Invisible IRC Project v1.10
User's Manual

Invisible IRC Project

What is it about?

---

by codeshark

IIP stands for Invisible IRC Project. The goal of the project is to secure the anonymity of users participating in IRC. In October, 2001, [0x90] released the project. Three short months later, on December 3 version 1 final was released featuring all the tools required to chat on an IIP network or act as a relay to an IIP server. The IIP server software was distributed separately.

IIP has been exhibited as a versatile point to point anonymity encryption protocol. Since its original inception, it has been demonstrated in a number of client-server networks such as ICECAST and SHOUT cast multi-media streaming. In this way, it provides the framework for any point to point communications client or server to seamlessly integrate in to a network of anonymous nodes.

Invisible IRC Project

Printed: April 2002 in (whereever you are located)

Publisher

iip dev team

Managing Editor

codeshark

Technical Editors

[0x90]
cohesion

Cover Designer

codeshark

Team Coordinator

[0x90]

Production

codeshark

Special thanks to:

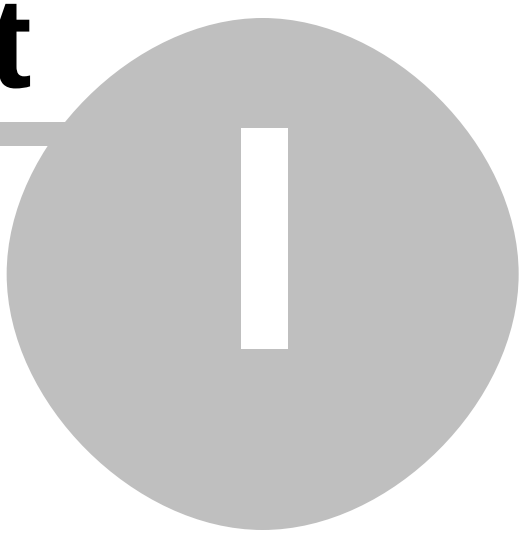
All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...

Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.

Table of Contents

Index 0

Part



Overview

1 Overview

1.1 Introduction

The Invisible IRC Project (IIP) was created for the obvious intent of bringing people interested in privacy and free speech together in a forum where development of these goals could be done in an equaly secure environment. The project's inspiration came mainly from the growing interest in the Freenet Project and Freenet's incapacity to carry real-time communication.

The software is provided AS IS without warranty, either expressed or implied. The IIP development team cannot be held responsible for the actions of the users of this software.

1.2 License

This software and document are released under the GPL license.

1.3 About This Document

This document should be able to provide you with the following understandings:

- the history, status and goals of the IIP project.
- how to set up an client/end user node, a relay node, or a new IIP network.
- how to use the network.
- how IIP works and, further, how one might use the source code to make improvement.
- the philosophy of the development group.

This document is maintained by cohesion. Any revision, comment, or question can be dirrected to the project leader, 0x90 or the writer, cohesion.
The html help is maintained by codeshark.

HTML Help compiled at 10.04.2002 01:46 by codeshark.
Revision: 1.1.0 (based on iip_1.1.2.html)

1.4 Credit

The following people get our heart felt kudos for devouting their valuable time and resources to this project:

- 0x90
- UserX
- delta407
- mids:
- codeshark
- Chocolate with a Zed
- magnade
- toobolar
- SkyRat
- fish
- Herod, our loving chat bot
- and last, but not least, Mrs. 0x90
- anyone else I inadvertantly forgot

Copyrights are held by the respective authors. Anything not explicitly copyrighted in the code base is

copyright 0x90.

