GIG-GOSSIP - P2P PROTOCOL FOR THE GIG ECONOMY

A PREPRINT

Sonof Satoshi*
sonof.satoshi@donttrustverify.org
www.Gig-gossip.org
Public Key (GPG) of the author 7

February 9, 2023

ABSTRACT

Gig-gossip protocol is a P2P, mobile-first protocol, that enables the decentralisation of the gig economy. It is built on top of Bitcoin financial infrastructure with Lightning Network for micropayments. It is spam and DDoS protected with its countermeasure on Proof of Work. Gig-gossip network nodes are financially gratified for enabling prompt communication between parties. Final payment settlements are controlled by trusted parties that certify gig workers and help in conflict resolution. These parties - payment settlers - are competing on a free market and are rewarded for the quality of their service.

Keywords gig economy · gossip protocol · distributed system · P2P network · Bitcoin · Lightning Network

1 Motivation

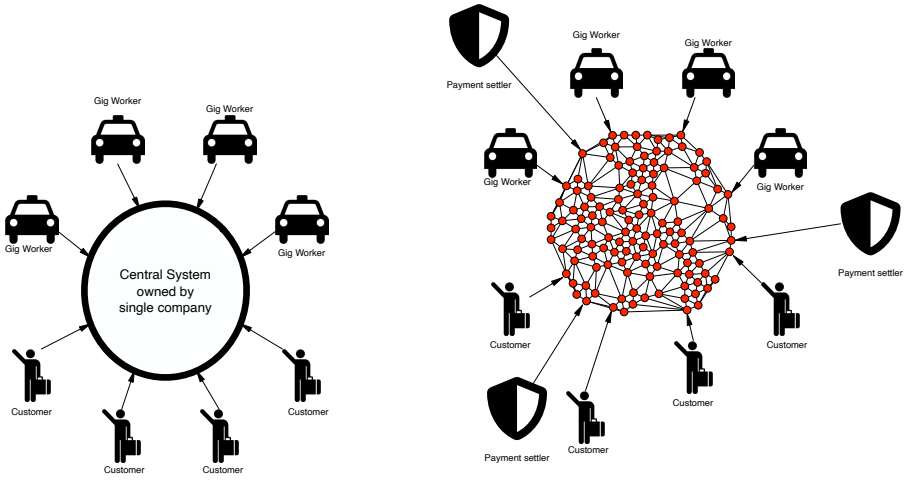
The gig economy refers to the work done by casual workers coordinated by a software system. At the time of this writing, the end customer of the gig economy interacts with a centralized, cloud-based platform (app). This platform is also used to pay for the services after the job is done to the platform, which in turn is sharing the revenue with the assigned gig worker. The actual job is done by the gig worker for their customer, making the online platform a tool that supports and manages the effectiveness of the job.

This kind of cybernetic system uses the human power of gig workers managed by AI, to extract value for the company shareholders. The AI component of the platform uses behavioural data represented as all the user (both gig workers and customers) interactions with the platform to maximize the total revenue generated by the system and the underlying company that operates it. To optimise the global goal, which is the company revenue, the platform is implementing gamification and for-purpose misinformation techniques. It is possible because gig workers and customers have no other option but to trust the platform for its efficiency and the underlying operation of the platform is opaque to users and configurable only by the operating central company. Therefore the platform operator:

1. dictates the revenue sharing and can change this anytime dependingly on the socioeconomic circumstances without giving any reason to gig workers
2. punish workers that are not behaving properly, that is aligned with company benefit, e.g. by blocking them from access to the platform

We are proposing a P2P protocol designed for the gig economy that, by eliminating the need for central online platforms, will create a new decentralized, P2P gig economy (Figure 1). Gig workers will be engaged directly by the end customer and can accomplish their tasks and earn money on a free market without the need for the existence of a central organization. Safety of the service delivery and conflict resolution is preserved by payment settlers - companies that earn their revenue by ensuring the quality of service delivery.

*We are the children of Nakamoto



(a) Centralised system (b) Decentralised system

Figure 1: Centralised vs Decentralised Gig-economy

The lack of central organization also means that a minimal volume of data is shared between gig workers customers and payment settlers. Just enough to fulfil the job according to the protocol-driven off-chain smart contract that uses P2P money i.e. Lightning Network [6] on Bitcoin [5], therefore forming layer 3 protocol from the Bitcoin perspective.

2 Gig-gossip P2P Network

Gig-gossip P2P Network is a global, symmetric, P2P network, meaning that there is no direct need to run any operation critical services in the cloud or any other centralised computing environment. Gig-gossip node is a software module that is run by every device that uses Gig-gossip protocol and forms a basis of communication. Gig-gossip nodes can be implemented as apps and run efficiently on cheap modern mobile devices. The need for implementation of supporting services that are cloud-based or edge-computing-based helps make the service more user-friendly but is never critical for the network operation.

It is important to state explicitly that we are not inventing any new coin or crypto token, but rather we are speaking about how Gig-gossip protocol forms a layer 3 protocol on top of the Lightning Network (being itself a layer 2 network sitting on top of Bitcoin network), therefore if any, the Bitcoin is a native token of the Gig-gossip network.

Gig-gossip P2P network preserves:

- P2P Symmetry - every node does the same thing
- Permissionlessness - anyone with internet access can join Gig-gossip P2P network
- Mobile first - the cost of running a Gig-gossip node is marginal on modern mobile devices. Also, the protocol handles mobile connectivity issues.
- Privacy - the communication is encrypted
- Anonymity - any personal information about the people behind the nodes is hidden
- DDos and Spam protection - it uses Proof of Work (PoW) to protect the network from DDoS and Spam
- Sustainability - the protocol is designed so all its participants benefit from joining the network
- Implicit punishment - the protocol does not explicitly punish unhonest participants, but rather makes honest participants benefit more than unhonest ones

Gig-gossip Protocol is a gossip protocol [7], that allows a network to broadcast in a similar way to gossip spreads. Assuming that each Gig-gossip node is connected to its peers and that the network graph is connected, each node works independently and in the event of receiving a message that needs to be broadcasted. It selects several peers and sends the message, in its owner's interest, to peers making the message spread over the network like gossip (Figure 2).

