Bitmessage and P2P Email Security

Mark McCurry & Xiaoyun Yang

December 9, 2013

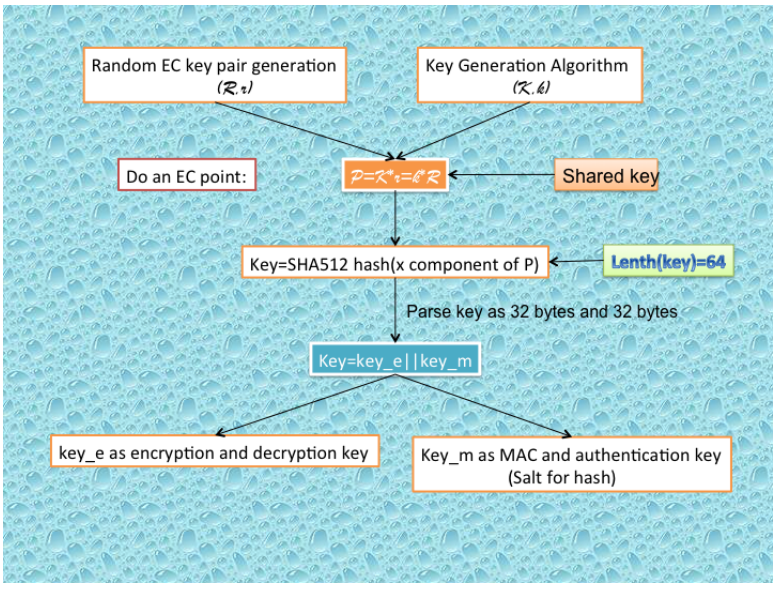
Abstract

Bitmessage is a P2P email replacement used to send encrypted messages without any trusted entities. Strong authentication and encryption are used on the messages to hide the content as well as metadata such as senders and receivers which is commonly leaked by other protocols. This protocol however appears to remain venerable to traffic analysis attacks which could easily be mounted by governments or individuals with a number of unique machines. This traffic analysis can be avoided to some degree, but clients with defaults enabled produce patterns from which a fair amount of identifying information can be gathered.

Bitmessage is a trustless decentralized peer-to-peer protocol. Unlike traditional email protocols you don't need a trusted third party, such as email server, to transmit a message with Bitmessage. This is done in an effort to avoid revealing the identity of the sender and receiver and content of the message, which is difficult or impossible to hide in most existing protocols.

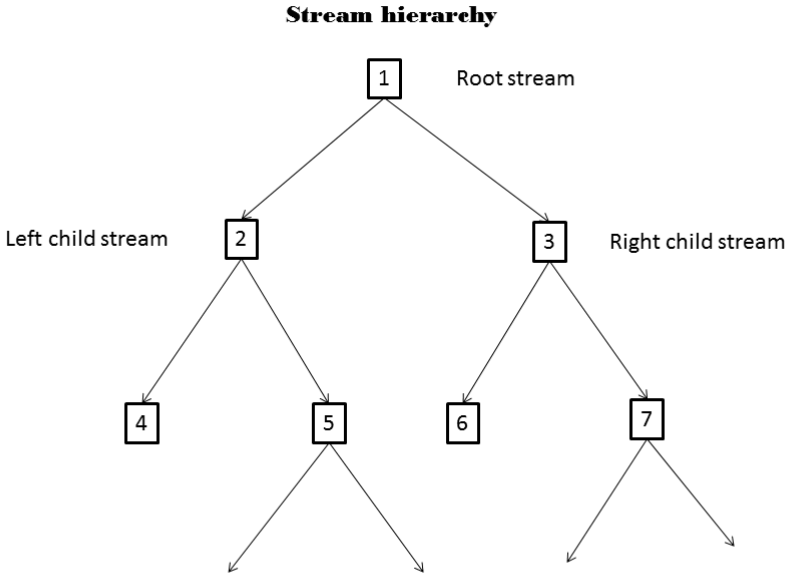
For this protocol to work, the sender and receiver need to exchange their public key which is identified by their email addresses. An example address would be *BM-2nTX1KchxgnmHvy9ntCN9r7sgKTraxczzyE*. While this is much more cumbersome than normal email addresses, it is still acceptable considering that a Bitcoin address has similar format and length, and the addresses can be easily aliased into something meaningful on individual clients.

