Assumption 5. The channels from the voter to the ballot boxes are anonymous.

Why: otherwise coercion resistance trivially violated. Failure has no impact on verifiability.

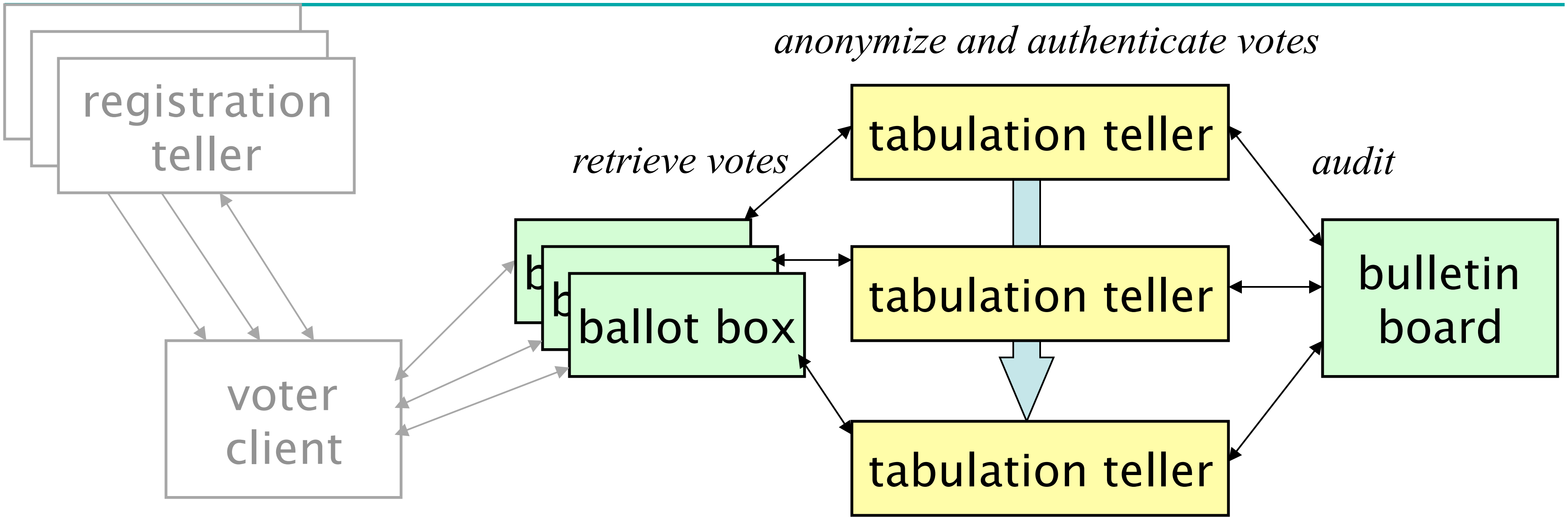
Voting



Assumption 6. Each voter trusts at least one ballot box to make vote available for tallying.

Why: expensive fault tolerance not required.

Tabulation



Assumption 7. At least one tabulation teller is honest.

Why: keeps tellers from decrypting votes too early or cheating throughout tabulation.

Setup Phase

- Supervisor posts public keys for registrar, registration tellers, tabulation tellers
- Registrar posts identities and public keys of voters
- Tabulation tellers generate public key K_{TT} for a distributed cryptosystem
  - Everyone knows public key
  - Each teller has *share* of private key
  - Anyone can encrypt; participation of *all* tellers required to decrypt
- Registration tellers generate credentials for voters
  - Each credential is a pair of public/private values
  - Each teller responsible for generating one share of the full credential for each voter
  - Public credential share posted on bulletin board
  - Private credential share stored; later released to voter

El Gamal Encryption

- K_{TT} is a public key for an El Gamal cryptosystem:
  – El Gamal works similarly to RSA (same assumptions)
  – El Gamal is *malleable*: Given C = Enc(m,K) anyone can find D such that D ≠C but Dec(D,k) = m
  – Enc(m,K) * Enc(n,K) = Enc(m*n, K) *Homomorphic*

- The private share of a credential is s_i
- The corresponding public share is Enc(s_i, K_{TT})
- The complete private share is: $s = s_1 * s_2 * \ldots * s_n$
- The complete public share is Enc($s_1 * s_2 * \ldots * s_n$, K_{TT})
  – The latter is computable because of the homomorphic property of El Gamal

Voting Phase

- Voter retrieves private credential shares
  - Contacts each registration teller, authenticates with public key posted by registrar
  - Establishes an AES key using Needham-Schroeder-Lowe
  - Voter combines all shares to produce s, the full private credential
- Voter votes
  - Submits a copy of vote to each ballot box:

$$Enc(s; K_{TT}), Enc(choice; K_{TT}), P$$

  - P is a proof that vote is well-formed:
    - In particular, it proves that the voter had access to s and choice simultaneously

