Multiple paths to transmit private data

Scalable method to improve privacy and preserve anonymity.

Martin Vassor

January 21, 2014

Abstract

The idea is to divide a given message into small pieces in such a way that all are needed to recompose the message. Then, we send all these pieces to the receiver through different paths. Thus, an observer wouldn't be able to know if he catches all the pieces. Moreover, we want the method to guarantee anonymity of the sender. First, we will propose a formal definition of privacy. Then how to strongly decompose a message. After that, we will see which modifications in OSI model are needed. Last but not least, we will see the impact on security protocols.

1 Introduction

1.1 What is 'privacy' ?

Common definition : There are two types of data. Public data on one hand, and private data on the other hand. Most of the time, we consider that a data is private when it only a few people can access it, and it is public when most of the population can access it. However, this definition is quite unprecise, where is the limit ? Well, to find a good definition, we should first define what we are working on.

Formal Definition : We are working on a set of n people $\{p_1,\ p_2,\ \dots\ ,p_n\}$. Let's say that this set forms a universe $U = \{p_i,\ \forall i \in [1, n]\}$. Moreover, the privacy depends on the data. Thus, we can define a logical proposition $private(data)$ which indicate if the given data is private. As we have seen before, people should either be able to access the data or not. We can thus partition U into two subsets. First, a subset A which contains people who should access the data, and a set B which contains people who should not access the data. Obviously, the partitionning depends on the data, some people can only access some datas. Then, we have a function $partition(data)$ which returns A and B. We can now formally define $private(data)$ following the idea that a data is public if and only if everybody is authorized to access it.

$$\boxed{private(data) \Leftrightarrow \forall p_i \in U : p_i \in A} \tag{1.1}$$

A few quite useless properties : We have the following properties :

$$U = A \cup B$$

$$\Leftrightarrow A = U \backslash B \Leftrightarrow A = \overline{B}$$

$$(1.1) \Leftrightarrow \neg \exists p_i \in U : \neg(p_i \in A)$$

$$\Leftrightarrow \boxed{private(data) \Leftrightarrow \neg \exists p_i \in U : p_i \in B} \tag{1.2}$$

$$\Leftrightarrow \boxed{private(data) \Leftrightarrow B = \emptyset} \tag{1.3}$$

1.2 What means 'respect privacy' ?

A data is private if there exists someone who should not access that data. But sometime a person can access a data even if he should not. For instance secret services try to get information they are not concerned by. Thus the concept of respecting privacy. We can say that the privacy is respected if and only if only people that access the data are allowed. More formally, in addition to U, A and B as above, we can define A' and B' as the set of people that indeed access and respectively not access the data. Thus, the privacy is respected if all persons that access the data are allowed. Notice that do not mean that all allowed person indeed access the data. As above, we define the proposition $respectPrivacy(data)$ which indicate if the privacy of the data is respected.

$$\boxed{respectPrivacy(data) \Leftrightarrow \forall p_i \in A' : p_i \in A} \tag{1.4}$$

$$(1.4) \Leftrightarrow A' \subseteq A$$

$$\boxed{\Leftrightarrow B \subseteq B'} \tag{1.5}$$

1.3 Concept of 'complete privacy'

Even if an external observer[1] can not deduce the data from a message, if it knows that the message exists, it can deduce implicits information, such that the fact that the transmiter try to communicate with the receiver. And sometimes, it is necessary that these persons remains anonymous. We define the concept of 'complete privacy' as the fact than we can not deduce any information about the transmiter and the receiver when observing the message, whenever or wherever we observe it. That is, we have a complete privacy when we guarantee both privacy as seen above, and anonymity.

2 Decomposition method

2.1 Introduction :

The idea is to use the power of collectivity in a network to hide and cipher the data. To achieve that, Alice will 'xor-decompose' its message into m submessages. Then she will send all that stuff to Bob using different paths. Then Bob just have to xor all the submessages to recompose the original message.

[1]An external observer is a person which should not access the data. p_e is an external observer if and only if $p_e \in B$

2.2 Xor decomposition :

Motivation : The Xor decomposition is an easy way to divide a message into submessages. It is very similar than a simple xor encrypting, where Alice and Bob both have to know a secret key, but with that method, there is no difference between the key and the cyphered text. Moreover, it is possible to have as much submessage as we want without compromise the privacy, because all submessage are needed to uncypher the original message.

