Drexel University

Updated Protocol Design

Secure Two-Way Messaging Protocol
Version 1

Group 9

Marcos Zegarra
Sergey Matskevich
Scott McHenry
Zainul Din

June 4, 2016

1 Definitions

Peer or Host: a participant in the network, which has S2WMP client installed and is listening to request.
Friend: a peer that has been added to the list of frequent contacts.
Bootstrap server: a server machine, which provides the service of finding peers in the network.
D-H: Diffie-Hellman public key exchange algorithm.
Client: is a user facing software that is running on top of S2WMP

2 Service Description

This document describes version one of the Secure Two-Way Messaging Protocol (S2WMP), which provides secure chat services. These services include atuomatic peer finding in the local network, exchange of text messages and end-to-end encryption. Version 1 of the protocol focuses on providing services only to local area network with a possible extension to work over internet later. Current version of the protocol is build on top of TCP and relies on it for reliable message delivery. Future versions of the protocol can be extended to work over UDP, details of which described in Section 5. Protocol provides a way to discover nodes in the local network using two possible ways: using Bootstrap server or LAN broadcast.

2.1 Peer Discovery

Finding peers in the network is a key issues for pee-to-peer protocols. S2WMP provides several ways to find peers in the network. First, is utilizing bootstrap servers, which store id's of clients and their location. In version 1 of S2WMP we will have only one bootstrap server, but we will be able to add more servers, which also will be in the pee-to-peer network by utilizing *Distributed Hash Tables (DHT)*. If the server is online, a peer can make a request of it to find particular user id. If the server is not online, however, a peer can scan the network for other friends or other peers. In this version S2WMP uses broadcast over IPv4, but support for IPv6 and multicast can be added in later versions (see Section 5).

2.2 Messaging

Since S2WMP is a chat prtocol which is designed with security in mind, it provides a service for secure message exchange. All messages over S2WMP are encrypted. The choice of encryption algorithm can be made by a client, but all of the supported algorithms use assymetric encryption. The default algorithm is RSA and the keys are exhanged with Diffie-Hellman algorithm. Each peer will generate one set of public/privite key pair, which could be regenerated on demand.

3 Message and Packet Definitions

3.1 Messages

This section contains a list of messages, which is sent by the protocol. The messages are defined in the following section in the form of packets. Each packet corresponds to one or more messages. S2WMP uses following messages:

Broadcast Server (BSM): a message that is broadcasted in order to find bootstrap server in the network.

Broadcast Server Response (BSR): a message which is sent by the server in response to BSM.

Broadcast to Peer (BPM): a message that is broadcasted when server is not found. This messages is looking for other peers and friends on the local network.

Broadcast Peer Response (BPR): a message which is sent by peers in response to BPM.

Status Change (SC): a message that is send when a client have changed status.

Message (M): a message that have data in it for the other peer.

ACK (ACK): an acknowledgment that data message have been delivered.

Registration (REG): a message sent by peer to bootstrap with request to add it to the list of peers.

Reg. Reply (RREG): a message sent after processing registration request. Can contain success or failure status.

Lookup (LP): a message sent by peer to the bootstrap server when a peer wants to add a friend. This mechanism will work even across the Internet.

Lookup Response (LPR): reponse of the bootstrap for the lookup request. Contains friend address in case of success or error status in case of failure.

ID Change (IC): a message which is sent in case two client IDs have collision. They negotiate about changing their ID and then this message is sent to the friends and bootstrap server.

Chat Request: a message peer sends when adding a new friend. It is essentially a public key initiation message.

Chat Request Response: response to Chat Request. Can send public key or can deny request.

3.2 Packet Definitions

All client and User IDs are 128 bit hashes and are stored as strings in hex format. Custom wrapper is used in the project in order to simplify their usage. Also we converted all fields to even bytes, so we don't have to deal with partial bits and shifting arrays as a result. Using full bytes allows us to just read it in and convert to the right data type. All the strings are US-ASCII strings, the protocol right now does not support unicode or any other encoding. All date fields are timestamps, represented in milliseconds. S2WMP defines several different packets.