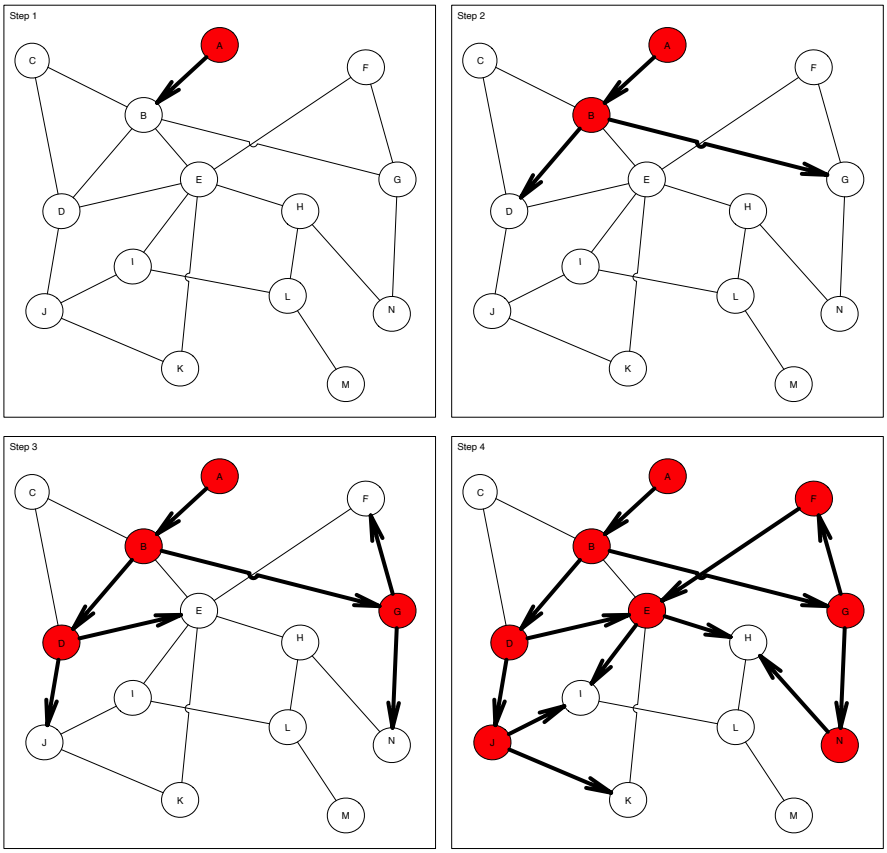


Figure 2: The intuition behind gossip protocol

Gig-gossip is a protocol, meaning that it only specifies the minimal set of rules to make it beneficial for all the nodes. It doesn't say explicitly how the network node should be implemented. The node implementation is free to do whatever is best to make it beneficial for the node owner.

3 The protocol

The main purpose of Gig-gossip protocol is to broadcast a job proposal (topic) to interested parties and collect job offers (reply messages) from interested contractors. Economically, the customer is interested in exchanging their money for the service, while gig worker is interested in being paid for the job done. The network to sustain its existence needs to reward broadcasters for the quality of the broadcasting. Therefore the protocol is constructed in a way that the customer broadcasts a topic, that contains an anonymous job description. This job description is delivered by the network to the gig worker that responds with a network-encrypted reply message. The message is delivered back to the customer that can verify the basic properties of the gig worker with their digital certificate but to decrypt the contact information of the gig worker is obligated to pay the network. The payment settlement is secured by payment settlers - companies that secure service quality.

For sake of clarity, we use the following naming convention:

1. Topic - the job proposal broadcasted through the network
2. Reply Message - the job offer for a specific topic sent from the contractor
3. Peer - any gossip network node. Every peer maintains a list of their peers.
4. Sender - Peer that is the source of the topic. From the gig economy point of view, it is a customer.
5. Originator - Peer that is currently broadcasting the topic to its peers.
6. Middleman - Peer that is passing the broadcasted topic further as well as bringing back the reply message

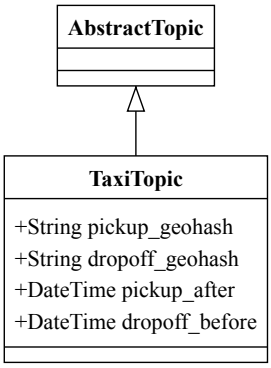


Figure 3: The topic frame

7. Replier - Peer that is replying to the broadcasted topic with a reply message.
8. Settler - the specific payment-settler

We assume that nodes of the Gig-gossip network are already connected to their peers via some internet transport protocol (e.g. TCP, UDP with or without hole punching [4], mobile mesh etc.) and all the peers are accepting Gig-gossip protocol. How the nodes discover their peers is not a part of the protocol.

In short, the Gig-gossip protocol can be summarised as follows:

1. **Asking for broadcast** If the originator wants to broadcast the topic the first step is to ask its peer (the middleman) about the condition on which the peer is happy to broadcast the specific topic. If the middleman accepts this kind of topic, it replies to the originator with specific POW properties the originator needs to compute to be able to broadcast the topic by this middleman.
2. **Broadcast with POW:** In order to use this middleman the originator must compute a hash (e.g. SHA256) that is less or equal to the specific target for the specific POW scheme. The computed POW is passed with the topic to the broadcaster and if the broadcaster validates the hash it broadcast the topic to its peers.
3. **Replying:** If instead, the middleman is interested in accepting the job it becomes the replier. Replier constructs the reply message. The reply message has to be encrypted by the selected payment settler. The message is then passed back to the sender by the network. The reply message contains all the information that is required to pay for the reply message delivery to all the middlemen involved.
4. **Paying for the reply message:** Once the message reaches the sender, the sender needs to pay the network to obtain the key allowing for message description. Once paid the reply message is revealed and the originator can begin direct communication with the replier. The payment is settled by the payment settler and the settlement of the payment involves also the payment to the network.

3.1 Topic

The topic is a data structure that defines the basic requirements for the job. It is application specific and by design, it should not reveal any information about senders allowing for their identification.

Let's use a taxi app as an example Figure 3. The topic of this taxi app has a form of two geo-hashes and time intervals describing from where and when the ride can be executed. Geo-hash here is a way of encoding a specific geographical place (a geographical cell that has a form of rectangle) in a form of a string where the length of the geo-hash determines its precision.

For example, Legal Services Council in Sydney is located at the following coordinates latitude= -33.8647 and longitude=151.2096 and the corresponding geo-hash of precision 7 is equal to 'r3gx2g5'.

The precision of geo-hash determines the size of the cell (Table 1) and to be useful for the taxi app it needs to be at least 7 so the cell has a size lower than 200m (see table below)

On the other hand, we do not want to be too specific and we might want to restrict the size of geo-hash to at most 8, so it is not possible to precisely locate the originator (customer) at this stage, but on the other side, the precision is enough for the taxi driver to accept/reject to the job.

