OnionPhone V0.2a:
VOIP add-on for TorChat

Van Gegel[1]

The idea. OnionPhone was inspired by the previous TorFone [1] presented in 2012 - the first out-of-box solution for VOIP over Tor network suitable for practical use. Later homegrown experiment grew into a small project which is due to its relevance now is cited in publicists [2] and academic [3–6] works.

Despite the uniqueness of the TorFone has been criticized for mono-platform design (Windows only), the use of C ++ (hard-to-trace language-specific bugs are possible), the interweaving of the functional code and GUI and for deprecated cryptographic solutions from Philip Zimmermann's PGPFone [7] (1995) on which the TorFone was fully based.

Therefore given the drawbacks OnionPhone was developed [8] and inherited the basic functionality of TorFone. Thanks to Roger Dingledine for a name of the project which is now is more appropriate to its nature[2].

Since OnionPhone uses advanced cryptographic primitives and protocols it is not compatible with TorFone or any other VOIP solutions. Choosing between compatibility and efficiency we prefer the latter and provide strong optimization under Onion conditions. New project was designed with high latency of Onion transport in mind and can be effectively used to explore the possibility of VOIP over Tor further [3].

To Do: the android version of OnionPhone and simple cross-platform Qt GUI connected over control port.

[1] torfone@ukr.net

[2] TorFone and OnionPhone are produced independently from the Tor anonymity software and carries no guarantee from The Tor Project.

[3] Alpha version is presented "as is" and has not yet been checked by anyone except the developer. Do not use in "life or death" cases.

1 General description

OnionPhone is a VOIP tool for calling over Tor network which can be used as a VOIP plugin for TorChat [9] . Call is targeted by the Onion address of the recipient (its hidden service HS). The recipient can initialize a reverse Onion connection to the originator's HS and use a faster channel periodically resetting the slower channel to reduce overall latency.

The ability to switch to a direct UDP connection (with NAT traversal) after the connection is established over Tor (Tor instead of SIP without a specialized server, any registration or metadata collection) is also provided. In addition, OnionPhone can establish a direct UDP or TCP connection to the specified port on IP-address or host.

OnionPhone uses a custom protocol (not RTP) with only one byte unencrypted header that can provide some obfuscation against DPI.

OnionPhone provides independent level of p2p encryption and authentication that uses modern cryptographic primitives: Diffie-Hellmann key exchange with Elliptic Curve 25519 and Keccak Sponge Duplexing encryption. In the case of a call to the Onion address Tor protects against MitM attacks. Also the recipient can verify identity of the originator's Onion address (only with the permission of the sender) similarly to the TorChat authentication. Other possible ways of multifactor authentication are:

- voice (biometrics);
- using previously shared password (with the possibility of hidden notification of coercion);
- using a long-term public keys signed by PGP.

OnionPhone provides Perfect Forward Secrecy (uses a fresh key for each call) and Full Deniability for using long-term public keys of participants (as a fact and a content of the conversation): like a deniable SKEME [10] protocol uses for initial key exchange.

OnionPhone can use a wide range of built-in voice codecs (C source included) from ultra low-bitrates up to high quality. The full list is: MELPE-1200, MELP-2400, CODEC2-1300, CODEC2-3200, LPC10-2400, CELP4800, AMR-4750/12200+DTX, LPC-5600+VOCODER, G723-6400, G729-8000, GSM-HR-5600, GSM-FR-13200, GSM-EFR-12400, ILBC-13333, BV16-16000, OPUS-6000VBR, SILK-10000VBR, SPEEX-15200VBR+R. Some of them are free, some require a license (check regional laws). Additional features: Implemented noise suppressors (NPP7 and SPEEX) of environment sounds and automatic mic gain control. Built-in LPC vocoder [11] featuring irreversible voice chage (robot, whisper etc.). Specially designed dynamic adaptive buffer useful for smart jitter compensation in high latency Onion environment. Radio mode (Push to Talk) and voice control mode (Voice Active Detector with generation a short signal when transmission is completed) are available.

The OnionPhone source (available on github: [12]):

- uses command-line style, does not require installation and can be run from removable disk or TrueCrypt container;
- is fully open source, developed on pure C at the possible lowest level and carefully commented;
- is statically linked, does not require additional third-party libraries and uses a minimum of system functions;
- can be compiled under Linux OS (Debian, Ubuntu etc.) using GCC or under Win32 OS (from Windows 98 up to 8) using MinGW.

