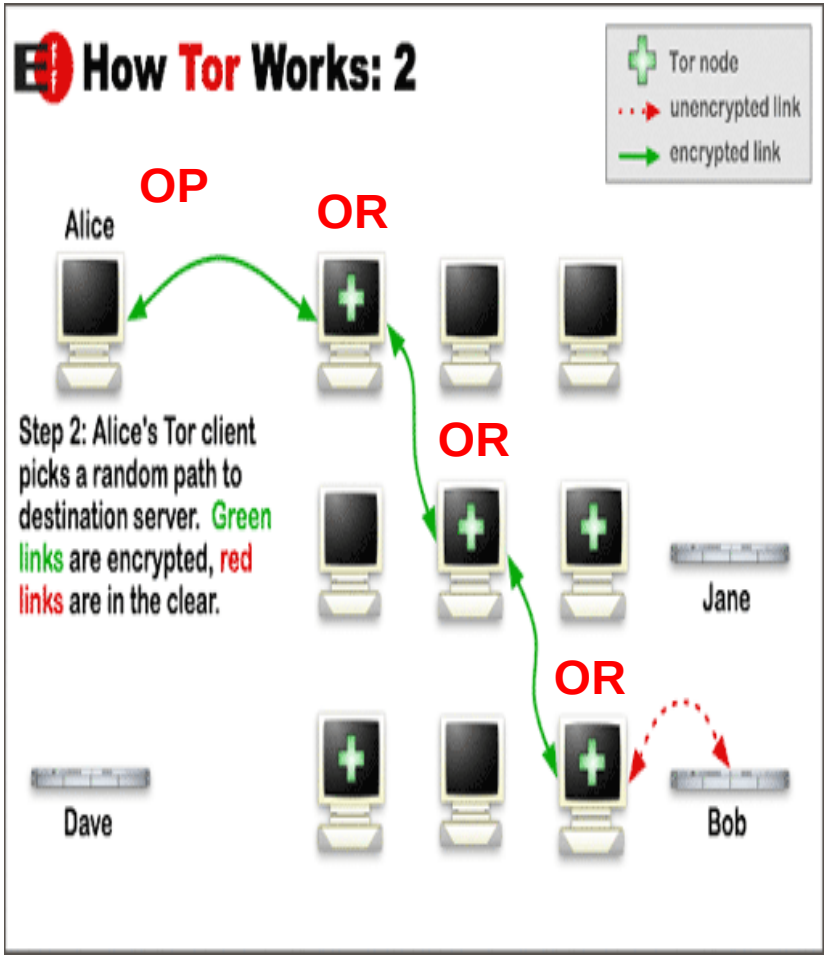
Reading Group Report on [DMS04]

Tor: The Second-Generation Onion Router

Overview

- A typical use case:
  - What is a circuit? How to set up one?
  - How to encrypt/decrypt messages iteratively?
- The security properties Tor tries to achieve:
  - semantic security
  - end-to-end integrity (**not hop-to-hop**)
- Possible future work

Tor's Basic



- A user (Alice), sets up a circuit to a server (Bob), before she begins to send any message to him.
- Each hop on the circuit is called an OR.
- Each OR keeps:
  - A shared key
  - Identities of its predecessor and successor.
- Alice is called OP (onion proxy)

The Setup of a Circuit

- Say Alice would like to set up a circuit of two ORs.
- She first chooses 2 ORs from a *directory server*, learns their IP addresses and their long-term public keys.
- She begins to set up the circuit in a *telescoping approach*.