1.5 Features

- end to end and node to node encryption
- fake traffic to thwart traffic analysis
- three tiered network to protect the identies of the users and the server
- working implementation of nickserv-like and chanserv-like services through "Trent" service
- self-maintaining node reference files to ensure users are always able to connect to the network
- full traditional IRC support with the exception of CTCP and DCC: compatible with all IRC clients
- OS support for all flavors of *nix including BSD, Mac OSX, and Solaris and Microsoft Windows 95 and higher
- node level flood control
- GUI interface for Windows

1.6 Versions and Version Enumeration

- Version 1.0.0: 09102001 Initial release.
- Version 1.1.0: 2002-03-31 This release. Added end to end encryption, node level flood control, routing, automated node.ref updating and more interface improvements (Windows GUI).

Changes in fundamental network architechture increment the first numerical possition (x.0.0). The addition of features increments the second numerical possition (0.x.0). Non-function-altering bug fixes increment the third numerical possition (0.0.x).

This document's revision numbers correspond to the projects first two numbers. Changes of this document occur when an increment of either of the first two possitions occurs, that is, if any program functionality changes. If the current revision is 1.1.23, this document should be labled and updated acordingly to 1.1.x. The third possition indicates the editorial revision. If this is the second revision of this document for version 1.1.x of the software, the document revision number is 1.1.2

1.7 Future Plans

Our goal is to incrementally move toward an increasingly distributed network. Version 2.0.0 should be completely distributed and autonomic.

Part II

Installation

2 Installation

2.1 Microsoft Windows Systems

The software for the windows environment comes in a single executable file named iip.exe. Download and place this file in folder by itself (ie. C:\iip\). Execute the application from the run menu. (Start > Run > C:\iip\iip.exe > OK) Upon initializing, the software will check for the presence of it's configuration files and, if they are not present, will create them in the folder it's residing in. These configuration files are user editable and we encourage you to explore them. They are as follows:

- iip.log: a log file of your proxy's connection status but not communications
- isproxy.ini: user's settings
- node.ref: a list of known relays to connect to
- listen.ref: settings for becoming a relay
- mynode.ref: information about your relay, if enabled
- seed.rnd: a Yarrow randomizer working file

If any other files appear, you may have a compromised copy of the software. Discontinue use immediately.

After file creation is complete, IIP will appear in the system tray on your taskbar and a dialog will appear requesting an available localhost TCP port to accept connections on (ie. 7777). Enter a number and press OK. You may now start your IRC client and connect to localhost on the port you opened.

/server localhost 7777

After a few tries, you should be connected to the network. See the Trent section on setting up your username and perhaps creating a new channel. I suggest joining channel #anonymous and getting to know your fellow IIPers.

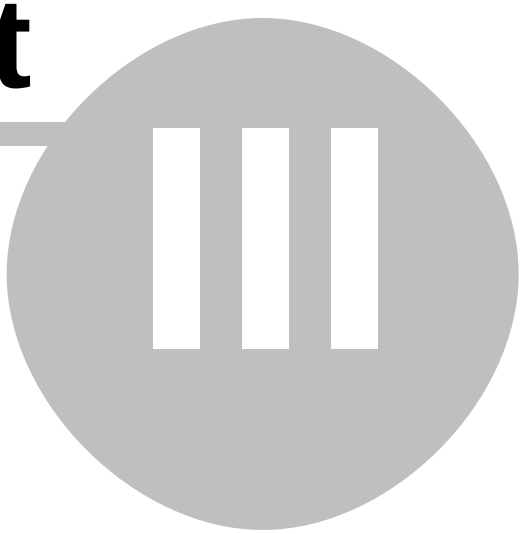
2.2 *nix Systems

Waiting for copy of 1.1.0 *nix software command interface completion.

compile requirements:

```
$ make isproxy
```

Part

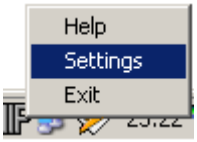


Become a part of the "Distributed Relay Architecture"