How to find a xor decomposition ? Let a be a sequence of n bits, and m the desired number of terms for the decomposition. We want to find $\{a_1,\ a_2,\ \dots, a_m\}$ such that

$$a = a_1 \oplus a_2 \oplus \cdots \oplus a_m \tag{2.1}$$

Finding such a_i-s is easy, and there are multiple ways. For instance, Alice may create $m-1$ random sequences of size n. Then compute

$$a_m = a \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_{m-1} \tag{2.2}$$

To reconstruct a. Bob just have to xor-sum all a_i.

Proof of correctness :

$$(2.2) \Leftrightarrow a_m = a \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_{m-1}$$

$$\Leftrightarrow a_m \oplus a = a_1 \oplus a_2 \oplus \cdots \oplus a_{m-1}$$

$$\Leftrightarrow a = \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_{m-1} \oplus a_m$$

```
Algorithm: Xor-decompose
Data: $a$ - The sequence to decompose
Result: $A$ - An array of size $m$ containing all $a_i$
$A$ = Array of size $m$;
$a_m = a$;
for $i$ from 1 to $m-1$ do
    $a_i \leftarrow$ RandomOfSize($n$);
    A[i] $\leftarrow a_i$;
    $a_m \leftarrow a_m \oplus a_i$
end
```

```
Algorithm: Xor-assemble
Data: $A$ - The array of received data
Result: $a$ - The original message
$n = A.length$ $a = A[1]$ for $i$ from 2 to $n$ do
    $a \leftarrow a \oplus A[i]$
end
```

Pseudo-code

3 Complete sender pattern

A complete picture is printed in page 5

3.1 Application layer :

Formatting the message : The application layer changes a lot. Indeed, it has to decompose the given message into submessages. As we have seen, the source address has to be into the data. That is why the application layer must have a reading access to that information. Then, the message is formatted.

Decomposing the message : Moreover, the application layer xor-decompose the message. Then it transmit all the submessages to the transport layer.

Port : When handshaking, Alice have to send to a default port. Then, Bob will dedicate a socket for the connection. Then Alice should send to the new port.

3.2 The transport Layer :

The transport protocol is just a UDP protocol.

3.3 Network Layer :

Routing : The routing algorithm should trace the whole route of the packet, instead of just choosing the next hop. Designing such an algorithm with an acceptable running time is hard and exceeds this essay. Indeed, the algorithm should not only choose a good-enough path, but it should also change the path each time.

Network header : In fact, there is not a network header, but a stack of headers. One for each node of the choosed path.

4 Complete receiver pattern

A complete picture is printed in page 7

4.1 Network Layer :

When receiving a packet : If the header stack contains only one element, the packet is for Bob. Thus decapsulate the packet, and pass it to the transport layer. Otherwise, remove the top header, and forward the packet.

4.2 Transport Layer :

It is just a UDP protocol.