Tabulation Phase

Tabulation tellers:

1. Retrieve data
2. Verify vote proofs
3. Eliminate votes with duplicate credentials
4. Anonymize votes and credentials
5. Eliminate votes with unauthorized credentials
6. Decrypt choices in remaining votes

Tellers constantly post proofs that they are performing the protocols correctly; yields verifiability

Voters Lie to Resist Coercion

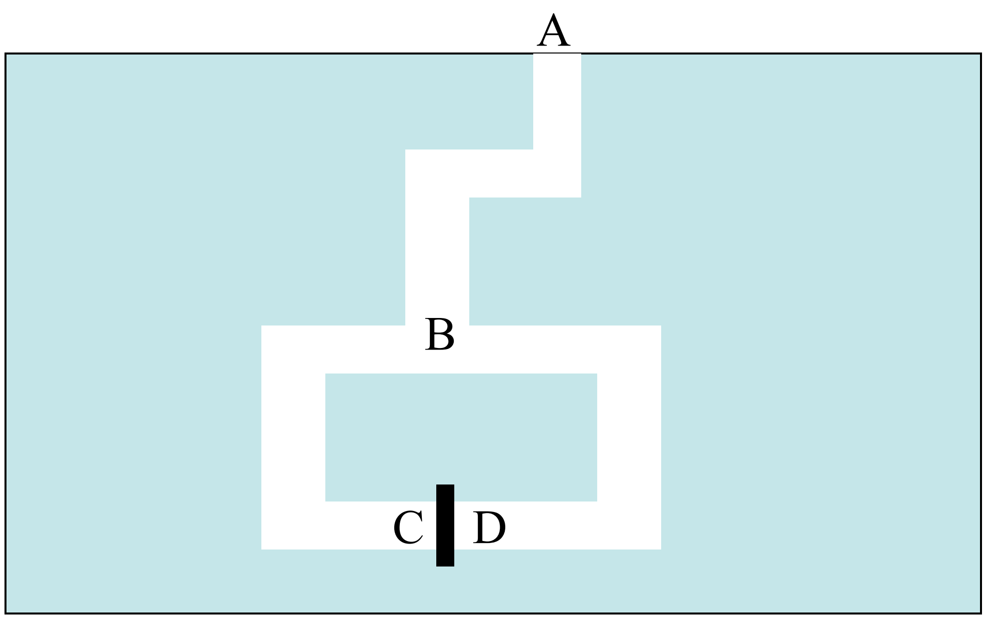
- Voter picks random “fake” private credential share
- Constructs new “fake” private credential
- Uses “fake” credential to behave exactly as coercer demands
  - Give it to adversary
  - Submit any vote demanded by adversary
  - Voter needs some time not under coercer’s control to use his real credential
- Yields coercion resistance

Zero-Knowledge Proofs

- Standard proofs in math class: student (prover) writes something that TA (verifier) checks.
  - Convinces verifier of statement made by prover
- But standard proofs also reveal knowledge to the verifier
  - Prover: “I know the password to the Federal Reserve”
  - Verifier: “I don’t believe you!”
  - Prover: “It’s XYZZY”.
  - Verifier: Logs into Fed to check if password works.
  - Verifier: “Thanks. Now I know it too.”

Zero-Knowledge Proofs

- A *zero-knowledge* proof allows P to prove to V that a statement is true without revealing any more information.
- Example: The magic word and the cave



Zero-Knowledge Proofs

- Commit:
  - V stands at A
  - P walks into cave to either C or D
  - V walks to B
- Challenge:
  - V shouts to P to either come out the left or the right passage
- Response:
  - P complies, using the magic word if necessary
- P and V repeat until V is convinced P knows the magic word

Zero-Knowledge Proofs

- Zero-knowledge:
  - V never learns the magic word
  - V can’t convince anyone else that P knows the magic word
    - P & V could have agreed on series of “left/right” in advance
- Large classes of problems have ZK proofs
- This proof was *interactive*
  - Based on challenge/response
  - Can make *noninteractive* by using a special kind of hash function to generate the challenge

- Plaintext Equivalence Test
  - Special kind of ZK proof
  - Collections of hosts can prove (as a group) that Dec(c) = Dec(c’) without anyone learning what Dec(c) or Dec(c’) actually are.
  - That is, it is possible to convince a third party that two encrypted plaintexts are the same, without revealing what the plaintext is!