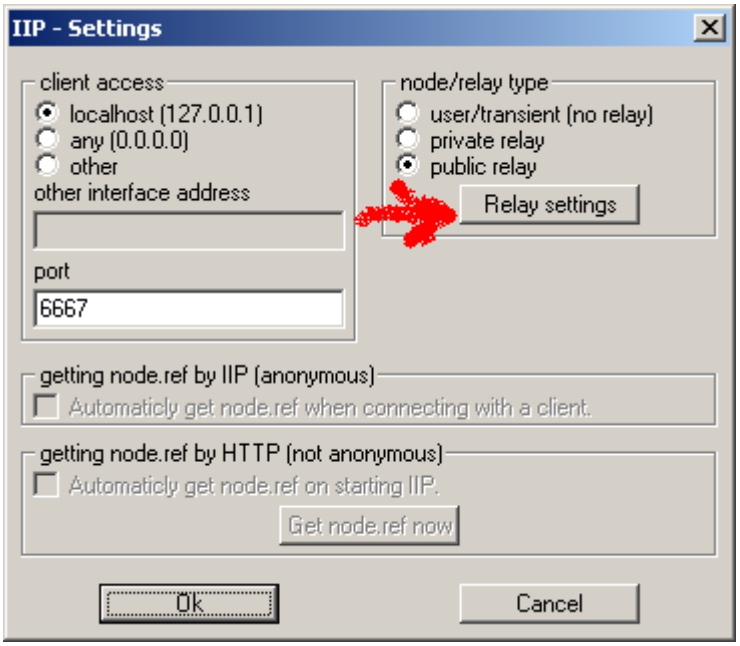
3 Become a part of the "Distributed Relay Architecture"

3.1 Microsoft Windows Systems

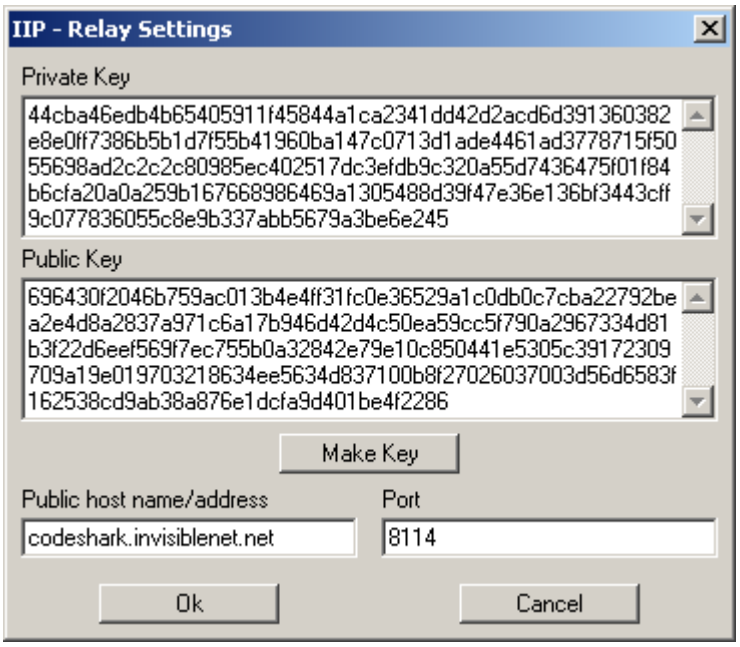
If you have fairly stable connection and would like to contribute to the network's virility, you could run a relay. All it requires is a computer that is connected to the internet all of the time and a little setup.



- Right click the IIP icon in the system tray.
- Select Settings.



- Mark the check box for Relay.
- Select Relay Settings.



- Type in your host name.
- Select Make Key and wait for it to complete.
- Select OK.
- Select Yes to announce to the network.

Be sure to add IIP to your Start Up group so that in the event of power failure, IIP will come immediately back online.

3.2 Microsoft Windows Systems

Awaiting linux CUI completion.