Geo-hash length	Cell width	Cell height
1	$\leq$ 5,000km	$\times$ 5,000km
2	$\leq$ 1,250km	$\times$ 625km
3	$\leq$ 156km	$\times$ 156km
4	$\leq$ 39.1km	$\times$ 19.5km
5	$\leq$ 4.89km	$\times$ 4.89km
6	$\leq$ 1.22km	$\times$ 0.61km
7	$\leq$ 153m	$\times$ 153m
8	$\leq$ 38.2m	$\times$ 19.1m
9	$\leq$ 4.77m	$\times$ 4.77m
10	$\leq$ 1.19m	$\times$ 0.596m
11	$\leq$ 149mm	$\times$ 149mm
12	$\leq$ 37.2mm	$\times$ 18.6mm

Table 1: Geo-hash precision

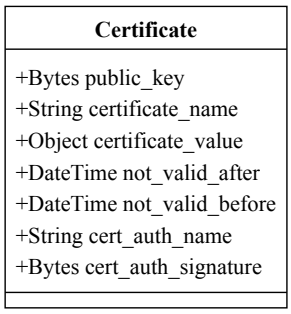


Figure 4: The digital certificate

3.2 Digital Certificates

Every gig economy environment needs to be safe for both customers and gig workers. Safety means here the ability to have physical protection from fraud during the service delivery, either if it is a taxi ride, delivering food or programming a website. The way to implement physical levels of trust in the internet is done with Digital Certificates implemented as public-key certificates (e.g. X.509 certificates). These certificates are issued by certification authorities that can be either trusted 3rd parties or communities. For a taxi driver, the minimal certification requires having a valid driving licence and no criminal record. The trusted 3rd party can issue this kind of certificate by signing it with its private key so anyone can verify that the specific certificate was truly issued by this trusted 3rd party. If the certificate is revoked the information about it is published by a trusted 3rd party in form of a revocation list. Public key certificates contain also a public key of the certified person (Figure 4), so it is possible to use it to encrypt a message that is targetted for this person and verify their signatures.

A trusted 3rd party can be a payment settler. This will make the certification authorities financially benefit from being a part of the Gig-gossip protocol and allow them to focus on their work.

3.3 Asking For Broadcast

The first step of Gig-gossip protocol is to send the AskForBroadcastFrame (Figure 5) to the potential broadcaster.

AskForBroadcastFrame contains ask_identifier and digitally signed RequestPayload. Signed RequestPayload is made of unique payload_id, topic (e.g. TaxiTopic), sender_certificate and sender signature obtained by signing the RequestPayload with the sender private key that is complementary to the public key stored within the sender_certificate. Anyone can verify the RequestPayload by validating its signature with the sender's public key from the certificate. The sender certificate can always be verified using the public key of the certification authority and checking its published revocation list.

Ask_identifier identifies the frame during the originator-middleman ping-pong communication (Figure 6). In the gossip protocol, the same broadcasting message may hit the same gossip node many times, so payload_id is to remain a unique

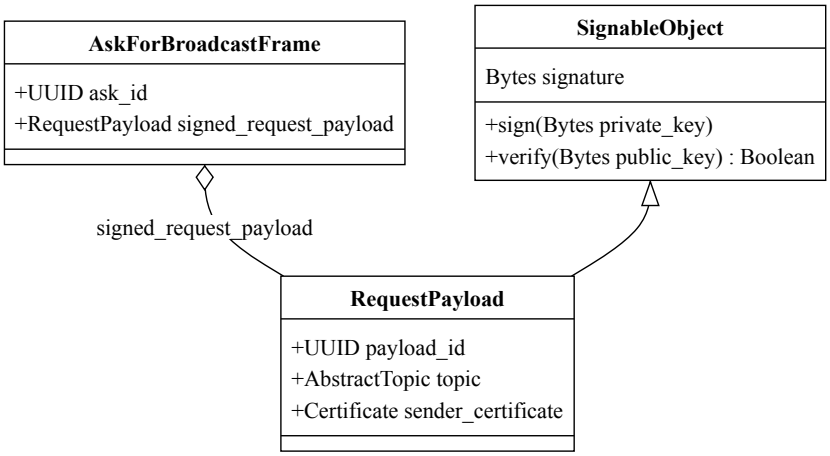


Figure 5: The AskForBroadcastFrame

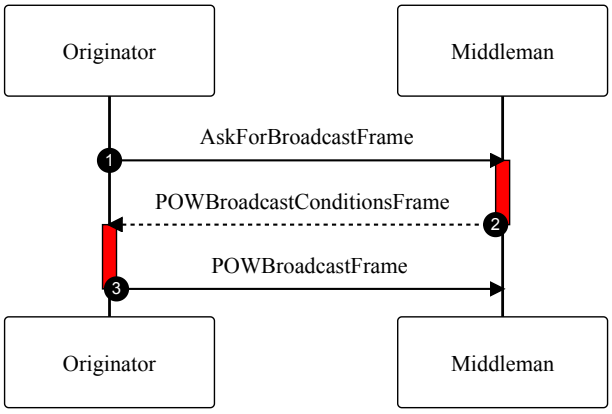


Figure 6: Ping Pong communication sequence

identifier that allows one to determine this situation and react accordingly to the node policy. Some nodes might wish not to broadcast the RequestPayload more than once, others if working on small fanout might wish to broadcast the same RequestPayload many times. Anyway, it is the requested responsibility to ensure the uniqueness of payload_id, risking if it is not unique it will be lost during the broadcast as other nodes can decide that it was already broadcasted if the payload_id was already seen before.

3.4 Proof Of Work (POW)

Message broadcast in Gig-gossip is protected with the idea of Proof of Work. POW was introduced in HashCash [1] and famously implemented in bitcoin mining but was originally introduced to limit email spam. The thinking here is that if the originator needs to make some significant computation to be able to send the message it will significantly increase the cost of spam and DDoS attacks.

There are many possible POW schemas (e.g. SHA hash-based POW, similar to the one implemented in the bitcoin network [5]). In Gig-gossip, given the topic, the middleman decides the kind and how complex POW is required to be computed by the originator to allow him to further spreading of this topic. For example in the case of hash-based POW, the task is to compute the hash of the specific payload so the hash itself is lower or equal to a specific target. The larger target is the more complex and costly the computation is. On the other hand, once the hash is computed, it is easy to verify that it fits into the specific target, so the broadcaster has an easy task here to make sure that the originator has done the work to compute the correct hash.

