Secure Transport Protocols

Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria (DEI)
Politecnico di Milano

alessandro.barenghi - at - polimi.it

Common Use scenarios

- After analyzing all the required cryptographic primitives (symmetric and asymmetric ciphers, hashes), we can tackle:
  1. The TLS secure transport protocol (HTTPS and Secure Mail Transport)
  2. The Secure SHell (SSH) protocol (secure interactive terminal access)
  3. The Tor Onion Routing protocol (Transport anonymity)
- In all cases the bulk of the data is encrypted with an ephemeral key with a block cipher
- The ephemeral key is agreed via an asymmetric key cipher
- All protocols employ public key authentication mechanisms to provide authenticated connections

TLS (SSL) - Basics and Standards

Description and use scenario

- The Secure Socket Layer (SSL) suite was designed back in 1995 by Netscape communications to provide a transparent secure socket layer, and published starting from the v2.0
- In 1999 the IETF standardised the (Transport Layer Security) TLS v1.0, patching a number of security issues in SSL v2.0 and enhancing the security margin of SSL v3.0
- TLS has updated since then and has finally come to v1.2, encompassing EC based cryptography
- SSLv3 and TLS are similar, but, due to some details, they do not interoperate. TLS allows connection downgrades to SSLv3

TLS (SSL) - Basics and Standards

Overview

- TLS v1.2 (RFC5246) employs a hybrid encryption scheme:
  - Public key ciphers are employed to provide authentication and exchange securely an ephemeral session key
  - Secret key ciphers are employed to provide high throughput data transmission, employing the session key as a shared secret
- X.509 certificates are employed to provide authentication to the communication endpoints to avoid impersonation attacks
- The TLS transport layer provides no enforced integrity guarantee, save when the mode of operation of the symmetric cipher does so
- Abuses of the missing integrity check have been exploited in practical attacks (e.g. BEAST), leading to a strong recommendation of authenticated encryption modes

TLS (SSL) - Basics and Standards

Ciphersuites

- The set of primitives employed by TLS is known as ciphersuite
- It is expressed as `TLS_A_WITH_B_C`, where `A,B,C` are:
  - `A` Asymm. cipher, used in authentication (e.g. `RSA`) during key exch.
    - Default kex performed via RSA, Diffie-Hellman (`DH_RSA`) and EC-based Diffie-Hellman (`ECDH_RSA`) available
    - Authentication can be skipped specifying `DH_anon` to have an unauthenticated Diffie-Hellman based kex
  - `B` Symm. cipher + mode of operation , used in data encryption, (e.g., `AES_GCM`, `3DES2_CBC`,`RC4`,`Salsa20`)
    - No padding strategy for block ciphers (random padding allowed)
    - `NULL` can be specified to employ TLS for endpoint auth. only
  - `C` Hash function, for MAC/signing (e.g. `SHA256`, `SHA` $\rightarrow$ SHA-1)
    - The TLS standard advises against `MD2`,`MD4`,`MD5` only after v1.1
  - Example of a ciphersuite : `TLS_DHE_RSA_WITH_AES_256_CBC_SHA256`

Practical key sizes

A brief primer on security margins

- A proper security margin is ensured choosing a large enough keyspace for all the ciphers involved
- The table obtained from the estimates at `www.keylength.com` by both academia and national standard bodies

Safe up to year	Secret/Private Key length or Digest length [b] RSA	ECDSA	DSA (key/group)	AES Equiv.	Hash
1986	512	111	122/515	60	128
2010	1024	144	144/1024	80	160
2030	2048	224	224/2048	112	224
2060	4440	256	256/4440	128	256
Fores. Future	26268	512	512/26268	256	512

TLS (SSL) - Server only authentication

Single sided authentication

1. [CLI $\rightarrow$ SRV] **ClientHello** message w/ highest TLS version supported and the supported cipher suites in clear
2. [SRV $\rightarrow$ CLI] **ServerHello** message w/ the protocol version and ciphersuite choice, and a session random number (nonce) in clear
3. [SRV $\rightarrow$ CLI] `X.509 Certificate`, and a **ServerHelloDone** message
4. [CLI,local] Checks the certificate, [CLI $\rightarrow$ SRV] **ClientKeyExchange** message with `premasterkey` enciphered with the server's public key
5. [SRV and CLI,local] Compute `session key` from `nonces` and `premasterkey`
6. [CLI $\rightarrow$ SRV] **ChangeCipherSpec** message, notifying that the communication will be encrypted and **Finished** message, containing a MAC of all the previous messages ()
7. [SRV,local] decrypts the **Finished** message and [SRV $\rightarrow$ CLI] an identical one

