# Lecture Notes for Advanced Web Security 2016

Part 2 — Anonymity

Martin Hell

## 1 Anonymity

Many people are under the impression that they are anonymous while communicating on Internet. This is to some extent true. If you participate in forum discussions under a pseudonym, it is very difficult for other participants to find out who you are unless you reveal your identity yourself. As long as you trust the administrator of the forum not to reveal your IP address, you will remain anonymous. Of course, clicking links which redirects your browser to another server will reveal your IP to this server as well. But even if your IP address is disclosed there is still the problem of linking this to the real person behind the IP address. If your IP address is assigned by your ISP, a reverse DNS lookup will usually not provide helpful information. Instead, it is necessary to get co-operation from the ISP in order to link the IP address to a person. Thus, there are two entities that enforce your anonymity, the communication partner which knows about your IP address, and the ISP which can link this to a real person. In the case when the communication partner tries to identify the user, only the ISP remains to be trusted.

In the definition of anonymity used here, it will be assumed that neither the ISP, nor the communication partner can be trusted. Anonymity should still be obtained. If anyone should be trusted, it should be possible to choose who to trust. Further, since a user's identity on internet is the IP address, the extra layer provided by the ISP will not be used. If the IP address is disclosed, anonymity is broken. Similarly, if the IP address is kept hidden anonymity is obtained.

Anonymous communication can have several different properties. The properties that are most interesting in this case are:

- **Sender anonymity.** The identity of the party sending a message is hidden.
- **Receiver anonymity.** The identity of the party receiving a message is hidden.

- **Unlinkability of sender and receiver.** Two parties can not be identified as communicating with each other.

Anonymity can be useful in several situations. Some usage of anonymity can be considered good and some as bad. Which usage is good or bad depends on which type of person the user is, but a possible enumeration of pros and cons of anonymity could be:

**Advantages**

- **Discussion of sensitive and personal issues.** Two people discussing sensitive issues may not want people to know that they are talking to each other. Even if the sensitive issues can be hidden using encryption, an anonymity solution is needed to hide the fact that they are communicating.
- **Freedom of speech in intolerant environments.** Not all countries have the same view on freedom of speech. In some countries, enforcing freedom of speech requires anonymity.
- **Polls/Surveys.** A prerequisite for a fair voting system is that people can trust that a vote remains anonymous and can not be linked to a certain person.
- **Counceling for drug or alcohol abuse.** It might be easier to help these people if their anonymity can be guaranteed.
- **Criminal victims.** As a victim of a crime, anonymity could help the process of bringing the criminals to court.
- **Snitches and rats.** Undercover agents, deserting agents and other people willing to give out important or senstive information might require anonymity.

**Drawbacks**

- **Accusations** False accusations directed towards people that you personally do not like can be made anonymously.
- **Spreading of propaganda** Illegal spreading of views and opinions can be anonymous.
- **Spreading of copyright protected material** Movies, music and software can be distributed anonymously.
- **Other illegal acts**

While moral, ethical and legal issues are important, they are out of scope in this chapter. Only the technology providing anonymity will be of interest here.

One goal of an anonymity protocol is to protect against *traffic analysis*. Traffic analysis is the use of data patterns to deduce information. In a broad sense, traffic analysis can be used to get any type of information, but in the context of anonymity the goal is to reveal who is talking to whom and when they talk. Information that can be used is timing, size of packets, amount of packets, etc. Even if the data is encrypted, traffic analysis can sometimes be used to help identify the plaintext (see e.g., [5]). A related issue is *traffic confirmation*, in which the goal is to decide if two given peers are communicating at a certain time. As will be seen, it is very difficult to protect against traffic confirmation.

The *anonymity set* is the set of people among which an identity is unknown. The degree of anonymity is decided by the size of this set. Assume that someone wants to identify who sent a certain email. If it is known that only two computers made a connection to the mail server around the time the message was sent, the anonymity set is of size two. The real sender must be one of these two computers and a simple guess will have a 50% chance of being right. However, if the number of computers making connections to the mail server is 100, this probability is only 1%. Thus, an important goal of an anonymity providing system is to make the anonymity set as large as possible.

Anonymity solutions are divided into two different categories, high latency and low latency designs. As the name suggests, the high latency designs have the common characteristic that there is a significant delay between the sending of a message and when the message is delivered to the intended recipient. The typical use for high latency designs is email messages. In low latency designs, a message is delivered (almost) immediately after it is sent. This is often required for web browsing.