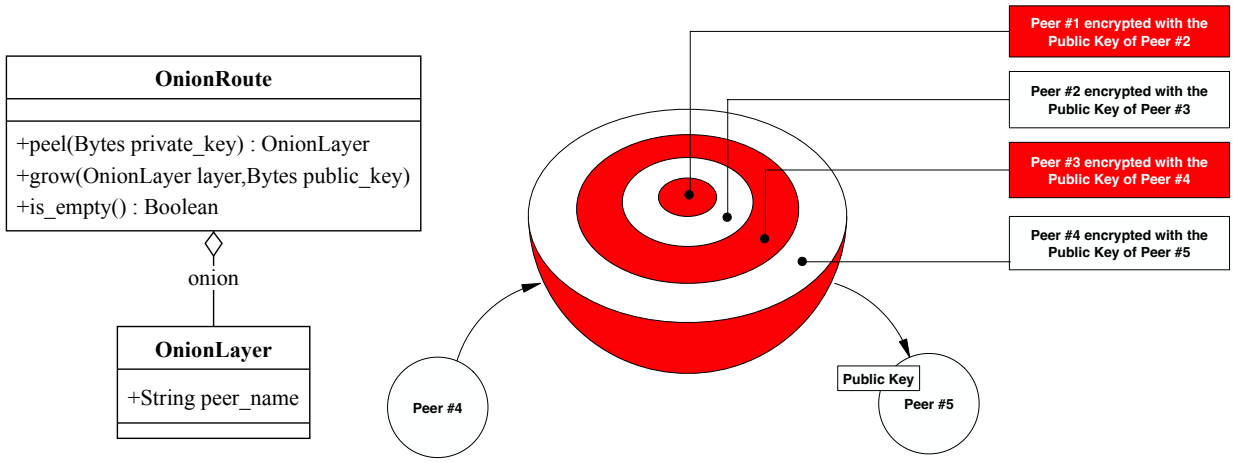


Figure 7: Onion-routing

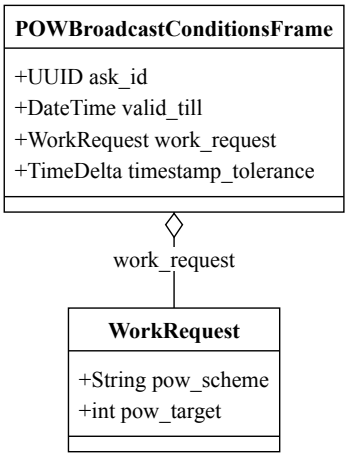


Figure 8: POWBroadcastConditionsFrame

3.5 Onion-routing

Gig-gossip is using the onion-routing technique to hide the message reply route (the route that the replies go from the replier to the sender) from the participating middlemen. During the broadcast phase, the onion grows layer by layer. Active peer appends its address to the onion and is using the public key of the next peer to encrypt the grown onion, therefore only the next peer can decrypt that layer of the onion. Once encrypted the onion is passed to the next peer.

This way of constructing the onion allows peeling the onion back to the sender through the network in a way that none of the nodes knows the sender nor the distant peers.

3.6 Broadcast with POW

If the middleman accepts the topic specified in the AskForBroadcastFrame, it sends back the POWBroadcastConditionsFrame (Figure 6). This frame describes the properties of POW expected to be computed by the originator and payment instructions expected by the peer for delivering the reply.

Starting with ask_id, which matches with AskForBroadcastFrame, and valid_till timeout meaning that the middleman will wait only till the specific time for the POWBroadcastConditionFrame (Figure 8) from the originator, it contains also WorkRequest that describes properties of POW. Additionally, to prevent reusing POW the timestamp_tolerance is sent, meaning the maximal time-distance from the timeout that is accepted by the broadcaster.

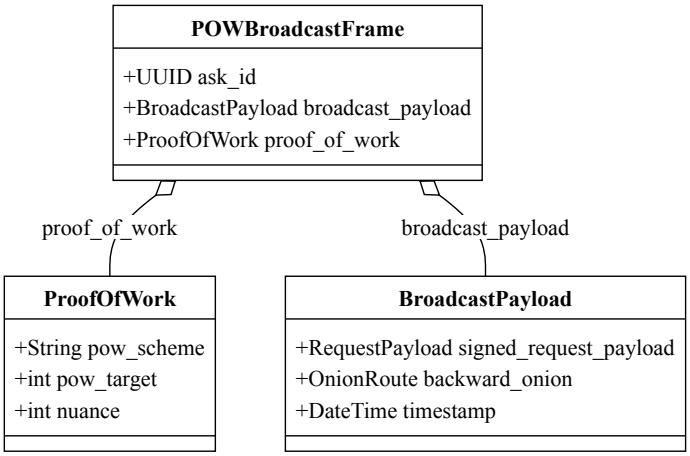


Figure 9: POWBroadcastFrame

The originator is replying to it with POWBroadcastFrame which is also marked with the corresponding ask_id. The main part is a broadcast_payload that contains the original signed_request_payload (the one that was a part of AskForBroadcast and was already signed by the originator) and backward_onion that implements the onion routing.

POWBroadcastFrame also contains ProofOfWork that contains a hash value (nuance) that fits below pow_target for the specific pow_scheme and was computed as a hash of broadcast_payload part, therefore middleman can easily verify nuance value by computing the hash of broadcast_payload and checking if it is lower or equal to the pow_target.

Additionally, POWBroadcastFrame contains the timestamp that makes the frame POW impossible to use after the timestamp_tolerance is reached. In other words, the following must hold:

$$\text{timestamp} \leq \textbf{now} \leq \text{timestamp+timestamp\_tolerance}$$

Each step of the broadcast involves passing a specific BroadcastPayload that consists of RequestPayload that is never changed and protected by the cryptographic signature. Additionally, every BroadcastPayload adds a new layer to the backward_onion.

In the gossip protocol nodes are randomly selected from the list of all the known peers of the originator. This number is sometimes referred to as fanout [7] of the gossip protocol. Once selected the broadcasting process is performed.

It's important to notice that while in vanilla gossip we are focused on making sure most of the nodes have retrieved the message, in Gig-gossip it is beneficiary for the network to build multiple routes that will provide the customer with a way to choose the one that has the minimal network fee. Therefore, here we allow for multiple crossing of the same node with the same topic on purpose, being a decision made by the node owner.

