The Onion Routing (TOR) – Cryptography and Anonymity in routing

Marek Kumpošt

xkumpost@fi.muni.cz

Literature

- "Tor: The Second-Generation Onion Router", in Proceedings of the 13th USENIX Security Symposium, August 2004. (http://www.onion-router.net/Publications/tor-design.pdf)
- "Anonymous Connections and Onion Routing," IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection, 1998. (http://www.onion-router.net/Publications/JSAC-1998.pdf)
- Onion Routing Home Page – www.onion-router.net

Motivation for anonymity

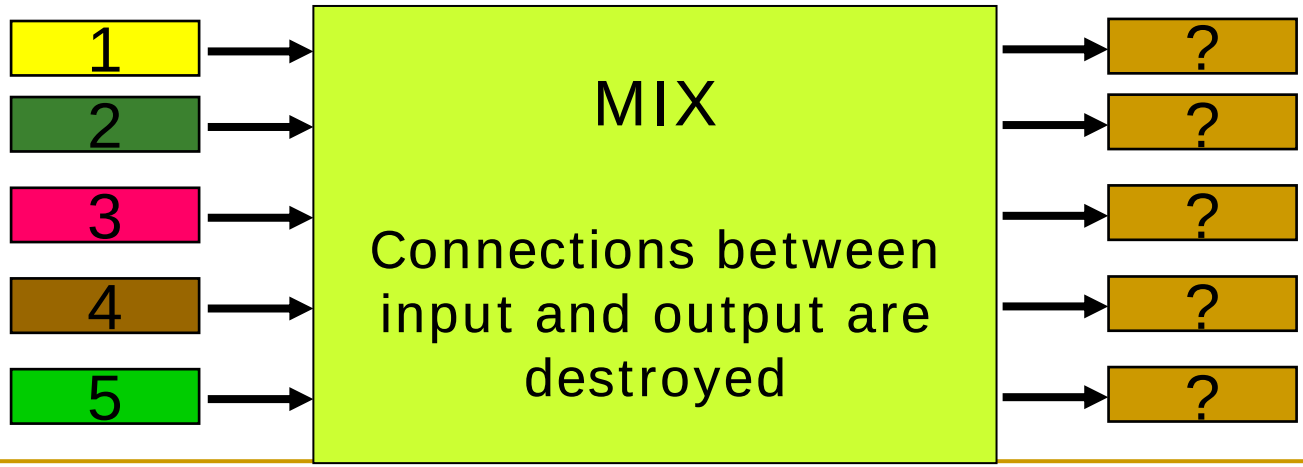
- privacy protection
- user's, location, transaction anonymity
- anonymity is one part of systems for privacy protection
  - pseudonymity
  - unlinkability
  - unobservability
- when we need to ensure anonymity?
  - information about health
  - electronic elections
  - …

Mix systems

- network traffic is observable and data is connected to its originator
- mixes are routers that changes data flow
  - input can not be mapped to output
  - content of a message is secured
  - data flow is changed (delays, messages are shuffled, dummy traffic)
- anonymous communication networks
  - mix networks
  - peer-to-peer systems

Types of mixes

- Chaum's threshold mix (1981)
  - collects N messages
  - shuffles them
  - sends messages (flush)



Types of mixes (2)

- depends on message processing algorithm
  - pool mixes (extension of the original design)
    - messages are stored in local memory (pool)
    - messages are processed in batches
    - different conditions for sending messages
    - different approaches for selecting messages from pool
  - stop-and-go mixes
    - messages are delayed by the mix
    - problems with low data flow in the network

Mixminion

- www.mixminion.net
- for sending anonymous emails
  - user specify a route through the mix network
- SURB – Single Use Reply Block
  - used if answering to an anoymous email
  - limited validity
  - routing is encrypted in SURB
  - replies can not be distinguished from normal messages

Introduction to Onion Routing

- What is Onion Routing?
  - system for private communication over a public network
  - system for providing bi-directional anonymous connection
  - provides near real-time anonymous connection for various services in the Internet
  - freely available system
- TOR
  - second-generation Onion Routing system