The Setup of a Circuit

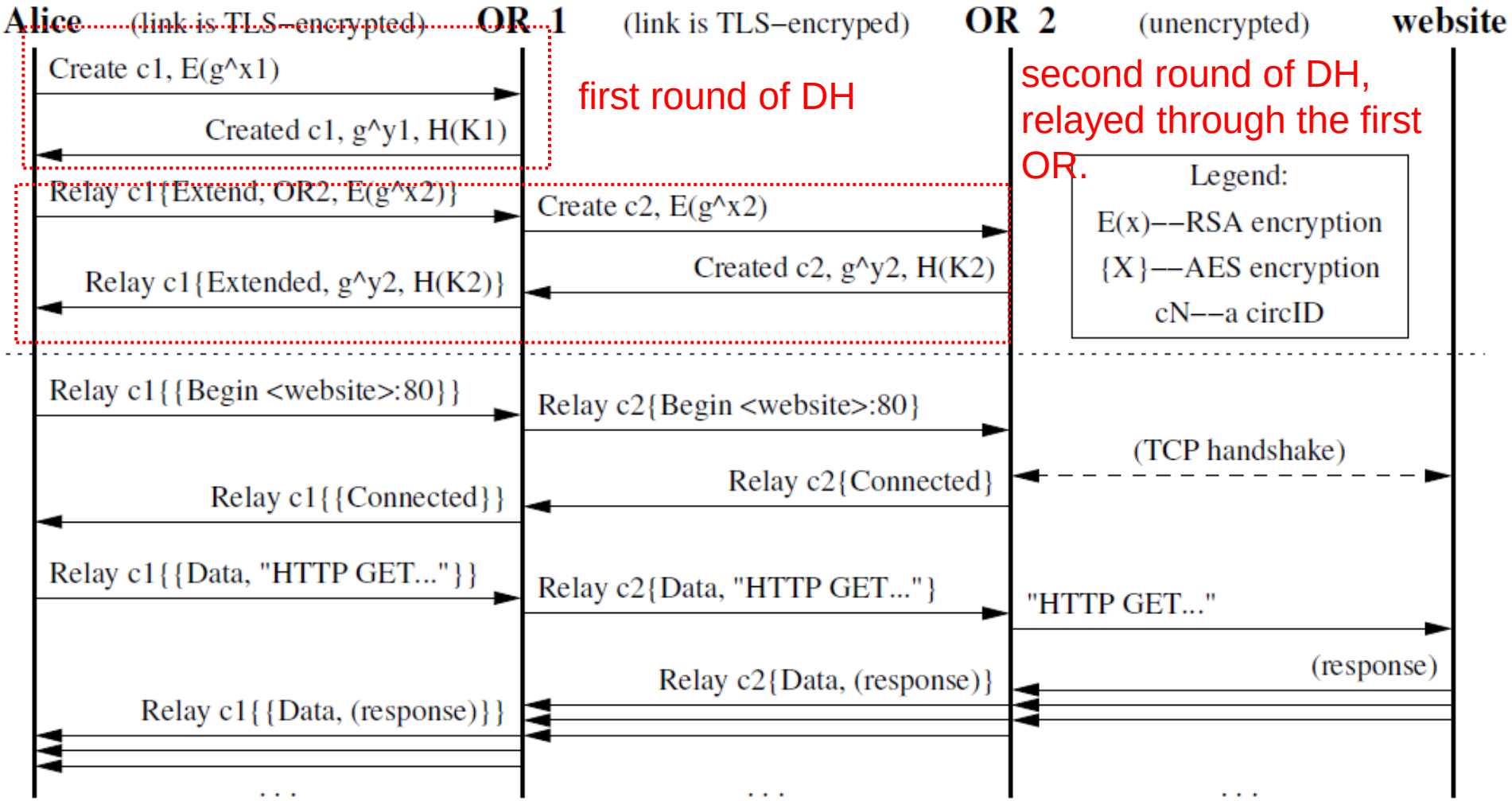


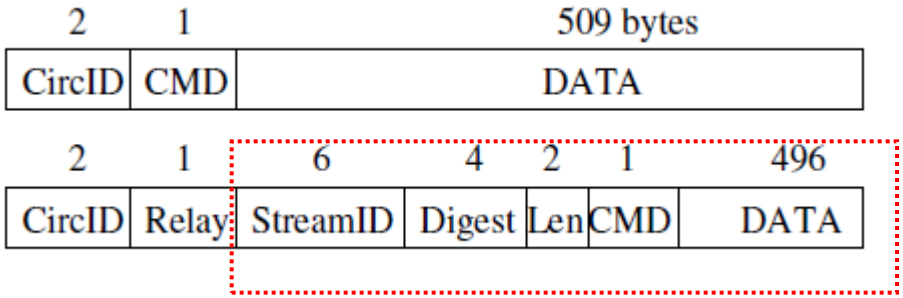
Figure 2 from [DMS04]: Alice constructs a 2-hop circuit to a website

The Setup of a Circuit - Remarks

- The round complexity is quadratic.
  - more reliable
  - forward secrecy
- The circuit has a direction:
  - Forward (Alice to Bob): First, Alice iteratively encrypts a message several times; then, each OR decrypts away one layer.
  - Backward (Bob to Alice): Each OR would encrypt one layer; then, Alice iteratively decrypts the incoming message.
- An OR would maintain several circuits, hence the necessity of circuit IDs in the above.