3.7 Lightning Network, HODL invoices, payments, preimages and payment-hashes

Lightning Network is a layer 2 network built on top of the Bitcoin network that allows for cheap and fast micropayments. It is built around the concept of channels. Once the channel is opened (that usually means funding it with some amount of Bitcoin), it can be used to issue and pay the invoice. Payment generates proof.

The interesting kind of invoices implemented in Lightning Network is HODL invoices. This idea extends the invoice-payment process with the cryptographic concept of preimage for payment hash in the following way:

1. Invoice is issued by the issuer and as one of the files it contains a specific payment hash. Payment hash is a hash of preimage that itself is a number known only to its creator, that in the case of HODL invoices is usually a different actor: a settler. The payment hash is the only thing that is exposed on the invoice.
2. Payer is paying the HODL invoice under the condition of having the preimage published. In other words, the payment means that the invoice is paid if the preimage is revealed and that this preimage has the payment hash that was presented on the invoice. Once paid the invoice is called Accepted.
3. Accepted invoice is Settled once the preimage is revealed.

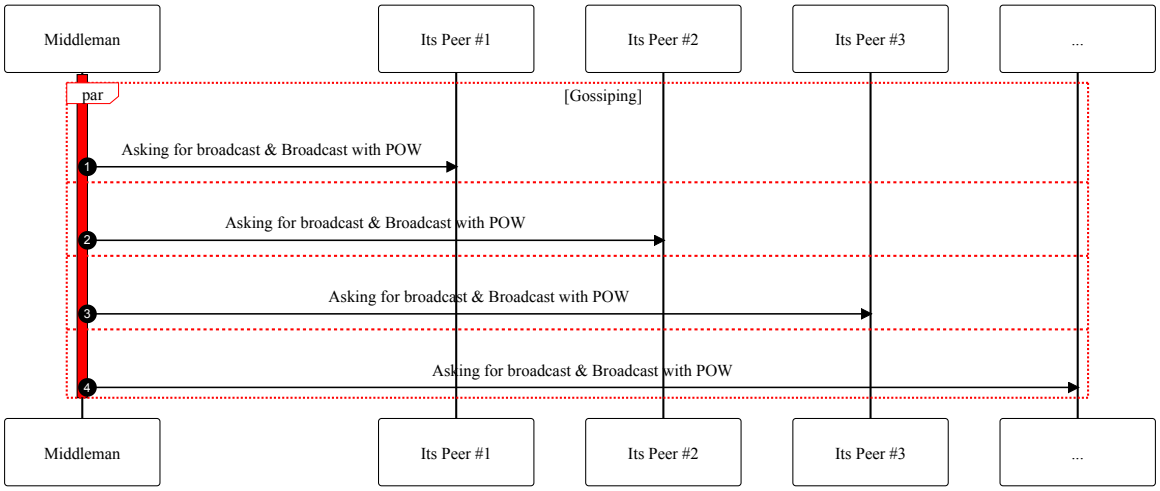


Figure 10: Broadcast Sequence

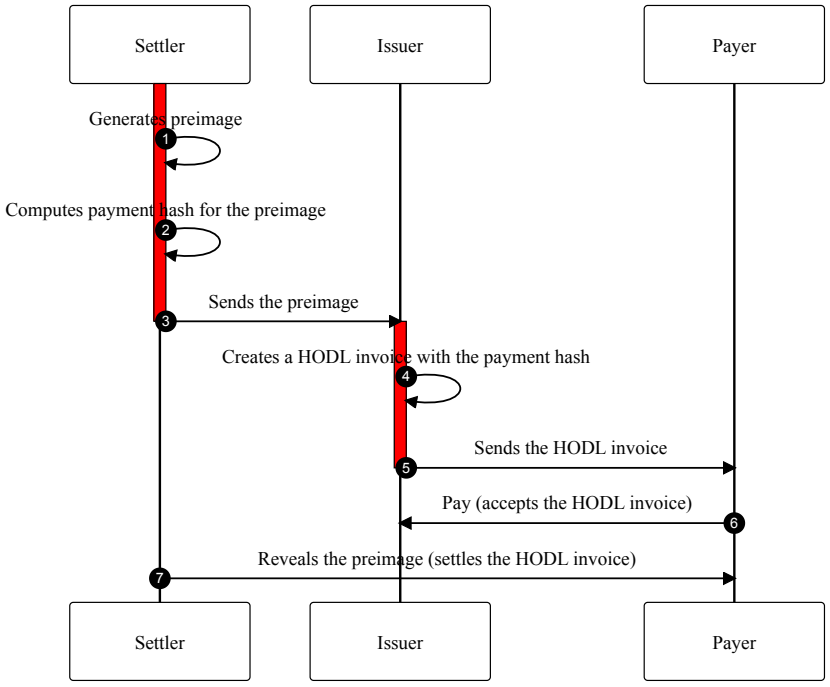


Figure 11: Lightning Network HODL Invoice Sequence

3.8 Cryptographic Payment Chains base on HODL invoices

If one uses cryptographic keys as a preimage for generating payment-hash for the invoice it implements a payment transaction for cryptographic keys. These keys might allow the description of the specific encrypted message. In other words, having a message that is encrypted with a specific key we can construct an invoice using the key as a preimage and compute its payment hash. To decode the message payer need to pay the invoice and obtain the key (preimage).

We can also create a chain of issuers that are becoming middlemen between the payee and settler (Figure 12). The settler generates the preimage (cryptographic key) and uses it to encrypt the specific message that is revealed to the public. The settler issue the invoice containing the payment hash for the generated preimage. Issuers in the chain, one after another are generating invoices using the same payment hash received from the settler. Once the last one - a payee is reached, the payer can decide to pay the final invoice so they can receive the preimage = key to decode the message. HODL invoice ensures the issuers that the payment was done (invoice is accepted) but to have control on the money, they need to use the preimage, so all the chain needs to see the preimage, therefore they are forced to pay invoices of their peers. Once the last invoice is paid to the settler, the settler reveals the preimage and all the invoices are settled at the same time. Also as the preimage = key, the message can be decoded.

There are two issues with this scheme that make the settler a special entity that manages the trust:

1. The preimage = key corresponds to the encrypted message, but it is not possible to see that just looking at the payment hash and the encrypted message, therefore all the issuers and the payer need to trust the settler that he has encrypted the message with the specific preimage, that corresponds to the payment-hash on all the invoices.
2. Preimage must be revealed only **after** the settler receives the payment. If for some reason the settler decides to reveal the preimage=key before receiving the payment the payer does not need to pay the invoice and therefore no one will be paid.