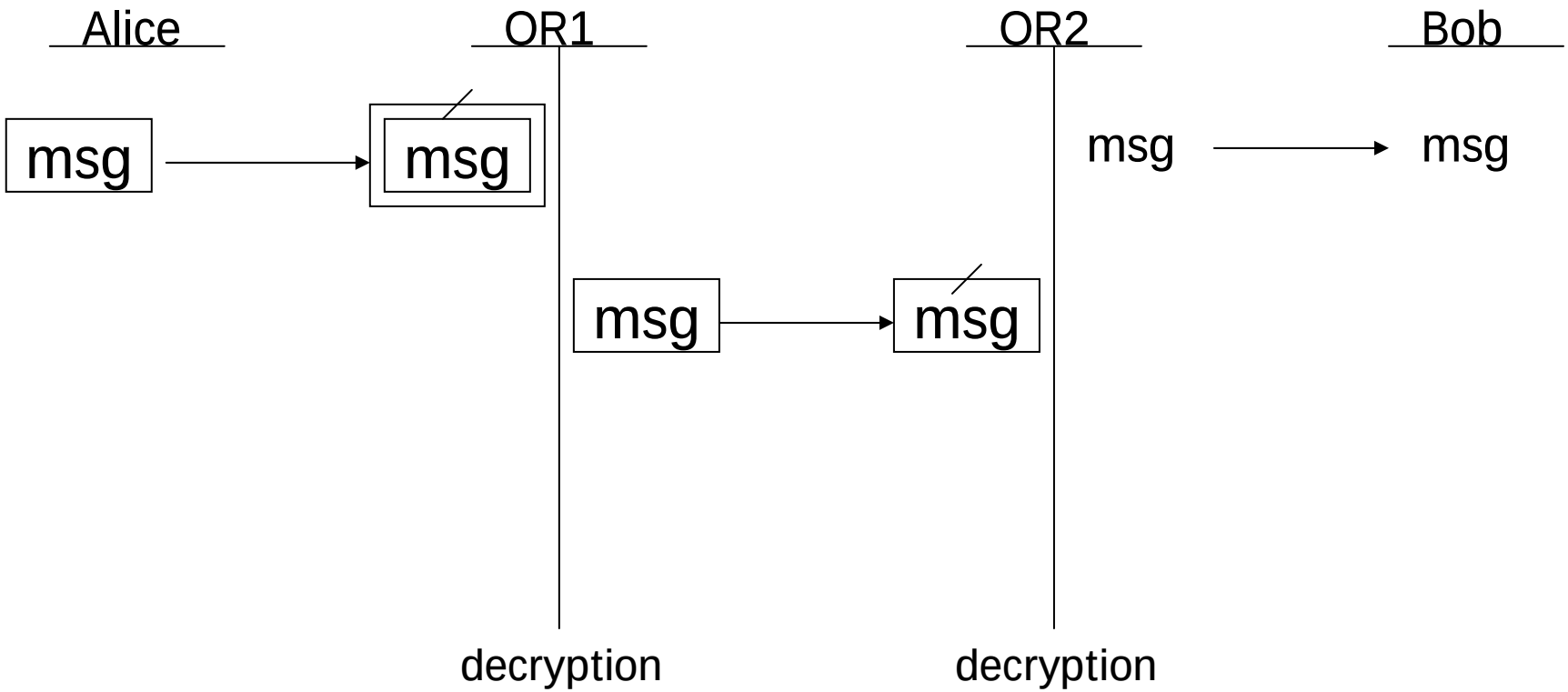
Overview of Onion Routing

- Why do we need onion routing?
  - encrypted messages can still be tracked, revealing who is talking to whom – care of message *context*, not only *content* protection (traffic analysis)
  - users may not wish to disclose their identity to the rest of the world
  - there is a need for a protocol that can relay traffic from various Internet services anonymously without modifying these services (SSH, RLogin, web browsing, Virtual Private Networks, …)
    - it works as a proxy