3.2.1 Messaging Packet

Figure 1 shows a packet, which two peers exchange in order to send messages and information to each other. The fields in the packet are as follows:

V Protocol version, one byte long

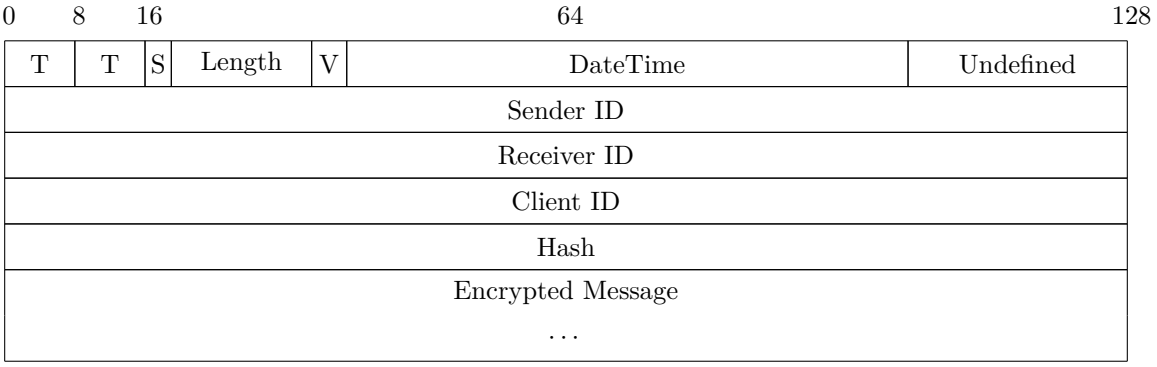

Figure 1: A packet for main communication in S2WMP

T is byte long type field. It identifies the type of packet which is being sent. Clients will have predefined enumeration of packets.

S is byte long a status field. The values are the following: 00 - online, 01 - Busy, 10 - Away, 11 - Invisible. Other values are not defied, they can be added later as an extension.

Length is a 16-bit field, which specifies the length of the encrypted message in bytes.

DateTime is a 64-bit long field, which contains the date and time of when the message was sent. It is presented as a timestamp in the milliseconds.

Undefined is a 25 bit space, which allows for future extensions of the protocol.

Sender ID is 128 bit cryptographic ID of the sender. This helps identify the contact who sends a message and process it accordingly. It is basically a hash value.

Receiver ID is 128 bit cryptographic ID of the receiver. This field is used to deliver the message. It is basically a hash value.

Client ID is 128 bit encrypted ID of the client. This identifies what client a peer uses. This field is for security, so anauthorized clients would not be able to fake messages.

Hash is a 128 bit cryptographic hash of the original message. Serves as integrity check. Does not play significant role in version 1, but will be important if switched to UDP.

Encrypted Message is a message of up to 64KB in length. This packet is only used for sending text messages, possibly with Unicode encoding, so this length of a single message should be plenty, including emojis and special symbols, and accounting for text size increase due to encryption.

3.2.2 Chat Request Packet

This packet is sent to the peer for friend request. It start D-H exchange.

V Protocol version, one byte long

T is a type field, one byte long. The same as in Figure 1.

Requestor ID is 128 bit cryptographic ID of the user, who requests to be added to the peer list. This ID can also be checked for validity to avoid unauthorized requests.

Public Key is a cryptographic 128 bit public key of the peer for D-H.

User ID 128 bit cryptographic ID of the person who this message was suppose to go. Used for validation.

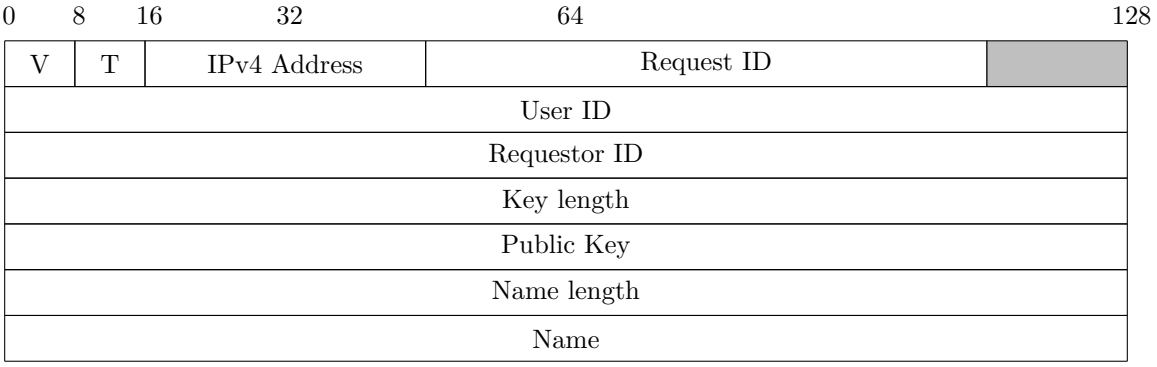

Figure 2: Registration packet

Key Length is a field, which specifies how long is the public key. In the D-H keys can vary at runtime after they are being converted to bytes. This is a 4-byte integer.

