Tor's Circuit-Layer Cryptography

Attacks, Hacks, and Improvements

Isis Agora Lovecruft
Core Developer, The Tor Project

October 13th, 2016

University of Waterloo

Why should you care about privacy?

"There is an entire genre of YouTube videos devoted to an experience which I'm certain that everyone in this room has had. It entails an individual, who, thinking they're alone, engages in some expressive behaviour – wild singing, girating dancing, some mild sexual activity – only to discover that, in fact, they are not alone, that there's a person watching and lurking, the discovery of which causes them to immediately cease what they're doing in horror. The sense of shame and humiliation in their face is palpable: it's the sense of 'this is something I'm willing to do only if no one else is watching.' This is the crux of the work on which I have been singularly focused for the sixteen months: the question of why privacy matters."

—Glenn Greenwald, TED Talk, October 2014

Introduction to Tor

Background: Anonymising proxies

- Typically application-specific proxies, e.g. HTTP proxies, or generic request-based proxies, e.g. SOCKS proxies
- Requests to online services come from the proxy
- Users behind the proxy should be indistinguishable
- Proxies can be chained together
- Various problems:
  1. Single point-of-failure

Background: Anonymising proxies

- Typically application-specific proxies, e.g. HTTP proxies, or generic request-based proxies, e.g. SOCKS proxies
- Requests to online services come from the proxy
- Users behind the proxy should be indistinguishable
- Proxies can be chained together
- Various problems:
    1. Single point-of-failure
    2. Relatively trivial to correlate ingoing/outgoing traffic

Background: Anonymising proxies

- Typically application-specific proxies, e.g. HTTP proxies, or generic request-based proxies, e.g. SOCKS proxies
- Requests to online services come from the proxy
- Users behind the proxy should be indistinguishable
- Proxies can be chained together
- Various problems:
  1. Single point-of-failure
  2. Relatively trivial to correlate ingoing/outgoing traffic
  3. (Usually) no crypto protecting the connection between the user and the proxy

Background: Anonymising proxies

- Typically application-specific proxies, e.g. HTTP proxies, or generic request-based proxies, e.g. SOCKS proxies
- Requests to online services come from the proxy
- Users behind the proxy should be indistinguishable
- Proxies can be chained together
- Various problems:
  1. Single point-of-failure
  2. Relatively trivial to correlate ingoing/outgoing traffic
  3. (Usually) no crypto protecting the connection between the user and the proxy
- Can add cryptographic or obfuscational features to the proxy, e.g. The Tor Project's "Pluggable Transports" ...

Background: Anonymising proxies

- Typically application-specific proxies, e.g. HTTP proxies, or generic request-based proxies, e.g. SOCKS proxies
- Requests to online services come from the proxy
- Users behind the proxy should be indistinguishable
- Proxies can be chained together
- Various problems:
  1. Single point-of-failure
  2. Relatively trivial to correlate ingoing/outgoing traffic
  3. (Usually) no crypto protecting the connection between the user and the proxy
- Can add cryptographic or obfuscational features to the proxy, e.g. The Tor Project's "Pluggable Transports" ...
- ...this (usually) still does not solve problems 1 and 2

Background: Anonymising proxies

- Typically application-specific proxies, e.g. HTTP proxies, or generic request-based proxies, e.g. SOCKS proxies
- Requests to online services come from the proxy
- Users behind the proxy should be indistinguishable
- Proxies can be chained together
- Various problems:
  1. Single point-of-failure
  2. Relatively trivial to correlate ingoing/outgoing traffic
  3. (Usually) no crypto protecting the connection between the user and the proxy
- Can add cryptographic or obfuscational features to the proxy, e.g. The Tor Project's "Pluggable Transports" ...
- ... this (usually) still does not solve problems 1 and 2

Designs like anonymising proxies which place ultimate trust in any single node in the network cannot provide any strong guarantees to anonymity, because these single points-of-failure can be exploited—legally or otherwise—to deanonymise users.

Background: Mix Networks

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.

Background: Mix Networks

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.
- There should exist some well-known public key for each mix.

Background: Mix Networks

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.
- There should exist some well-known public key for each mix.
- Messages are split up into blocks and encrypted to the mix's public key.

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.
- There should exist some well-known public key for each mix.
- Messages are split up into blocks and encrypted to the mix's public key.
- The first few blocks conceptually contain some "headers", which are stripped at each hop, then some random cruft is added to the end of each message.

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.
- There should exist some well-known public key for each mix.
- Messages are split up into blocks and encrypted to the mix's public key.
- The first few blocks conceptually contain some "headers", which are stripped at each hop, then some random cruft is added to the end of each message.
- Each mix only knows the nodes immediately before and after it, making the network resistant to malicious mix nodes.

Background: Mix Networks

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.
- There should exist some well-known public key for each mix.
- Messages are split up into blocks and encrypted to the mix's public key.
- The first few blocks conceptually contain some "headers", which are stripped at each hop, then some random cruft is added to the end of each message.
- Each mix only knows the nodes immediately before and after it, making the network resistant to malicious mix nodes.
- Supposed to achieve *bitwise unlinkability* between the source of the message and the destination, making it difficult for an adversary to trace end-to-end communications end-to-end.

Background: Mix Networks

Mix networks are routing protocols that create hard-to-trace communications by using a chain of nodes, known as mixes, which receive messages from multiple senders, shuffle them in some manner, and send them to the next destination.

Chaum's Original Mix Networks

- Idea for anonymous electronic mail by David Chaum, in 1981.
- There should exist some well-known public key for each mix.
- Messages are split up into blocks and encrypted to the mix's public key.
- The first few blocks conceptually contain some "headers", which are stripped at each hop, then some random cruft is added to the end of each message.
- Each mix only knows the nodes immediately before and after it, making the network resistant to malicious mix nodes.
- Supposed to achieve *bitwise unlinkability* between the source of the message and the destination, making it difficult for an adversary to trace end-to-end communications end-to-end.
- Message shuffling in order to achieve unlinkability.

Problems with Chaum's Original Mix Network Scheme

- Pfitzmann and Pfitzmann (1990) demonstrated that Chaum's original work did not acheive the desired unlinkability property.

Problems with Chaum's Original Mix Network Scheme

- Pfitzmann and Pfitzmann (1990) demonstrated that Chaum's original work did not acheive the desired unlinkability property.
- *Tagging attacks* are possible: each encrypted message block, using RSA, is not dependent on those before or after it, and thus can be substituted or reused.

Problems with Chaum's Original Mix Network Scheme

- Pfitzmann and Pfitzmann (1990) demonstrated that Chaum's original work did not acheive the desired unlinkability property.
- *Tagging attacks* are possible: each encrypted message block, using RSA, is not dependent on those before or after it, and thus can be substituted or reused.
- Most of Chaum's work was done in the late 1970s. Unsurprisingly, it used RSA in ways now known to be unsafe, i.e. without padding and applying the modular exponations directly to messages.

Problems with Chaum's Original Mix Network Scheme

- Pfitzmann and Pfitzmann (1990) demonstrated that Chaum's original work did not acheive the desired unlinkability property.
- *Tagging attacks* are possible: each encrypted message block, using RSA, is not dependent on those before or after it, and thus can be substituted or reused.
- Most of Chaum's work was done in the late 1970s. Unsurprisingly, it used RSA in ways now known to be unsafe, i.e. without padding and applying the modular exponations directly to messages.
- Because the RSA operation was applied directly, an adversary could trick a mix into signing a message by applying the decryption operation.

Problems with Chaum's Original Mix Network Scheme

- Pfitzmann and Pfitzmann (1990) demonstrated that Chaum's original work did not acheive the desired unlinkability property.
- *Tagging attacks* are possible: each encrypted message block, using RSA, is not dependent on those before or after it, and thus can be substituted or reused.
- Most of Chaum's work was done in the late 1970s. Unsurprisingly, it used RSA in ways now known to be unsafe, i.e. without padding and applying the modular exponations directly to messages.
- Because the RSA operation was applied directly, an adversary could trick a mix into signing a message by applying the decryption operation.
    - This attack could be further hidden from the mix by applying a signature blinding technique.

Problems with Chaum's Original Mix Network Scheme

- Pfitzmann and Pfitzmann (1990) demonstrated that Chaum's original work did not acheive the desired unlinkability property.
- *Tagging attacks* are possible: each encrypted message block, using RSA, is not dependent on those before or after it, and thus can be substituted or reused.
- Most of Chaum's work was done in the late 1970s. Unsurprisingly, it used RSA in ways now known to be unsafe, i.e. without padding and applying the modular exponations directly to messages.
- Because the RSA operation was applied directly, an adversary could trick a mix into signing a message by applying the decryption operation.
  - This attack could be further hidden from the mix by applying a signature blinding technique.
  - To be fair, Chaum invented RSA blind signing two years later, in 1983.