The *Global Passive Adversary* (GPA) is the strongest passive adversary imaginable. A GPA has the ability to eavesdrop *every* node in a network. All traffic entering a node and all traffic exiting a node can be seen by the GPA. From this it follows that the GPA knows the source and destination address of all packets in a network. In general

- Low-latency designs cannot protect against the GPA.
- High-latency designs aims at protecting against the GPA.

## 2 Chaum Mix

The Mix is a fundamental building block in the anonymity solutions that will be covered. It was proposed in 1981 by Chaum [2]. The exact functionality of the Mix depends on the implementation and goals of the protocol. This section will introduce the Mix in the way it was presented by Chaum. The Mix was proposed as a building block that allowed *anonymous encrypted emails.* To achieve this goal, public key cryptography is used. It was a high-latency design with the goal of achieving anonymity and confidentiality in the presence of a GPA. Thus, it is assumed that anyone can learn the origin and destination of every packet in the network. In particular anyone can learn the origin and destination of all packets entering and leaving the Mix. Also, the design protects against active attacks,

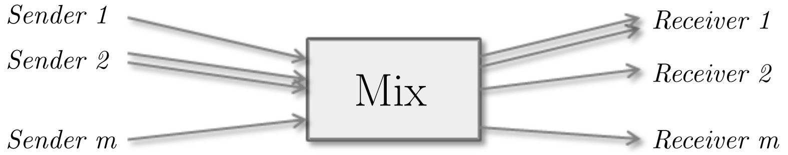

Figure 1: The Mix needs to remove all duplicates, otherwise it will be easy to link messages to the sender and receiver.

such as injecting arbitrary messages into the network, deleting messages, and modifying messages.

The main functionality of a Mix is to hide the correspondence between input and output messages. In practice, the Mix is simply a network connected computer that has the ability to perform cryptographic operations and the ability to collect input messages, scramble their order, and output them to a given destination.

Since an adversary knows origin and destination of incoming and outgoing messages, it is easy to relate messages if there is not a significant time delay between the input and output of a message in the Mix. Thus, it needs to be a design with high latency. Moreover, if messages are sent relatively sparingly on the network, high latency also allows the anonymity set to be large.

When receiving messages, the Mix needs to check to see if there are any duplicates, and in that case throw away the duplicate. If this is not done, it is easy to determine the link between sender and receiver. Assume that an adversary wants to know the receiver of a message sent by *sender 2*, see Figure 1, then the adversary can duplicate that message and observe the messages output by the Mix. The recipient receiving two identical messages will then be linked to sender 2.

## 2.1 Sending Messages

Assume that a sender $S$ wants to use a Mix to send an anonymous encrypted email to an addressee $A$. The steps involved in preparing the message are the following:

1. Add some randomness, $R_0$, to the message
2. Encrypt the message using the public key belonging to $A$, denoted $K_a$
3. Add the address of the receiver $A$ and the randomness $R_1$
4. Encrypt the message with the public key of the Mix, denoted $K_1$

The message entering the Mix will be

$$K_1(R_1, K_a(R_0, M), A). \tag{1}$$

Note here that public keys are used, which means that it will be necessary to somehow guarantee the link between the public key and the receiver/Mix. This problem is not addressed in this design, but is important in an actual implementation.

Inside the Mix, the following steps are performed:

1. Decrypt the received string using the private key of the Mix.
2. Remove the randomness $R_1$.
3. Read the destination address, A, and forward the message to this destination with the correct batch.

The message that is output by the Mix is given by

$$K_a(R_0, M), A. \tag{2}$$

The randomness $R_0$ is used to avoid a guessing attack on the message. Without $R_0$, the message leaving the Mix would be $(K_a(M), A)$. The public key is assumed known to everyone, and to verify if a certain message $M'$ was sent, it is enough to just encrypt this guess with the public key, $K_a(M') \overset{?}{=} K_a(M)$.

The use of randomness $R_1$ is to hide the correspondence between inputs and outputs. Without $R_1$ the message entering the Mix would be $K_1(K_a(R_0, M), A)$. Then, by just taking the output $K_a(R_0, M), A$ and encrypting it with the public key of the Mix, it would be easy to identify the correspondning input message.

When the message is finally delivered, the receiver decrypts the message using its private key, and removes the randomness $R_0$.