TLS (SSL) - Mutual authentication

Mutual Authentication

- In the previous scheme, only the client authenticates the server identity: TLS supports also mutually authenticated communications
- The mutual authentication is achieved through the client sending his own certificate before the **ClientKeyExchange** message
- After sending the **ClientKeyExchange** message, the client sends a **CertificateVerify** message, which is a signature over all the previous messages, allowing the server to check that the one who initiated the communication was effectively the client
- In this case, both the server and the client can close the connection if the identity of the other is not the correct one preventing impersonation attacks

TLS (SSL) - Issues and Solutions

Common Issues

- Issue: Public key based key exchange is expensive, especially if there are many sessions in a short amount of time
- Solution: fast session resume: the two endpoints exchange only the **Hello** messages, with the random nonce of the session they want to resume and switch to encrypted mode.
  - The authentication between the parties is implicitly handled by the fact that they both know the same session secret
- Issue: Long term private key leak implies that all the previous communications can be decrypted: no forward secrecy
- Solution: Enclose the secret session key in a Diffie-Hellman key exchange protocol, allowing the DH parameters to be renewed by the server. This is known as Ephemeral Diffie-Hellman (`DHE`).

TLS - Uses

Common Uses

- HTTPS: HTTPS is basically plain HTTP traffic carried by an SSL/TLS transport layer
- Stunnel: The Stunnel utility provides a TLS transport wrapper for any protocol you like acting as a portmapper
- Secure SMTP/IMAP/POP3: all these protocols can (and should) be used with an TLS transport layer
- OpenVPN: OpenVPN exploits TLS (instead of IpSEC or PPTP) as the transport protocol for point-to-point VPNs

TLS - Common pitfalls

Issues and mitigation

- Weak ciphersuites open up to attacks: Don't use DES (small key) , RC2 (small key), RC4 (broken in 2013 in the TLS use scenario)
- An attacker may force a downgrade of the ciphersuite altering the **ClientHello** message: Remove from accepted ciphersuites the vulnerable ones altogether
- Cipher renegotiation allows an attacker to splice a request into a TLS setup: a fix is standardized in RFC5746 and implemented in recent OpenSSL/GNUTLS/NSS
- The Heartbleed bug: affecting OpenSSL, it is not a TLS weakness, it's simply a CS-101 programming error (missing check on the size of a memcpy): get a fixed implementation
- When possible, use an authenticated mode of operation (e.g., GCM) and TLS v1.2 to fix padding-based attacks (e.g. BEAST)

SSH - Basics

Overview

- The Secure SHell Protocol suite was designed at first (1995) to protect remote login/shell applications from eavesdropping (i.e. provide confidentiality)
- In the years it has grown to a full fledged protocol suite (encompassing 15 RFC) with separated layers for authentication, transport and multiple connection handling
- Most popular implementations (OpenSSH and Putty) support advanced features such as providing a forwarding tunnel for a TCP connection flow and a dedicated file transfer protocol (SFTP)
- SSH uses a hybrid (asymmetric+symmetric) encryption scheme similar to TLS, while allowing different methods for authentication

SSH - Session authentication

Machine Authentication Protocol

- SSH provides a simple server endpoint authentication through asking the user whether the fingerprint of the server public key is trusted to be the one of the server
- If the user acknowledges that the server key is authentic, the key is added to a local storage, and considered to be authentic for all the future connections
- The local key storage is a simple text file with one IP address-public key pair per line (have a look at `~/.ssh/known_hosts`)
- Trust revocation is simply performed via removing the offending key from the storage file

SSH - Session authentication

User Authentication Protocol

- Once the server to which the client is connecting is authenticated, SSH authenticates the user which wants to log in
- The authentication of the user typing on client host can be performed with four methods (RFC4252):
  - **Password**: a simple password is sent encrypted with the public key of the server: the server checks if the password is the correct one
  - **Public Key**: at bootstrap time, the user saves in a secure manner a copy of its personal public key on the server he wants to connect to. The authentication is simply performed through verifying a user-signed nonce on the server with the public key previously stored
  - **Host Based**: The user is authenticated by signing a nonce with the host wide private key, the check is done with the host public key
  - **Delegated Authentication**: via PAM (e.g. to use Kerberos)
  - **None**: RFC4252 Explicitly notes that this is not recommended.