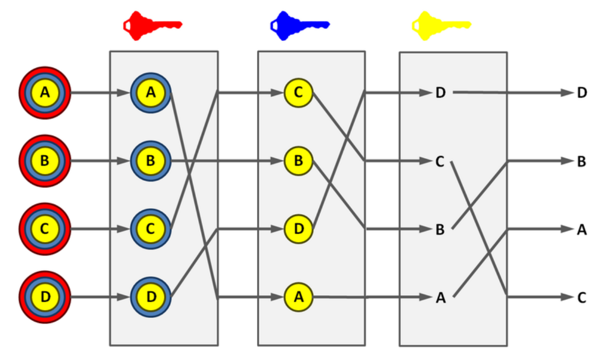
Background: Mix Network Designs – Cascading Mixes

Chaum noted that relying on only one mix is not resilient against malicious nodes, so the function of mixing should distributed. Mixes can be chained to ensure that, even if just one of them remains honest, some anonymity is provided.

Background: Mix Network Designs – Cascading Mixes

Chaum noted that relying on only one mix is not resilient against malicious nodes, so the function of mixing should distributed. Mixes can be chained to ensure that, even if just one of them remains honest, some anonymity is provided.

First proposed way to chain mixes together is called *cascade mixing*, and uses all nodes in the network, in a specific order (grey boxes), each of which shuffles the order of outgoing messages:



The second way is to allow users to arbitrarily select which mixes their message will pass through, and is what is now generally referred to as a *mix network*.

The second way is to allow users to arbitrarily select which mixes their message will pass through, and is what is now generally referred to as a *mix network*.

This design has some problems:

- Berthold, Pfitzmann, and Standtke (2000) argue that mix networks do not offer some properties that cascades offer.

The second way is to allow users to arbitrarily select which mixes their message will pass through, and is what is now generally referred to as a *mix network*.

This design has some problems:

- Berthold, Pfitzmann, and Standtke (2000) argue that mix networks do not offer some properties that cascades offer.
- They illustrate a number of attacks to show that, if *only one* mix is honest in the network, the anonymity of the messages going through it can be compromised.

The second way is to allow users to arbitrarily select which mixes their message will pass through, and is what is now generally referred to as a *mix network*.

This design has some problems:

- Berthold, Pfitzmann, and Standtke (2000) argue that mix networks do not offer some properties that cascades offer.
- They illustrate a number of attacks to show that, if *only one* mix is honest in the network, the anonymity of the messages going through it can be compromised.
- These attacks rely on compromised mixes which exploit some knowledge of their position in the chain . . .

Mix Network Designs: Mix Networks

The second way is to allow users to arbitrarily select which mixes their message will pass through, and is what is now generally referred to as a *mix network*.

This design has some problems:

- Berthold, Pfitzmann, and Standtke (2000) argue that mix networks do not offer some properties that cascades offer.
- They illustrate a number of attacks to show that, if *only one* mix is honest in the network, the anonymity of the messages going through it can be compromised.
- These attacks rely on compromised mixes which exploit some knowledge of their position in the chain ...
- ... or multiple messages using the same sequence of mixes through the network.

Background: Mix Networks vs. Anonymous Proxies

Mix Networks

Anonymous Proxies

Background: Mix Networks vs. Anonymous Proxies

Mix Networks

+ No single point-of-failure (with cascading)
+ Generally strong anonymity guarantees
+ Inbound/outbound traffic analysis does not deanonymise

Anonymous Proxies

Background: Mix Networks vs. Anonymous Proxies

Mix Networks

+ No single point-of-failure (with cascading)
+ Generally strong anonymity guarantees
+ Inbound/outbound traffic analysis does not deanonymise
– High latency
– Slow public-key cryptography

Anonymous Proxies

Background: Mix Networks vs. Anonymous Proxies

Mix Networks

+ No single point-of-failure (with cascading)
+ Generally strong anonymity guarantees
+ Inbound/outbound traffic analysis does not deanonymise
− High latency
− Slow public-key cryptography

Anonymous Proxies

+ Low latency
+ Fast, symmetric cryptography

Background: Mix Networks vs. Anonymous Proxies

Mix Networks

+ No single point-of-failure (with cascading)
+ Generally strong anonymity guarantees
+ Inbound/outbound traffic analysis does not deanonymise
− High latency
− Slow public-key cryptography

Anonymous Proxies

+ Low latency
+ Fast, symmetric cryptography
− Single point-of-failure
− No strong anonymity guarantees
− Inbound/outbound traffic analysis may deanonymise

Background: Mix Networks vs. Anonymous Proxies

Mix Networks

+ No single point-of-failure (with cascading)
+ Generally strong anonymity guarantees
+ Inbound/outbound traffic analysis does not deanonymise
– High latency
– Slow public-key cryptography

Anonymous Proxies

+ Low latency
+ Fast, symmetric cryptography
– Single point-of-failure
– No strong anonymity guarantees
– Inbound/outbound traffic analysis may deanonymise

Onion Routing: Combine the advantages of each system

- Use (non-cascading) mix (called a Tor *circuit*) of proxies, which are called Tor *relays* or Tor *nodes*
- Use asymmetric cryptography for establishing an (one-way) authenticated, encrypted channel, then use fast symmetric cryptography.

What is Tor?

Tor is an anonymity network, which uses onion routing to encapsulate client traffic in a manner such that each node in the client's chosen path only knows the destinations before and after it.

How Tor Works: Directory Authorities and Consensus Protocol

Tor uses a semi-centralised design, in which certain specific nodes, called *Directory Authorities* are trusted ultimately.

How Tor Works: Directory Authorities and Consensus Protocol

Tor uses a semi-centralised design, in which certain specific nodes, called *Directory Authorities* are trusted ultimately.

- The Directory Authorities participate in a voting protocol to decide upon a canonical decision regarding the nodes within the network.

How Tor Works: Directory Authorities and Consensus Protocol

Tor uses a semi-centralised design, in which certain specific nodes, called *Directory Authorities* are trusted ultimately.

- The Directory Authorities participate in a voting protocol to decide upon a canonical decision regarding the nodes within the network.
- They vote upon their views of the network, and eventually derive a *consensus* document which is distributed to clients.

How Tor Works: Directory Authorities and Consensus Protocol

Tor uses a semi-centralised design, in which certain specific nodes, called *Directory Authorities* are trusted ultimately.

- The Directory Authorities participate in a voting protocol to decide upon a canonical decision regarding the nodes within the network.
- They vote upon their views of the network, and eventually derive a *consensus* document which is distributed to clients.
- Despite the misleading name, “consensus” documents are created by majority vote.

How Tor Works: Consensus Retrieval

Initial Setup

- Directory Authority public keys are compiled into the client software.

How Tor Works: Consensus Retrieval

Initial Setup

- Directory Authority public keys are compiled into the client software.
- Client uses one of these keys to establish an encrypted connection to a Directory Authority.

How Tor Works: Consensus Retrieval

Initial Setup

- Directory Authority public keys are compiled into the client software.
- Client uses one of these keys to establish an encrypted connection to a Directory Authority.
- Client downloads the consensus from the Directory Authority and checks the Directory Authorities signatures on the document.

Initial Setup

- Directory Authority public keys are compiled into the client software.
- Client uses one of these keys to establish an encrypted connection to a Directory Authority.
- Client downloads the consensus from the Directory Authority and checks the Directory Authorities signatures on the document.
- Client creates a list of all relays within the consensus, weighted by the relays' bandwidths.

How Tor Works: Consensus Retrieval

Initial Setup

- Directory Authority public keys are compiled into the client software.
- Client uses one of these keys to establish an encrypted connection to a Directory Authority.
- Client downloads the consensus from the Directory Authority and checks the Directory Authorities signatures on the document.
- Client creates a list of all relays within the consensus, weighted by the relays' bandwidths.
- Client chooses a relay from the weighted list to act as its *Guard* relay. This Guard will be the client's entry into the network for some set amount of time (currently approximately 2 months).

How Tor Works: Consensus Retrieval

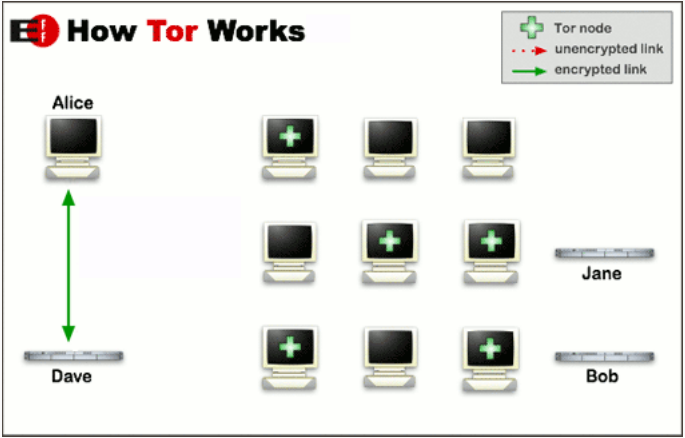
Initial Setup

- Directory Authority public keys are compiled into the client software.
- Client uses one of these keys to establish an encrypted connection to a Directory Authority.
- Client downloads the consensus from the Directory Authority and checks the Directory Authorities signatures on the document.
- Client creates a list of all relays within the consensus, weighted by the relays' bandwidths.
- Client chooses a relay from the weighted list to act as its *Guard* relay. This Guard will be the client's entry into the network for some set amount of time (currently approximately 2 months).