How does the encryption/decryption work in Tor?

Before that, let's first have a look at Tor's packet format:

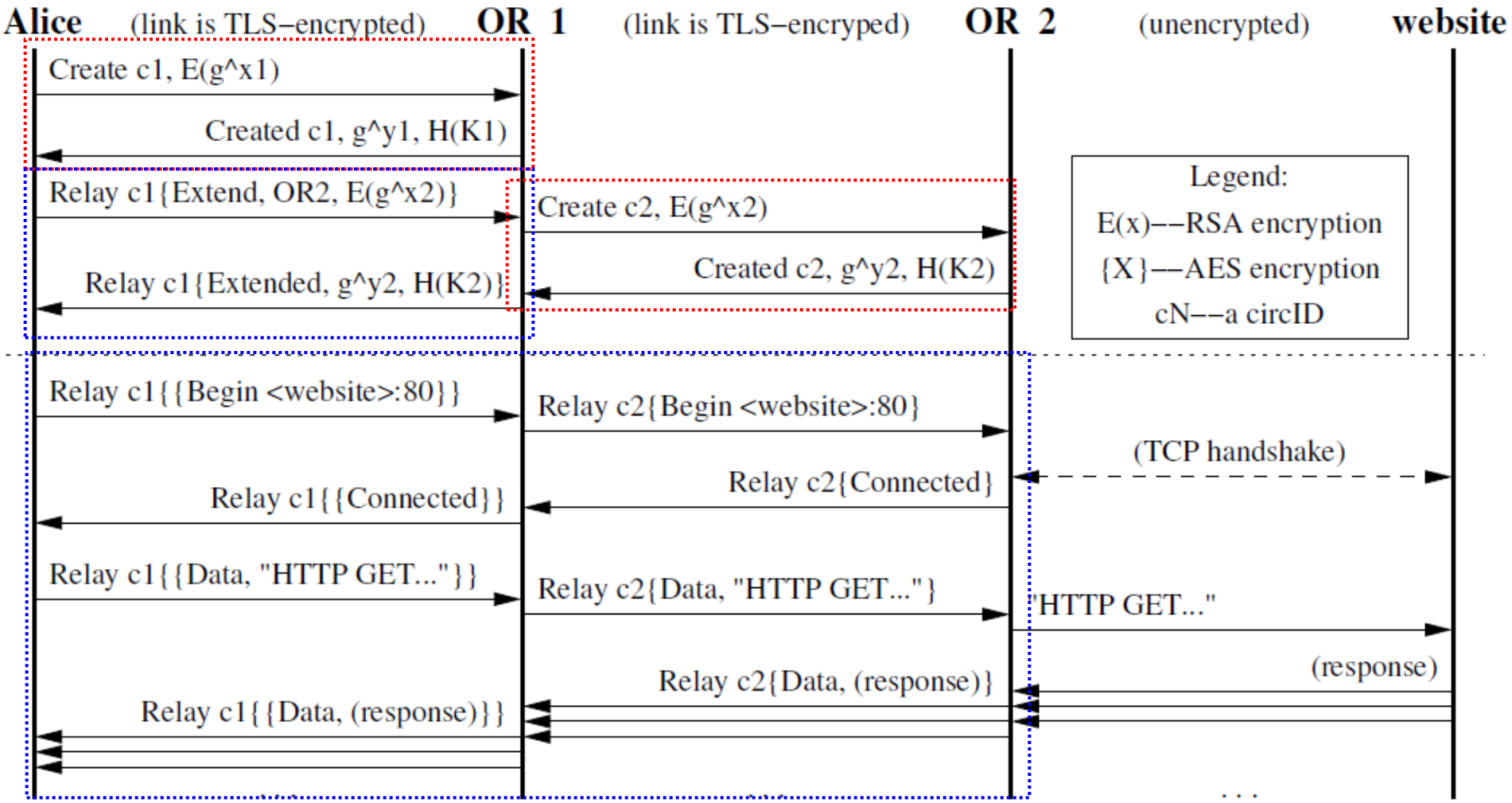


encrypted until the last hop

- Two kinds of cells:
  - Control Cells: when CMD is not RELAY. (The upper picture)
    - Examples in the previous slides: CREATE, CREATED
    - Different semantics/processing for different commands.
  - Relay Cells: when CMD is RELAY. (The lower picture)
    - The processing is the same (more on this later)

create cell

relay cell



Back to Our Example

- Say Alice has constructed a 2-OR circuit. Now she wants to connect to a web server, Bob.
- She would:
  - set the subCMD to "RELAY BEGIN".
  - set the streamID to all-zero.
  - the digest appropriately (More later)
  - include in the DATA a random streamID and the identity of Bob
- She then encrypts the above 509 bytes by k2 and k1.
- Finally she sets the circID as already constructed; and the CMD to "RELAY".