2 Quick start and easy usage

2.1 Installation

The easiest way is to use OnionOhine as a VOIP plugin for TorChat:

- Step 1: Put the OnionPhone folder on the hard disk, removable media or TrueCrypt container (preferred).
- Step 2: Edit the TorChat configuration file /torchat/bin/Tor/torrc.txt: immediately following the line:

  ```
  HiddenServicePort 11009 127.0.0.1:11009
  ```

  add the a new line:

  ```
  HiddenServicePort 17447 127.0.0.1:17447
  ```

  then run the TorChat.
- Step 3: Right click on myself icon of TorChat contact list and copy ID to clipboard. Edit the OnionPhone configuration file conf.txt: specify our Onion address using copied ID, for example:

  ```
  Our_onion=gegelcy5fw7dsnsn
  ```

  Once started the OnionPhone is ready to receive incoming and make outgoing calls.

2.2 Usage

- To accept an incoming call press Enter.
- To make an outgoing call as a guest to guest (without use of personal public keys) type command: `-Oremote_onion_address` and press Enter then wait 10-30 sec for connection over Tor to be established.
- To toggle continuous voice transmission press Enter. Hold down / release the Tab for Push-to-Talk mode.
- To apply the voice codec from 1 to 18 use the command `-Ccodec_number`. Smaller numbers correspond to low bitrate codecs, greater ones are for high quality. Numbers from 16 to 18 correspond to the variable bit rate codecs.
- To enable security vocoder use the command `-Qmode`, where mode=3 corresponds to "whisper" voice (recommended), modes 6-255 correspond to "robot" etc. To deactivate the vocoder use the command `-Q-3`.
- To use the chat feature type a message and send it by pressing Enter.
- To switch to direct UDP connection use the command `-S` (both parties must do this).
- To return into Tor from direct UDP connection use the command `-O`.
- To end the call use the command `-H`.
- To exit the OnionPhone use the command `-X` or hit Esc twice for emergency exit.

3 Keyboard interface

The keyboard is used to control the voice transmission, typing the commands and navigating in menu. Control keys are:

- **Back** removes the last typed character.
- **Del** clears the typed string.
- **Tab** activates voice transmission while key is held down (Push-to-Talk mode).
- **Sift+Tab** (Linux) or **Ctrl+Tab** (Windows) activates the voice detector.
- **Up**, **Down** arrows are used to navigate between menu units.
- **Left**, **Right** arrows are used to select menu items.

- **Enter**:
  - answers while incoming call is waits for being accepted;
  - enables / disables continuous voice transmission if command line is empty;
  - if the first char of command string is "**-**" (command was typed) processes the command;
  - otherwise sends typed chat message.
- **Esc**:
  - rejects while incoming call is waits for being accepted;
  - if hit twice immideately exits the program clearing the memory.

4 Menu

Console menu allows to quickly enter commands which are most often performed by the user. The menu consists of 9 sections. Navigation between sections is performed using the Up and Down keys. Selecting an item in the section is done using Left and Right keys. The selected item is applied using the Enter key. Menu sections are:

1. **Controls** - termination of the call, exiting the application, obtaining information about the current state, measuring the latency of connection, voice processing controls (loopback).
2. **Low bitrate codecs** - the selection of codec, obtaining information information about the current codec.
3. **High-quality codecs** - the selection of codec, obtaining information about the codec used by the other party.
4. **Voice processing** - enabling / disabling the automatic gain control, noise suppression and vocoder.
5. **Jitter compensation** - setting fixed buffer size or auto-adjustment depending on the parameters of the communication channel.
6. **Voice detector** - activation the transmission mode using mic level or voice analyzer.
7. **Signal levels** - activation of the three-tone alarm received from the opposite side after voice message and noise substitution of lost fragments.
8. **Switching** - establish direct UDP connection using selected STUN-server for NAT traversal.
9. **Contacts** - view the address book and select the contact for an outgoing call.
10. **Others** - setup the backup counter Onion connection, reset application etc.

5 Commands

The command should start with a "-" symbol followed by a capital letter followed by an optional parameter. No spaces can be inserted between letter and parameter. Some commands can contain several subcommands separated by spaces. After typing the commands press Enter for processing. While typing use Back to correct the typed characters or Del to cancel and clear the command string.

