Anonymity in XIA: Developer Tools and User Control

Nicolas Feltman and David Naylor

1 Introduction

1.1 Anonymity

1.2 XIA

2 Levels of Anonymity

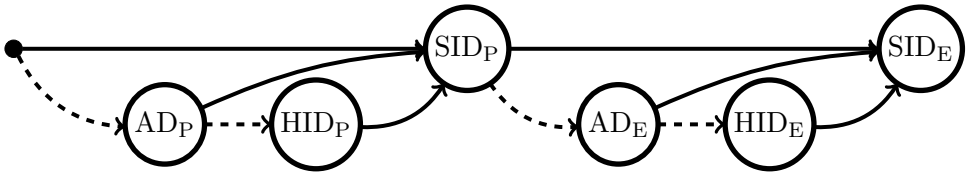
3 Approach: Proxies

The first approach we will discuss for providing anonymity is proxies. The basic idea behind a proxy is for a service requester to route all of its communications with a service provider through an auxiliary party called a proxy. Since the service provider only ever interacts with the proxy and never with the server requester directly, it does not need to know the machine address (i.e. identity) of the requester.

In this section we will discuss one possible way to implement a proxy protocol for persistent connections.

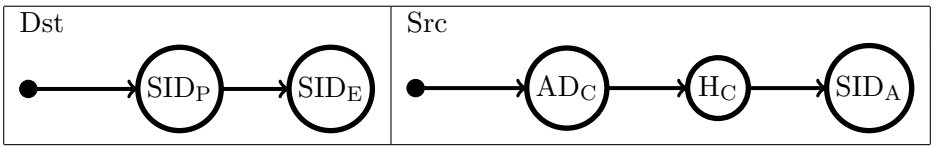
3.1 Proxy Addresses

One key feature of our proxy protocol design is including the proxy's address with the address of the service provider in a single DAG. This means that from the point of view of a client program developer, the use of a proxy can be expressed entirely within the service provider's address DAG. Although we provide an anonymous communication API, it is little more than syntactic sugar over the standard library. Although we considered introducing a new principle type for proxy services, we found that they could be expressed sufficiently with the standard service principle type. The DAG in figure (**HELP I DON'T KNOW HOW TO DO FIGURES**) represents communication with the end service $\mathrm{SID_E}$ via the proxy $\mathrm{SID_P}$, with backup paths for both. We do not cover in this document how the proxy's DAG is resolved.

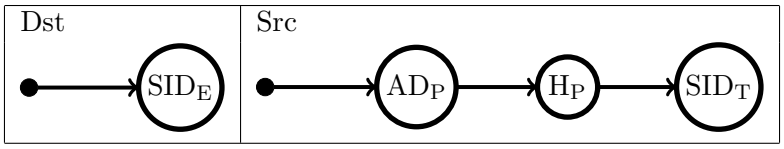


3.2 Protocol

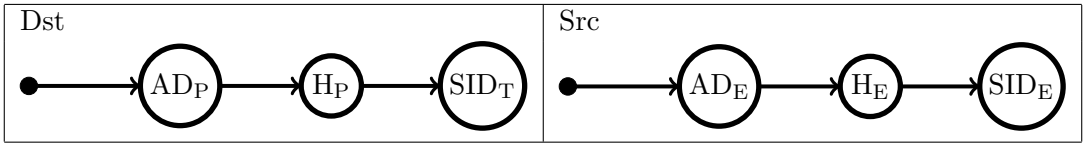
This version of the protocol assumes that the client application does not need to communicate any connection parameters with proxy. The client begins by resolving DAGs for the proxy and for the end service. Let the proxy service have the public address SID$_P$ and the end service have the address SID$_E$. Let the client application be running on a host H$_C$ with the local service identifier SID$_A$. It sends the initial packet with the following destination and source addresses (backup paths omitted):