## 2.2 Return Address

It has been shown how to anonymously send a message to a recipient. An extension to this is to allow the recipient to reply to the message, but still not revealing the address of the recipient. In addition to the encrypted message, an encrypted return address is sent together with the message. The return address, $A_x$, is encrypted by the public key of the Mix, after adding some randomness $R_1$. Also in order to allow the reply to be encrypted, a temporary public key, $K_x$, is sent to the recipient. Thus the message $K_1(R_1, A_x), K_x$ is sent to the recipient.

Upon reply, $A$ adds randomness $R_0$ to the return message and encrypts it with the public key $K_x$, $K_x(R_0, M)$. The purpose of $R_0$ is here the same as when sending the original message. This is returned to the Mix together with the encrypted return address. The Mix decrypts the return address with its private key, reads the address $A_x$ and forwards the return message to the original sender $S$. However, before forwarding the return message it is encrypted using $R_1$ as key. Thus, the randomness $R_1$ serves two purposes in this scheme

- It prevents adversaries, in particular the addressee $A$, from simply guessing the return address and verify the guess using the public key of the Mix.

- By encrypting the return message with $R_1$, it prevents an eavesdropper to compare the messages entering and leaving the Mix. Without this encryption, $K_x(R_0, M)$, would be seen on both sides of the Mix.

## 2.3 Using Several Mixes

The design presented above relies on the fact that the Mix has to be trusted. Since the Mix knows the link between all senders and receivers, a Mix controlled by an adversary provides no anonymity. To avoid this problem, several Mixes can be concatenated. Then anonymity can still be guaranteed if one or a few Mixes are controlled by adversaries. In fact, it is enough that only one Mix can be trusted. The preparation of messages when several Mixes are involved is similar to the case with only one Mix. All Mixes have their own private/public key pair. With $n$ Mixes there are $n$ public keys used in the message preparation, $K_1, K_2, \ldots K_n$, where $K_n$ is the public key of the first Mix in the cascade and $K_1$ the public key of the last Mix, i.e., the Mix that sends the message to the addressee. Then the message is prepared as

$$K_n(R_n, K_{n-1}(R_{n-1}, \ldots, K_2(R_2, K_1(R_1, K_a(R_0, M), A)) \cdots)). \tag{3}$$

The first Mix decrypts the message with its private key, removes the randomness $R_n$ and sends the message

$$K_{n-1}(R_{n-1}, \ldots, K_2(R_2, K_1(R_1, K_a(R_0, M), A)) \cdots) \tag{4}$$

to the next Mix in the cascade. At the end of the cascade, the last Mix outputs the message $K_a(R_0, M), A$.

A similar generalization can be made with the untraceable return address. The return address is sent to the addressee as

$$K_1(R_1, K_2(R_2, \ldots, K_{n-1}(R_{n-1}, K_n(R_n, A_x)) \cdots)). \tag{5}$$

This encrypted return address is sent in the reply together with the message $K_x(R_0, M)$. The first Mix in the return path, i.e., the Mix with public key $K_1$, decrypts the outmost encryption layer, removes the randomness $R_1$ and encrypts the reply message with $R_1$. Thus the output of the first Mix is

$$K_2(R_2, \ldots, K_{n-1}(R_{n-1}, K_n(R_n, A_x)) \cdots), R_1(K_x(R_0, M)). \tag{6}$$

The output of the last Mix, i.e., the message returned to the original sender is given by

$$A_x, R_n(R_{n-1}(R_{n-2}(\ldots R_2(R_1(K_x(R_0, M))) \cdots)). \tag{7}$$

## 2.4 Attacks on Mixes

There are many possible attacks on Mixes that an implementation has to be aware of and protect against. Recall that the goal of an adversary is to correlate inputs to and outputs from the Mix. Comparing the *size* of incoming and outgoing messages is one way of doing this. If the message sizes are different it is easy to decide the corresponding input message by looking at an output message. The protection against this attack is to pad all packages to the same size.

*Replay attacks* is a common class of attacks. It was noted in Section 2 that a Mix has to remove duplicates. A replay attack is similar to adding duplicates but instead of adding the duplicate to the same batch, the adversary can replay a message much later than the original message was sent. The idea is that the replayed message will be sent to the same destination as before. By comparing the output messages in the two batches, the replayed message can be identified and the addressee will be disclosed. Protection against this attack can be achieved by saving a hash value of each previous message, or by including a timestamp in order to verify that the message is fresh.