Part IV

Trent (nickserv and chanserv)

4 Trent (nickserv and chanserv)

4.1 Introduction

Trent is the service for IIP that controls nick and channel control. It is like the chanserv and nickserv that some other IRC networks have. Additionally it has some special features for the Instant Messenger.
The main purpose of Trent is to authenticate while keeping your anonymity. Gimmicks like auto-op and auto-voice lists are not implemented (and not planned to).
The user has some responsibility on security. Don't forget your password nor pick an easy one. There is no way to change it. If you give untrustworthy persons access to your channel with a high level, you might risk to loose it.
Nicknames and channels don't wear out, unless you drop them. In the future there might be period of inactivity after which your nick will be dropped.

4.2 Addressing Trent

The correct way to address Trent depends on your IRC client. Trent listens to the SQUERY command specified in RFC 2812. Not all IRC clients know that command or use it correctly. Both IRC commands and Trent commands may be upper- and lowercase, Trent can also be spelled in lowercase.

If you are lucky, your IRC client works like this:
```
/squery Trent command arguments
```
Otherwise you could try:
```
/squery Trent :command arguments
```
This should always work:
```
/quote squery Trent :command arguments
```

4.3 Most important commands

These are the most important commands for everyone, you should learn them thoroughly:

HELP <optionalcommandname>
`/squery Trent HELP <optionalcommandname>`
Help about commands.

NICKREG <password>
`/squery Trent NICKREG <password>`
Register the current nick you are using. The password should have at least 6 characters.

IDENTIFY <password>
`/squery Trent IDENTIFY <password>`
Identify yourself as the authorized user of your current nick.

NICKSTATUS <nick>
`/squery Trent NICKSTATUS <nick>`
Get the registration and authentication status of the specified nick. Know who you are talking with.

4.4 All commands

ADMIN
Info about the administrative contact

CHANADD <channel> <nick>
Add nick to channel access (level 2)

CHANDEL <channel> <nick>
Remove nick from channel access (level 2)

CHANDROP <channel>
Drop the channel

CHANINFO <channel>
Info about channel

CHANLEVEL <channel> <nick> <level>
Sets the access level for a nick on channel (level 2)

CHANLIST <channel>
List nicks with channel access

CHANREG <channel>
Register channel

CHANSITE <channel> <site>
Sets the channel website (level 2)

CLEAR <channel>
Deop everyone in the channel (level 2)

GHOST <nick> <password>
Rename the nick, gives you your original nick back

HELP <optionalcommandname>
Help about commands

IDENTIFY <password>
Identify with current nick

INFO
Info about the service and programmers

NICKDROP <password>
Drop the current registered nick

NICKINFO <nick>
Info about nick

NICKREG <password>
Register current nick

NICKSETSITE <url>
Sets your website

NICKSTATUS <nick>
Gives registration/identification status

OP <channel>
Op you in channel (level 1)

Part V

Philosophy

5 Philosophy

5.1 Why Anonymity

The purpose of this project is not to facilitate the evil, but to protect the capacity of individuals living in oppressive environments to communicate. IIP is a *tool for communication* just like your phone is a tool for communication. While it may be used for destructive endeavors, our belief is that far more good than bad will come of this.

See these excellent papers on the subject of freedom of expression and facilitating communication.

Part VI

The Innards Of IIP

6 The Innards Of IIP

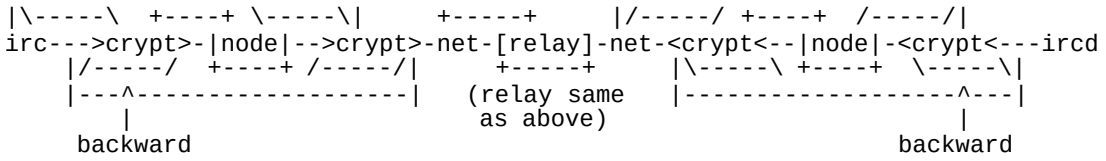
6.1 The Innards Of IIP

The network architecture is pretty self-explanitory from the docs above so this section will focus on the intricacies of specific features such as encyrption and dynamic node update.