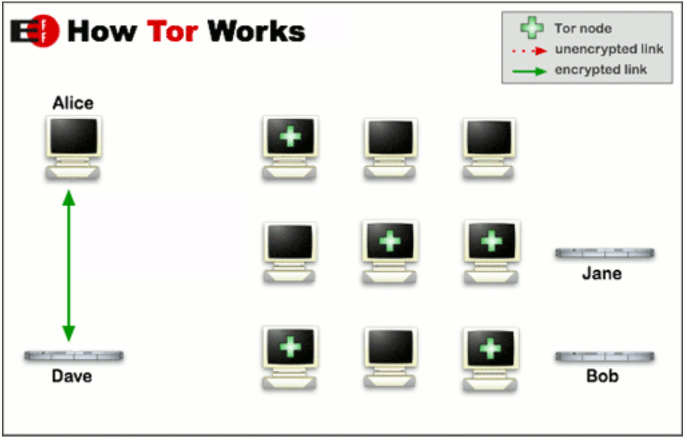
We currently require that all clients know about all valid nodes in the Tor network, in order to safeguard against *partitioning attacks* where an adversary uses a client's partial knowledge of the network topology in some manner to gain some advantage (usually to increase feasibility of further attacks, e.g. a correlation attack).

When a new *stream* is created (e.g. some data to be transmitted over Tor has arrived), a circuit is either chosen from a list of pre-constructed circuits, or a new circuit is created as needed. Using the same bandwidth-weighted list (as before), the Client selects a *Middle* relay and an *Exit* relay for the new circuit.

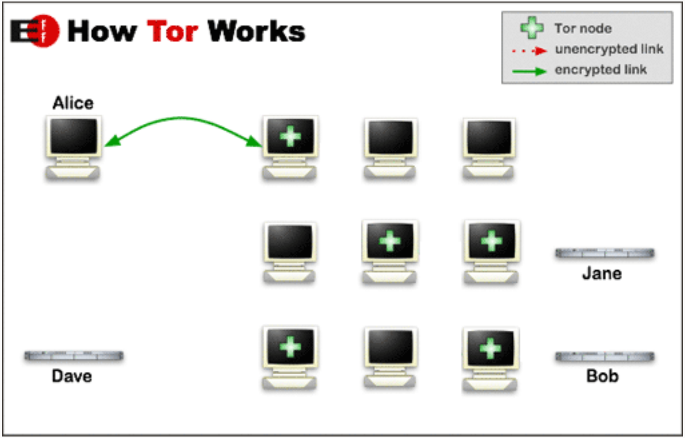
Establishing a Circuit



Request consensus from Directory Authorities (DirAuths)

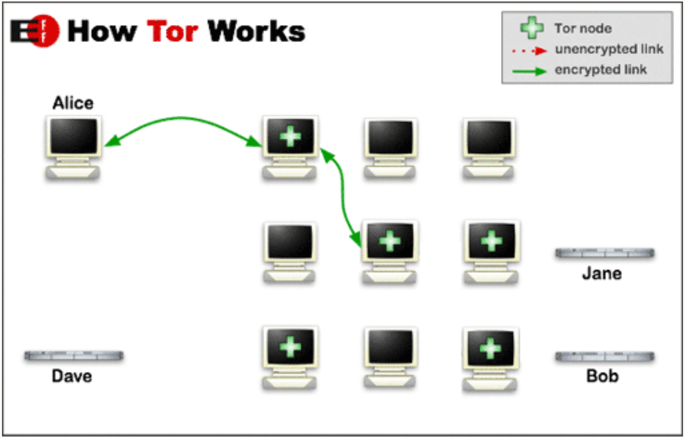


Pick entry, middle, and exit node; obtain their public keys from directory mirror (DirServ)



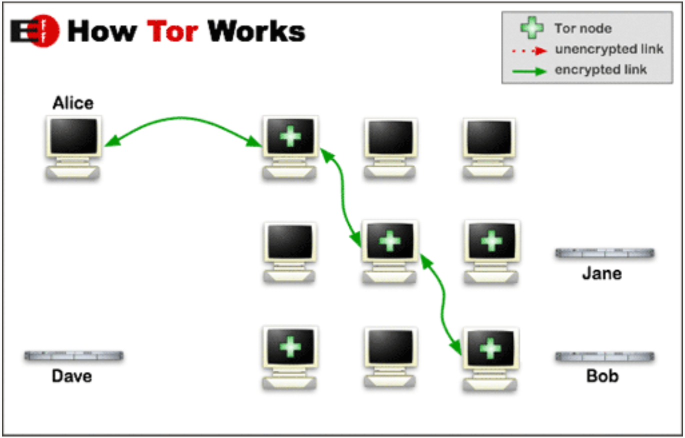
Exchange symmetric key with entry node (Diffie-Hellman)

Establishing a Circuit

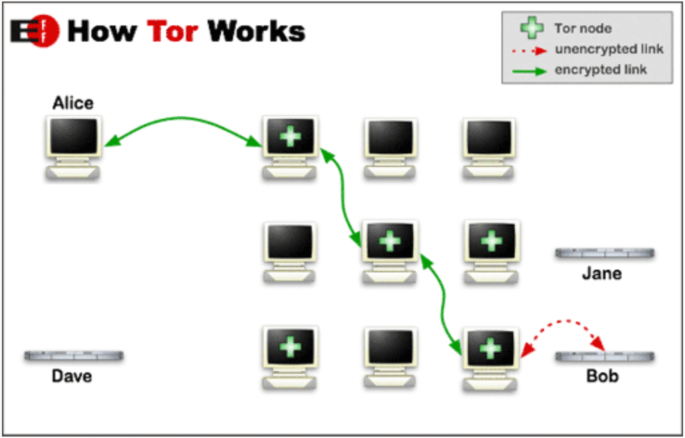


Exchange key with middle node (tunnelled through entry node)

Establishing a Circuit



Exchange key with exit node (tunnelled through middle node, tunnelled through entry node)



Communicate with Bob

Circuit Extension to the Middle Relay

- A TLS connection to the Guard $\mathbf{R_1}$ is established, $\mathbf{TLS_{R_1}}$.

Circuit Extension to the Middle Relay

- A TLS connection to the Guard $\mathbf{R_1}$ is established, $\mathbf{TLS_{R_1}}$.
- Through $\mathbf{TLS_{R_1}}$, the Client does a circuit-level handshake to setup shared keys with the Guard ($\mathbf{R_1}$) for the forward and backward paths, $\mathbf{KF_{R_1}}$ and $\mathbf{KB_{R_1}}$ respectively.

Circuit Extension to the Middle Relay

- A TLS connection to the Guard $\mathbf{R_1}$ is established, $\mathbf{TLS_{R_1}}$.
- Through $\mathbf{TLS_{R_1}}$, the Client does a circuit-level handshake to setup shared keys with the Guard ($\mathbf{R_1}$) for the forward and backward paths, $\mathbf{KF_{R_1}}$ and $\mathbf{KB_{R_1}}$ respectively.
- The Client next creates a **RELAY EXTEND** cell to extend the circuit to the Middle relay ($\mathbf{R_2}$) which contains the first stage of the circuit-level handshake with $\mathbf{R_2}$. It encrypts this relay cell with $\mathbf{KF_{R_1}}$ and sends it forward to $\mathbf{R_1}$, who decrypts with $\mathbf{KF_{R_1}}$ and packages the content into a **RELAY CREATE** cell, which is sent over a newly established TLS connection between R_1 and R_2, TLS_{R_2} who sends its half of the circuit handshake in response, packaged in a **RELAY CREATE** cell and reverse encrypted (with the corresponding $\mathbf{KB_i}$ keys) down the reverse path.

Circuit Extension to the Exit Relay

- After the Client's handshake with the Middle relay (R_2) completes, the Client creates another **RELAY EXTEND** cell to extend the circuit to the Exit relay, R_3. This is then tunneled over $\mathbf{TLS_{R_2}}$ (which is tunneled through $\mathbf{TLS_{R_1}}$). The cell itself is super-encrypted with $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{KF_{R_1}}, \mathbf{CELL}))$.

Circuit Extension to the Exit Relay

- After the Client's handshake with the Middle relay (R_2) completes, the Client creates another **RELAY EXTEND** cell to extend the circuit to the Exit relay, R_3. This is then tunneled over $\mathbf{TLS_{R_2}}$ (which is tunneled through $\mathbf{TLS_{R_1}}$). The cell itself is super-encrypted with $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{KF_{R_1}}, \mathbf{CELL}))$.
- This cell is sent to R_1, who decrypts with $\mathbf{KF_{R_1}}$ and sends it along to R_2. R_2 decrypts with $\mathbf{KF_{R_2}}$, sees that it's a **RELAY EXTEND** cell to R_3, packages the content into a **RELAY CREATE** cell (as R_1 did before), and sends it to R_3.