5.1 Address book management

`-V[filter]` displays all the entries of the address book containing the substring filter. Displays all uncommented entries if the parameter is omitted.
`-E[name]` outputs the command to call to the subscriber name defined in the address book. Outputs the most recently used call command (redials) if the parameter is omitted.

5.2 Application and Connection management

`-H` gracefully terminates the call.
`-X` gracefully shuts down the program.

5.3 Sound management

`-C[codec]` sets the codec for outgoing voice stream. Parameter codec is codec number from 1 to 18. If the parameter is "?", the commands shows the number of the current codec. If omitted, the default codec is applied.
`-J[jitter]` sets a fixed size of buffer to compensate transport jitter. Parameter jitter is average buffering delay in milliseconds. If you use the parameter="?" the current value is displayed. If it's "-1" the minimal buffering is applied (lower latency but bad resistance to jitter). If the parameter is omitted the automatic buffering adaptation to the communication channel will be used (default).
`-Q[voice]` sets the style of voice preprocessing. Parameter voice is: "0"-no automatic microphone gain control (AGC) and no noise suppressing (NPP), "1"-applies AGC, "2"-applies NPP, "3" - applies a vocoder in "whisper" mode, "4" - in "high" mode, "5"- in "deep" mode, "6"-"255" - in "robot" mode with different pitch, "-1"-disables AGC, "-2"-disables NPP, "-3" - disables the vocoder (unchanged voice will be sent), "?" - show current state of voice processing.

5.4 Key sharing

`-K[name]` sends specified public key name from the our phone book to remote side. This command is only possible if the connection is active. If parameter is omitted own public key specified in the configuration file is sent. The keys are received and automatically added to the address book with

option `-L` without parameter (untrusted). The user can view the address book and check out all the untrusted keys later (check the PGP signature into key file) and edit the trust level to that key.

5.5 Authentication

`-Yaccess` specifies password for private key correspondent for our_name used for processing all incoming and default outgoing calls. If private key is in encrypted format then this option must be applied immediately after OnionPhone is started otherwise you can not accept incoming calls. If the parameter is space applied key access will be cleared. If the parameter is omitted applied key access will be checked for validity for specified secret key corresponding to our default name.

The alternative way to introduce access is to specify the password as Our_secret_access parameter in the configuration file. In this case it is recommended to store OnionPhone folder in a protected place (TrueCrypt container etc.). The access also can be temporarily applied to the current outgoing call using `-Yaccess` into the call extended command `-N` (see below).

`-P[password]` applies a common password for authentication and initiates it. Passwords can contain from 3 to 31 characters and must be the same for both parties except for the last two characters: one of the participants swaps their places. The password can be used both during the established connection and when dialing (see the extended command for calls), as well as defined for each user in the address book (in this case it will be applied automatically when an incoming call from that person occurred). If parameter omitted applied password will be cleared.

In the case of manually entering the password (preferably for safety) it is possible to alert the other side to being under duress: user can change the last character of the password to any other. In this case the authentication on their side will look as usual but the participant on the opposite side will receive a warning. An attacker can not verify the correctness of the last letter of the password in any way, even when analyzing dumps of previous sessions, controlling current session, as well as with the active "fishing" by creating test sessions to both parties, both before and after.

`-W[interval]` sets the interval of how frequently to reconnect the slower of the two Onion connections (to reduce the overall latency of the connection). If parameter is omitted the default value is used. If the parameter is 0, the counter connection is not used for data exchanging and is closed immediately after verifying the Onion address. If the parameter is "-1" the counter connection is not established (also Onion address verification is not possible). If the parameter is too large (e.g., "1000000000"), the counter connection is used but never reconnected.

`-WI` command initiates the verification of originator's Onion address by re-creating reverse connection to the declared hidden service and exchanging the verifiers (like TorChat authentication procedure). The command can be used only if the Tor connection to Onion address is used and verification is allowed by originator. If the originator specified Onion address in configuration file and under the above conditions verification is performed automatically after the key agreement. Manually executing this command is useful if the process was frozen.

`-WR` command gracefully restores the doubling process by sending the other side an invitation with

a request to restart their outgoing connection.

5.6 Switch to direct connection

`-S[stun]` Initiates switching from Onion connection to direct UDP connection using specified STUN-server for the NAT traversal. If the parameter is omitted the default STUN server specified in configuration file will be used. If the parameter is "0", NAT traversal attempts are not performed (in this case the direct connection is only possible if both parties are on the same subnet or if at least one of the parties has the "real" IP-address on the interface of computer running OnionPhone.