Removing the need for trust settler is one of the challenges at this stage but do not limit the application of the protocol to the specific scenario, which is the decentralisation of the Gig-economy. Gig workers are fulfilling physical services, and therefore there is a need for their certification and managing trust anyway. The certification authorities are already needed and they can naturally become the payment settlers for Gig-gossip.

3.9 Replying

The node (replier) that is happy to accept the broadcasted message instead of broadcasting it further is replying to it back. It is done with ReplyFrame which is sent back to the node that was the sender of the topic.

The replyier_certificate has two meanings here:

1. It allows the sender of the topic to identify that the gig contractor is a credible service provider by checking the certificate and verifying its "hard" certification (e.g. driving licence) with the specific certification authority being a trusted third party (e.g. government agency or specialised certification verifier)
2. It contains the replier public_key that is used to verify the signature of the Reply Payload.

The main part of the reply is the message that is encrypted so it can be used only after the sender (customer) will pay the network. The encryption is done by the replier and its correctness is verified by the network while travelling back to the sender. This is done in the following way:

1. Replier encrypts the message with the sender public_key that is a part of the sender's Certificate in the RequestPayload
2. Replier generates reply_invoice that can be viewed by the sender and must be used to pay for the service after the service is delivered. The sender is not obligated to pay it if decides not to use the replier. On the other hand, the invoice is signed by the replier and the replier cannot change it after the sender accepts the service to be provided by the replier.
3. Replier generates a symmetric key that will be used as a preimage for payment hash used in network invoices.
4. Replier performs symmetric encryption using the symmetric key
5. Replier computes payment hash for the generated preimage
6. The encrypted message needs the symmetric key and the private key of the sender to be decrypted, making it obligatory for the sender to obtain the preimage, and therefore the network invoice needs to be settled.

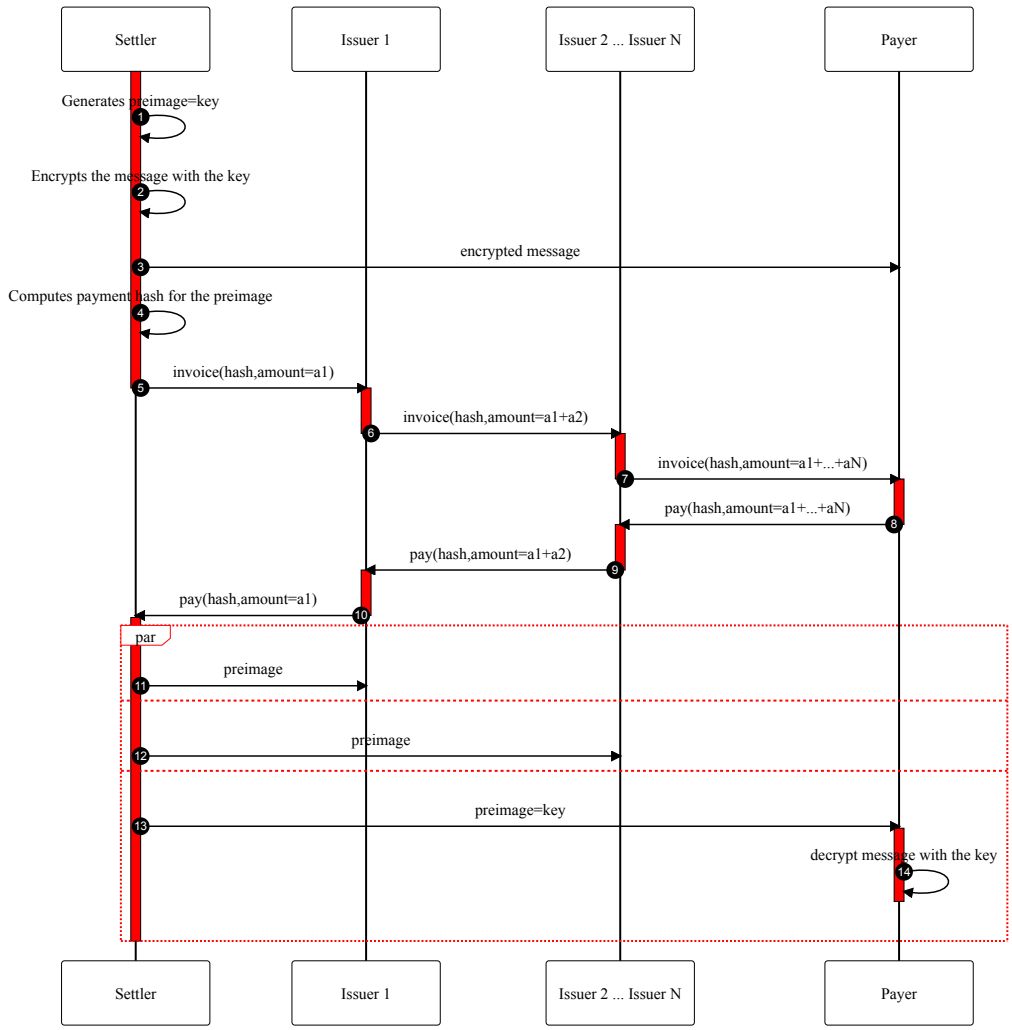


Figure 12: Cryptographic Payment Chain Sequence

7. Request payload, encrypted message, network_payment_hash and reply_invoice are signed with the private key of the replier, so its integrity can be verified with the replier Certificate

The ReplyFrame contains Replier Certificate, ReplyPayload and additionally:

- forward onion
- network invoice

When replying the forward onion is simply a copy of the backward onion, and then each of the middlemen is peeling one layer of the onion, using its private key and sending it to the node that is found there in the peel.

Replier generates the original network invoice and puts their precomputed payment hash and starts listening to its state change. If the state of this invoice changes to Accepted it settles the invoice with a known preimage. Then, while travelling back, each of the middlemen adds its price on top of the invoiced one and replaces the network invoice with its one and starts to listen to its Accepted state. Also, it makes sure that the payment hash of the network invoice is the same as stored in the ReplyPayload. Once it reaches the sender and the sender Accepts the network invoice the chain of acceptances reaches the replier. Once the replier settles its invoice the chain of settlement reaches the sender.

The sender retrieving the preimage uses it to decrypt the message.