Another attack is the *N-1 attack*. The idea here is to reduce the anonymity set for a user. If the adversary can control the generation of several messages that are input to the mix, then she also knows the recipients of these messages. Taking this to the extreme, assume that the adversary controls all senders but one. Then it is easy for the adversary to link the sender and receiver of the remaining message.

The *long term intersection attack* is a generic attack that is difficult to protect against. Assume that the adversary knows, or can guess with high probability, when a message entering the Mix, or a cascade of Mixes, is delivered to the recipient. Furthermore, assume that it is possible to *link* several messages to the same sender. A typical situation when this assumption is quite fair is messages posted anonymously to an Internet forum. If a user with pseudonym "Alice" sends her forum posts through one or several Mixes in order to be anonymous, it is still known that all messages from "Alice" are sent by the same user. Knowing *when* "Alice" sends her posts will also give the anonymity set of the real user, $S$, behind the pseudonym "Alice". In the attack, each time a message is sent that can be linked to the pseudonym "Alice", the adversary stores the anonymity set. Denote these sets by $AS_i, AS_{i+1}, AS_{i+2}, \ldots, AS_{i+T-1}$. A new anonymity set for $S$ can be found by taking the intersection of $AS_i$ and $AS_j$ and since $S$ will be in all stored sets, she will also be in the intersection between the $T$ sets. Thus, for large enough $T$, $S$ is revealed by taking

$$AS_i \cap AS_{i+1} \cap AS_{i+2} \cap \ldots \cap AS_{i+T-1}. \tag{8}$$

Of course, the required size of $T$ depends on several factors, e.g., the size of the batches (anonymity sets), the number of users in the system and the correlation between usage patterns of the users. The intersection will never increase the anonymity set for $S$. Instead, it is very likely that the anonymity set will decrease. While it is very difficult to avoid this attack, some measures can be taken to at least make it more difficult. If possible, the linkability between messages should be removed as far as possible. Moreover, dummy traffic from as many users as possible can be generated in order to maximize the anonymity sets. The generation of the dummy traffic can be implemented in the software used to send messages, but the dummy traffic is still limited to online users. Offline users cannot send any dummy traffic. The intersection attack is discussed in [1].

Another generic attack is the *disclosure attack*. It was described in [4]. In this attack it is assumed that a user, Alice, has $m$ communication partners, i.e., receivers that she regularly sends messages to. The goal of the attack is to reveal these $m$ communication partners. The total number of users in the anonymity system is $N$, the number of senders in each batch is denoted $b$, and the number of receivers in a batch is denoted $n$. Assume that all *senders* in a batch are distinct. Then, $n \leq b$ since more than one sender can send messages to the same receiver. The attack is divided into two phases, the *learning phase* and the *excluding phase*. In the learning phase the adversary records all *receivers* in a batch when a message from Alice is sent. This set of receivers is denoted by $R_i$. This is done until $m$ mutually disjoint sets are recorded, i.e., until

$$R_i \cap R_j = \emptyset \quad \forall i \neq j \tag{9}$$

for a total of $m$ sets. When this is fulfilled, each set, $R_1, R_2 \ldots R_m$, contains exactly one communication partner of Alice. This concludes the learning phase of the attack. The next phase, the excluding phase, a new set is taken, $R$, and if

$$R \cap R_i \neq \emptyset \text{ and } R \cap R_j = \emptyset \quad \forall j \neq i \tag{10}$$

then the set $R_i$ is replaced by the intersection $R \cap R_i$. This will reduce the size of the set $R_i$ (unless $R_i \subseteq R$). This procedure is repeated with a new set $R$ until all $m$ sets contain only *one* recipient. These $m$ recipients are then the communication partners of Alice. This attack will work as long as it is possible to find $m$ mutually disjoint sets, i.e., the Mix is insecure if

$$m \leq \lfloor N/n \rfloor. \tag{11}$$

Note that the exclusion phase can work even if $m = N/n$ since the same receiver can be present several times in one set.

All attacks presented in this section are independent of the actual implementation. They apply to all designs based on Mixes. In addition to this it could be possible to find attacks that completely depend on a specific implementation.