6.2 Network Architecture

Selections from UserX's emails on the subject:

```
Hopefully this little diagram helps make things a little clearer in
regards to the direction in which backPipe goes in a chain of pipes.

                          pipe           pipe           pipeface
          sock            crypt          spurt            raw
       +-----------+   +-----------+   +-----------+   +-----------+
NULL<----backPipe  |<----backPipe  |<----backPipe  |<----backPipe  |
       |forwardPipe--->|forwardPipe--->|forwardPipe--->|forwardPipe--->NULL
       +-----------+   +-----------+   +-----------+   +-----------+


The short description of the backward pipe is that it reverses the
direction of the previous pipe (the one to the left on a protocol
parameter).

A normal crypt pipe in action:
protocol = crypt:raw
       +------+    +------+    +------+
  in--->------>---->------>---->-->in |
       |socket|    |crypt |    | raw  |
 out---<------<----<------<----<--<out|
       +------+    +------+    +------+

A crypt pipe that has been made backwards:
protocol = crypt:backward:raw
(the A, B, C, D pairs are joined)
                   +--------------------+
                   |      backward      |
       +------+    |      +------+      |    +------+
  in--->------>---->-A  C->------>-D  B->---->-->in |
       |socket|    |      |crypt |      |    | raw  |
 out---<------<----<-C  B-<------<-A  C-<----<--<out|
       +------+    |      +------+      |    +------+
                   +--------------------+

One use for the backward protocol is to perform a local test of a
pipe.
protocol = crypt:backward:crypt:raw
        \-----\   /-----/ +----+
client--->crypt>-<crypt<--|node|
        /-----/   \-----\ +----+

Another use for it is to achieve end-to-end encryption.
This is achieved by putting crypt:backward in the protocols
of the listen noderef for entry into the network (what irc
clients connect to) and the final exit noderef (the one that
connects to ircd).

how things are with IIP v1.0
     [user]                      [relay]                      [server]
    |-iip----------|     |---------iip----------|     |---------iip--|
    |              |     |                      |     |              |
    |+----+ \-----\|     |/-----/ +----+ \-----\|     |/-----/ +----+|
irc--|node|-->crypt>-net-<crypt<--|node|-->crypt>-net-<crypt<--|node|--ircd
    |+----+ /-----/|     |\-----\ +----+ /-----/|     |\-----\ +----+|
    |--------------|     |----------------------|     |--------------|

how things will be with IIP v1.1
              [user]                                  [server]
    |----------iip----------|                |---------iip-----------|
    |                       |                |                       |
```



6.3 Encryption

6.3.1 Encryption

IIP contains two layers of encryption.

6.3.2 Node to Node

When a node connects to the network, whether it's a user, relay, or broadcast server, there is an encrypted connection that is established. The encryption used is 128-bit Blowfish Cipher and session key is exchanged between nodes via Diffie-Hellman Public Key Exchange Protocol. So every node that connects to another node must handshake. Every relay node that a client may connect to has a static 1024-bit Public Key attached to it for identity reasons. When another node connects to that public relay, it handshakes using the known public key that is contain in the node.ref. Example node.ref entry:

```
host = iip.relay.com
port = 1796
protocol = crypt:raw
publickey = ffb0d51c03a9149696730b739eb2472f7b051b59dd16
e11eec49e329de597e5f07bd4290f717b2c7ce86951ae6b6eda2e859
afa8484f67067dea46f9466a4370e925d624f32707a9e90c4135e009
d4beb87883b9617208c8c0c56ad424748e976a2aed3830581ccb185a
bc6025547da2d201ce538a3001d898d886089ddfb7e8
```

The Diffie-Hellman Key Pair is generated using an implementation of Yarrow (a secure pseudo-random number generator - http://www.counterpane.com/yarrow.html) and from that point, a Diffie-Hellman shared key is calculated for the private Blowfish session key which will is implemented throughout the node to node encrypted connection. Each node does this routine per connection. If the node that it connects to does not respond to the public key that the connecting user expects from the relay, the connection is terminated and considered invalid. This is done throughout the network all the way up to the IRCd server. The main purpose of this is to protect the privacy from outside monitoring connections. This will not protect you from an "evil" node deciding to decrypt the messaging inside and monitoring his connection for messages. Hence the reason for the end to end encryption.