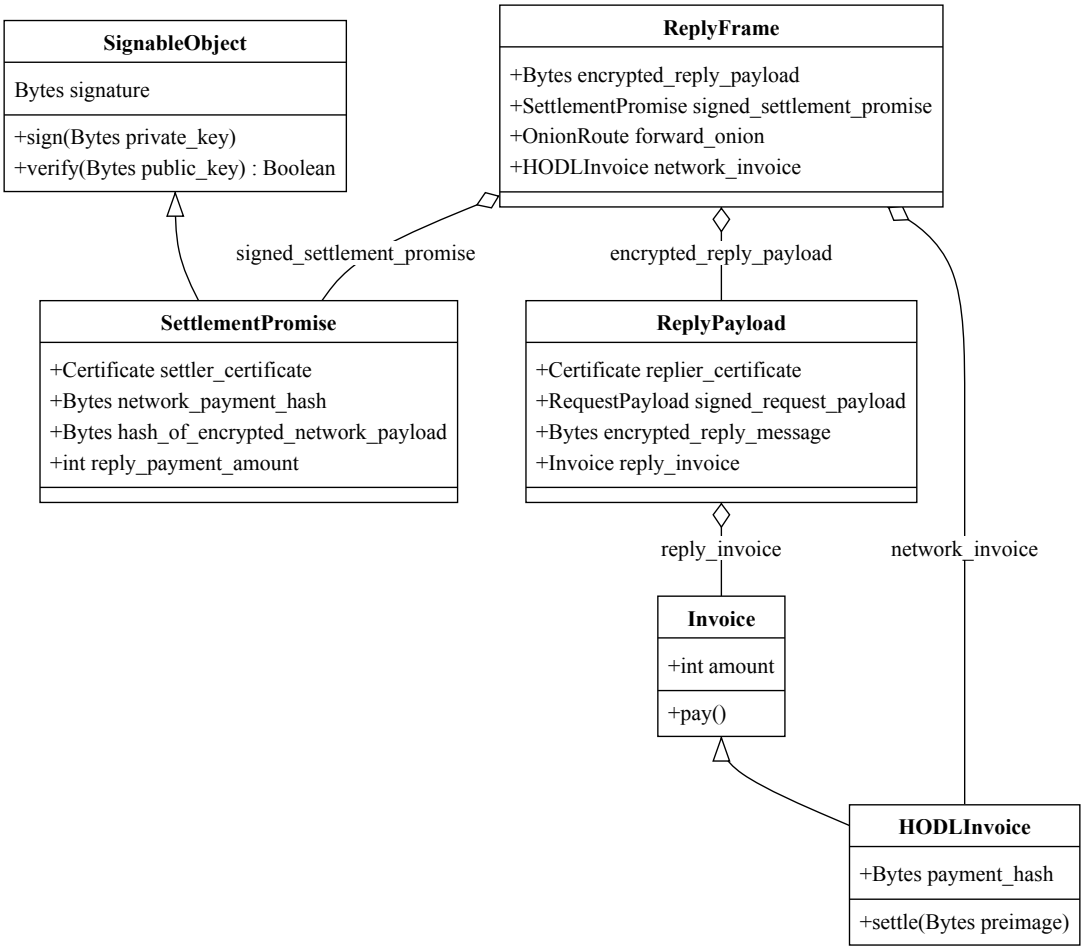


Figure 13: Reply Frame

If one of the settlements fails for some reason, it can break the preimage delivery process, therefore it is important to publish the preimage in an alternative way.

[This concludes the protocol.]

4 Discussion

4.1 Distributed Trust

Certification Authority hierarchy is used commonly in modern internet design however it is based on the idea of trusted 3rd parties. Having Certification Authorities that coexist in a free market helps with the decentralization of trust.

The other approach for distributed trust is based on the idea of a track record. Track record means that the specific customer (sender) or gig worker (replier) was already involved in a series of successful transactions. This can be implemented using proof of payments that can be traced back to the specific service being delivered.

The other one is based on the idea of reputation (transitive recommendations), so trustworthy participants are recommending other participants put their reputation on a table.

4.2 Mobile device connectivity issues

Holepunching [3]