Currently there have been two major versions of the Bitmessage protocol and encryption scheme. The most recent one gives the strongest security. It uses AES-256-CBC as the symmetric encryption scheme which is widely used and has been shown to be secure. The key generation algorithm for the shared key exploits the efficient and secure Elliptic Curve methods. Essentially, the sender will ask for the public key K of the receiver before sending the message, which will be used to generate a shared key P combined with the private component r of the key pair generated using the EC method by the sender himself. Then P will be used to encrypt the actual message. On the receiver side, P will also be generated by using its own private key k and the public component R of the key pair generated in the encrption process. And the receiver then use P to decrypt the message. We can see this more clearly in the following graph.



So how can you hide the identity of the sender and receiver when a message is sent to the receiver? Clearly, if you include the destination IP address into your message, you have already revealed some important information of the receiver, so directly sending the message is not possible. Similarly, when an acknowledgement comes back from the receiver, the identity of the sender could also be revealed. That is where the unique part of the Bitmessage protocol comes in. According to Bitmessage, messages are broadcast to *all* clients in a stream, which forms the topological local network of clients. So messages are flowing freely in a stream, but only the real receiver can decrypt the message. Although all the messages can be seen, one client cannot know where they come from, who they are going to, and what they contain. The Bitmessage clients just check whether the message is for them and forward it along to clients that don't already have it.

In order for this protocol to scale practically, there is a hierarchy stream structure, which is limits to whom the messages freely propagate to. This prevents clients from getting flooded with messages as the number of Bitmessage users grows. Basically, each new user will be assigned a stream number which means that it belongs to that stream and its messages will only be broadcast in that stream by default. When a stream has enough active users, two children streams will be created. Each node maintains a list of streams from which it can find its own connections and the two child stream. When the client can't find connections in the parent stream, it will attempt to find connection with peers in a child stream. When a sender try to send a message out, it will firstly connect with a peer in the destination stream according to the stream number encoded in the email address and send the message to it. And then the message will be broadcast in that stream until it reaches the receiver and get decrypted. The following graph gives an idea of how the stream structure looks like.



And the stream number can be easily calculated by

$$\begin{aligned} parent\ of\ n &= [\frac{n}{2}]\ if\ n > 1 \\ &= null\ if\ n = 1 \\ children\ of\ n &= [2n, 2n+1] \end{aligned}$$

Bitmessage is not flawless even though it appears to be a well designed protocol. About a year ago, a paper on the Bitcoin magazine pointed out several flaws that Bitmessage previously had. Bitmessage used RSA directly on messages which was both inefficient and insecure. The encryption scheme's use of RSA was quite similar to a cypherblock in ECB which lead to trivial attacks and the ability to flood

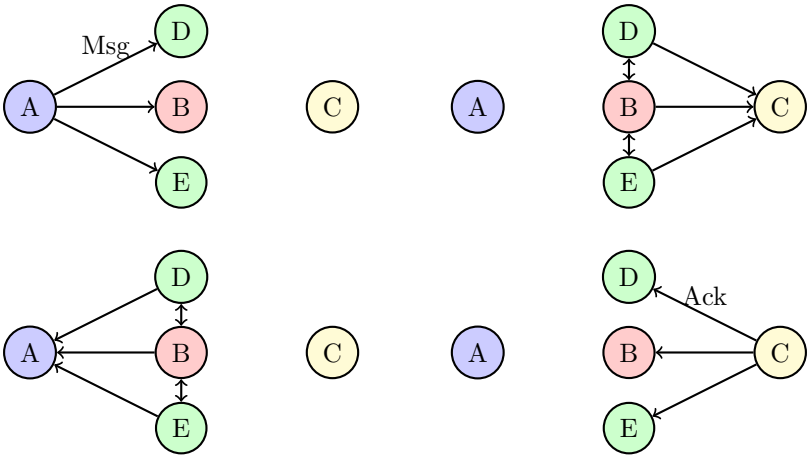


Figure 1: Example Message Transmission from A to C read clockwise from upper left

the network with seemingly valid messages. This is no longer the case as Bitmessage now uses 2084-bit keys in conjunction to Elliptic Curve Cryptography, which is known to have better security and higher efficiency than the RSA system.