## 3 Onion Routing

High latency designs are usually suitable for emails and messages posted to forums etc. Web browsing and voice over IP protocols are typical examples where low latency is required. It would be very annoying and inefficient to have to wait for hours each time a webpage is requested. Any usage that requires some interaction between client and server is likely to favour a low latency design.

While one Mix would be enough in high latency designs, this is not enough for low latency designs. The reason is that the anonymity set is very small and by looking at the timing of incoming and outgoing packets, it would be easy to deduce the link between sender and receiver. In fact, against a GPA with the ability to eavesdrop the entire network, using several Mixes is also not enough. Using timing, a message can be traced along its path through all Mixes and the link between sender and receiver can be found (almost) just as easily. With this in mind, it is clear that low latency designs cannot, at least theoretically, protect against a GPA. Instead, focus has to be on making it as hard as possible for the GPA, and instead try to protect against adversaries with less power. Some measures taken to achieve this goal are

- Many Mixes are used at the same time.
- Have a large volume of traffic flowing through the network at all times.
- Use synthetic traffic when network usage is low.
- Mix the order of all packets that arrives within a fixed short time interval.

## 4 Tor - The Onion Router

Tor is the most widely used implementation of onion routing. The Tor design was proposed in [3] and details can be found in that paper. This section will give a brief overview of the design.

A design decision in Tor is to not use a public key of each router to encrypt the messages as originally proposed by Chaum, see Section 2.3. The reason is that asymmetric cryptography is far slower than symmetric cryptography. Still, each router has a private/public key pair and a variant of Diffie-Hellman key exchange is used to negotiate a symmetric session key with each router. This session key is then used to encrypt the message.

The Tor network consists of many onion routers and a user uses a subset of these routers to connect to a remote server, e.g., a web server. All connections between onion routers use TLS, and also the connection between the user and the first onion router. However, it is important to note that the final connection, i.e., the connection between the last router in the network and the remote server, is not encrypted. Tor does not offer any confidentiality. This is because the remote server does not have to be aware of the fact that Tor is used.

The data in the Tor network is sent in 512-byte cells. To avoid attacks based on packet size, all cells have this size even if the data to send is less than 512 byte. There are two types of cells, *control cells* and *relay cells*. Control cells are always interpreted by the receiving router, or node. These cells are typically used in the Diffie-Hellman key negotiation but can also be used to send padding data and to tear down a circuit. Relay cells are typically used to relay data between nodes in the network. These cells are only interpreted by the last node in the Tor network.

The cells have the following fields:

- **Circuit ID** A 2-byte ID for the connection to the next router.

- **CMD** A command that determines what to do.
- **Recognized** Helps identifying if an incoming packet should be interpreted or not.
- **StreamID** The ID used for this particular stream. Several streams can be multiplexed and use the same CircID.
- **Digest** A hash value of data to check integrity.
- **Length** The number of bytes in the data field that is real data. The rest is just padding in order to get a 512-byte cell.

The Circuit ID is an ID that is known only to adjacent nodes. The first and the second node has a shared ID for one circuit. This may not be the same as the ID shared between the second and the third node for the same circuit. The second node simply keeps a table that links incoming and outgoing IDs.

The digest field is used to provide integrity protection of the message. It is a hash value of the current cell, with digest set to 0, together with all previous cells and information from the handshake. Since this field is encrypted in the network, only the last node can use it to check the integrity, but every node can use it to check if a cell is to be relayed or interpreted. To speed this process up, the recognized field is used together with the digest field. Before encrypting the cell with the symmetric keys of the routers, the 2-byte recognized field is set to zero. Each router then checks this field and if it is nonzero, the cell is relayed to the next router determined by the CircID, without having to compute the hash value. If the field is zero, then the hash has to be computed and compared to the value of the digest field. This means that a hash value needs to be computed only once in $2^{16}$ cells, except at exit nodes where the hash is always computed since the recognized cell then must be zero. This provides a significant speedup. The total number of bytes used is 6 so the probability of failure is $2^{-48}$.

## 4.1 Negotiating Symmetric Keys

As noted before, each onion router has a private/public key pair that is used to negotiate a symmetric session key with a user. The key negotiation is based on Diffie-Hellman. One drawback with this type of authentication is the lack of entity authentication. You know that you have a shared key with *someone*, but you do not really know *who*. One solution is the Station-To-Station (STS) protocol, in which the parameters sent are digitally signed using a private key. In Tor, a slightly different approach is taken, but still using asymmetric cryptography. The user, Alice, does not have an asymmetric key pair and packets from her are not authenticated.