Processing of a relay cell

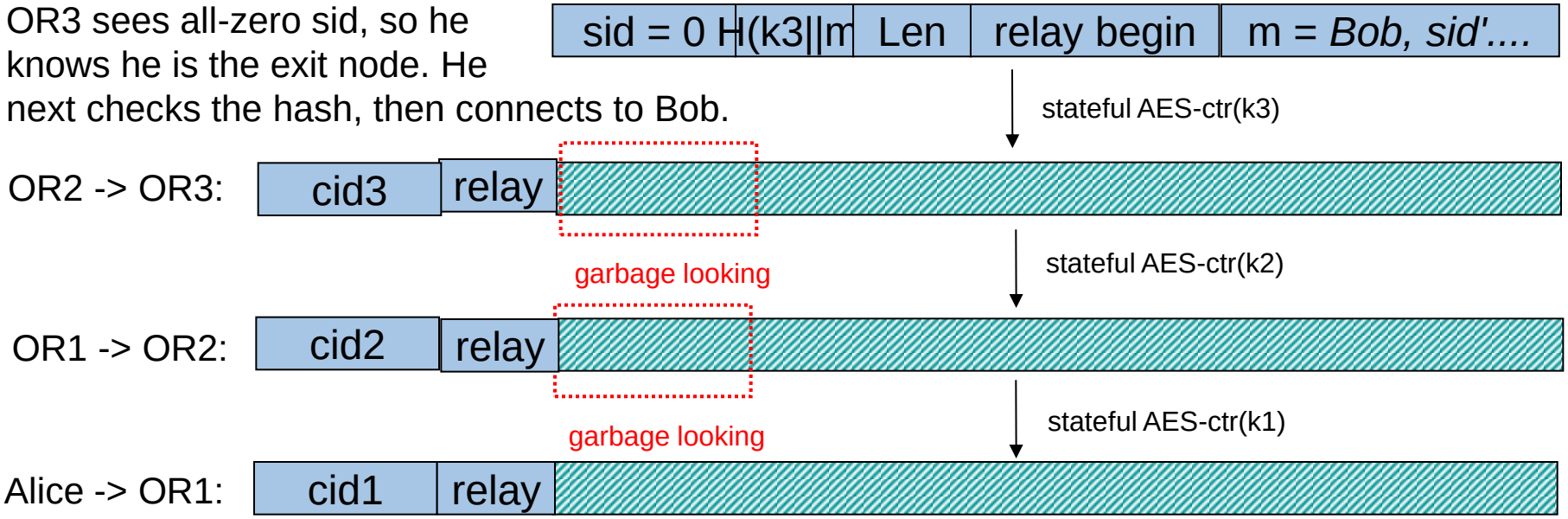
- first decrypts the DATA part.
- If streamID is all-zero:
  - extract the actual streamID and the identity of the server from DATA.
  - record that streamID on that circuit.
  - initiate a normal TCP connection to the server.
- Else if streamID is one that has been recorded as above, relay the data to the server accordingly.
- Else, relay the data further down the circuit.

End-to-end integrity

- The digest field is set to the first 4-bytes of a hash of the concatenation of the shared key and all relay cell's data flowing through this OR so far.
- ORs and OPs maintain such a digest for each circuit, and update them accordingly upon receipt of relay cells.
- **end-to-end integrity, not hop to hop!**

A Graphical Example

OR3 sees all-zero sid, so he knows he is the exit node. He next checks the hash, then connects to Bob.



Possible Future Work

- Formalization of the circuit construction phase:
  - non-UC (UC is really complicated!!!!!!!!)
  - game-based, like the formalization of AKE by Bellare, Rogaway, et.al.
  - capture the notion of "no more than the identities of predecessor and successor should be revealed", and more.
- Formalization of the onion-style encryption applied in Tor.
  - Basic challenge: no message expansion allowed.
  - Is it possible to ensure hop-to-hop integrity in this case?