The command can be executed only if the Onion connection is already established. Both parties must execute this command to start the process. If both parties are behind full-cone NATs, or both are in a common subnet with a "gray" IP address (for example, home internet provider), and then each of them behind a router, the NAT traversal can fail. In this case users will be able to continue communication over Tor. After a successful switch to direct UDP-connection the Onion connection is still available but not actually used for communication . At any time any user can stop the direct connection and return to the use of the Onion connection. To do this at least one of the participants must execute command `-O` with no parameter.

5.7 Extended call command

`-N[name] destination_subcommad [options_subcommands]` initiates a connection with the subscriber `name` (the subscribers public key must be added to the address book). If the name is omitted the default `guest` will be used. Public and private keys for the guest included in the OnionPhone by default and are available for each user (and also for the attacker). The authentication using `guest` key is not performed and the danger of a MitM-attack is present in the case of initiation of direct connection. In the case of connection over Tor the key agreement is going on inside Tor so MitM is likely impossible (while you trust Tor).

Parameter `destination_command` specifies the type of transport (UDP, TCP, Onion) and the address of the recipient (IP or hostname and optionally port number separated by ":"), for example:

```
-U178.95.218.29:17447
```

```
-Talice.dyndns.org
```

```
-Or4kxspnzpnsel4fu
```

Connection type `-U` initiates direct UDP-connection, `-T` for direct TCP-connection, `-O` for TCP-connection over Tor. Address can be specified as a string IP, domain name or Onion address with or without the suffix *.onion*. The port can be specified after the address separated by a colon. If no port is specified the default port 17447 will be used.

Also, these commands can be used independently to install unauthenticated connections `guest` to `guest`.

Subcommands `-P[password]` (see above) and `-I[our_name]` can follow, separated by spaces: our_name parameter specifies how the name of the originator should appear to the recipient. If this option is omitted Our_name from the configuration file is sent instead. If `-I` option is used without a parameter the default name guest will be used (see above). If the parameter is specified it will be used as a name for the current session. In this case the initiator should have files of both public and private keys for this name. In addition this public key must be added to the address book. Of course the recipient also must have a public key for this name and this key must be added in their address book otherwise the connection will not be established.

Option `-Yaccess` can follow specifying the password for the secret key our_name. Note that this password temporary (only for this call) overwrites access to default private key used for incoming calls. If both parties connect for the first time, the only key they have in common is probably the guest key. Thus the first connection must be initiated as untrusted guest to guest:

```
-N -I -Oonion_address
```

This connection is still encrypted with the session key but its resistance to MitM depends fully on Tor. Also the parties can't be sure of the authenticity of each other (except for the possible verification Onion addresses similar to TorChat). Preinstalled untrusted connection can share PGP-signed long-term public keys using `-K` command (this is denied because keys are public and attacker can't be sure that customers really declare own key: user can also send multiple keys of other various users) . Received keys will be automatically added to the address books and marked as untrusted. After manually checking the PGP signatures users can set the desired level of trust by editing contacts entries in the address book file. Later user can re-initialize the connection to be fully trusted, for example:

```
-Nalice -Or4kxspnzpnsel4fu -Ibob
```

5.8 Others commands

`-RI` command runs voice self-test (audio loopback) and useful for checking the sound quality in your system. While sound test is running there are no sound transmission to remote side. `-R` command cancels sound test and enables sound transmission. `-RI` command shows the OnionPhone current status, `-RL` command provides measurement of the latency (connection must be active), `-RS` command discovers local and external UDP listening interfaces (if UDP listener was enabled in configuration settings).

6 Key management

OnionPhone uses public keys to authenticate the subscribers to each other using PGP. Also keys in this format will be used in other projects in the future (audio chat group, encryption radio and over-GSM-audio channels, etc.). Console utility `./addkey` is useful for key management:

Step 1: To generate a new key pair run:

```
./addkey -Gname [-Yaccess] [options]
```

where name is your identifier, access is password that will be used to encrypt your private key and options will be passed to other participants. If `-Y` is omitted the private key will be stored unencrypted. In this case it is recommended to store OnionPhone folder in a protected place (TrueCrypt container etc.).