Blind Signatures

- Digital signature scheme equipped with a commutative *blinding* operation
  - Signer never learns what they signed
  - Like signing an envelope with a window (or with carbon paper)
  - I.e.: unblind(sign(blind(m))) = sign(m)
- Voting scheme:
  - Voter prepares vote *v*, blinds, and authenticates to Authorization server, and sends vote. Server checks off voter, signs vote, and sends back to voter. Voter unblinds and now has sign(*v*).
  - Voter anonymously sends sign(*v*) to Tabulation server. Server checks signature, then counts vote.

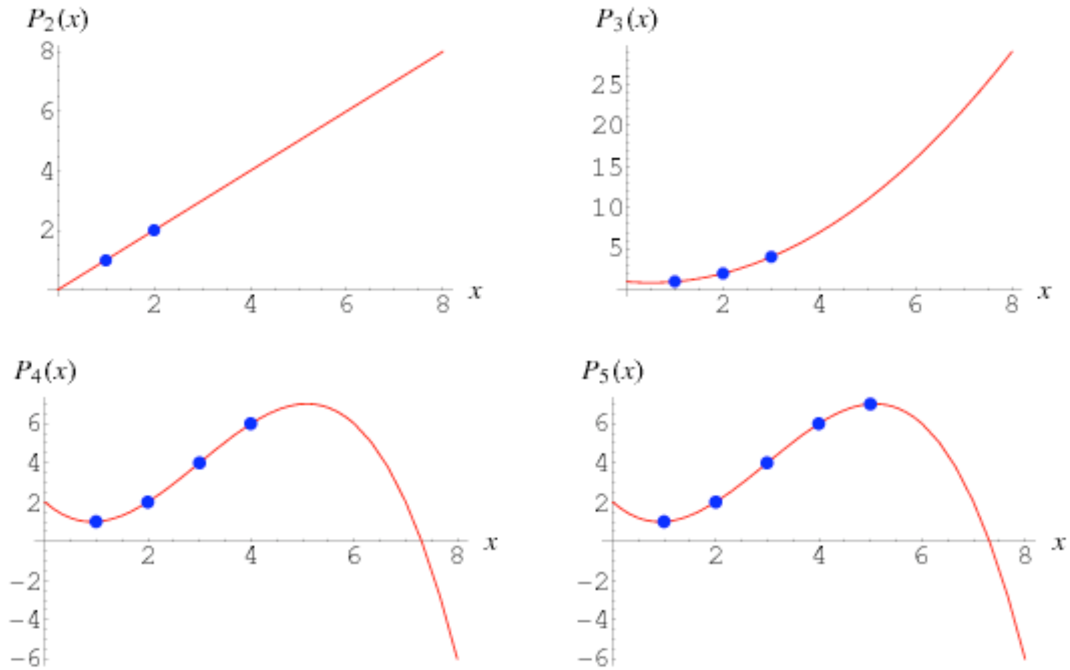
Homomorphic Encryption

- A *homomorphic* encryption scheme has an operator ★ such that Enc(m) ★ Enc(n) = Enc(m ★ n). ★ is usually either + or $\times$, never both.
  - E.g. both RSA and El Gamal have $\times$.
- Voting scheme:
  - Suppose scheme has + as homomorphism and votes are either 0 or 1.
  - Voter prepares Enc(0) or Enc(1) as vote, authenticates to Tabulation server, and submits vote.
  - Tabulation server sums all the votes, then decrypts result. Individual votes never decrypted.

Secret Sharing

- How to share a secret among N+1 players:
  - Owner of the secret generates N random bitstrings R1 … RN
  - Player 0 gets $S \oplus R1 \oplus \ldots \oplus RN$
  - Player j > 0 gets Rj
  - All N players can cooperate to recover S -- they just XOR their shares.
- *Threshold* schemes allow k-out-of-N players to recover the secret:
  - Owner of the secret picks a random polynomial f with degree (k-1) such that f(0) = S
  - Player j > 0 gets f(j)
  - If any k players get together, they can use Lagrange interpolation to calculate f(0)
  - If fewer than k players get together, there's no information about f(0).

Lagrange Interpolation



The Lagrange interpolating polynomial is P(x) that passes through n points:
$(x_1, y_1 = f(x_1)), \ldots , (x_n, y_n = f(x_n))$

$$P(x) = \frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)}\,y_1 + \frac{(x-x_1)(x-x_3)\cdots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)}\,y_2 + \cdots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\cdots(x_n-x_{n-1})}\,y_n.$$

Example: 3-out-of-N Secret

- Suppose the secret is S = 7
- I generate (at random) $f(x) = 2x^2 - 3x + 7$
- Then S = f(0) = 7
  - Share s1 = f(1) = 6
  - Share s2 = f(2) = 9
  - Share s3 = f(3) = 16
  - Share s4 = f(4) = 27

- To recover secret and obtain 3 shares:
  - Example: given s2, s3, s4 = (2,9) (3,16) (4,27)
  - Calculate P(x) as on the previous slide [see blackboard]