Remove identifying information

- filtering data before sending it to network
  - removing all identifying information about the originator of the data
  - attacker is unable to learn anything about the participants of the communication
  - traffic analysis is not possible
- data about identity must be passed as ordinary data through the anonymous connection

Message processing scheme



Data processing (1)

- Through a sequence of Onion Routers (OR) instead of direct connection to the responder
  - ORs network allows anonymous connection between client and server over a public network
  - each OR knows only its predecessor and successor
  - ORs in the network are connected by long-standing (permanent) connections
  - communication route is strictly defined at connection setup

Data processing (2) – proxies

- OR network is accessed through series of proxies
  - application makes a connection to a application proxy
  - application proxy transforms the messages to a specific form that is accepted by OR network
  - application proxy makes a connection to Onion Proxy which establish a communication circuit
  - the circuit can then carry users data

Data processing (3) – circuit

- proxy constructs layered data structure – onion and sends it to the network (PK cryptography is used in this step)
- each OR peels off one layer of the message, takes keys seed material for generating symmetric key, and pass the massage to the next hop
- last onion router forwards data to the „responder“
- the connection is now established

Data processing (4)

- every OR keeps track of received onions until they expire
  - payload of expired onions is not forwarded
  - they cannot be used to uncover the route information
- data are encrypted using stream ciphers
  - data will look differently each time it passes through a properly operating OR

TOR

- TOR is a circuit-based low-latency anonymous communication service
  - TOR is a second-generation Onion Routing system
  - the original Onion Routing protocol design has not been updated for years
- TOR provides following improvements over the old Onion routing design
  - perfect forward secrecy
  - separation of „protocol cleaning“ from anonymity
  - many TCP streams can share one circuit
  - leaky-pipe circuit topology
  - congestion control

TOR enhancements

- TOR provides following improvements over the old Onion routing design
  - directory servers
  - variable exit policies
  - end-to-end integrity checking
  - rendezvous points and hidden services
  - does not require OS kernel patches
  - TOR is also available under a free license

Perfect forward secrecy

- in the original OR design, a single hostile node could record traffic, attacker then should compromise successive nodes and force them to decrypt that traffic
- TOR uses *telescopic* path-building design instead of single multiply encrypted onions
  - initiator negotiates session keys with each node in the circuit path
  - once these keys are deleted, subsequently compromised nodes cannot decrypt old traffic
- the whole process of building circuits is now more reliable

Separation of “protocol cleaning”

- original design required separate application proxy for each supported application protocol
  - most of them were never written
- TOR uses the standard SOCKS proxy interface
  - supports most TCP-based programs without modification

TCP streams circuit sharing Leaky-pipe circuit topology

- many TCP streams can share one circuit
  - original OR built a separate circuit for each application
  - many public key operations for every request
  - building many communication circuits
  - in TOR design many streams can share one circuit
- leaky-pipe circuit topology
  - senders can direct traffic to any node in the circuit
  - allows traffic to exit the circuit from the middle
  - attacker can catch nothing if observing the end of circuit

Directory servers
Variable exit policies

- directory servers
  - old design – flooding state information through the net
  - TOR – some more trusted nodes act as *directory servers* (DS)
  - DSs provide information about known routers and their current state (users get this information via HTTP)
- variable exit policies
  - each router advertise policy describing the hosts and ports to which it will connect
  - user can decide which node will be the exit node

End-to-end integrity checking Rendezvous points and hidden services

- original Onion Routing did no integrity checking
  - nodes on the circuit could change the data (tagging attacks)
  - TOR – verifies data integrity before it leaves the network
  - the integrity depends on all traffic between A and B
- Rendezvous points and hidden services
  - for providing responder anonymity
  - old design used long-lived “reply onions“
  - TOR – client negotiates *rendezvous point* to connect with hidden servers
  - prevents DoS attacks on hidden servers

TOR design goals

- deployability – system will be deployed in the real world
  - not expensive to run (e.g. bandwidth requirement)
  - not be difficult or expensive to implement (by requiring OS kernel modifications)
- usability
  - hard-to-use system => only few users => less anonymity
  - anonymity systems hide users among users
  - usability is therefore a security requirement
  - not require modifying applications, no delays, easily implementable on all common platforms