Alice uses the private parameter $x_i$ in the key exchange with the $i$th router and the private parameter used by the $i$th router is denoted $y_i$. The key negotiation with the *first* onion router can be divided into the following steps.

1. Alice sends $g^{x_1} \bmod p$ encrypted with the public key of the first router, $OR_1$. This is sent in a control cell with the *create* command.

2. $OR_1$ decrypts $g^{x_1} \bmod p$ and uses its own private parameter to compute the shared key, $K_1 = (g^{x_1})^{y_1} \bmod p$.

3. $OR_1$ returns $g^{y_1} \bmod p$ together with a hash of the negotiated key $H(K_1)$.

4. Alice computes the shared key as $K_1 = (g^{y_1})^{x_1} \bmod p$ and verifies that the key is correct by checking the hash value.

This protocol provides *unilateral entity authentication* since the message from $OR_1$ is authenticated, but $OR_1$ does not know it is communicating with Alice. Not only is this necessary since Alice does not have a key pair, but it also supports the goal of anonymity. Alice knows she is communicating with the intended router, but she does not reveal her own identity through a public key. In this case, when communicating with the first onion router, her identity is revealed through her IP address. However, as will be seen, when communicating with the other routers in the network, she will be anonymous. The protocol also provides *unilateral key authentication.* Alice knows that only $OR_1$ and herself knows the shared key $K_1$.

The same procedure is repeated when Alice negotiates symmetric keys with the second node. The only difference is that the messages are relayed by the first node. The procedure can be described as follows (see Figure 2).

1. Alice sends a relay extend cell to the first onion router. The data consists of the name of the second node, $OR_2$, together with the encrypted (using $K_{OR_2}$) Diffie-Hellman key exchange message intended for the $OR_2$. This data is encrypted with the symmetric key, $K_1$, shared between Alice and $OR_1$.

2. $OR_1$ decrypts the message using $K_1$ and sends $K_{OR_2}(g^{x_2} \bmod p)$ in a control cell to $OR_2$.

3. Similar to the case with the first node, $OR_2$ decrypts $g^{x_2} \bmod p$ and uses its own private parameter to compute the shared key, $K_2 = (g^{x_2})^{y_2} \bmod p$. Also the hash $H(K_2)$ is computed.

4. $OR_2$ returns $g^{y_2} \bmod p$ to $OR_1$ together with $H(K_2)$.

5. $OR_1$ puts this data into a relay cell, encrypts it with $K_1$ and returns it to Alice.

6. Alice decrypts the cell using $K_1$ and computes the shared key as $K_2 = (g^{y_2})^{x_2} \bmod p$. Finally she verifies that the hash of $K_2$ is correct.

The Circuit ID used in the connection between Alice and $OR_1$ is not the same as the Circuit ID between $OR_1$ and $OR_2$. Instead, $OR_1$ keeps a table describing that all incoming relay cells from Alice with $ID_1$ should be forwarded to $OR_2$ with Circuit ID $ID_2$. Thus, it is not possible to correlate incoming and outgoing packets from a router by looking at the Circuit ID. Also, Alice does not need to know the Circuit ID used in the communication between $OR_1$ and $OR_2$.

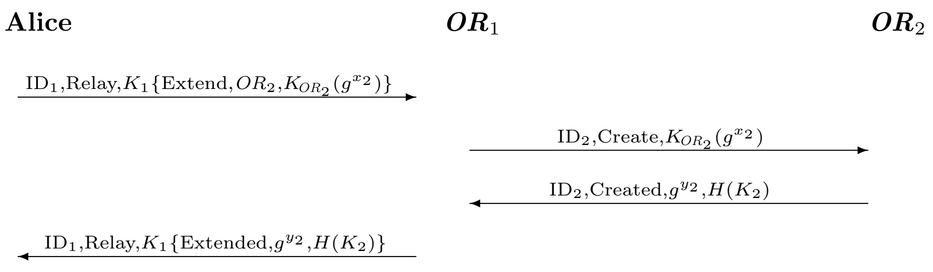
**Alice**
$\boldsymbol{OR}_1$
$\boldsymbol{OR}_2$
$\mathrm{ID}_1$,Relay,$K_1\{$Extend,$OR_2$,$K_{OR_2}(g^{x_2})\}$
$\mathrm{ID}_2$,Create,$K_{OR_2}(g^{x_2})$
$\mathrm{ID}_2$,Created,$g^{y_2}$,$H(K_2)$
$\mathrm{ID}_1$,Relay,$K_1\{$Extended,$g^{y_2}$,$H(K_2)\}$

Figure 2: Negotiating a symmetric key with $OR_2$, where $K_x\{\}$ denotes symmetric encryption, $K_x()$ denotes asymmetric encryption and $H()$ is a hash function.

