WASTE

WASTE is a tool for secure distributed collaboration and communications for trusted groups of individuals. WASTE allows easy and effortless communication and data transfer. WASTE requires minimal administration and involves no central server.

WASTE provides a generic virtual secure private network for various services.

- **Instant Messaging** – allows users to communicate with other users on a private WASTE network in much the same way as when using AIM/ICQ/etc. This feature is primarily accessed through the main WASTE window.
- **Group Chat** – allows two or more users to chat on a WASTE network in much the same way as when using AIM/ICQ/IRC/etc. This feature is primarily accessed through the main WASTE window.
- **Presence** – allows users to see what other users are currently on a private WASTE network. This feature is primarily accessed through the main WASTE window, and facilitates ease in Instant Messaging.
- **Key Sharing** – allows hosts on the WASTE network to exchange public keys so that they can directly connect to each other (which helps the network optimize itself)

- **File Index** – allows users to browse a virtual directory structure for each user on the network. Each user can specify a list of directories to make available to other users on the network. This feature is primarily accessed through the WASTE Browser window.
- **File Search** – allows users to search other users' databases. Each user can specify a list of directories to make available to other users on the network. Currently searching for filenames and directory names is all that is supported, but full-text searching and meta-searching would be easily added. This feature is primarily accessed through the WASTE Browser window.
- **File Transfer** – allows users to transfer files to or from other users. Files can be found via the file browsing and file searching features, or files can be uploaded to other users manually. This feature is accessed through many interfaces, and can be managed with the WASTE File Transfer window.

Protocol

WASTE routes all data through a distributed ad-hoc network. The network structure can adapt for traffic, and is fairly organized based on capacity. When moving large amounts of data, the network is redundant and load-balanced (though the load balancing is currently sub-optimal, it does work). Because all data transfer is accomplished through this distributed network, firewalls do not impair function as long as there are sufficient hosts on the network that are accessible from everywhere.

WASTE keeps the private network private by only connecting or allowing connections between known users, and by using strong encryption to secure those links.

Once a WASTE network is up, users do not have to worry about IP addresses to connect to, firewalled machines, or other network topologies. As long as the user can connect to any other host on the WASTE network, the user can access all of the services of the WASTE network. All of this happens automatically.

WASTE network architecture

WASTE is built upon an underlying distributed network architecture that is similar to that of Gnutella. It consists of a distributed “peer to peer” network that allows communication between hosts based on the model of broadcast request routed reply, where a host sends out a broadcast message to the network, and zero or more hosts send routed replies that follow the path of the broadcast message back to the sender. WASTE uses 128 bit IDs for each new broadcast message, so that each node can track which broadcast messages it has seen, and so that it can route routed messages back to where the original broadcast message came from. Due to the logic of each node on the network, if there are multiple paths to a particular node from another node, the path that took the least time to broadcast is used for the routed reply.

Nodes on the WASTE network can decide whether or not to rebroadcast or route traffic based on their connection type (modem nodes communicating with nodes on T1s/DSL will generally not want to route).

Each node organizes a queue of messages for each connection, and prioritizes messages in the queue as appropriate for optimal network performance.

WASTE has a basic protocol for sending messages that involves the following information per message:

- 16 byte MD5 of message
  For verifying the integrity of the messages
- 1 byte TTL of message
  Used to prevent broadcast messages from saturating the network in the rare instance where multiple hosts have their routing tables overflowed, or a slow node gets very far behind in broadcasting.
- 1 byte message priority
  Tells how to prioritize the message in sending (0=highest, 255=lowest)
- 4 byte message type
  Contains information on what kind of message this is, as well whether or not it is a broadcast message, routed message, or local message.
- 2 byte message length (max of 32kb for routed messages, 2kb for broadcast messages)
- 16 byte (128 bit) message ID
- <message length bytes> message data, dependent on the message type

WASTE service implementations

WASTE currently provides numerous basic services for users on the network. Here are basic descriptions of how each service is implemented:

- Instant Messaging – text messages are broadcasted on the WASTE network, with information on the sender and the recipient. Routed replies inform the sender of the instant message when the recipient has received the message, and how long it took to go round trip.
- Group chat – text messages are broadcasted on the WASTE network, with information on the sender and the destination channel name. Automated notification messages, such as when a user joins or parts a channel, are sent via the same means. Routed replies are sent when a user receives a channel message, so that the sender can see who on the channel has gotten the message, and if not, the client can determine that the user has “pinged out”.
- Distributed presence – Two methods are used to let each user have a reliable prediction of who is on the network at any given time. The first method consists of each user periodically broadcasting (especially on each new connection brought up) its existence on the network, so that other users can see when a new user comes on, and detect when the user is no longer broadcasting their existence. The second method is a user can send a broadcast message to request replies with user names. This allows a user to quickly get a full list of who is on the network. Users detect when other users go offline when no activity from that user has been seen in a specified amount of time.

- File browsing – File browsing is accomplished by sending a broadcast message with a browse path to the network, to which each host may send routed replies with any results it may have.
- File searching – File searching is accomplished by sending a broadcast message with a search specification to the network, to which each host may send routed replies with any results it may have.
- File transfer – Efficiently implementing file transfer is a bit more complex than the other services, but it also demonstrates the flexibility of the underlying network architecture.

  When a node wishes to download a file (or portion of a file) from another node, the requesting node broadcasts a message with information on which file it is requesting (including host ID, length, file index, filename hash, etc), which portions of the file it wants sent (in 4kb blocks, up to 64 per request (these are run-length encoded for size considerations), and so on.

