Decall: Decentralized Phone System Concept

Written By Ben Stokman (ben.stokman@protonmail.com)

Outline

1: Abstract
2: Basic Design Needs
    2.1: No Standards
    2.2: No authority whatsoever
3: Defining terms
4: JSON Standard
    4.1: JSONless
        4.1.1: Security risks
        4.1.1.1: Manual key entry
    4.1.2: Potential Benefits
5: Layer Protocol
    5.1: Top Level Connection
    5.2: Handshake
    5.3: Data transfer
6: Encryption Standards
7: Sources

1.0

Abstract

The phone system that we use today is old, slow, expensive, vulnerable , and way too useless for today's needs. Even with the internet backbone that the phone system is currently run on, it still costs more than domestic rates to call internationally.[1] With some exceptions, notably how in the United States, calls to Canada and Mexico cost the same as long distance calls.

The phone system is also very insecure and vulnerable.; open to scams[2] and for corporate espionage[3]. People can spy on you, and your private communications. Which could end up with an affair turning violent, or a government becoming oppressive with no way to easily stop it.

Centralized services, even ones like telegram, where privacy is its foundation, are not adequate as they can be easy targets to compromise, block, or destroy. And, since its owned by someone, it could go offline or a change in ownership or management could compromise its security without public knowledge. What is needed is an open source service that does not rely on any system - centralized or decentralized; except obviously the internet

2.0

Basic Design Needs

This section is about the design necessities of Decall. It is not the final design specifics of the system and should never be consulted for anything technical or for any standards.

2.1

No Permanent Standards (sorta)

The entire design of Decall is based around the idea that changing things should be as easy as possible. The whole reason why I can't just append some security features onto the current phone system is because it was designed with absolutely zero thought of improvement.

Decall must have no permanent standards, besides very basic ones

2.2
No Authority Whatsoever

Decall is designed around having no authority to rely on, kind of like other internet standards like torrents. The clients do not query a centralized or decentralized system on how to make a connection, but instead will query an independent service.

Decall relies on the DNS system. A phone number can be an email or a domain, or something else, but those are those would be the most useful. The calling client will query the DNS server TK

2.3
Layer Protocol

Because of its expandable nature, the protocol will have to be devided up to allow a mix and match of how things are connected, and the actual data transfered.

Decall will be split up into three layers. In order they are: Top level connection, Handshake , and Data Transfer

1. Top Level Connection

The top level connection (TLC) is what protocol the other two layers are tunneled through. This can include HTTP, HTTPS, and ONION. There is also an option of no top level connection for direct connections. This layer also includes a sub-layer where the clients verify that the TLC is allowed to be a TLC for the number.

2. Handshake

The handshake layer is where the clients verify that who they are connected to is the correct number, and “negotiate” an encryption key.

The handshake process is simple: each client picks a random 256 bit integer, and sign it with their private key. They then exchange these signed numbers, and then they each sign the others’ signed integer, and then send it back to each-other for verification.

Once they have verified each-others validity They then generate another random 256 bit integer and then encrypt it with the other’s public key. They then send the encrypted integers to each-other, and then they checksum the two together, starting with the lowest one, to create a 256 ECDSA private key which all of their communications for the current session are encrypted with.

3. Chatting

This layer is the actual data being transferred between clients. This can be audio calling, video calling, text messaging, file transfer, or basically anything. For security reasons, there must be a new session for every new type of chatting protocol. This includes every new call and video call. It is recommended to refresh

3.0
Defining Terms

This section contains the definition of technical terms used throughout this paper.

4.0
JSON Standard

When I was first designing this, I was wondering how on earth I was going to make the numbers displayed. IP addresses was stupid, especially considering someone's IP address changes many times daily when they move to work or onto a cuellar network, and that does not mention the fact that multiple people use the same IP address, and people would have to communicate a port number on top of the crazy complicated IP address. And on top of that, the new IPs would have to be communicated through some other service; but the goal of Decall is to be the universal, all-encompassing service, so that wouldn't work.

Then I thought about email. Everyone has one, except for the type of people who don't use the internet, it's an authority that is difficult to attack, is (usually) free, and incredibly easy to memorize. After a bit of thinking, I decided that the JSON object would be hosted on the incoming email server(s) of the domain.

The steps are as follows:

1. The caller would enter an email address they want to call (ex: ben.stokman@protonmail.com).
2. The caller's device would get the MX record of the domain (ex: mail.protonmail.ch 10).
3. The caller's device would request the JSON object at the domain name with the email and then ".json" after it (ex: https://mail.protonmail.ch/ben.stokman@protonmail.com.json).