Any onion router only knows about the node before and after. If 3 routers are used, the first and the third router have no information about each other's identity. In the implementation of Tor, the user updates the circuit, i.e., the set of nodes used, about once per minute. Every time a new circuit is constructed there are public key operations involved. This means that it can take some time to construct a circuit. To avoid letting the user wait for a new circuit to be established, new circuits are built preemptively. Thus, every time a new circuit is used, the symmetric keys for the new circuit have already been negotiated and the delay is minimized.

### 4.2 Relaying Data Through Nodes

Once a circuit has been setup and symmetric encryption keys have been negotiated between Alice and the routers, it can be used to e.g., anonymously send HTTP requests to web servers. An example of this is given in Figure 3, where Alice uses two onion routers to setup a TCP connection to a website. The different steps can be summarized as follows.

1. Alice constructs a *relay begin* cell, identifying the website. The cell is first encrypted with $K_2$ and then encrypted again with $K_1$. It is sent to $OR_1$.

2. $OR_1$ decrypts the cell with $K_1$, checks the *recognized* field (and possibly the *digest field*), and forwards it to $OR_2$.

3. $OR_2$ decrypts the cell with $K_2$, checks the *recognized* and *digest* fields and determines that it is the exit node. The name of the remote website is read and a TCP handshake is made against this site.

4. $OR_2$ creates a relay cell with information that there is now a connection to the website. This is encrypted using $K_2$ and sent to $OR_1$.

5. $OR_1$ receives the cell and adds a new encryption layer using $K_1$, before the cell is forwarded to Alice.

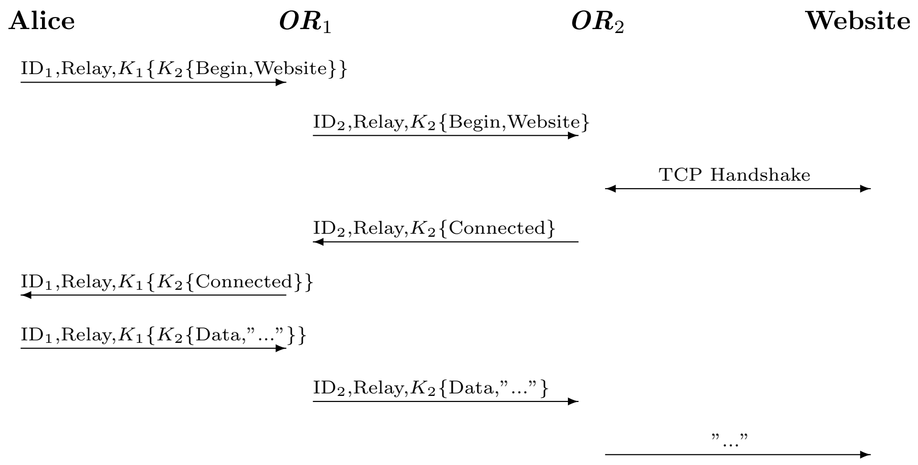
Alice
$OR_1$
$OR_2$
Website
ID$_1$,Relay,$K_1${$K_2${Begin,Website}}
ID$_2$,Relay,$K_2${Begin,Website}
TCP Handshake
ID$_2$,Relay,$K_2${Connected}
ID$_1$,Relay,$K_1${$K_2${Connected}}
ID$_1$,Relay,$K_1${$K_2${Data,"..."}}
ID$_2$,Relay,$K_2${Data,"..."}
"..."

Figure 3: Example of messages when two onion routers are used. Note that only symmetric encryption is used.

6. Alice decrypts the cell first using $K_1$ and then $K_2$.

The procedure above indicates why the name onion routing is used. The encryption is added in layers, just like layers of an onion. In the decryption process, layers are removed one by one.

The symmetric encryption scheme used in Tor is 128-bit AES in counter mode.