SSH - Key material and its storage

Key handling features

- SSH implementations support both GPG and X.509 standard certificates for key storage seamlessly
- The keys of a user are typically stored in a directory within his home directory on the system (`~/.ssh` on Linux, pub keys ending in `.pub`)
- The stored keys are:
  - the known keys, server keys which have been recognised as authentic
  - the authorised keys to authenticate the user on the host at login
  - the actual keypairs the user may employ when using the SSH client
- Using Pluggable Authentication Modules (PAM) allows SSH implementations to use a smartcard, secure token or other delegates to perform authentication
  - Secure tokens require the server to have a corresponding PAM module requesting a signature to them

Authentication

Kerberos

- Willing to reduce the number of passwords and still perform sound authentication, the Kerberos system was designed at MIT in late '80s [1]
- After refinement the fifth version of Kerberos became standardised in RFC1510 which has been currently superseded by RFC4120 (drop from block cipher set single DES and add AES)
- The Kerberos protocol authenticates users registered on a centralized authentication point and thus share with it a *common secret*
- The whole authentication procedure is performed without the use of public key primitives and is thus very fast
- Kerberos is based on authentication proofs known as *tickets*

Kerberos - Pros and Cons

- Pro: The user needs to remember/type a single password
- Con: single point of failure for the authentication: the AS *must* always be available or the whole authentication mechanism freezes (usually coped with through keeping multiple AS as backups)
- Con: Due to the ticket timestamps, all the machines in the authentication domain should be synchronized[a]: this is usually coped with through the use of the (cleartext, unauthenticated) Network Time Protocol to perform synchronizations
- The password administration protocol is not standardised, although a proposal is described in RFC3244

[a] default configuration mandates a lag among them lesser than 5 minutes

Mixing Networks

Anonymous network access

- Goal: provide anonymous access to contents via Internet
- First attempt: anonymizing proxies: a simple L5 proxy not performing any caching and exposing his own identity (i.e. IP address)
- Main issues: single point of failure, relies on the trustworthyness of the actual anonymizing proxy
- Further attempts made:
  - First generation P2P mixing networks (e.g. Tarzan): everyone may either be a router or be an actual endpoint, traffic bounced around
  - High latency mixing networks (e.g. Babel, MixMaster): Seek to hide everything, including content fruition latencies, from an eavesdropper, encrypt fully the traffic, insert long artificial delays

Second Generation Mixing networks

Tor: a low-latency, onion routing mixing network

- First generation mixing networks were trying to solve multiple problems (content anonymization, source anonymization, latency hiding) with a single approach
- In the light of the ideas and shortcomings of the first generation was designed Tor [3, 2, **?**]
- Designed to be:
  - Reliable in transport: Tor provides anonymization of TCP data flows
  - Low latency: it should be possible to employ it for interactive workloads
  - Easy to deploy: no modifications to the transport/network stack
  - Providing confidentiality and integrity of transmitted data
  - Perfect forward secrecy: avoid issues with routers seized by malicious entities
- Not designed to be: concealing latencies, performing application-layer content anonymization (e.g. User-Agent removal in HTTP)

Tor - Basics

Tor actors

- **Relays**: The actual nodes of the mixing network, they relay the information around, all communications among them are performed over TLS. They are typically known as Onion Relays (OR)
- **Clients**: A client connects to the Tor network via a so-called Onion Proxy (OP), sets up a circuit for communication choosing randomly the relays it should pass through, and uses them to communicate with another endpoint, the target, outside the Tor Network
- **Exit point**: A relay which agrees to act as the last relay of the communication and communicates with the target endpoint

Tor - Communication

Tor communication principles

- A circuit is a transport path in the Tor network, akin to a physical wire: it can be built or torn down by an OP
- All Tor communications are on top of TLS (= contained as the payload of a TLS connection) acting as blind "piping"
- TLS performs only server authentication, no client authentication
- Circuits are designed to be built telescopically, i.e. acknowledging each hop added to them: they need to be involving at least 3 relays
- The basic transport unit in Tor is the cell:
  - control cells: they relay control messages for circuit setup and teardown
  - relay cells: they are the actual cells carrying the user data to be relayed
- Each cell contains a 2 byte circuit ID: the relays can multiplex many circuits over a single TLS connection

Tor - Communication

Tor Circuit Setup (2 hops long for the sake of clarity)