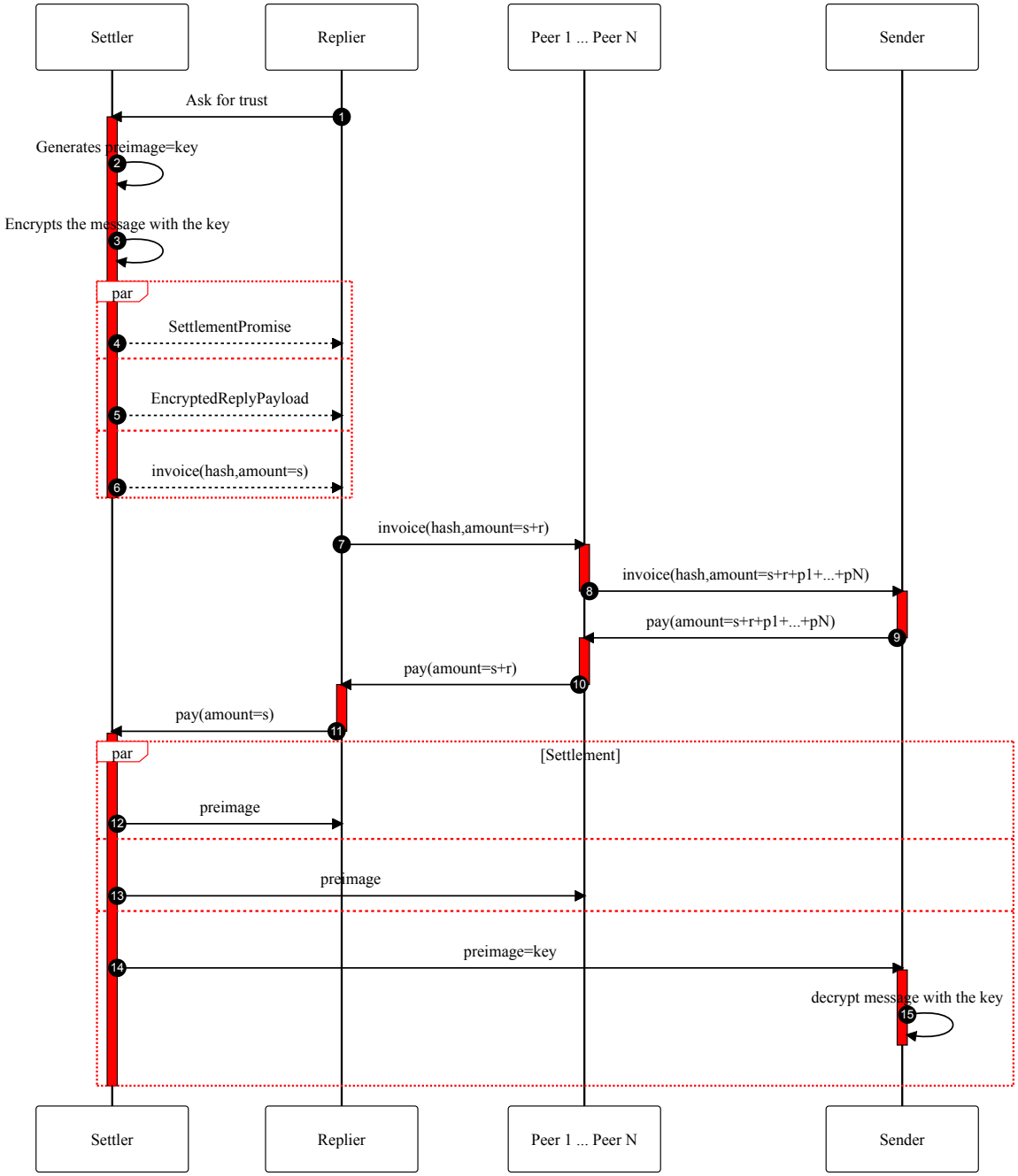


Figure 14: Reply/Payment/Decryption Sequence

Attack	Gig-gossip defense
Spam	POW
DDoS	POW
Silent	Multibroadcast > 1
Chatterbox	POW
Sybil	POW
Eclipse	Multibroadcast >1
Censorship	-
Convert Flash	-

Table 2: Attacks

Broken connection

4.3 Attacks

Here we are discussing common attacks that can be performed on the Gig-gossip network.

4.3.1 The Silent Attack

The adversary tries to distort the distribution of the gossipy nodes to cause propagation failure by failing to respond to the gossip protocol [2]. The same behaviour can be just a result of node failure that can happen naturally, especially with mobile devices.

Multibroadcast > 1

4.3.2 Chaterbox Attack

A malicious node retransmits repeatedly the same message [2].

4.3.3 Sybil Attack

This is the most common form of attack in P2P networks, since creating large numbers of identities is generally, cheap resource-wise, unless cryptographic puzzles are included as part of joining the system [8].

4.3.4 Eclipse Attack

This attack can be carried out against a single victim or the whole network. The objective is to silence the victim by refusing to propagate messages from it or to distort its view by delaying message propagation towards it [8]. Here the attack can be performed against specific Certification Authority

4.3.5 Censorship Attack

Sybils seek to establish themselves in the mesh and propagate all messages except those published by the target peer. In contrast to the Eclipse Attack, in the Censorship Attack Sybils appear to behave properly from all vantage points, but hairpin-drop the victim's messages. The objective of the attacker is to censor the target and prevent its messages from reaching the rest of the network [8].

4.3.6 Covert Flash Attack

In the Covert Flash Attack, Sybils connect to the network but behave properly for some time in order to build up the score. Then, they execute a coordinated attack whereby they stop propagating messages altogether in an attempt to completely disrupt the network. The attack is difficult to identify before the attackers turn malicious as they behave properly up to that point and build a good profile [8].

5 Supporting Services

5.1 Settler towers

5.2 KYC services as Certification Authorities

5.3 Push notification servers

5.4 Map servers

6 Applications

Gig-gossip Protocol is an enabler for building P2P apps.

7 Public Key

This is the Public Key (GPG) of Sonof Satoshi.


```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mI0EY8pcpgEEAN+bUaEdg+ylWkdNc6U9LNkWb4ii0Neay4kUyU2NntHMlFAZPNSC
wxJ8PlbrQnOGeeGNyfZtjZKTSn0Jor5YT4pHNlubGFj3/BrihJCBRSJ878qO2ct9
4RJXiNADVg1w3jRKRrk1CimmmL7VVK7oFZHd0311+8r/qIT4WNOITydNABEBAAG0
MVNvbm9mIFNhdG9zaGkgPHNvbm9mLnNhdG9zaGlAZG9udHRydXN0dmVyaWZ5Lm9y
Zz6I0QQTAQgAOxYhBIaALVutWo8Bqg5fDYtkf+QR5XMgBQJjylymAhsDBQsJCAcC
AiICBhUKCQgLAgQWAgMBAh4HAheAAAoJEItkf+QR5XMg2PYEAMcEB370PgxAaV+e
Kt458OPymI/rZOWO6Cm9E6BqMdNqNx7d4udxbQYutkUr1xhLmLH1JTxJwFhe3oMv
/3MUjm/VjIYrdnXAhHqvZA3502AyiWEQ66OQ9whj57PY04YYcBZP/NDe4QuoUX9r
b3XzYIeJcqHUNg0zjjQJQ7bU7gcwuI0EY8pcpgEEAK/nkFTpiOiGtUI1RqWD46HA
nH7wTVXy2BVwHefRiDHz2hGgQiHXF6EU8mk9F2SVBjOTBNHGAwvXssT97Y8jiq6i
vJosx7VtolxBEDRL1PFMOH4whwu1rjDg8QR3KPkB3kMcXvD9ZHIB6FspVvhx2/Jk
V+PKLQ0ThhQITxFIKz4nABEBAAGItgQYAQgAIBYhBIaALVutWo8Bqg5fDYtkf+QR
5XMgBQJjylymAhsMAAoJEItkf+QR5XMgRngD/0GbcDFoL8hqppvuBuXBLHJVMLGh
fF/3fZyd1ZkjE+Il/LX5G/WSsLcAm/dmAVd8L1zat3PvdL57RHY06BEE4kdDEo8m
DlZ8SycI1yGaSS8DdGCMaAFLzOxrJgER3NnXxg7BxCfREcUTawq1CEO1QYx/71ib
GoTF/wPiCY/JQ1Ed
=Ei5q
-----END PGP PUBLIC KEY BLOCK-----
```


References

[1] A. Back. Hashcash - a denial of service counter-measure. 09 2002.

[2] M. Burmester, T. V. Le, and A. Yasinsac. Adaptive gossip protocols: Managing security and redundancy in dense ad hoc networks. *Ad Hoc Networks*, 5(3):313–323, 2007.

[3] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. *CoRR*, abs/cs/0603074, 2006.

[4] A. B. Johnston and D. C. Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, St. Louis, MO, USA, 2012.

[5] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.

[6] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[7] S. Verma and W. Ooi. Controlling gossip protocol infection pattern using adaptive fanout, 2005.

[8] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras. Gossipsub: Attack-resilient message propagation in the filecoin and ETH2.0 networks. *CoRR*, abs/2007.02754, 2020.