## 4.3 Security in Tor

As mentioned previously, in low latency designs it is very difficult to protect against a GPA. Tor is not an exception to this as it does not claim to protect against a GPA. If someone has the ability to listen to traffic to and from all nodes in the Tor network, the low latency would make it possible to relate incoming and outgoing messages in nodes and then also determine the sender of a message arriving at a destination. It can be argued whether such strong adversaries are likely in practice, but it should also be noted that for the attack to be successful, the GPA does not have to be able to listen to all traffic. It only needs to be able to listen to all nodes that are actually used at a given time. Moreover, the adversary can be adaptive and possibly decide which subset of nodes that she will listen to.

It is possible for users to run their own Tor node. Then they are not only using Tor for their own anonymity, they are also helping others to be anonymous. Users running their own Tor node are also more anonymous than users that do not. This is because they generate traffic that is not only their own traffic. Since all packets are encrypted, an adversary has no way of knowing if cells leaving the node are just relayed from another cell or actually origins in the cell.

The Diffie-Hellman key exchange provides *perfect forward secrecy*. This means that old session keys remain secure even if the long term key is broken. The long term key in this case is the private/public key pair belonging to the onion router and used to encrypt/decrypt the parameters sent from the user to the router. Consider the case when an eavesdropper records encrypted traffic on the Tor network and later steals or somehow gains knowledge of the router's private key. If the session keys are deleted after use, there is no way the eavesdropper can use this key to decrypt the recorded information. This can be compared to the case when a client generates a session key and uses the public key of the server to transmit the session key to the server. This is (roughly) the case in the key exchange used in SSL/TLS when RSA is used. This key exchange does not have perfect forward secrecy.

## 4.4 Rendezvous Points and Hidden Services

Up to this point, Tor has been described as a protocol that allows users to be anonymous when using a web service. Another feature is to use *hidden services*. This will allow, not only the user, but also the web server to be anonymous. The basic idea is to use so called *rendezvous points*, i.e., the user does not connect to the server. Instead, both user and server connects to a common onion router acting as server to the client and as client to the server.

Let Alice be the user and Bob be the service provider. The communication using rendezvous points is divided into two parts. First, Bob initializes the service to make it available as a hidden service. Second, Alice sets up the communication to Bob's service.

### 4.4.1 Bob Initializes Service

Bob's service is identified by his public key. Bob chooses a set of introduction points, denoted IP, and records all IPs for his service in a database. Then Bob connects to each IP using Tor. Note that these connections are not direct, but are made through other onion routers, keeping Bob anonymous to each IP.

### 4.4.2 Alice Connects to Service

When Alice wants to use Bob's service she looks up his public key in the database. The corresponding entry will give her a set of IPs that she can use to connect to Bob. Alice picks another router, the rendezvous point, denoted RP, that is to be used as a meeting node for Alice and Bob. She picks one of the IPs, makes a Tor connection to it, and notifies it about her chosen RP. The IP sends the communication request to Bob with the identity of the RP. Finally, Bob connects to the RP where Alice is waiting and the anonymous communication can begin.

## 5 Final Remarks

It is important to remember that anonymity is not the same as encryption. Chaum presented a protocol which provided both anonymity and end-to-end encryption, but this is not the case in e.g., Tor, which does not provide end-to-end encryption. Traffic is only encrypted inside the Tor network. The last node, the exit node, decrypts the packet and sends it in clear to the destination. If end-to-end encryption is needed, then this must be explicitly set up with the destination using e.g., SSL/TLS.

Using Tor for logging in to email accounts and sending instant messages can be a bad idea since the exit node can read all communication.

In September 2007, about 100 email accounts were posted on the Internet. These accounts belonged to e.g., embassies and government agencies so the information in emails was potentially very sensitive. It turned out that the accounts were captured by an exit node in Tor, but many of the account owners did not even know what Tor was. Hence, it is likely that the email accounts were stolen by other means. In order to hide their identity, the people that stole the accounts used Tor when logging in to the accounts with the stolen credentials.

## References

[1] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*, volume 2482. Springer-Verlag, 2002.

[2] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24:84–88, February 1981.

[3] R. Dingledine, N. Mathewson, and P. Syverson. The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[4] D. Kedogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In *Revised Papers from the 5th International Workshop on Information Hiding*, volume 2578 of *Lecture Notes in Computer Science*, pages 53–69. Springer-Verlag, 2002.

[5] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, 2001.