Circuit Extension to the Exit Relay

- After the Client's handshake with the Middle relay (R_2) completes, the Client creates another **RELAY EXTEND** cell to extend the circuit to the Exit relay, R_3. This is then tunneled over $\mathbf{TLS_{R_2}}$ (which is tunneled through $\mathbf{TLS_{R_1}}$). The cell itself is super-encrypted with $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{KF_{R_1}}, \mathbf{CELL}))$.
- This cell is sent to R_1, who decrypts with $\mathbf{KF_{R_1}}$ and sends it along to R_2. R_2 decrypts with $\mathbf{KF_{R_2}}$, sees that it's a **RELAY EXTEND** cell to R_3, packages the content into a **RELAY CREATE** cell (as R_1 did before), and sends it to R_3.
- The Exit relay R_3 receives this **RELAY CREATE** cell, does $Dec(\mathbf{KF_{R_3}}, \mathbf{CELL})$ and receives the traffic the client had intended to proxy (which is hopefully further encrypted with some application-layer encryption, e.g. TLS, SSH, etc).

How Tor Works: Relay Cells on the Forward Path

- Assumption: all relays in the network have well-known public keys

How Tor Works: Relay Cells on the Forward Path

- Assumption: all relays in the network have well-known public keys
- Use relay public keys to setup an authenticated and encrypted channel, which is used to establish symmetric keypairs for the *forward* and *reverse paths*:
  - Entry relay R_1 (keys KB_{R_1}, KF_{R_1})
  - Middle relay R_2 (keys KB_{R_2}, KF_{R_2})
  - Exit relay R_3 (keys KB_{R_3}, KF_{R_3})

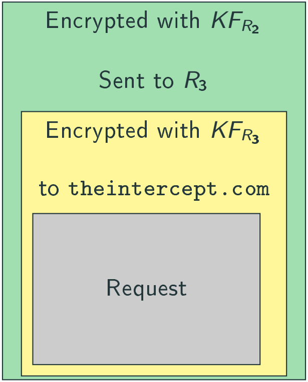
How Tor Works: Relay Cells on the Forward Path

- Assumption: all relays in the network have well-known public keys
- Use relay public keys to setup an authenticated and encrypted channel, which is used to establish symmetric keypairs for the *forward* and *reverse paths*:
    - Entry relay R_1 (keys KB_{R_1}, KF_{R_1})
    - Middle relay R_2 (keys KB_{R_2}, KF_{R_2})
    - Exit relay R_3 (keys KB_{R_3}, KF_{R_3})
- Wants to anonymously send request to `theintercept.com`

How Tor Works: Relay Cells on the Forward Path

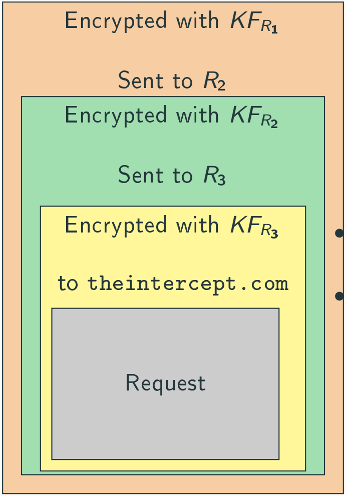
- Assumption: all relays in the network have well-known public keys
- Use relay public keys to setup an authenticated and encrypted channel, which is used to establish symmetric keypairs for the *forward* and *reverse paths*:
    - Entry relay R_1 (keys KB_{R_1}, KF_{R_1})
    - Middle relay R_2 (keys KB_{R_2}, KF_{R_2})
    - Exit relay R_3 (keys KB_{R_3}, KF_{R_3})
- Wants to anonymously send request to `theintercept.com`
- Prepares Tor *relay cell* as follows:
    - Create request for `theintercept.com` and encrypt with KF_{R_3}

How Tor Works: Relay Cells on the Forward Path



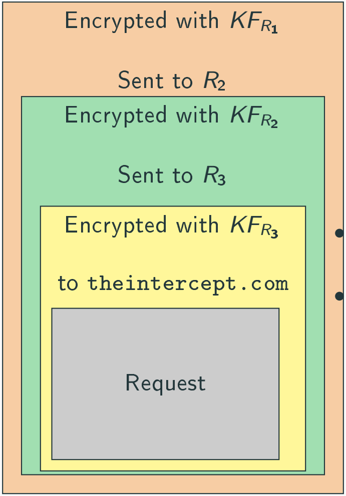
- Assumption: all relays in the network have well-known public keys
- Use relay public keys to setup an authenticated and encrypted channel, which is used to establish symmetric keypairs for the *forward* and *reverse paths*:
  - Entry relay R_1 (keys KB_{R_1}, KF_{R_1})
  - Middle relay R_2 (keys KB_{R_2}, KF_{R_2})
  - Exit relay R_3 (keys KB_{R_3}, KF_{R_3})
- Wants to anonymously send request to `theintercept.com`
- Prepares Tor *relay cell* as follows:
  - Create request for `theintercept.com` and encrypt with KF_{R_3}
  - Set destination as R_3 and encrypt with KF_{R_2}

How Tor Works: Relay Cells on the Forward Path



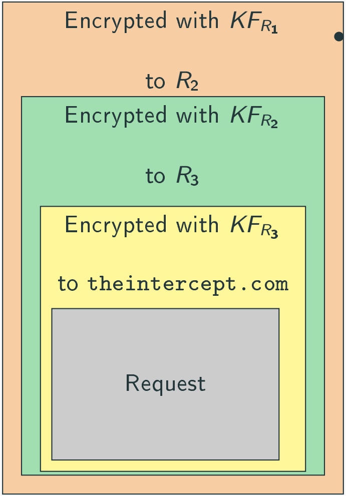
- Assumption: all relays in the network have well-known public keys
- Use relay public keys to setup an authenticated and encrypted channel, which is used to establish symmetric keypairs for the *forward* and *reverse paths*:
  - Entry relay R_1 (keys KB_{R_1}, KF_{R_1})
  - Middle relay R_2 (keys KB_{R_2}, KF_{R_2})
  - Exit relay R_3 (keys KB_{R_3}, KF_{R_3})
- Wants to anonymously send request to `theintercept.com`
- Prepares Tor *relay cell* as follows:
  - Create request for `theintercept.com` and encrypt with KF_{R_3}
  - Set destination as R_3 and encrypt with KF_{R_2}
  - Set destination R_2 and encrypt with KF_{R_1}

How Tor Works: Relay Cells on the Forward Path



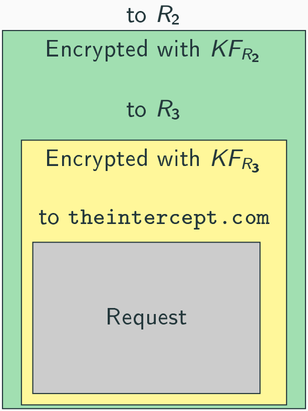
- Assumption: all relays in the network have well-known public keys
- Use relay public keys to setup an authenticated and encrypted channel, which is used to establish symmetric keypairs for the *forward* and *reverse paths*:
  - Entry relay R_1 (keys KB_{R_1}, KF_{R_1})
  - Middle relay R_2 (keys KB_{R_2}, KF_{R_2})
  - Exit relay R_3 (keys KB_{R_3}, KF_{R_3})
- Wants to anonymously send request to `theintercept.com`
- Prepares Tor *relay cell* as follows:
  - Create request for `theintercept.com` and encrypt with KF_{R_3}
  - Set destination as R_3 and encrypt with KF_{R_2}
  - Set destination R_2 and encrypt with KF_{R_1}

How Tor Works: Relay Cells on the Forward Path



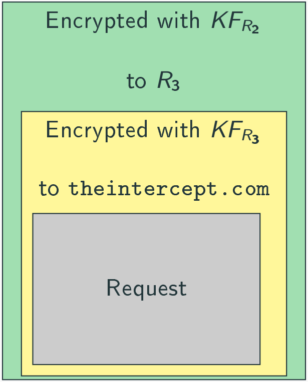
- R_1 receives packet, removes encryption with KF_{R_1}

How Tor Works: Relay Cells on the Forward Path



- R_1 receives packet, removes encryption with KF_{R_1}
- Sees next destination: R_2, forwards

How Tor Works: Relay Cells on the Forward Path



- R_1 receives packet, removes encryption with KF_{R_1}
- Sees next destination: R_2, forwards
- R_2 receives packet, removes encryption with KF_{R_2}

- R_1 receives packet, removes encryption with KF_{R_1}
- Sees next destination: R_2, forwards
- R_2 receives packet, removes encryption with KF_{R_2}
- Sees next destination: R_3, forwards

How Tor Works: Relay Cells on the Forward Path

- R_1 receives packet, removes encryption with KF_{R_1}
- Sees next destination: R_2, forwards
- R_2 receives packet, removes encryption with KF_{R_2}
- Sees next destination: R_3, forwards
- R_3 receives packet, removes encryption with KF_{R_3}

How Tor Works: Relay Cells on the Forward Path

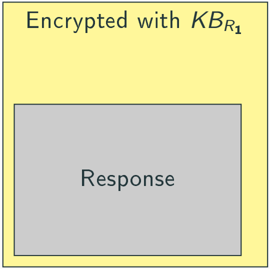
- R_1 receives packet, removes encryption with KF_{R_1}
- Sees next destination: R_2, forwards
- R_2 receives packet, removes encryption with KF_{R_2}
- Sees next destination: R_3, forwards
- R_3 receives packet, removes encryption with KF_{R_3}
- Sees next destination: `theintercept.com`, sends request

How Tor Works: Relay Cell on the Reverse Path

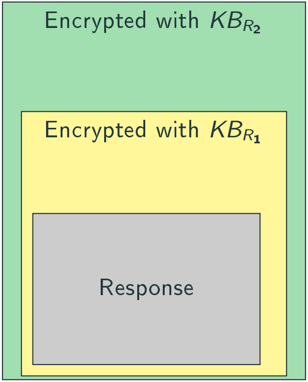
- R_3 receives response from `theintercept.com`.