The most common use is option `-Oour_onion_address` to present own address. Upon receipt of your key this option will be automatically copied in address books of other participants. For example, after the command

```
./addkey -Galice -Y1234 -Or4kxspnzpnsel4fu
```

files alice.sec (the private key encrypted using password 1234) and alice (the public key) will be created in the folder keys.

Step 2: To sign the public key open it as a text and sign their content using PGP. Signature must be added to the key file. Once the key has been signed further edition is not allowed (renaming is still possible).

Step 3: Before using the key you must add it to your address book:

```
./addkey -Aalice
```

To use this key as its own default edit the OnionPhone configuration file conf.txt specifying the key's name: `Our_name=alice`

Step 4: Now you can send the key to other users. For transmission of the key establish an un-authenticated call guest to guest with the other party (see above) and execute `-Kalice` (or `-K` without parameter if this key is assigned as your own default). Key will be automatically added to the remote address book with the lowest possible level of trust. After manual PGP signature check of the the key file other parties can set the desired trust level by editing `-L` parameter to the corresponding entries in the address book file keys / contacts.txt, for example:

```
[alice] {FNrjEjGmlZtvKXzBQkNIDA ==} #alice -Or4kxspnzpnsel4fu -L1
```

You can also pass the key in any other way (for example, by email). The recipient verifies PGP signature in the key file and determines the necessary trust level, then add the key to the address book indicating the trust level as `-Llevel`, for example:

```
./addkey -Aalice -L1
```

7 Cryptography design

Normally OnionPhone establishes a connection over Tor network. Despite the use of Tor as a secure transport OnionPhone provides own layer of a p2p encryption and multifactor authentication. This makes it possible to use the direct connection outside Tor.

7.1 Cryptograhic Primitives

- Asymmetric cryptography is Diffie - Hellmann on EC25519 [13] used curve25519_donna C-code [14] ;
- Symmetric cryptography is stream encryption used Keccak Sponge Dulpexing library [15] ;
- SPRNG is Keccak Sponge Duplexing RNG [16, 17] with Havege reseeding.
- Hashing, MAC, KDF are Keccak in 576/1600 mode [18] ;

We don't use special MAC and KDF functions i.e. Keccak is RO-PRF and can be directly used for this purpose.

7.2 Symmetric encryption

Stream cipher based on Keccak is used. Both parties maintain synchronized counters *ctr* separately for sent and received packets. TCP transport is lossless and provides synchronization of counters by himself. For UDP each packet's header contains last 7 bit of sender's outgoing counter value for synchronization incoming counter on receivers side. For each packet Keccak Sponge initiates and absorbs 128-bits session key s, 32-bits counter *ctr* and 8-bits flag O is 0 for an originator and 1 for an acceptor of the call, then squeezes gamma $g = H(s \parallel ctr \parallel O)$.

Gamma XOR-ed with plainest p (packet bytes except first byte as a header h) provides ciphertext: $c = g \oplus p$. After packet was encrypted 32-bits MAC is computed: $mac = H(s \parallel ctr \parallel O \parallel h \parallel c)$, After this counter is incremented by 1. There is no way to rewind the counter.

Encrypted packet is h, p, mac. Decryption is similar to encryption.

7.3 Key formats

OnionPhone uses ECDH25519 256 bits private keys and 256 bits Montgomery public keys providing approximately 128 bit security level.

Public key is a text file containing: **Info**, **Key**, **Sig**(Info, Key), where:

- **Info** string begins with "#" symbol, folowed by name and options. Name is a string up to 31 characters and options is symbol "-", followed by option type (capital letter), followed by string of parameters. Options are separated by spaces. All options are passed to address book for further processing by application.

Some other strings are optional comments etc.

- **Key** string is base64 representation of ECDH25519 public key (in figure brackets).
- **Sig** is a PGP signature as a part of key file: info-string and key string must be signed by PGP.

Key stamp (ID) consists of the first 128 bits of Keccak hash of public key file. After public key was generated and signed noone can modify it but it's possible to rename the file. Owner must add their own public key to address book before using it. All other parties also must add this key before using.

Note: public key is not self-signed. An attacker could re-sign someone else's key submit it as their own thereby trying to mount an UKS attack. This can be prevented including ID in the authentication process. Implementing initial key exchange protocol completely solved this problem.