I have created an example JSON object. It is as follows (also available at example-benjistokman.json in this repository):

```
{
  "number":"ben.stokman@protonmail.com",
  "callerID": {
    "name":"Ben Stokman",
    "email":"ben.stokman@protonmail.com",
    "webpage":"https://benstokman.me",
    "pgpFingerprint": {
      "type":"SHA256",
"value":"b48e5fcb760d7c819c66f3db752f17ec3dda360cf2fbb5d29f0cf06e66
8cd13c"
    },
    "imageURL":"https://benstokman.me/profile-picture.jpeg",
    "language":"en-us"
  },
  "rules": {
    "callLimit":"none",
    "callTimezone":"none",
    "callTimeout":30,
    "verification":"enforce",
    "encryption":"enforce",
    "robots":"deny"
  },

"verificationKey":"18a8e0a603625e7687ee18801b45c70f15943d4637b10c77
db029a264f51ba91",
  "methods":[
    {
      "priority":0,
      "Server":"mail.protonmail.ch",
      "userID":"ben.stokman",

"publicKey":"855910b5fbca6d856d3d87b1b0dde1d9e010a5022e467362063feb
152710c9",
      "ssl":"enforce",

"sslFootprintSHA256":"16:FA:5B:FB:62:D4:DC:82:EC:19:AF:9F:07:FF:10:
2E:12:8D:67:0B:AD:EF:A7:CD:5B:30:85:AD:55:6E:F8:1D"
    },
  ]
}
```

This whitepaper will not include a description of every individual field, as they may change in the future for what is more useful or what needs to be added. However, this whitepaper will include an explanation of the fields that are important to the inner workings of Decall.

“number”: The reason why JSON was chosen is because its human readable. This field is just for bookkeeping purposes.

“rules”: This section contains some changeable rules on how the number works, such as how long the caller should be waiting before retrying because something went wrong, or encryption settings.

“verificationKey: This key is used at layer two of the connection protocol as a way to make sure that the caller is connected to an email server that is allowed to host the number. This is mainly to securely establish an encryption key between the email server of the caller and the email server of the receiver. It also acts as a failsafe in case the DNS system is compromised or misconfigured by accident.

“publicKey”: This key is the key that each of the end clients will use to verify each other and negotiate and encryption key.

5.0

Layer Protocol

Decall will be operated under three layers: top level connection, handshake, and chatting. In this section, the three layers will be gone into a lot of depth. Since this is more or less the entire system, this will be an incredibly long and detailed section.

The system is based off of three layers; in order, they are:: top level connection, handshake connection, and the data transfer layer.

5.1

Top Level Connection

As discussed in section 4, the client will use the JSON object to find how to connect to a client. This connection may be a direct connection to the client, but for privacy and safety reasons, Decall has a layer that is built for the purpose of relaying traffic through more secure means, like someone’s email provider.

This layer is specified in the `methods` array in the JSON object. Every entry in that array is a new top level connection, in the example, there is only one:, but a client can specify any number of places to connect.

The default is to create a UDP connection to any of the servers listed at port 20000 (an old voip port) tunneled through SSL. The caller can also use a proxy, and the server can refuse non-encrypted connections.

No matter what the clients do, or how they connect - the only purpose of this layer is to connect the caller and receiver. If the caller wants, and the receiver allows for it, they can connect directly to each other.

There are also peer to peer connections. Peer to peer connections are connections directly between two devices that works behind a firewall that does not allow any incoming connections. Each client just needs to know what port the other is listening on and what ip each of them have. For more information, see the original creator's page on the topic at reference 4.

5.2
Handshake Connection

This is where the fun stuff happens.

This layer is where the caller and the receiver verify each other, and negotiate an encryption key. The steps are rigid, and require no mistakes, or else the connection could become compromised or outright useless.

The caller sends the receiver their identity (their email) and public key. The receiver then goes through the steps outlined in section 4 to verify the authenticity of the caller. If the key does not line up with the identity, or there is no identity (anonymous calling); the receiver can either ignore the request to connect, or warn the user that the caller fails the verification requirements.

Then the clients "negotiate" an encryption key. I put "negotiate" in quotes because it's more of agreeing on an encryption key. Each client chooses a random 256 bit string, encrypts it with the others' public key, and sends it to each other. They then create a combined 256 bit string by checksumming the **combined hexadecimal utf-8 text** where the caller's string is first. The reason why it is the hexadecimal text is for debugging purposes and so humans can better understand what is happening, for transparency.

This is then the private key for a 256 bit ECDSA pair. Each packet in the data transfer layer is encrypted with the corresponding public key before being sent.

5.3
Data Transfer

This is a really boring layer. No offense to it, not that it can take any, its inanimate.

This layer does not have any specifics that need to be made at the moment. The only thing that can really be said is that a new handshake connection has to be made for every single type of data transfer. Examples for this layer are: calling, texting, video calling, sending files, syncing calendars. More types of data transfers can be made in the future if the community wants it.

6.0

Sources