  When a node that has the file receives the broadcast message requesting a file, it routes one or more replies, that include information on the file that it is sending, and up to 64 of the 4k blocks of the file. If the file is larger than 64 blocks, or if any of the blocks are lost during transit (which the receiver can detect by timing out or other means), then the receiver can request more blocks (when it does so, it also includes information on what the last request was, so that the sender can efficiently manage the download). Because each request for more blocks consists of a new broadcast message, the route that blocks get sent back to the receiver can change throughout the transmission of a file.

The sender and receiver in a file transfer can compute SHA-1 hashes of the file data, to ensure reliable transfer.

Finally, to accomplish an upload, the sender sends a broadcast message to the recipient requesting the upload, which the recipient can optionally accept. Once the recipient accepts the upload, the recipient downloads the file as it would any other.

- Key distribution – WASTE also distributes public keys for connection negotiation by periodically broadcasting them on the network. If a host encounters a new public key on the network, it can optionally accept it (often with user approval), and can optionally send a routed reply to the message with its own public key.

<u>WASTE network design limitations</u>

The underlying design of the WASTE network and the basic services that run on it requires that the following conditions be met for the WASTE network to function well:

- The number of nodes on the network should be small, since the amount of traffic on the network scales more than linearly with the number of users.
- Each node on the network should trust other users on the network, since messages are inherently broadcasted (often unnecessarily) to many nodes on the network, and data is routinely routed through other nodes on the network.

WASTE cryptography

Since WASTE requires a small trusted network to function efficiently, it benefits greatly from cryptography. Using public-key encryption for session key negotiation and user authentication allows both the prevention of unknown users from joining the network as well link data security to prevent unknown users from “sniffing” network traffic.

WASTE also provides for an additional “network name or ID” that can be used to secure a network against people who do not have the name or ID. This can be useful if you wish to easily prevent multiple networks from merging, or change it to easily remove access of user(s) without having to make everybody ban those user(s) public keys.

WASTE uses a (hopefully) cryptographically secure random number generator based on the implementation in the RSA reference code. The code uses a 32 byte state, with 16 bytes of counter and 16 bytes of system entropy constantly mixed in, and produces random values by using MD5.

WASTE connections use RSA (with 1024 bit or greater public key sizes) for exchange of 56 byte Blowfish session keys, and 8 byte PCBC initialization vectors.

The link connection negotiation, where A is connecting to B, goes something like this:

1. A sends B 16 random bytes (randA), or blowFish(SHA(netname),randA) if a network name is used.
2. A sends B blowFish(randA, 20 byte SHA-1 of public key + 4 pad bytes).
3. B decrypts to get the SHA-1 of A's public key.
4. If B does not know the public key hash sent to it, B disconnects.
5. B sends A 16 random bytes (randB), or blowFish(SHA(netname),randB) if a network name is used.
6. B sends A blowFish(randB,20 byte SHA-1 of public key + 4 pad bytes).
7. A decrypts to get the SHA-1 of B's public key.
8. If A does not know the public key hash sent to it, A disconnects.
9. A looks up B's public key hash in A's local database to find B's public key (pubkey_B).
10. A generates sKeyA, which is 64 random bytes.
11. If a network name is used, A encrypts the first 56 bytes of sKeyA using the SHA-1 of the network name, to produce EsKeyA. Otherwise, EsKeyA is equal to sKeyA.
12. A sends B: RSA(pubkey_B,EsKeyA + randB) (+ = concatenated).
13. B looks up A's public key hash in B's local database to find A's public key (pubkey_A).
14. B generates sKeyB, which is 64 random bytes.
15. If a network name is used, B encrypts the first 56 bytes of sKeyB using the SHA-1 of the network name, to produce EsKeyB. Otherwise, EsKeyB is equal to sKeyB.
16. B sends A: RSA(pubKey_A, EsKeyB + randA), (+ = concatenated).
17. A decrypts using A's private key, and verifies that the last 16 bytes are equal to randA.
18. B decrypts using B's private key, and verifies that the last 16 bytes are equal to randB
19. If a network name is used, A decrypts the first 56 bytes of sKeyB using the SHA-1 of the network name.
20. If a network name is used, B decrypts the first 56 bytes of sKeyA using the SHA-1 of the network name.
21. Both A and B check to make sure that the first 56 bytes of sKeyA does not equal the first 56 bytes of sKeyB. If they do (which is statistically unrealistic and would lead one to believe it is an attack), they disconnect.
22. Both A and B check to make sure the final 8 bytes of sKeyA differs from the final 8 bytes of sKeyB. If they are equal, disconnect.
23. A uses the first 56 bytes of sKeyA XOR sKeyB to initialize Blowfish for send and receive. A uses the final 8 bytes of sKeyA as the PCBC IV for send, and the final 8 bytes of sKeyB as the PCBC IV for receive.
24. B uses the first 56 bytes of sKeyA XOR sKeyB to intialize Blowfish for send and receive. B uses the final 8 bytes of sKeyB as the PCBC IV for send, and the final 8 bytes of sKeyA as the PCBC IV for receive.
25. All further communications in both directions are encrypted using the initialized Blowfish keys and PCBC IVs.
26. A sends B the constant 16 byte signature ("MUGWHUMPJISMSYN2")
27. B decrypts verifies the signature
28. B sends A the constant 16 byte signature ("MUGWHUMPJISMSYN2")
29. A decrypts and verifies the signature
30. Message communication begins (each message uses a MD5 to detect tampering – if detected, connection is dropped)