Private key is a binary file containing 256 bits ECDH25519 secret. Private key can be saved unencrypted (32 bytes) or encrypted with password (64 bytes). Both formats are acceptable but encrypted format requires the use of command `-Yaccess` when working with the OnionPhone. Symmetric key derived from the password is used for private key encryption: a Keccak Sponge is initialised, which then absorbs the salt string "$OnionPhone_salt", then absorbs password string 32768 times in a row, and then squeezes a 16-bytes key: $s = H(salt \parallel password \parallel password \parallel \ldots \parallel password)$.

To perform encryption Keccak Sponge is initialized, then absorbs key s and randomly generates 16-bytes $nonce$, then squeezes a 32-bytes gamma: $g = H(s \parallel nonce)$. Gamma is XOR-ed with private key body x to produce encrypted data: $c = g \oplus x$. After packet was encrypted 16-bytes MAC is computed: $mac = H(c \parallel s \parallel nonce)$. Encrypted private key is $c, nonce, mac$.

Key pair "guest" is a private key as a 32 bytes binary representation of string "Guest secret http://torfone.org" and corresponding public key. These keys are included by default and can be used for performing primary unauthenticated connection with an unknown receiver.

7.4 Deniability authenticated Key agreement

The core of implemented protocol is SKEME. The shared secret is derived only using the Diffie - Hellman exchange. Public parameters are signed by another key outputted by another protocol performed in parallel. The original SKEME [19] uses the Abadi protocol [20] as auxiliary. For this purpose we use a slightly modified KEA+ protocol [21] . This modification is known as the TripleDH and is used, for example, as a initial key exchange (IKE) in the Axolotl protocol [22] .

OnionPhone's IKE provides key agreement (produces fresh session key for each run achieving Prefect Forward Secrecy) and fully deniable authentication using long-term ECDH public keys signed with PGP. If the parties can deny having exchanged a key with the other party then the rest of the communication can also be denied.

Proposed IKE is intuitively viewed as deniable in both Deng at all [23] (full deniability) and Raimondo and Gennaro (forward deniability) [24] scenario even with a cooperating judge: while the dishonest party cooperates with judge before protocol running (but not during IKE) it is still will not be possible to convince the judge after IKE will be completed that the other party had really participated in the protocol. The initiator's view of the protocol can be simulated by an simulator that does not know the secret key of the acceptor and vice versa. The value of the session key also can be part of the output of the simulation so all subsequent session messages can also be simulated. Furthermore provided is the ability to reveal the auxiliary MAC key after completing the IKE. This is a not a standard cryptographic way, thus this feature is optional.

The IKE looks resistant to UKS and wPFS attacks. Note that full deniability property conflicts with KCI resistance. Therefore after the leak of user's private key the attacker can impersonate anybody to him. Note the IKE is sensitive only to the leak of ECDH private key but not to the leak of PGP private key used to sign.

Protocol: let p,q be two large primes satisfying $q|(p-1)$, G_1 an additive group of order q on an elliptic curve 25519. Public key $T = g^t$, where $t \in Z_q$ is the private key and $g = 9$ is the base point of G_1. H is a one-way hash function with RO-PRF property (SHA3 Keccak Sponge).

When Alice wants to exchange session key s with Bob, they cooperatively perform the following steps. Here, Alices and Bobs identities, public and private keys are id_A, A, a and id_B, B, b respectively. Both participants randomly pick nonces N_A and N_B, two ephemeral secret keys x, v and y, w and compute public keys X, V and Y, W respectively.

Beginning participants anonymously exchanges their public keys V, W and commitments of X, N_A and Y, N_B.

At the next step participants computes theirs hidden IDs and exchanges them and public keys X, Y and nonce N_A, N_B. As a result, agreed the session and auxiliary secrets s, K.

Finally participants sign session parameters using an auxiliary secret K and exchanges authenticators M_A, M_B confirming ownership of their secret keys a and b.