How Tor Works: Relay Cell on the Reverse Path

- R_3 receives response from `theintercept.com`.
- R_3 encrypts with $Enc(\mathbf{KB_{R_3}}, Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL)))$, and sends to R_2.

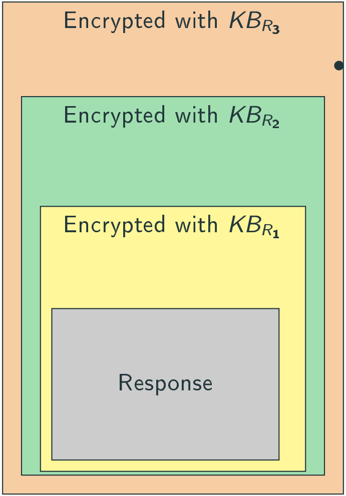


How Tor Works: Relay Cell on the Reverse Path



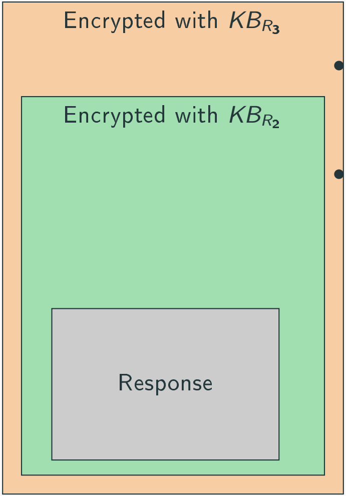
- R_3 receives response from `theintercept.com`.
- R_3 encrypts with $Enc(\mathbf{KB_{R_3}}, Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL)))$, and sends to R_2.

How Tor Works: Relay Cell on the Reverse Path



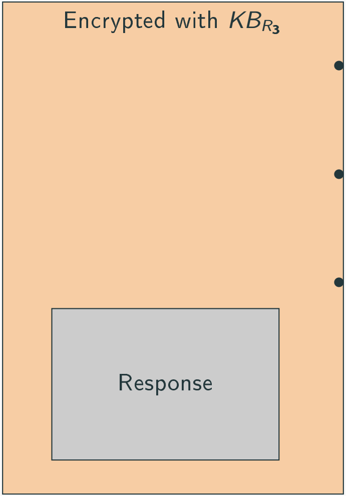
- R_3 receives response from `theintercept.com`.
- R_3 encrypts with $Enc(\mathbf{KB_{R_3}}, Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL)))$, and sends to R_2.

How Tor Works: Relay Cell on the Reverse Path



- R_3 receives response from `theintercept.com`.
- R_3 encrypts with $Enc(\mathbf{KB_{R_3}}, Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL)))$, and sends to R_2.
- R_2 decrypts with $\mathbf{KB_{R_3}}$, giving $Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL))$, and sends to R_1.

How Tor Works: Relay Cell on the Reverse Path



- R_3 receives response from `theintercept.com`.
- R_3 encrypts with $Enc(\mathbf{KB_{R_3}}, Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL)))$, and sends to R_2.
- R_2 decrypts with $\mathbf{KB_{R_3}}$, giving $Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL))$, and sends to R_1.
- R_1 decrypts with $\mathbf{KB_{R_2}}$, giving $Enc(\mathbf{KB_{R_1}}, CELL)$, and sends to the Tor Client.

How Tor Works: Relay Cell on the Reverse Path

- R_3 receives response from `theintercept.com`.
- R_3 encrypts with $Enc(\mathbf{KB_{R_3}}, Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL)))$, and sends to R_2.
- R_2 decrypts with $\mathbf{KB_{R_3}}$, giving $Enc(\mathbf{KB_{R_2}}, Enc(\mathbf{KB_{R_1}}, CELL))$, and sends to R_1.
- R_1 decrypts with $\mathbf{KB_{R_2}}$, giving $Enc(\mathbf{KB_{R_1}}, CELL)$, and sends to the Tor Client.
- The Tor Client decrypts with $\mathbf{KB_{R_1}}$ and thus receives the response.

Tor as Censorship Circumvention Mechanism

Tor as censorship circumvention

- Various countries filter Internet traffic by destination address
- Most prominent example: Great Firewall of China

Tor as censorship circumvention

- Various countries filter Internet traffic by destination address
- Most prominent example: Great Firewall of China
- Firewalls and gateways cannot see the true destination of Tor traffic
- Tor is a powerful tool to circumvent online censorship (e.g., in China, Iran, Turkey, Kazakhstan, Ethiopia, others)

Tor as censorship circumvention

- Various countries filter Internet traffic by destination address
- Most prominent example: Great Firewall of China
- Firewalls and gateways cannot see the true destination of Tor traffic
- Tor is a powerful tool to circumvent online censorship (e.g., in China, Iran, Turkey, Kazakhstan, Ethiopia, others)
- Can also use Tor to circumvent country filters:
  - Need an IP address that isn't in Germany (e.g. because of GEMA restrictions on YouTube): can use Tor access YouTube from a non-German IP address.

Censorship of Tor

- Easy solution for censors:

- Easy solution for censors:
    - Obtain list of Tor nodes from the Directory Authorities

Censorship of Tor

- Easy solution for censors:
  - Obtain list of Tor nodes from the Directory Authorities
  - Block access to the Tor network (all public relays)

- Easy solution for censors:
  - Obtain list of Tor nodes from the Directory Authorities
  - Block access to the Tor network (all public relays)
  - Even simpler: block access to the Directory Authorities (Iran, Ethopia, Kazakhstan, and others have done this historically).

Censorship of Tor

- Easy solution for censors:
  - Obtain list of Tor nodes from the Directory Authorities
  - Block access to the Tor network (all public relays)
  - Even simpler: block access to the Directory Authorities (Iran, Ethopia, Kazakhstan, and others have done this historically).
- Solution: Tor Bridges

Tor Bridges

- Tor *Bridges* are unpublished entrances to the Tor network used to circumvent online censorship when the public relays in the consensus are blocked.

Tor Bridges

- Tor *Bridges* are unpublished entrances to the Tor network used to circumvent online censorship when the public relays in the consensus are blocked.
- Bridge IP address and other connection information must be distributed out-of-band.

Tor Bridges

- Tor *Bridges* are unpublished entrances to the Tor network used to circumvent online censorship when the public relays in the consensus are blocked.
- Bridge IP address and other connection information must be distributed out-of-band.
- Deep Packet Inspection (DPI) or an active adversary is required to identify Bridges.

Tor Bridges

- Tor *Bridges* are unpublished entrances to the Tor network used to circumvent online censorship when the public relays in the consensus are blocked.
- Bridge IP address and other connection information must be distributed out-of-band.
- Deep Packet Inspection (DPI) or an active adversary is required to identify Bridges.
- Distributed via a centralised system called BridgeDB. Clients can currently obtain bridges by:
    - visiting `https://bridges.torproject.org/`
    - writing e-mail to bridges@torproject.org

Tor Bridges

- Tor *Bridges* are unpublished entrances to the Tor network used to circumvent online censorship when the public relays in the consensus are blocked.
- Bridge IP address and other connection information must be distributed out-of-band.
- Deep Packet Inspection (DPI) or an active adversary is required to identify Bridges.
- Distributed via a centralised system called BridgeDB. Clients can currently obtain bridges by:
    - visiting `https://bridges.torproject.org/`
    - writing e-mail to bridges@torproject.org

Since 2010, various nation state adversaries have been conducting active probing and enumeration attacks to attempt to collect all of Tor's bridges. Since then, an arms race to distribute the bridge addresses to honest clients without these adversaries obtaining them has ensued.

Bridge Enumeration Attacks, Part I

The first stage of the arms race was to simply identify the ways in which a Tor Client's traffic could be distinguished from normal traffic. This is obviously also effective against Tor Bridges.