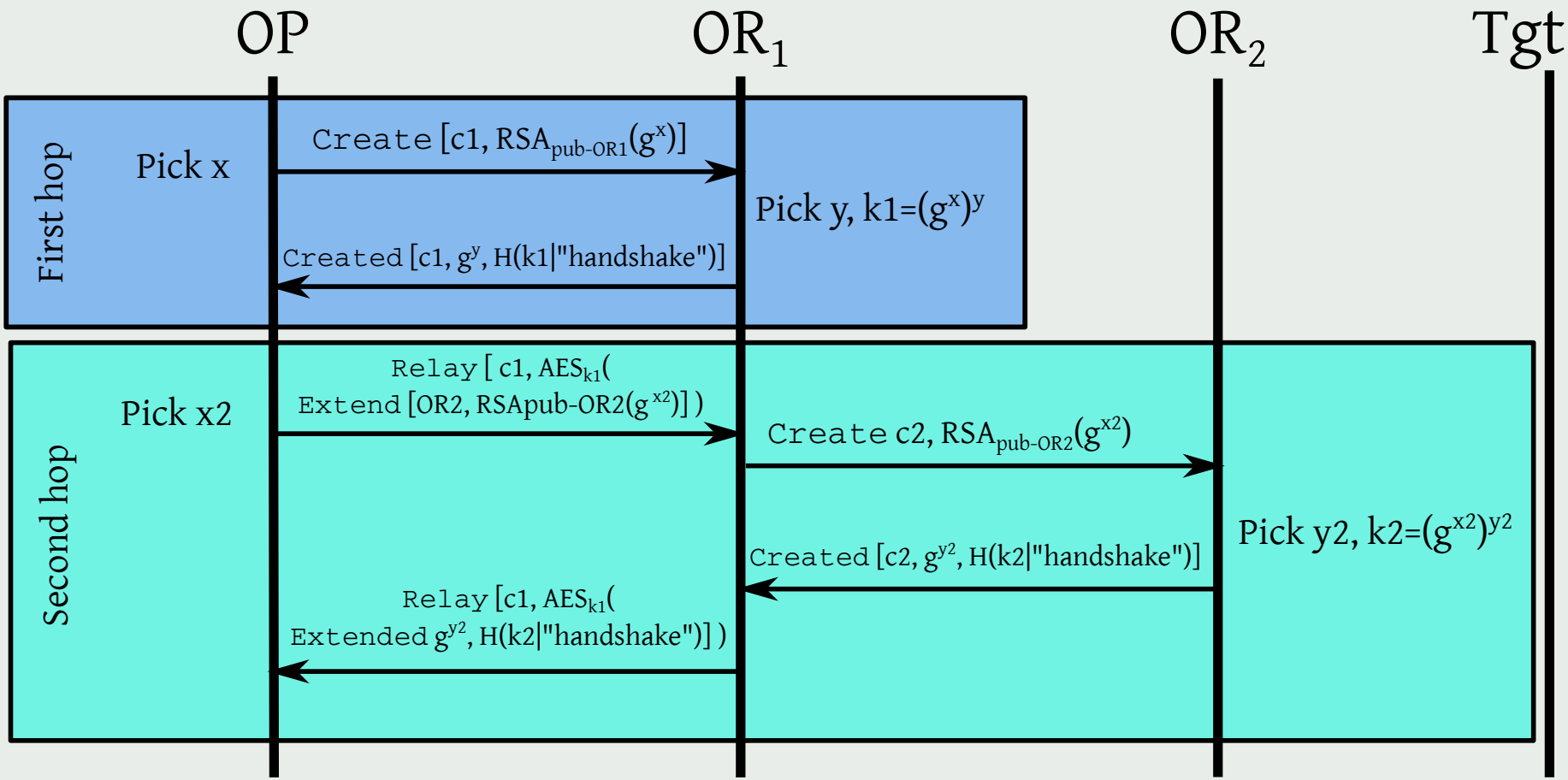


Figure: c1,c2 cell IDs; commands in `teletype font`, cell content within [ ]