Thus, parties A (originator) and B (responder) shall exchange the following messages:

A: $V = g^v, X = g^x$, where $v \in Z_q$, $x \in Z_q$ at random
$C_A = H(X \parallel N_A)$, where N_A at random
$A \to B:$ V, C_A

B: $W = g^w, Y = g^y$, where $w \in Z_q$, $y \in Z_q$ at random
$C_B = H(Y \parallel N_B)$, where N_B at random
$B \to A:$ W, C_B

A: $D_A = H(id_A \parallel B^v \parallel W^v)$
$A \to B:$ D_A, X, N_A

B: checks $C_A =?H(X \parallel N_A)$
searches contact in id-list matching $D_A =?H(id_? \parallel V^b \parallel V^w)$
$D_B = H(id_B \parallel A^w \parallel V^w)$
$B \to A:$ D_B, Y, N_B

A: checks $C_B =?H(Y \parallel N_B)$
checks $D_B =?H(id_B \parallel W^a \parallel W^v)$
$K = H(W^a \parallel B^v \parallel W^v)$
$M_A = H(K \parallel V \parallel W \parallel X \parallel Y \parallel N_A \parallel N_B)$
$s = H(X^y) \oplus N_A \oplus N_B$
$A \to B:$ M_A

B: $K = H(A^w \parallel V^b \parallel V^w)$
check $M_A =?H(K \parallel V \parallel W \parallel X \parallel Y \parallel N_A \parallel N_B)$
$M_B = H(K \parallel W \parallel V \parallel Y \parallel X \parallel N_B \parallel N_A)$
$s = H(Y^x) \oplus N_A \oplus N_B$
$B \to A:$ M_B

A: checks $M_B =?H(K \parallel W \parallel V \parallel Y \parallel X \parallel N_B \parallel N_A)$

$A, B:$ shares secret s
$A \to B:$ K (optionally reveals auxiliary key)

7.5 Voice authentication

Voice (biometric) authentication is similar to SAS authentication in ZRTP [25] and can prevent a MitM attack if the participants know each other by voice. Participants must read each other a list L of words based on the shared secret s. Since uses short authenticators (only 32 bits for the list of four words of the 256 available) commitment is needed to prevent the dishonest party influence of the shared secret to obtain the same SAS during the MitM attack:

1. $A \to B$: $C = H(N_1)$, where N_1 at random
2. $B \to A$: N_2 at random
3. $A \to B$: N_1, $L = H(N_1 \parallel N_2 \parallel s)$
4. $B \to A$: $L = H(N_2 \parallel N_1 \parallel s)$

7.6 Optional authentication using pre-shared password

This authentication can be performed to detection a MitM attack after the key is agreed on and the shared secret is derived. Password must be identical in both parties except last two characters: one of the parties swaps their places. Let Alice's password be $pass \parallel p \parallel q$ and Bob's password be $pass \parallel q \parallel p$. The participant under enforcement must change the last character of their password with any other. Authentication will look like correct from their side but the other party will be notified. Each party can initiate authentication independently. For example, Alice is the originator of the call and initiates authentication using pre-shared password:

1. $A \to B$: $U1_A = H(O \parallel s \parallel pass)$

   where s is shared secret, O is byte 0 for originator and 1 for acceptor of call;

2. B : Checks $U1_A$, rejects invalid;

   $B \to A$: $U1_B = H((O+2) \parallel s \parallel pass)$, $U2_B = H((O+2) \parallel s \parallel pass \parallel p)$

3. A : Checks $U1_B$, rejects invalid; checks $U2_B$, warns invalid;

   $A \to B$: $U2_A = H(O \parallel s \parallel pass \parallel q)$

4. B : Checks $U2_A$, warns invalid;

7.7 Optional Originator's onion address verification

After originator has connected to the acceptor's Hidden Service (HS) he is sure of acceptor's Onion address (while he trust Tor). But acceptor can't know initiator's Onion address: initiator must present it to acceptor. Acceptor can trust it only after its verification.

Let Alice be the originator and Bob accepts the connection to his HS, s is shared secret, O is byte 0 for originator and 1 for acceptor of the call:

1. A : Connects to Bob's HS using their Onion address.

   $B \leftrightarrow A$: Parties agree secret s as describe above using channel 1.

2. B : Requests for Alice's Onion address using channel 1:

   $B \rightarrow A$: ?

3. A : Sends own Onion address to Bob using channel 1 (encrypted):

   $A \rightarrow B$: *alice.onion*

4. B : Connects to HS specifies by received Onion address, sends using channel 2:

   $B \Rightarrow A$: $I_R = H((O + 4) \parallel s)$

5. A : Checks I_R, rejects invalid; sends using channel 2:

   $A \Rightarrow B$: $I_S = H((O + 4) \parallel s)$

6. B : Checks I_S, warns invalid.

   Now Bob can be sure that remote Onion address matches submitted by Alice.