Name Length length of the user name in bytes. The field is an integer 4 bytes long.

Name textual user name.

3.2.3 Registration Packet

This packet is sent to the bootstrap server in order to be added to the list.

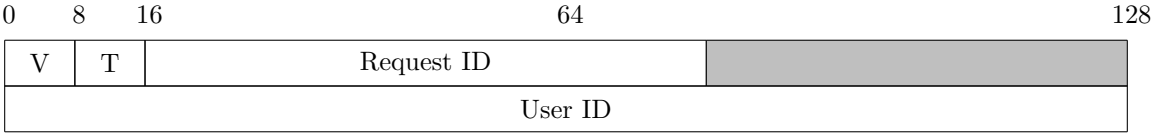

Figure 3: Registration packet

V Protocol version, one byte long

T is a type field of one byte long. The same as in Figure 1.

User ID is 128 bit cryptographic ID of the user, who requests to be added to the peer list. This ID can also be checked for validity to avoid unauthorized requests.

3.2.4 Registration Response Packet

This packet is sent back by the server in response to registration request.

Figure 4: Server registration responce packet

V Protocol version, one byte long

T is a type field of 4 bits long. The same as in Figure 1.

Request ID is Request ID field from Figure 3.

S is a 3 bit status field. The values are the following:

- 00: Operation was completed successfully.

- 01: Key or address collision.
- 10: Invalid Client ID.
- 11: Invalid User ID.

3.2.5 Lookup Packet

This packet is used for looking up peer address in the bootstrap server and for LAN discovery. It contains minimum information required for peers to find each other in the network or for a peer to find the server in the network.

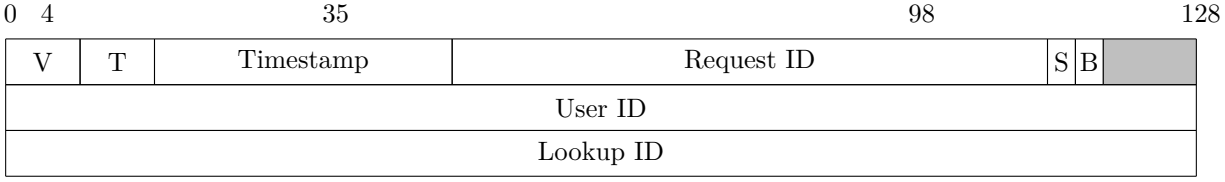

Figure 5: Lookup Packet

V Protocol version, one byte long

T is a type field, one byte long. The same as in Figure 1.

Timestamp indicates a when the lookup was requested. In case of an error, the server will keep request for some time and then discard when it becomes too old.

Request ID is Request ID field from Figure 3.

B is a flag that indicates it's a broadcast packet.

S is a byte long status field. Currently only the following values are defined and therefore valid:

- 00: User found;
- 01: User not found;
- 10: Invalid Client ID;
- 11: Invalid User ID;

User ID is 128 bit cryptographic ID of the user, who requested lookup. This ID can also be checked for validity to avoid unauthorized requests.

Lookup ID is 128 bit cryptographic ID of a peer which is being looked up.

3.2.6 ACK Packet

ACK packet is used to acknowledge or reject messages.

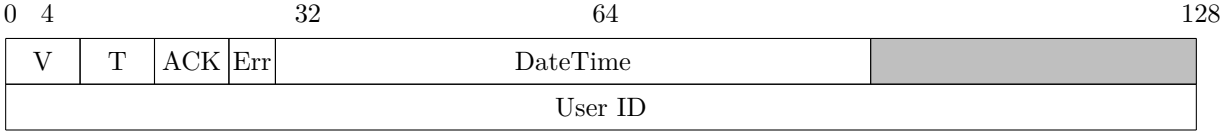

Figure 6: Lookup Packet

V Protocol version, one byte long.