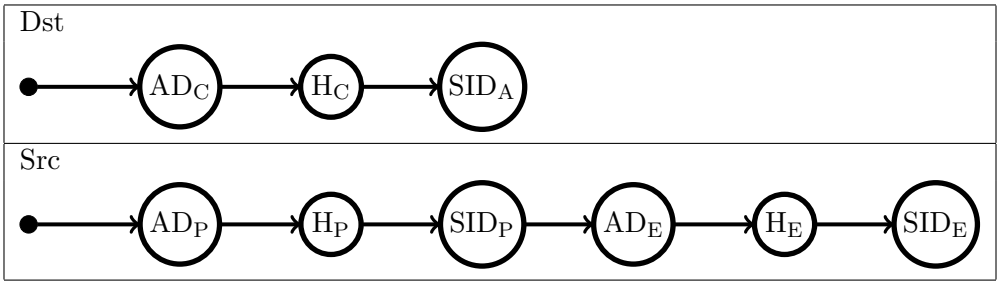
Upon receiving this initial packet, the proxy service chooses a particular machine, henceforth H$_P$, to handle this connection. If AD$_C$.HID$_C$.SID$_A$ is not already in this machine's rerouting table, the proxy creates a new local service ID, SID$_T$ (for tunnel), and adds this pair to its rerouting table. The machine then forwards the intial packet with the following destination and source addresses:



The end service is not aware that it is communicating via a proxy. It chooses a machine, H$_E$, to handle this connection, and responds with the following packet header:



The proxy, in the interest of keeping minimal state, does not remember this source address but instead sends it back to the client in what the client sees as the first response:



The client then binds to this source DAG, sending all future communications to it. Thus, the client continues communicating with the host via the proxy, the proxy swaps references to $HID_C.SID_A$ with $HID_P.SID_T$, and the end service thinks it is maintaining a persistent connection with $HID_P.SID_T$.

4 Approach: Temporary Service IDs

Our second method for anonymization involves explicit cooperation of the client's AD. In this method, the client requests a temporary service ID, or *pseudonym*, from the AD through a special mechanism. It then uses this pseudonym (bound within the AD) as its return address. This level of anonymity is acceptible for some applications.

4.1 Protocol

The client, running on host HID_C, starts by requesting a psuedonym from its administrative domain, which has the address AD_C. The AD, which must explicitly support this operation, then generates a unique service identifier, SID_P, and adds $SID_P \rightarrow HID_C$ to its forwarding tables. From here, the client simply uses the following DAG as the source address in its packets:

5 An API for Developers

As we have discussed above, XIA allows for simple, if not elegant, ways to employ techniques for anonymization. For instance, we explored how an application can achieve fine-grain control over the use of a proxy service through DAG manipulation. We also proposed the use of temporary SIDs as pseudonyms.

Both of these strategies incorporate anonymity by leveraging core features of XIA. Thus, it is both possible and useful to a standard set of tools implementing these techniques to

application developers — it would be silly for each developer to implement his/her own functions for adding a proxy service to all outgoing DAGs, for example.

We have implemented these tools in the form of the XAnonSocket API, an extension to the XSocket API. (The XSocket API, as the name suggests, allows developers to communicate with sockets over XIA, much like network applications today use TCP or UDP sockets.) The XAnonSocket API allows developers to specify how they want to achieve anonymity (i.e., by routing traffic through a proxy of their choice) just once, after which they can send packets using the specified service with no extra effort. In § 5.1 we present the interface of the XAnonSocket API.

5.1 API Interface

Function	Description
`XAnonSocket()`	Creates an anonymous XIA socket
`XAnonRegisterAnonymizer(sock, dag)`	Specifies the DAG of an anonymization (e.g., proxy) service all packets sent through `sock` should be routed through
`XAnonUseTempSID(sock, duration)`	Future packets should be sent from a temporary SID with HID elided. A new SID is generated after `duration` seconds
`XAnonConnect(sock, dag)`	Connect to `ddag` with current anonymization settings
`XAnonSend(sock, payload)`	Sends a packet with current anonymization settings
`XAnonGetStatus(sock)`	Retrieves the current anonymization settings for `sock`

6 Example Application: Web Browser