Bridge Enumeration Attacks, Part I

The first stage of the arms race was to simply identify the ways in which a Tor Client's traffic could be distinguished from normal traffic. This is obviously also effective against Tor Bridges.

- Tor traffic has trivial distiguishers: it's "disguised" as TLS traffic, but:

Bridge Enumeration Attacks, Part I

The first stage of the arms race was to simply identify the ways in which a Tor Client's traffic could be distinguished from normal traffic. This is obviously also effective against Tor Bridges.

- Tor traffic has trivial distiguishers: it's "disguised" as TLS traffic, but:
    - It uses random domain names,

Bridge Enumeration Attacks, Part I

The first stage of the arms race was to simply identify the ways in which a Tor Client's traffic could be distinguished from normal traffic. This is obviously also effective against Tor Bridges.

- Tor traffic has trivial distiguishers: it's "disguised" as TLS traffic, but:
  - It uses random domain names,
  - It has a characteristic packet-size distribution,

Bridge Enumeration Attacks, Part I

The first stage of the arms race was to simply identify the ways in which a Tor Client's traffic could be distinguished from normal traffic. This is obviously also effective against Tor Bridges.

- Tor traffic has trivial distiguishers: it's "disguised" as TLS traffic, but:
    - It uses random domain names,
    - It has a characteristic packet-size distribution,
    - Historically, it presents a unique TLS ciphersuite list

Bridge Enumeration Attacks, Part I

The first stage of the arms race was to simply identify the ways in which a Tor Client's traffic could be distinguished from normal traffic. This is obviously also effective against Tor Bridges.

- Tor traffic has trivial distiguishers: it's "disguised" as TLS traffic, but:
  - It uses random domain names,
  - It has a characteristic packet-size distribution,
  - Historically, it presents a unique TLS ciphersuite list

In 2012, Ethiopia began blocking all TLS (and hence blocking all Tor) traffic by looking for the client HELLO. Any packet with the string **TLS_DHE_RSA_WITH_AES_256_CBC_SHA** in it is dropped. If you pick **TLS_DHE_RSA_WITH_AES_128_CBC_SHA** instead, or fragment the ciphersuite list, it works anyway.

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

- Observe Tor client's TCP connection to the Bridge

Bridge Enumeration Attacks, Part I

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

- Observe Tor client's TCP connection to the Bridge
- For Tor<0.2.3.17-beta, identification was based upon Tor's unique ciphersuite list

Bridge Enumeration Attacks, Part I

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

- Observe Tor client's TCP connection to the Bridge
- For Tor<0.2.3.17-beta, identification was based upon Tor's unique ciphersuite list
- A seemingly random machine from somewhere in China (possibly using IP-spoofing) will connect to the Bridge's IP:port and attempt to complete the first couple steps of the handshake

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

- Observe Tor client's TCP connection to the Bridge
- For Tor<0.2.3.17-beta, identification was based upon Tor's unique ciphersuite list
- A seemingly random machine from somewhere in China (possibly using IP-spoofing) will connect to the Bridge's IP:port and attempt to complete the first couple steps of the handshake
- The Bridge is blocked by IP:port

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

- Observe Tor client's TCP connection to the Bridge
- For Tor<0.2.3.17-beta, identification was based upon Tor's unique ciphersuite list
- A seemingly random machine from somewhere in China (possibly using IP-spoofing) will connect to the Bridge's IP:port and attempt to complete the first couple steps of the handshake
- The Bridge is blocked by IP:port
- The GFW sometimes spoofs a RST from Bridge to the client

Bridge Enumeration Attacks, Part I

Since 2010, China's GFW began active probing Tor Bridges, usually in the following manner:

- Observe Tor client's TCP connection to the Bridge
- For Tor<0.2.3.17-beta, identification was based upon Tor's unique ciphersuite list
- A seemingly random machine from somewhere in China (possibly using IP-spoofing) will connect to the Bridge's IP:port and attempt to complete the first couple steps of the handshake
- The Bridge is blocked by IP:port
- The GFW sometimes spoofs a RST from Bridge to the client

Solution: Tor's Pluggable Transports

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for "plugging" obfuscating proxies together, called *Pluggable Transports* (PTs).

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for "plugging" obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for “plugging” obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

- Pluggable Transport API allows communication between an obfuscating SOCKS proxy and Tor client

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for “plugging” obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

- Pluggable Transport API allows communication between an obfuscating SOCKS proxy and Tor client

Currently, the most widely-used and effective Pluggable Transport is obfs4proxy:

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for "plugging" obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

- Pluggable Transport API allows communication between an obfuscating SOCKS proxy and Tor client

Currently, the most widely-used and effective Pluggable Transport is obfs4proxy:

- Created by Yawning Angel.

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for "plugging" obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

- Pluggable Transport API allows communication between an obfuscating SOCKS proxy and Tor client

Currently, the most widely-used and effective Pluggable Transport is obfs4proxy:

- Created by Yawning Angel.
- It uses Tor's NTor handshake with public keys obfuscated via the Elligator 2 mapping.

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for "plugging" obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

- Pluggable Transport API allows communication between an obfuscating SOCKS proxy and Tor client

Currently, the most widely-used and effective Pluggable Transport is obfs4proxy:

- Created by Yawning Angel.
- It uses Tor's NTor handshake with public keys obfuscated via the Elligator 2 mapping.
- The link layer uses NaCl secret boxes (Poly1305, Xsalsa20).

Pluggable Transports

In 2011, fellow Tor developer George Kadianakis came up with an idea for a simple SOCKS-proxy based API for "plugging" obfuscating proxies together, called *Pluggable Transports* (PTs).

PTs are generally not meant to provide security benefits, because Tor traffic is tunneled through the obfuscating proxy. Instead, they often provide countermeasures to distinguishers, or packet-distribution or timing analysis.

- Pluggable Transport API allows communication between an obfuscating SOCKS proxy and Tor client

Currently, the most widely-used and effective Pluggable Transport is obfs4proxy:

- Created by Yawning Angel.
- It uses Tor's NTor handshake with public keys obfuscated via the Elligator 2 mapping.
- The link layer uses NaCl secret boxes (Poly1305, Xsalsa20).

Yawning has recently created a newer PT, called "basket2" which uses a hybrid handshake between Ed448 Goldilocks and NewHope.

Pluggable Transports: Magick Potion Ingredients

Pluggable Transports are generally considered a “finished” research field. There’s an incredibly simple formulae for creating one which works, although the “hard” problems (i.e. distributing the requisite shared secrets) are shoved under the rug.

Pluggable Transports: Magick Potion Ingredients

Pluggable Transports are generally considered a “finished” research field. There’s an incredibly simple formulae for creating one which works, although the “hard” problems (i.e. distributing the requisite shared secrets) are shoved under the rug.

- The handshake should be uniform.

Pluggable Transports: Magick Potion Ingredients

Pluggable Transports are generally considered a "finished" research field. There's an incredibly simple formulae for creating one which works, although the "hard" problems (i.e. distributing the requisite shared secrets) are shoved under the rug.

- The handshake should be uniform.
- The PT should use some pre-shared key material for server authentication.

Pluggable Transports: Magick Potion Ingredients

Pluggable Transports are generally considered a "finished" research field. There's an incredibly simple formulae for creating one which works, although the "hard" problems (i.e. distributing the requisite shared secrets) are shoved under the rug.

- The handshake should be uniform.
- The PT should use some pre-shared key material for server authentication.
- The PT should encrypt starting with the client's first message (i.e. encrypt the first stage of the handshake).

Pluggable Transports: Magick Potion Ingredients

Pluggable Transports are generally considered a "finished" research field. There's an incredibly simple formulae for creating one which works, although the "hard" problems (i.e. distributing the requisite shared secrets) are shoved under the rug.

- The handshake should be uniform.
- The PT should use some pre-shared key material for server authentication.
- The PT should encrypt starting with the client's first message (i.e. encrypt the first stage of the handshake).
- An anthenticated encryption cipher should be used at the transport layer.

Pluggable Transports: Magick Potion Ingredients

Pluggable Transports are generally considered a “finished” research field. There’s an incredibly simple formulae for creating one which works, although the “hard” problems (i.e. distributing the requisite shared secrets) are shoved under the rug.

- The handshake should be uniform.
- The PT should use some pre-shared key material for server authentication.
- The PT should encrypt starting with the client’s first message (i.e. encrypt the first stage of the handshake).
- An anthenticated encryption cipher should be used at the transport layer.

While there’s probably not any remaining research problems in Pluggable Transports for producing academic papers, writing new PTs is an incredibly fun project (suitable for Master’s, or sufficiently-motivated Bachelor’s, students) because you get to be super #yolo and use experimental new crypto.

While Pluggable Transports effectively obfuscate the link between a Client and a Bridge—and modern Pluggable Transports make the traffic data look uniformly indistinguishable from random—this turns out to be insufficient to prevent Bridge *enumeration attacks*.