T is a type field, one byte. The same as in Figure 1.

ACK is acknowledgment flag. If it's 1, the message is accepted, otherwise it's rejected.

DateTime is a date and time of the message, the same as Figure 1. It can be used in conjunction with **User ID** as a strong identifier for the message.

Err is a byte long error field, indicating what kind of error happened. The following are the valid valies in this field:

- 00: message is accepted, no error.
- 01: user id couldn't be validated.
- 02: hash of the message didn't match.
- 03: message could not be parsed because some fields were corrupted.
- 04: message could not be decrypted.

4 DFA/Finite State Machine Diagram

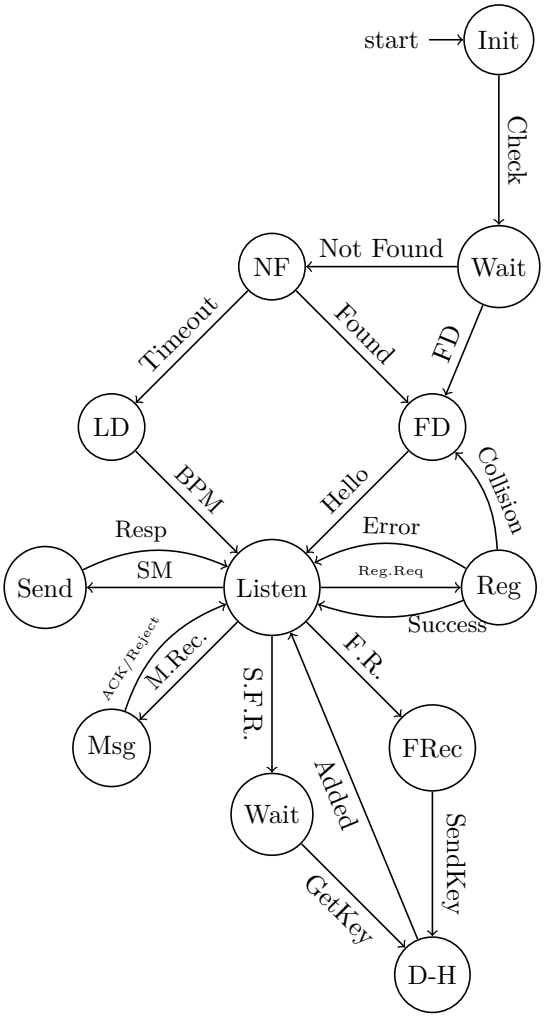

Figure 7: S2WMP State Machine

4.1 Initialization

Initialization phase (*Init*) starts when the client is invoked. First, client checks if there is a known bootstrap server in the network (*Wai* state). When communications are done over the Internet, this should always be true. The server address can be stored in the config or acquired in some other way. If the server is found, then the peer sends hello packets to all of the friends in order to tell them it is online (*FD* transition to *Listen* state). If the server is not found (state *NF*), then a broadcast message is sent to look for one. This produces 2 options: a server could exists in the network , or it might be down. If the server exists, it will respond to the *BMS* with it's IP address (transition *Found* to state *FD*). Otherwise the client will eventually timeout (transition to state *LD*). From state *FD* the peer will send **Hello** packets to it's friends, notifying them it's online. From *LD* state the peer will send *BPS* message in order to discover peers in LAN. After the notification or broadcast is made, peer will go into idle state (*Listen*), where it will listen for incoming messages. A this step the protocol initialization is complete and the client will process each message accordingly.

4.2 Adding Friends

Friends can be added by either sending or receiving requests. If friend request is received (state *FR*), then the peer already have public key of the requestor. It now sends it's own public key and the friend is added. In case the request is made, then peer sends friend request (transition *S.F.R.* to state *Wait*). Then the peer waits for the other party to respond with it's public key. Once the exchange is complete, the other peer is added to the list. Note that the packet which is sent in this process is the one depicted in Figure 3, which means that after **D-H** is completed, we also have all required information for communication as well, so no further exchanges are needed.

4.3 Messaging

The main purpose of this protocol is to provide ability to send messages. The actual messaging part is fairly straightforward. When the message is received (*Msg* state), it is checked for validity. Client ID and Friend ID is verified, as well as message integrity is checked using **Hash** field. If the problems are found, message is invalid and the packet is rejected. If the message is valid, it is handed over to client for processing and the ACK is send to the sender.