To prevent unwanted verification originator can skip specifying their Onion address in the configuration file. Thus the recipient cannot know the Onion address of the originator and the originator remains completely anonymous.

References

[1] Van Gegel. (2012). TOR Fone - p2p secure and anonymous VoIP tool. (Web link)

[2] Leslie Meredith. 3 Privacy Tools to Stop Gov. Phone Snooping. TechNewsDaily [Jun 7, 2013] 1:(1)[4 screen]. (Web link)

[3] Maimun Rizal. A Study of VoIP Performance in Anonymous Network - The Onion Routing (Tor) [dissertation] Georg-August Univ.; Gottingen, 2014. (Full text)

[4] Georgios Karopoulos, Alexandros Fakis, Georgios Kambourakis. Complete SIP message obfuscation: PrivaSIP over Tor (University of the Aegean). The Ninth International Workshop on Frontiers in Availability, Reliability and Security (FARES); University of Fribourg, Switzerland; Sep. 8th - 12th, 2014. (Full text)

[5] Georgios Kambourakis. Anonymity and closely related terms in the Cyberspace: An analysis by example. Journal of information security and applications XXX (2014), 1:16. (Full text)

[6] Keen Sung, Brian Neil Levine, Marc Liberatore. Location Privacy without Carrier Cooperation (Univ. of Massachusetts Amherst). IEEE Symposium on Security and Privacy (SP) San Jose, CA Mobile Security Technologies (MoST); May 17, 2014. (Full text)

[7] Phil Zimmermann. (1999) PGPfone - Pretty Good Privacy Phone. (Web link)

[8] Van Gegel (2014). OnionPhone - VOIP add-on for TorChat. (Web link)

[9] Bernd Kreuss (2012) TorChat: Decentralized anonymous instant messenger on top of Tor Hidden Services. (Web link)

[10] Mario Di Raimondo, Rosario Gennaro, Hugo Krawczyk. Deniable Authentication and Key Exchange. Proceedings of the 13th ACM conference on Computer and communications security; 2006; 400:409. (Full text)

[11] William Andrew Burnson, Wenjia Zhou. Real-Time Voice Conversion: A Multirate 8kHz LPC Vocoder. [Thesis Research] Univ. of Illinois, 2011. (Full text)

[12] Van Gegel (2013) OnionPhone - VOIP add-on for TorChat. (Web link)

[13] Daniel J. Bernstein Curve25519: new Diffie-Hellman speed records. In Public Key Cryptography (PKC), Springer-Verlag LNCS 3958, 2006. (Full text)

[14] Adam Langley (2011) Implementations of a fast Elliptic-curve Diffie-Hellman primitive. (Full text)

[15] Van Gegel (2013) Portable C implementation of universal Sponge construction based on compact Keccak source code. (Web link)

[16] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche. Sponge-based pseudo-random number generators. Cryptographic Hardware and Embedded Systems, CHES 2010. 12th International Workshop, Santa Barbara, USA, August 17-20, 2010; 33:47. (Full text)

[17] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. Selected Areas in Cryptography. 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers, 2011; 320:337. (Full text)

[18] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche. The Keccak reference. Version 3.0.; January 14, 2011.

(Full text)

[19] Hugo Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. Network and Distributed System Security; San Diego, CA, 1996; 114:127. (Full text)

[20] Martin Abadi, Cedric Fournet. Private Authentication In Software Security Theories and Systems. Mext-NSF-JSPS International Symposium (ISSS02); 2003; 317:338. (Full text)

[21] Kristin Lauter, Anton Mityagin. Security Analysis of KEA Authenticated Key Exchange Protocol. PKC 2006, volume 3958 of LNCS; 2006; 378:394. (Full text)

[22] Trevor Perrin, Moxie Marlinspike (2013). Advanced cryptographic ratcheting. (Web link)

[23] X.Deng, C.H.Lee , H.Zhu. Deniable authentication protocols. IEE Proc.Comput.Digital Techniques, 2001, 148(2):101-104. (Web link)

[24] Mario Di Raimondo, Rosario Gennaro. New Approaches for Deniable Authentication. Journal of Cryptology, 2009, 22(4):572-615. (Full text)

[25] Phil Zimmermann. ZRTP: Media Path Key Agreement for Unicast Secure RTP. Network Working Group Internet-Draft; June 17, 2010; 1:31. (Web link)