Bridge Enumeration Attacks, Part II

While Pluggable Transports effectively obfuscate the link between a Client and a Bridge—and modern Pluggable Transports make the traffic data look uniformly indistinguishable from random—this turns out to be insufficient to prevent Bridge *enumeration attacks*.

Another possibility is to simply automate obtaining Bridges in the same manner an honest client would.

The proposed solution uses attribute-based credentials to record honest users' good behaviour (i.e. the bridges not being censored/blocked), which also serves to effectively lock censoring adversaries out of the distribution system.

A Social Protocol for Bridge Distribution

The proposed solution uses attribute-based credentials to record honest users' good behaviour (i.e. the bridges not being censored/blocked), which also serves to effectively lock censoring adversaries out of the distribution system.

Wang, Q., Lin, Z., Borisov, N., & Hopper, N. (2013, February). rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In NDSS.

Original rBridge Design

A Social Protocol for Bridge Distribution

Original rBridge Design

- Users are given "brownie points" for "good behaviour".

A Social Protocol for Bridge Distribution

Original rBridge Design

- Users are given "brownie points" for "good behaviour".
- Users with enough brownie points might win the chance to invite their friends.

Original rBridge Design

- Users are given “brownie points” for “good behaviour”.
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.

A Social Protocol for Bridge Distribution

Original rBridge Design

- Users are given “brownie points” for “good behaviour”.
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.

Original rBridge Design

- Users are given “brownie points” for “good behaviour”.
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.
- Some odd crypto choices, minor mistakes, and efficiency sacrifices for very little added privacy.

Original rBridge Design

- Users are given “brownie points” for “good behaviour”.
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.
- Some odd crypto choices, minor mistakes, and efficiency sacrifices for very little added privacy.
    - K-TAA signature scheme

Original rBridge Design

- Users are given "brownie points" for "good behaviour".
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.
- Some odd crypto choices, minor mistakes, and efficiency sacrifices for very little added privacy.
    - K-TAA signature scheme
    - Pedersen commitments on vectors

Original rBridge Design

- Users are given “brownie points” for “good behaviour”.
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.
- Some odd crypto choices, minor mistakes, and efficiency sacrifices for very little added privacy.
    - K-TAA signature scheme
    - Pedersen commitments on vectors
    - Oblivious Transfer

Original rBridge Design

- Users are given "brownie points" for "good behaviour".
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.
- Some odd crypto choices, minor mistakes, and efficiency sacrifices for very little added privacy.
  - K-TAA signature scheme
  - Pedersen commitments on vectors
  - Oblivious Transfer
  - Ad-hoc anonymous credential construction from k-TAA signatures and a Camenisch-Stadler NIZK proof-of-discrete-logarithm.

Original rBridge Design

- Users are given "brownie points" for "good behaviour".
- Users with enough brownie points might win the chance to invite their friends.
- Censors lock themselves out of the system via their own bad behaviour.
- Hopefully nobody is friends with the censors enough to give them an invite ticket.
- Some odd crypto choices, minor mistakes, and efficiency sacrifices for very little added privacy.
  - K-TAA signature scheme
  - Pedersen commitments on vectors
  - Oblivious Transfer
  - Ad-hoc anonymous credential construction from k-TAA signatures and a Camenisch-Stadler NIZK proof-of-discrete-logarithm.

Currently, I'm redesigning the protocol and implementing the scheme using an anonymous credential based on algebraic MACs.

The best game a censor can play against the rBridge scheme is to exhibit good behaviour in order to slowly amass brownie points, trading them in for new Bridges and invite tickets. Using an *Event-Driven Blocking Strategy*, that is, waiting until some important event, e.g. a political protest, and blocking all known Bridges en masse, is the most effective.

A Social Protocol for Bridge Distribution

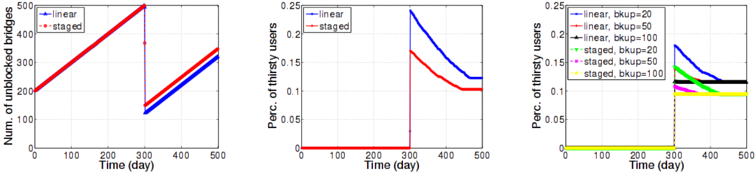
The best game a censor can play against the rBridge scheme is to exhibit good behaviour in order to slowly amass brownie points, trading them in for new Bridges and invite tickets. Using an *Event-Driven Blocking Strategy*, that is, waiting until some important event, e.g. a political protest, and blocking all known Bridges en masse, is the most effective.

Some honest users whose Bridges are blocked, and who do not currently posess enough brownie points for new, unblocked bridges, will effectively be locked out of the system as collateral damage.

A Social Protocol for Bridge Distribution

The best game a censor can play against the rBridge scheme is to exhibit good behaviour in order to slowly amass brownie points, trading them in for new Bridges and invite tickets. Using an *Event-Driven Blocking Strategy*, that is, waiting until some important event, e.g. a political protest, and blocking all known Bridges en masse, is the most effective.

Some honest users whose Bridges are blocked, and who do not currently posess enough brownie points for new, unblocked bridges, will effectively be locked out of the system as collateral damage.



(a) Unblocked bridges (b) Thirsty users (c) Thirsty users with backup bridges

Figure 4: Event-driven blocking ($f = 5\%$)

Bridge Enumeration Attacks, Part III

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Bridge Enumeration Attacks, Part III

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Ling, Z., Fu, X., Yu, W., Luo, J., Yang, M. (2011).
Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery.

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Ling, Z., Fu, X., Yu, W., Luo, J., Yang, M. (2011).
Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery.

Bridge Enumeration by Running a Middle Relay

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Ling, Z., Fu, X., Yu, W., Luo, J., Yang, M. (2011).
Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery.

Bridge Enumeration by Running a Middle Relay

- Run a Middle relay. Unlike running a Guard or Exit relay, there is no waiting period to do this. Even if you were previously an Exit relay running

Bridge Enumeration Attacks, Part III

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Ling, Z., Fu, X., Yu, W., Luo, J., Yang, M. (2011).
Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery.

Bridge Enumeration by Running a Middle Relay

- Run a Middle relay. Unlike running a Guard or Exit relay, there is no waiting period to do this. Even if you were previously an Exit relay running
- Wait until you see something connecting to you which isn't listed in the consensus.

Bridge Enumeration Attacks, Part III

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Ling, Z., Fu, X., Yu, W., Luo, J., Yang, M. (2011).
Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery.

Bridge Enumeration by Running a Middle Relay

- Run a Middle relay. Unlike running a Guard or Exit relay, there is no waiting period to do this. Even if you were previously an Exit relay running
- Wait until you see something connecting to you which isn't listed in the consensus.
- Running 20 malicious routers, each with bandwidths of 10MB/s, results in a 90% probability of discovering any one particular Bridge.

Even with these measures being implemented, there are other schemes for discovering the locations of Tor bridges.

Ling, Z., Fu, X., Yu, W., Luo, J., Yang, M. (2011).
Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery.

Bridge Enumeration by Running a Middle Relay

- Run a Middle relay. Unlike running a Guard or Exit relay, there is no waiting period to do this. Even if you were previously an Exit relay running
- Wait until you see something connecting to you which isn't listed in the consensus.
- Running 20 malicious routers, each with bandwidths of 10MB/s, results in a 90% probability of discovering any one particular Bridge.
- These researchers claim to have run this attack on the live Tor network in 2011, enumerating 2369 Bridges in just 14 days.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathsf{KF}_{\mathsf{R}_1}, Enc(\mathsf{KF}_{\mathsf{R}_2}, Enc(\mathbf{R}_3, CELL)))$ to produce $Enc(\mathsf{KF}_{\mathsf{R}_2}, Enc(\mathbf{R}_3, CELL))$.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathbf{KF_{R_1}}, Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL)))$ to produce $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$.
- Then re-encrypting $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$ to any additional relay(s) of its choice.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathbf{KF_{R_1}}, Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL)))$ to produce $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$.
- Then re-encrypting $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$ to any additional relay(s) of its choice.
- Forward this re-encrypted cell to the first additional hop.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathbf{KF}_{\mathbf{R_1}}, Enc(\mathbf{KF}_{\mathbf{R_2}}, Enc(\mathbf{R_3}, CELL)))$ to produce $Enc(\mathbf{KF}_{\mathbf{R_2}}, Enc(\mathbf{R_3}, CELL))$.
- Then re-encrypting $Enc(\mathbf{KF}_{\mathbf{R_2}}, Enc(\mathbf{R_3}, CELL))$ to any additional relay(s) of its choice.
- Forward this re-encrypted cell to the first additional hop.
- Each additional hop decrypts the cell as usual.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathbf{KF_{R_1}}, Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL)))$ to produce $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$.
- Then re-encrypting $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$ to any additional relay(s) of its choice.
- Forward this re-encrypted cell to the first additional hop.
- Each additional hop decrypts the cell as usual.
- Eventually, the cell will be fowarded to the client's chosen $\mathbf{R_2}$.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathbf{KF_{R_1}}, Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL)))$ to produce $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$.
- Then re-encrypting $Enc(\mathbf{KF_{R_2}}, Enc(\mathbf{R_3}, CELL))$ to any additional relay(s) of its choice.
- Forward this re-encrypted cell to the first additional hop.
- Each additional hop decrypts the cell as usual.
- Eventually, the cell will be fowarded to the client's chosen $\mathbf{R_2}$.
- The client never learns that their traffic traverse additional hops.