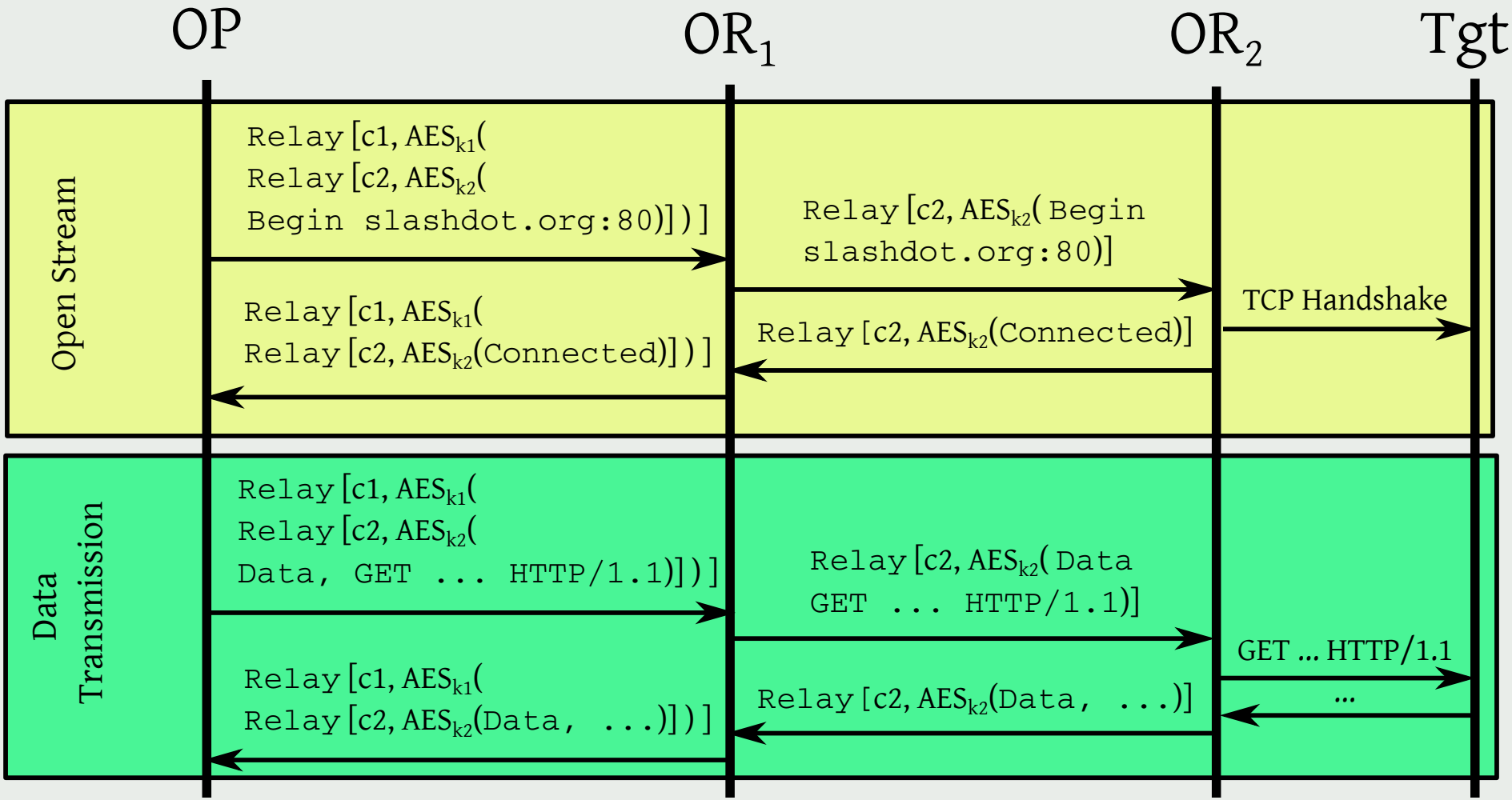
Tor - Communication

Tor Circuit Characteristics

- Every OR knows the network identity of its neighbours only
- Only the OP has full knowledge of the ORs involved in the circuit!
- Long term RSA OR public keys are made available to the OP via a set of trusted OR which act as key directories
  - They mantain a list of the active Tor ORs, together with their long term public keys, thus acting as notaries
  - Tor supports EC-Based asymmetric crypto for higher efficiency
- Once a circuit is set up, it requires an active teardown procedure to be destroyed (akin to TCP Fin-Fin/Ack-Ack)
  - This allows to reuse the same circuit for more than a single TCP connection of the OP, to reduce latencies
  - The OP is the one in charge of asking for a circuit teardown

Tor - Communication

Tor Data Relay (2 hops circuit for the sake of clarity)



Tor - Communication

Tor Data Relay

- The client opens a communication with the target via a `begin` SOCKS command, which is encrypted multiple times with the symmetric keys of the OR in the circuit
- Each OR peels away a layer of the encryption, and executes the command specified in the cell which he has just decrypted
- Only the exit node knows the actual content of the transmission, however he does not known who generated such content
- Note that Tor is not designed to solve user content confidentiality: data exiting from the Tor network are not automatically encrypted
- An external eavesdropper only sees TLS traffic among the different ORs and the OP, with the exit nodes communicating with the targets: no inference can be made on the source of such communication, unless the attacker controls both the entry point and the exit node

Bibliography I

Clifford Neuman and Theodore Ts'o.

Kerberos: An Authentication Service for Computer Networks.
IEEE Communications 32 (9): 33-8, also available at
http://gost.isi.edu/publications/kerberos-neuman-tso.html.

The Tor Project.

Tor Protocol Specification.
https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt.

The Tor Project.

Tor Website.
https://www.torproject.org/.