TOR design goals (2)

- flexibility
  - protocol must be flexible and well-specified
- simple design
  - the protocol’s design and security parameters must be well-understood
  - TOR aims to deploy a simple and stable system that integrates the best accepted approaches to protecting anonymity

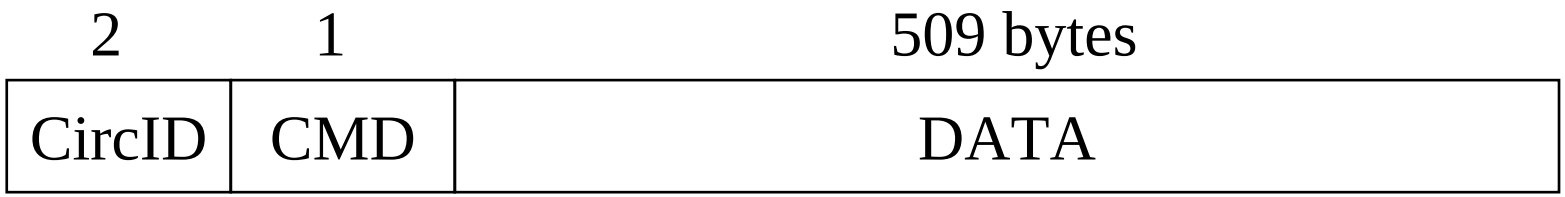
The TOR design

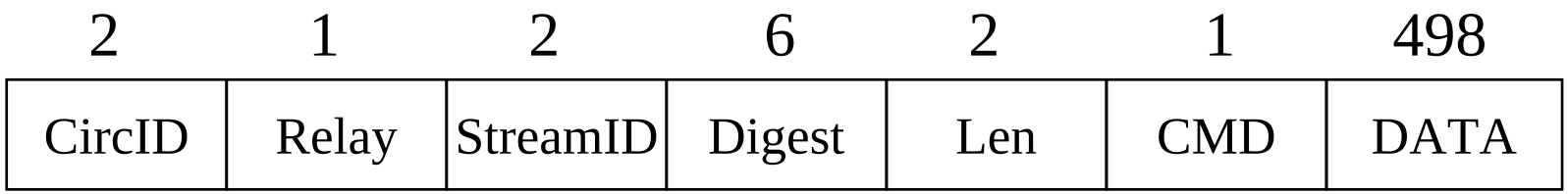
- ORs run as a normal user-level process without any special privileges
- each OR maintains a TLS connection to every other onion router
- each user runs local Onion Proxy (OP)
  - establish circuits, handles connection from user appl.
- each OR maintains long-term identity key (PK) and a short-term onion key (PK)
  - ID key – for signing the ORs *router descriptor* (a summary of its keys, bandwidth, exit policy,…)
  - OK – for decrypting set up circuit requests