Tor proposal #188: Bridge Guards

While we normally tell everyone the a circuit is (normally) three hops, and that a client chooses these hops, this is not entirely true.

Tor is loose-source routed

- Nothing prevents any relay along the client's chosen path from removing their layer of encryption, e.g. $\mathbf{R_1}$ can do $Dec(\mathbf{KF}_{\mathbf{R_1}}, Enc(\mathbf{KF}_{\mathbf{R_2}}, Enc(\mathbf{R_3}, CELL)))$ to produce $Enc(\mathbf{KF}_{\mathbf{R_2}}, Enc(\mathbf{R_3}, CELL))$.
- Then re-encrypting $Enc(\mathbf{KF}_{\mathbf{R_2}}, Enc(\mathbf{R_3}, CELL))$ to any additional relay(s) of its choice.
- Forward this re-encrypted cell to the first additional hop.
- Each additional hop decrypts the cell as usual.
- Eventually, the cell will be fowarded to the client's chosen $\mathbf{R_2}$.
- The client never learns that their traffic traverse additional hops.

We can exploit this unintended feature to give Bridges *their own* Guards, unbeknownst to the client, thus protecting Bridges from malicious Middle relays.

Future Improvements to Tor's Circuit-Level Cryptography

Tor currently uses AES256-CTR for the symmetric cryptography at the circuit level.

Current Circuit-Level Cryptography

Tor currently uses AES256-CTR for the symmetric cryptography at the circuit level.

A *Message Authentication Code* (MAC) is recomputed after each cell decryption, that is, cells are not end-to-end authenticated from the client.

Current Circuit-Level Cryptography

Tor currently uses AES256-CTR for the symmetric cryptography at the circuit level.

A *Message Authentication Code* (MAC) is recomputed after each cell decryption, that is, cells are not end-to-end authenticated from the client.

Although we've never witnessed an adversary take advantage of any of these, there are various known potentential issues.

Current Circuit-Level Cryptography

Tor currently uses AES256-CTR for the symmetric cryptography at the circuit level.

A *Message Authentication Code* (MAC) is recomputed after each cell decryption, that is, cells are not end-to-end authenticated from the client.

Although we've never witnessed an adversary take advantage of any of these, there are various known potentential issues.

Due to using CTR mode and re-MACing at each hop, a tagging attacks is possible.

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:
    - The Exit attempts to XOR the same tag back out, effectively removing it, then decrypts to produce the original cell.

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:
    - The Exit attempts to XOR the same tag back out, effectively removing it, then decrypts to produce the original cell.
    - The adversary has now confirmed that she is both the Guard and the Exit for the client's circuit.

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:
    - The Exit attempts to XOR the same tag back out, effectively removing it, then decrypts to produce the original cell.
    - The adversary has now confirmed that she is both the Guard and the Exit for the client's circuit.
- Otherwise, if the Exit is not colluding with the Guard:

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:
  - The Exit attempts to XOR the same tag back out, effectively removing it, then decrypts to produce the original cell.
  - The adversary has now confirmed that she is both the Guard and the Exit for the client's circuit.
- Otherwise, if the Exit is not colluding with the Guard:
  - The Exit decrypts the cell to produce garbage.

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:
  - The Exit attempts to XOR the same tag back out, effectively removing it, then decrypts to produce the original cell.
  - The adversary has now confirmed that she is both the Guard and the Exit for the client's circuit.
- Otherwise, if the Exit is not colluding with the Guard:
  - The Exit decrypts the cell to produce garbage.
  - The Tor Protocol says that the Exit **MUST** now send a **RELAY END** cell to tear down the circuit, and hence

A Known Tagging Attack on Tor's Circuit-Level Cryptography

Assumption: the Guard relay is controlled by an adversary, who also controls (some) Exit relay(s).

- When receiving cells from the target client, the Guard XORs a *tag* (e.g. some bits) into the decrypted cell, recalculates the MAC, and forwards to the Middle relay.
- The Middle relay successfully verifies the MAC, decrypts the cell, computes a new MAC, and forwards to the Exit relay.
- The Exit relay successfully verifies the MAC, and
- If the chosen Exit happens to be adversary-controlled:
    - The Exit attempts to XOR the same tag back out, effectively removing it, then decrypts to produce the original cell.
    - The adversary has now confirmed that she is both the Guard and the Exit for the client's circuit.
- Otherwise, if the Exit is not colluding with the Guard:
    - The Exit decrypts the cell to produce garbage.
    - The Tor Protocol says that the Exit **MUST** now send a **RELAY END** cell to tear down the circuit, and hence
- The adversary may repeat this attack until a colluding Exit relay is chosen by the client.

(Needed!) Future Improvements to Tor's Circuit-Level Cryptography

We would like to switch to using a chained, authenticated encryption, 509-byte wide block cipher.

(Needed!) Future Improvements to Tor's Circuit-Level Cryptography

We would like to switch to using a chained, authenticated encryption, 509-byte wide block cipher.

Doing so renders changes to a cell at any hop detectable.

(Needed!) Future Improvements to Tor's Circuit-Level Cryptography

We would like to switch to using a chained, authenticated encryption, 509-byte wide block cipher.

Doing so renders changes to a cell at any hop detectable.

There isn't such a cipher yet.

(Needed!) Future Improvements to Tor's Circuit-Level Cryptography

We would like to switch to using a chained, authenticated encryption, 509-byte wide block cipher.

Doing so renders changes to a cell at any hop detectable.

There isn't such a cipher yet.

Other potential (non-cryptographic) improvements to Tor's circuit protocol: There's not really any reasons we haven't considered disparate forward and reverse paths. Nothing in the crypto or protocol is technically preventing it. It would be an interesting area of research to see the changes (and, hopefully, improvements to anonymity guarantees) which might be derived from disjoint path selection.

Tor Relay Handshake

Tor's Relay Handshake Protocol: NTor

Tor's current handshake, NTor, is a one-way-authenticated, Diffie-Hellman based handshake which uses X25519.

Tor's current handshake, NTor, is a one-way-authenticated, Diffie-Hellman based handshake which uses X25519.

In the event of a quantum-capable adversary in the future, who is currently recording Tor handshakes now, we need a post-quantum handshake.

Proposed Post-Quantum Hybrid Handshakes for Tor

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

- Created by John Schanck, William Whyte, Zhenfei Zhang.

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

- Created by John Schanck, William Whyte, Zhenfei Zhang.
- Allows for composition of a classical handshake, e.g. Tor's current NTor handshake, with a key encapsulation mechanism (KEM) which is believed to be post-quantum secure.

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

- Created by John Schanck, William Whyte, Zhenfei Zhang.
- Allows for composition of a classical handshake, e.g. Tor's current NTor handshake, with a key encapsulation mechanism (KEM) which is believed to be post-quantum secure.
- Currently proposed post-quantum KEMs are:

Proposed Post-Quantum Hybrid Handshakes for Tor

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

- Created by John Schanck, William Whyte, Zhenfei Zhang.
- Allows for composition of a classical handshake, e.g. Tor's current NTor handshake, with a key encapsulation mechanism (KEM) which is believed to be post-quantum secure.
- Currently proposed post-quantum KEMs are:
    - Tor proposal #263: NTRU

Proposed Post-Quantum Hybrid Handshakes for Tor

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

- Created by John Schanck, William Whyte, Zhenfei Zhang.
- Allows for composition of a classical handshake, e.g. Tor's current NTor handshake, with a key encapsulation mechanism (KEM) which is believed to be post-quantum secure.
- Currently proposed post-quantum KEMs are:
    - Tor proposal #263: NTRU
    - Tor proposal #270: NewHope

Proposed Post-Quantum Hybrid Handshakes for Tor

A Modular Hybrid Handshake

Tor proposal #269: Transitionally secure hybrid handshakes
by John Schanck, William Whyte, Zhenfei Zhang.

- Created by John Schanck, William Whyte, Zhenfei Zhang.
- Allows for composition of a classical handshake, e.g. Tor's current NTor handshake, with a key encapsulation mechanism (KEM) which is believed to be post-quantum secure.
- Currently proposed post-quantum KEMs are:
  - Tor proposal #263: NTRU
  - Tor proposal #270: NewHope

John Schanck currently has a branch which integrates NTRU, and I'm currently working on an expirmental branch which implements the modular hybrid handshake (#269) and adds a plugin to implement the NewHope version (#270).

Questions & Contact

Isis Agora Lovecruft
isis@torproject.org
0A6A 58A1 4B59 46AB DE18 E207 A3AD B67A 2CDB 8B35