6.3.3 End to End

To solve the above problem, we added a network ID to the broadcast layer which is also a static 1024 bit public Diffie-Hellman Key which is placed at the top of the node.ref file as well as the protocol type, which is spurt:backward:crypt:backward:core. Spurt:backward is the fake chat size messaging protocol telling IIP to understand how that works without interfering with IRC server; crypt:backward is the cryptographic protocol also understanding the fake traffic (the backward is the traffic interpreter and director), and core is the raw IRCd server. Example of node.ref file with Network ID and node to node encryption.

```
networkid = ffb0d51c03a9149696730b739eb2472f7b051b
59dd16e11eec49e329de597e5f07bd4290f717b2c7ce86951a
e6b6eda2e859afa8484f67067dea46f9466a4370e925d624f3
2707a9e90c4135e009d4beb87883b9617208c8c0c56ad42474
8e976a2aed3830581ccb185abc6025547da2d201ce538a3001
d898d886089ddfb7e8
networkprotocol = spurt:backward:crypt:backward:raw
```

```
host = iip.relay.com
port = 1796
protocol = crypt:raw
publickey = 8484f67067dea46f9466a4370e925d624f32707
a9e90c4135e009d4beb87883b9617208c8c0c56ad424748e976
a2aed3830581ccb185abc6025547da2d201ce538a3001d898d8
86089ddfb7e8ffb0d51c03a9149696730b739eb2472f7b051b5
9dd16e11eec49e329de597e5f07bd4290f717b2c7ce86951ae6
b6eda2e859afa
```

So in this case, every node executes a key exchange and private session encryption in the same manner as the first layer, but this time it's exchanging it with the network broadcast layer which will encrypt the traffic from beginning point to end point throughout the network. Each node does this that is on the network, thus preventing reading of information being decrypted by relays, and making it less necessary to trust relays on the network.

6.3.4 Additional Encryption Methodology

In addition to just relying on standard encryption, we have implemented certain extra measures for prevention of traffic analysis and anonymity. We have implemented fake traffic which spurts throughout the network appearing to come from user clients on the network. Also, we XOR every 8 bytes of traffic so that you cannot analyze the encryption text. (e.g: Someone says test and it looks like 9z9s09er in ciphertext. If they repeat the word test, it will look completely different in ciphertext rather than the same 9z9s09er you saw before. Also, every 52 blocks of traffic (messages sent, fake or real), we rotate the session key with a new one generated from the shared key info so that both sides will negotiate trivially and without any notice on the network. This imposes a constantly moving key system which helps protect the privacy, integrity and security of the traffic on IIP. All of this is executed on both layers, end to end, and node to node.

6.4 Dynamic Node.Ref Updating

In order to ensure that clients have a consistently good connection to the network, it's important to keep the node.ref up to date with the most accurate information about the status of the network. This is done two ways:

6.4.1 Update on Join

When the end to end handshake occurs, nodes receive a short data stream from the IRCd server that contains the latest node.ref. The ability to disable this feature may be included in future versions.

6.4.2 Periodic Polling of Relays

Currently, when a node elects to become a relay, a message is sent to a inform server identifying it's host name, port number accepting connections, and its public key. The informserver periodically checks a relay for it's status. If the relay is found to be non-functioning for a certain period of time, then the relay is removed from the node.ref that the IRCd server is sending to users joining the network. These rules are called the Strict Contact Probability Algorithm and are implement to increase the reliability of the network.