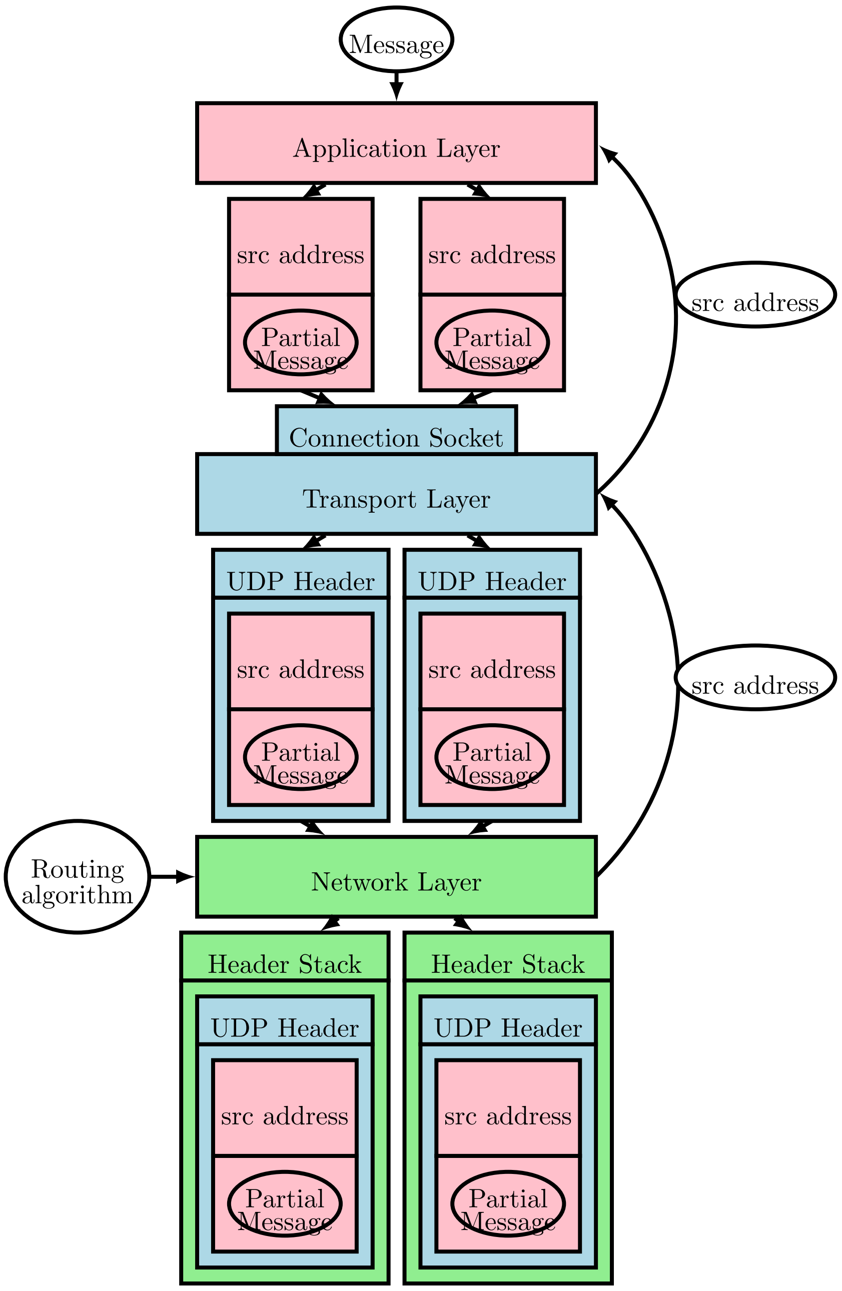

Figure 1: Pattern of modified OSI, sender side.

4.3 Application Layer :

On default port : When receiving a set of messages on the default port, Bob should xor-compose them. If it makes sense (i.e.: if it is a valid handshake), Bob should open a new socket, dedicated to the connection with Alice. Then he acknoledge from that socket.

On a dedicated port : Then, Bob should wait and compose until the message make sense. Indeed, since submessages goes through different paths, the RTT might be different. If the validity time is over, Bob drops the concerned submessage.

5 Resistance to attacks :

5.1 Settings :

We suppose to be in a huge enough network, such that an observer can not observe on all links of the network.

5.2 Eve :

For huge enough networks, Eve can do nothing. Indeed, to uncypher the message, one must have all the message from Alice to Bob. But Eve cannot even knows if a message she intercepts come from Alice or not. That is why, even if she catch all the messages, she cannot knonws that she has all the message.

5.3 Mallory :

Modification : Mallory can modify some bits of the message. Nevertheless, as Eve, she doesn't know what to modify. Then, Mallory's modification are similar to noise of a channel. Then we can use usual error detection and correction.

Reordering : Reordering has no sense. Usually, we reorder to cause an action. Here, Mallory doesn't know what message cause which action. To same submessage can cause different actions if the other submessages are different. And even if Mallory wants to reorder messages, she has to catch and re-send all messages. And Alice may include in the data of the message order number, etc...

6 Limitations :

6.1 Network size :

The method above should work well on huge networks. Nevertheless, on small network, it is easy for a single person to observe all links.