Sending message is going through *SM* path. First, the message is encrypted, then it is passed down to lower layers for sending. Once the responce with the ACK or an error code is received, the protocol returns this status to the client. In future version it can be extended with mechanisms for different error processing. It will be an easy task to add more states to this branch of the graph.

4.4 Registration with Bootstrap Server

Registration is a process of submitting peer ID to the bootstrap server. The peer first sends request packet to the server (*Reg.Req* transition), and if the server returns success, it's done. In case of error, if it is key collision, then the protocol will regenerate the key, update all of the peers in the list with new key and try to register again. In case some other error occurred, the behavior is undefined and the protocol just returns to the idle *Listen* state.

5 Extensibility

S2WMP is designed with extensibility in mind and can be extended in a variety of ways. First, is obviously extending functionality from LAN to work over the Internet. There are several mechanisms that built in in the S2WMP, which allows this extension to be done easy. The biggest problem with communicating over the internet is NAT. Version 1 of the protocol actually allows communication over the Internet, as long as the peers are not behind NAT. It is quite challenging to traverse NAT, but techniques such as UDP hole punching can be utilized for NAT traversal. This is also can be a security issue, so implementing this technique requires some time and effort in order to secure peers and test the use cases. Another requirement for communication over the Internet is the ability to find peers. This mechanism is already employed in the form of the bootstrap server. However current version of the protocol uses only one server, which is the problem once number of peers becomes large enough. Possible solution to this is using a network of bootstrap server with DHTs. This allows to perform a load balancing just by the nature of network structure and gives us an ability to have quick peer search time. Security concerns about this design are address in Section 6.

Our current methodology is to utilize TCP for communication, however this might bring new challenges in the future. For example it is harder to perform NAT traversal over TCP, as oppose to over UDP. Thus, a switch to UDP, or another similar protocol will be needed. This kind of extension will touch both clients, and the protocol. A message header in Figure 1 has **Hash** field, which can function as CRC check for the message once the switch from TCP is performed. Undefined fields in Figure Figure 1 can allow us to perform segmentation of the message and its transmission in chanks, if needed.

6 Security

Security is one of the key components of the S2WMP protocol, from the packet definitions of the **Messaging Packet** in Figure 1 to the **Registration Packet** in Figure 3 and even in the **DateTime** field in the **Lookup Packet** in Figure 6. However there are several security issues in version 1. First, it is possible to track the friend lists of peers through the bootstrap server by capturing the packet transmissions of the server when a client first registers with it. Another issue is the possibility of DOS attacks on the bootstrap server. Although this type of attack is inherent to the client-server architecture and extremely difficult to mitigate there are some options S2WMP can take. Since our protocol requires the registration of clients with the bootstrap server to first authenticate, a simple yet effective solution would be to ”ban” the client flooding the server with requests for authentication. As S2WMP increases in popularity and more bootstrap servers are implemented to handle the growing number of clients, redundancy servers can be implemented so that when one is attacked, it can be ”shutdown” and traffic routed to another server not under attack. Commercial products are also available to track incoming TCP connections in order to quickly identify these attacks and lessen the chance of disruption. The possibility of man-in-the-middle attacks is also present. If the attacker intercepts the initial key exchange and substitutes both keys for his own, then he will be able to listen and control the conversation. Methods to mitigate this attack have been explored through other protocols such as SSH. S2WMP can be set up to run over an SSH connection before reaching the lower layers of the stack. However, one of the main security designs of this protocol is its implementation of the Diffie-Hellman key exchange algorithm. Using this method combined with the **DateTime** and **UserID** fields will make it harder to conduct this type of attack. Another security concern mentioned in Section 5

deals with the implementation of DHT's. The basic concept of this implementation is the trusted notion that each client will provide its own true and correct **UserID** with correct routing tables. To address this issue, redundancy in tables could be duplicated in adjacent nodes of the DHT. Additional attacks can be mitigated by designating *Super Users* or *Privileged Users* of the network to hold these routing tables who can in turn send each other authentication integrity messages amongst themselves to monitor the health of the network. Trusted certificates can be distributed to these *users* in order to maintain their elevated status.