The TOR design – cells

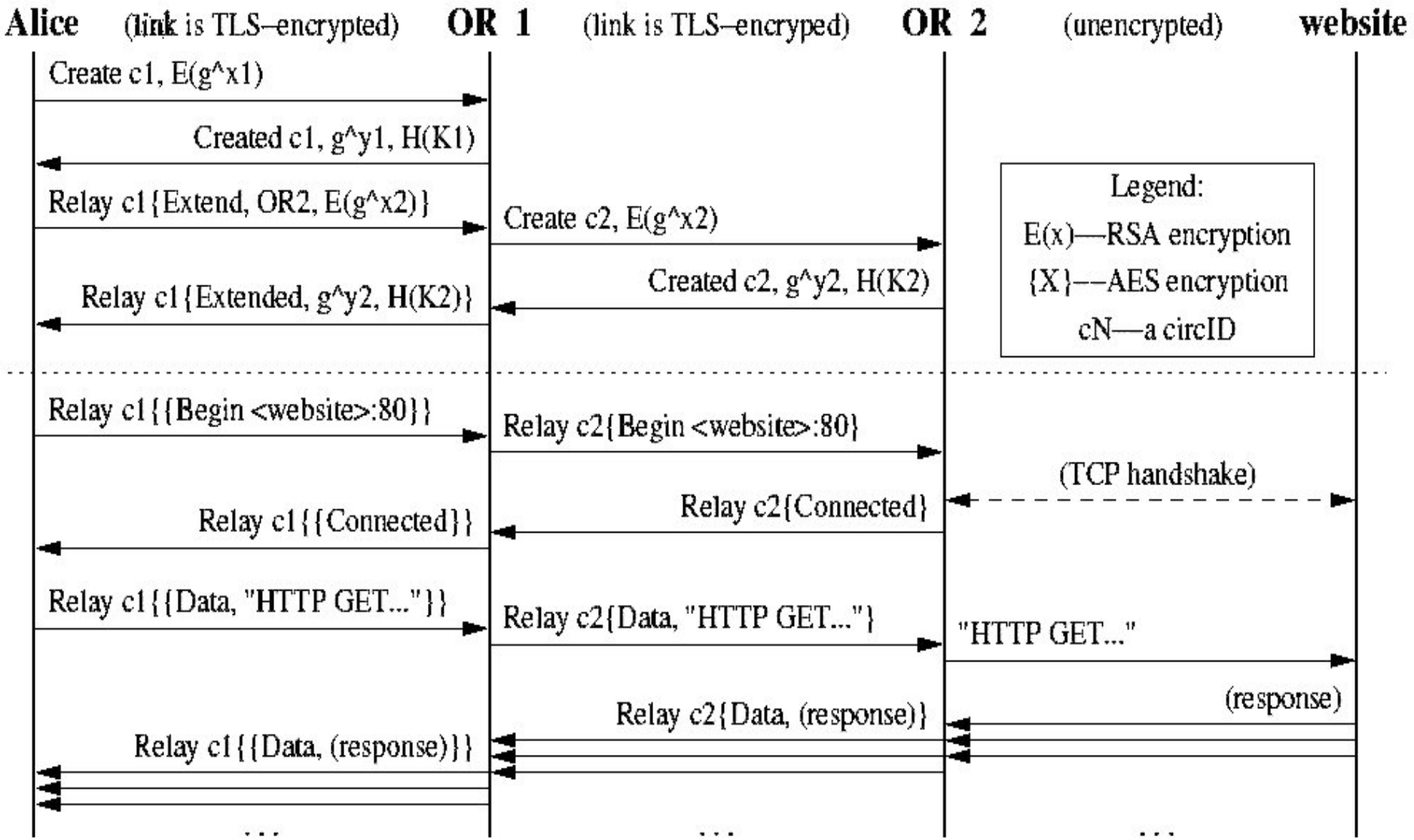
- traffic passes in fixed-size cells
- cell – 512 bytes (header and payload)
  - header includes circID and command
  - *control* cells (interpreted by node), *relay* cells (end-to-end data)
  - control cells commands are:
    - *padding* (to keepalive the connection)
    - *create* or *created* (used to set up a new circuit)
    - *destroy* (to destroy a circuit)
  - relay cells contains streamID, end-to-end checksum, length of the payload and a relay command

The TOR design – cells (2)





The TOR design – circuit



Rendezvous point – main idea

- used for *location-hidden services* (responder anonymity)
- allows responder to offer a service without revealing his IP address
- protects against DoS attacks
  - attackers are forced to attack the OR network
- the main goals are:
  - access-control – filtering the incoming traffic
  - robustness – maintain long-term pseudonymous identity even in the case of router failure (migration)
  - application-transparency – users must run special software but they don’t have to modify their applications

Rendezvous point – main idea (2)

- responder is allowed to advertise several onion routers – *introduction points* as contact points
- sender chooses an OR as his *rendezvous point*
- sender connect to one of responder's introduction point informs him about rendezvous point
- wait for responder to connect to the rendezvous point
  - responder can respond to some requests and ignore others
- sender and recipient can communicate via OR network

Any questions?