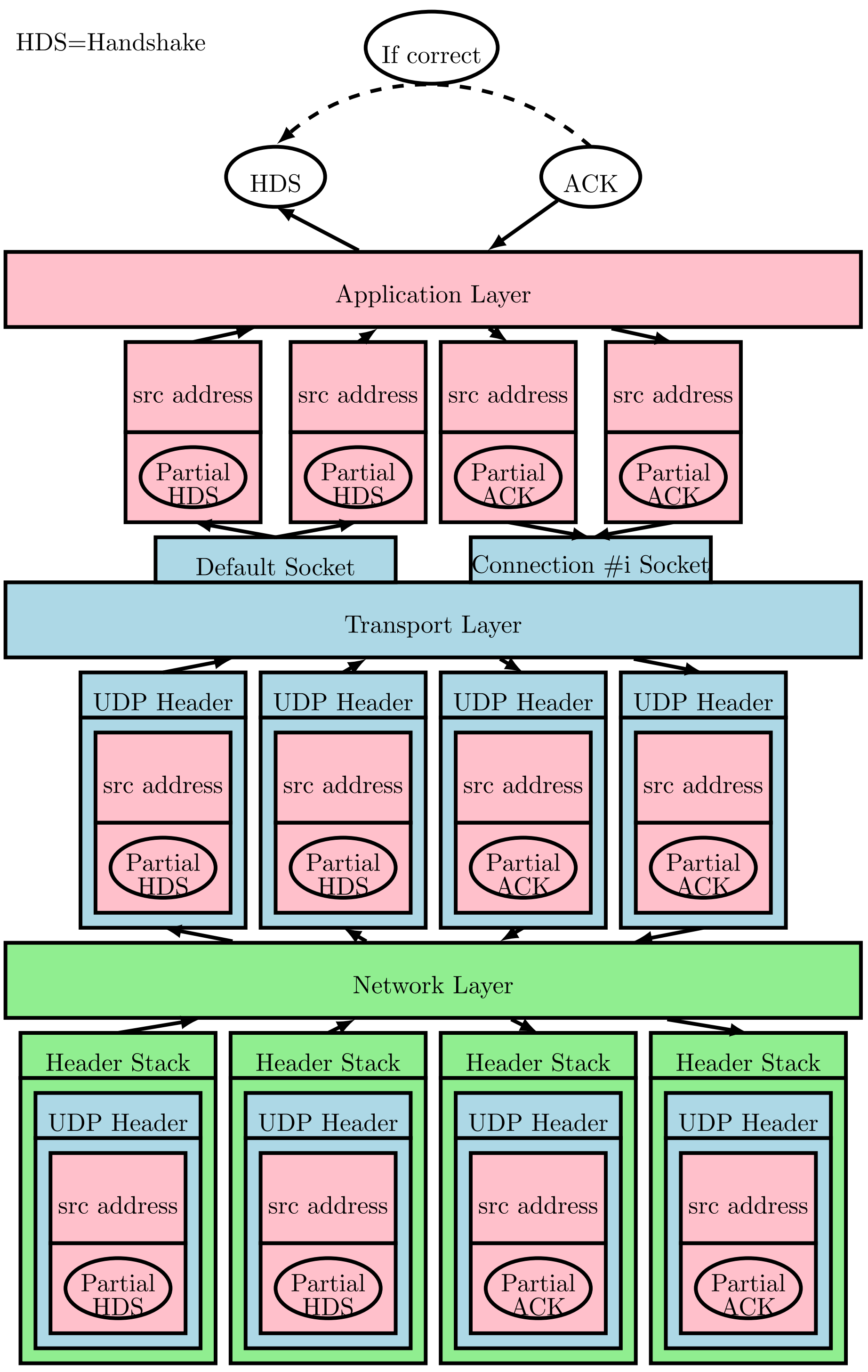

Figure 2: Pattern of modified OSI, receiver side, when handshaking

6.2 Isolated end-system :

Even on huge networks, if an observer wants to focus on one particular person, it is easy for the observer to catch all the messages coming in and out of this end-system. By comparing them, he can deduce messages produced by this particular end-system, and then the data. Nevertheless, we can cypher the data before xor-decomposing it, which reduce the risk of data leak.

6.3 IP address spoofing :

If the attacker knows the IP of the receiver, he can spoof it. Then we will send all data to the attacker. One should then take care to cypher the data before decompositing it.

7 Conclusion

7.1 Privacy :

By using multipath messages, we use the size of the network to preserve the privacy of users. We use the fact that all pieces of message are needed to understand every part of the message. This is achieved by the Xor-decomposition, and multipath routing (i.e.: Header stacks).

7.2 Anonymity :

To guarantee anonymity, the source address is not in the headers, which are not cyphered. This is achieved by the format of Network headers.

7.3 Weaknesses :

Isolated End-systems One should take care not to be isolated, but to multiply the number of links

Only on wide networks Otherwise, it is easy to monitor all links.

Only a good-enough solution[2] We can not guarantee that the data is not catched.

7.4 Strong Points :

Shared Data The sender and receiver doesn't need to share sensitive datas such as keys.

Scalability The bigger the network is, the more potential paths there are, and the harder it is for and opponent to monitor all links.

Few modification on current web architecture We only need to modify the application layer and the network layer.

Distributed system There isn't a server which manages all exchanges, each sender manages its data.

A Example

The following example shows an idea of implementation of the above method. We are given the network shown in FIG.3 p.9.