As a major objective of Bitmessage is to hide the sender and receiver of a message it is logical that some definition of security is created with respect to this. The important element is that the identity of both is hidden, though identity in this case can mean a variety of things. In the Bitmessage protocol defines addresses in relationship to the associated public keys. Thus, it is said that if an attacker can associate the message with either the public key of the sender or the receiver that the scheme is broken. This does not guarantee that the message can be associated with the identities of real world persons, but it does nicely define things within the given protocol.

Given these definitions it is possible to break security using either a passive or an active observer. In these attacks it is assumed that:

- any client can be replaced with another
- that all messages are transmitted in their encrypted format as specified in the protocol definition
- acknowledge messages are enabled on a large portion of clients
- it is possible to create an oracle that observes all internode communications
- many users maintain a somewhat static IP

Given observation of all node traffic it is possible to observe the owner of a key by observing the first node to display the key. This information appears to be encrypted between nodes, but in order to utilize the public keys it must be possible to decrypt them given the information in the announcement messages. Encrypted messages can be tracked within the network by observing where a specific message is observed as identified by it's non-changing binary signature. Even though the contents cannot be observed the cyphertext should be unique with some very high probability. By observing which node is the first node to broadcast a message it is possible to know that that node created that message. Given the outlined assumptions, this node should be at the same IP address as one of the origins of a key which might be associated with the message. This should reduces the set possible public keys associated with the message by two to four orders of magnitudes depending on the number of addresses at that IP address. As for the receiver of the message it will be possible to identify that based upon the timing of the acknowledge message. While code is in place to mitigate timing attacks for the decryption of messages, it should still be possible to link the message and the acknowledgement as only 100 messages traverse the entire network in an hour and the proof of work for an acknowledge only takes a few minutes. The above sequence of transmissions can be seen below:

After this passive attack a number of public key pairs should be identified from which some probabilistic social graph can be constructed. This is very non-ideal as users that did not use a service such

as tor to proxy all communications will have a possibly personally identifiable IP address linking them to various communications.

While complete network surveillance might seem like an unrealistic assumption, it can actually be achieved with modest resources. A large portion of the Bitmessage network can be observed though the use of a modified client which connects to a larger number of clients (eg. 100 peers). With a small number of these modified clients it should be possible to minimize the distance between an snooping node and any other node out there. Though no statistics on the number of active nodes were found it is estimated that there should be less than 50,000 active nodes in the network at this time, thus even for a user with modest resources it should be possible to gain a significant advantage.

These attacks do have weaknesses that can be fixed. Either it can be made more complicated to increase the difficulty of tracking the messages as they propagate, introducing ambiguity as to when a new message has come into existence, or background noise can be increased. Approaches such as periodically sending invalid messages could be implemented, but they would increase the strain on the network. Node to node encryption would stop the passive network observer, but not compromised peers. While it would not directly stop the attack encrypting between nodes would prevent passive network snooping from observing communications. One well documented idea that could be of use is the onion routing provided by tor. Currently there are recommendations to piggyback off the existing tor network in the discussion forums, but as things stand Bitmessage is insecure in this regard by default.

A more extreme option would be to use the ideas in onion routing to obfuscate the sources of various messages using the nodes within the Bitmessage network. Each message can be sent such that it has several layers of nested encryption. Each node that can decrypt one layer of the message does so and then retransmits it. If a sufficiently long delay happens after the reception of the message before retransmission, it should be very difficult to identify associations. This method might have some practical issues, such as forcing some clients to perform more work than normal or increasing network traffic much more than necessary, though it has been studied in some detail, so existing literature may be of use.

In conclusion, Bitmessage appears to have a secure protocol, but the trustless network is a weak point that needs further study and some better defaults. As things are currently implemented it may be possible to make the analysis attacks impractical by the use of rapidly changing mail and IP addresses via generating new keys and using services like tor. This however is at the inconvenience of the user, which mean many users may be under the false assumption of security, which is certainly non-ideal.

References

- Source Code - `https://github.com/Bitmessage/PyBitmessage`
- Discussion Form - `https://bitmessage.org/forum`
- Discussion Form - `https://pay.reddit.com/r/bitmessage`
- Protocol Spec - `https://bitmessage.org/wiki/Protocol_specification`
- 2012 Security Audit - `https://bitslog.wordpress.com/2012/11/30/bitmessage-completely-broken-crypto/`
- Network Statistics - `http://vps1.adammelton.com/`