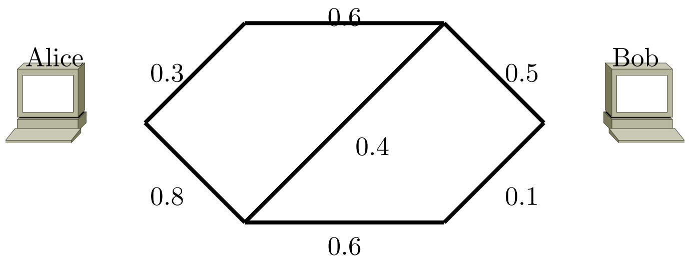

Figure 3: Example on a small network

A.1 Get receiver IP :

Directory IP : To get Bob IP, Alice should ask a directory (a bit like a phone book). But, we assume Alice doesn't know anything, i.e. she doesn't even know the directory IP. To learn it, she just have to send a broadcast message[3] asking for that IP. Alice obtain back a list of directories IPs (if people doesn't use the same IP). Alice now have a list of directories, but she deosn't know if these directories are corrupted or not. But, the bigger the network is, the more chance there are to have at least one correct directory.

Receiver IP : Alice should ask for every directory Bob IP. If a directory is corrupted, it may return a wrong IP, but we assume that the network is wide enough such that Alice has at least one correct directory. Then Alice has a list of Bob address, with at least one correct. Alice just as to send to every address a message (using the decomposition method), asking for each receiver to authenticate (with usual methods for instance). Then Alice may know which of the received addresses is correct. Notice that we can imagine a system of feedbacks on the directories, and also, when Alice ask for an address, she can inform of its address, to complete the database of the directories.

A.2 Cypher the plain text :

Alice just cyphers the text with any existing method. For instance with RSA, getting Bob public key with a system similar to the directory above. She sends the cypher text to Bob. Bob uncyphers it.

A.3 Sending :

Alice send two submessages as shown in FIG.4 p.10, with one common link. Each link is associate with a probability of being monitored. To compute the

[3]To limit the number of answer, Alice can set a short Time-to-Live.

global probability of success, we should reccursively merge serial and parallel links.

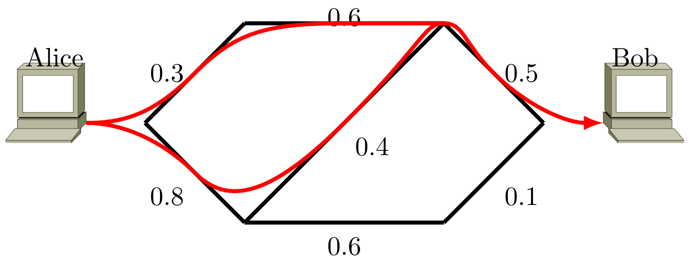

Figure 4: Pathes Alice chooses

A.3.1 Merging serial links :

To catch a submessage, one should only catch it once. That is why successfully transmit a data over serial links is successfully transmit data on every link of the path. Then we can merge serial links by multiplying success probabilities. When applying that to the given network, we get the paths shown in FIG.5 p.10.

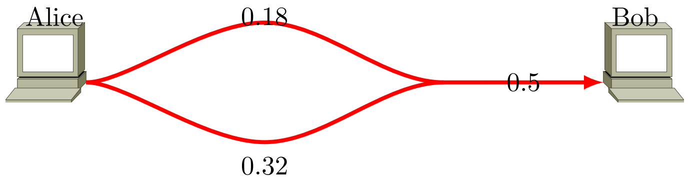

Figure 5: We merge the serial links.

A.3.2 Merging parallel links :

To get the whole message, an observer should get all submessages. That is, the transmission should fail on all separated links. Thus, to merge parallel links, we should multiply failure probability. In our example, we get the paths shown in FIG.6 p.10

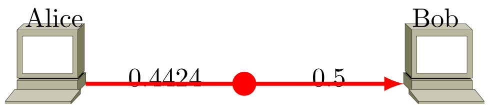

Figure 6: We merge parallel links.

A.3.3 Merging serial links :

As above, we multiply success probabilities. We finally find that the transmission has a success probability around 22% while the best path has a probability of 16% $(0.4 \times 0.8 \times 0.5)$. With only two paths, we improve the probability by approximatively 37%.

B About this essay

Writing conditions : This essay was written for the course of COMPUTER NETWORKS for second year students in computer science at EPFL, Switzerland. This course was given by Katerina ARGYRAKI.

Aknoledgements : First of all, I would like to thank Katerina ARGYRAKI, who gives us the opportunity to try to write something, and who endures all our questions. I would also thank the readers of this essay, who report a lot of mistakes.

Contact : If you have a remark, you can join me at <martin.vassor@epfl.ch>.