OnionPhone V0.1a

http://torfone.org
(C) Van Gegel torfone@ukr.net

General description:

OnionPhone is a VOIP tool for *calling over Tor network* and can be used as a VOIP plugin for TorChat. Call is made to the onion address of the recipient (its *hidden service*). The recipient can install a reverse connection to the originator and use a faster channel periodically resetting the slower channel to reduce overal latency.
Also provides the ability to switch to a *direct connection (with NAT traversal)* after the connection is established over Tor (Tor rather SIP without one specific server, the registration and collection of metadata). And also OnionPhone can establish a direct UDP or TCP connection to the specified port on IP-address or host. OnionPhone uses a proprietary protocol (not RTP) with only one byte unencrypted header that can provide *some obfuscation of the DPI.*

OnionPhone provides *independent level of p2p encryption and authentication* that is uses modern cryptographic primitives: Diffie-Hellmann key exchange on Elliptic Curve 25519 and Keccak Sponge Duplexing encryption.
In the case of a call to the onion address Tor protects against Man-in-Middle attacks. Also the recipient can verify identity of the originator's onion address (only with the permission of the sender) similar the TorChat authentication (like Abadi protocol uses).
Otherwise possible multifactor authentication:
- voice (biometric);
- using previously shared password (with the possibility of hidden notification of coercion);
- using a long-term public keys signed by PGP;

Onion Phone provides *Perfect Forward Secrecy* (uses a key agreement 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 SKEME protocol uses for initial key exchange.

OnionPhone can use a wide range of voice codecs (18 pcs) from *ultra low-bitrates* (MELPE at 1200 bps) up to *high quality* with redundance (SPEEX at 15000 bps).
Implemented *noise suppressor* of environment sounds and *automatic gain control* of the microphone. Built-in *LPC vocoder* with the possibility of irreversible change of voice (robot, breathy etc.). Specially designed *dynamic adaptive buffer* usefull for compensate jitter in high latency Tor connections. Available the radio mode (*Push to Talk*) and voice control (*Voice Active Detector*) with generation a noise-like short signal when transmission is completed.

The OnionPhone:
- is a *console* tool 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;
- *statically linked*, does not require additional third-party libraries and uses a minimum of system functions;
- can be compiled for *Linux* systems (Debian, Ubuntu etc.) using GCC or *Win32* (from Windows 98 up to 8) using MinGW.

Quick start and easy usage:

The easiest way to use OnionOhine as a VOIP plugin for TorChat:
Step 1: Put the OnionPhone folder on the hard disk, removable media, or TrueCrypt conainer (preferred).
Step 2: Edit the TorChat configuration file *'\torchat\bin\Tor\torrc.txt'*
add the following line
HiddenServicePort 11009 127.0.0.1:11009
new line
HiddenServicePort 17447 127.0.0.1:17447
Step 3: Right click on *'myself'* iteam in TorChat contacts list and copy ID to clipboard. Edit the OnionPhone configuration file *'conf.txt'*: specify parameter *'Our_onion'* using copied ID, for example:
Our_onion=r4kxspnzpnsel4fu
Step 4: Run the TorChat and OnionPhone

You are now ready to receive incoming and make outgoing calls:

- To accept an incoming call press *'Enter'*
- To make an outgoing call as a guest to guest (without using of personal public keys), type command: **–Oremote_onion_address** and press *'Enter'* then wait 10-30 sec for connecting over Tor.
- To continuous enabling / disabling of the voice transmission use the *'Enter'*. Hold down / release the *'Tab'* for Push-to-Talk mode.
- To select the voice codec from **1** to **18** use the command **-Ccodec_number** Smaller numbers correspond to low bitrate codecs, great - high quality. Numbers from **16** to **18** correspond to the variable bit rate codecs.
- To enable security vocoder use the command **–Qmode** (mode3 correspond 'breathy" voice (preffered), modes 6-255 correspond "robot" etc). For deactivation of vocoder use the command **–Q-1**
- 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 OnioPhone use the command **-X**

Users interface:

<Back> removes the last typed character
<Del> clears the typed string
<Tab> performed voice transmission while key hold down (Push-to-Talk mode)
<Sift+Tab> (Linux) or **<Ctrl+Tab>**(Windows) activates the voice detector
<Up>, **<Down>** arrows used to navigate between menu units.
<Left>, **<Right>** arrows used for menus item selection.
<Enter>:

- answers while incoming call is waits for accepting
- if command line is empty enables / disables continuous voice transmission
- if the first char of command line is '-' (command was typed) process the command
- otherwise sends typed chat message

<Esc>:
- rejects while incoming call is waits for accepting
- click twice for emergency exits the program clearing the memory

Commands:

The command should start with a *'-'* symbol followed capital letter followed optional parameter. No space can be between letter and parameter. Some commands may contain several subcommands separated by spaces. After typing the commands press <*Enter*> for processing. While typing use <*Back*> for correction of typed characters and <*Del*> for canceling and cleaning of command string.

Address book management:

-V[filter] displays all the entries in the address book containing the substring *'filter'*. If the parameter to be omitted displays all uncommented entries.
-E[name] outputs the command for call to the subscriber *'name'* defined in the address book. If parameter to be omitted outputs the most recently used call command (redialing).

Application and Connection management:

-A answer an incoming call (the same as pressing ENTER)
-H graceful terminated the call
-X graceful shutdown the program

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 applies.
-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 displays. If **'-1'** the minimal buffering applies (lower latency but bad resistance to jitter). If the parameter is omitted the automatic buffering adaptation by the communication channel will use (default).
-Q [voice] sets the style of voice preprocessing. Parameter *'voice'* is: **'?'** - displays the current settings, **'0'**-no automatic microphone gain control (AGC) and no noise supressing (NPP), **'1'**-applies AGC, **'2'**-applies NPP, **'3'** - applies a vocoder in "breathy" mode, **'4'** - in "high" mode, **'5'**- in "deep" mode, **'6'-'255'** - in "robot" mode with different pitch and **'-1'** - disables the vocoder (unchanged voice will be sended).

Key sharing:

-K [name] sent specified public key 'name' from the phone book to other user. This command is only possible if the connection is active. If parameter omitted sent own public key specified in the configuration file. The keys are receives and automatically adds to the address book with option **-L** without parameter (untrusted). Later user can view the address book and check out all the untrusted keys (for example, check the PGP-signature into key file) and edit the trust level to that key.

Authentication:

-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).
In the case of manually entering a password (preferably for safety) provides the ability to hidden alerts the other side being under duress: user can changes the last character of the password on any other. In this case the authentication on his side will look as usual but on the opposite side abonent receive a warning. An attacker can not verify in any way the correctness of the last letter of the password even when analyzing dumps of previous sessions, controling current session, as well as with the active "phishing" by creating test sessions to both parties, both before and after.

-W [My_onion] initiates verification originator's onion address by creating reverse connection to the declared hidden service and exchanging of verifiers (like TorChat autentication procedure). Command can be used only if the Tor connection to onion adress used and originator allowed verification. If the originator's specified onion address in configuration file and under the above conditions verification is performed automatically after the key agreement. Manually execution of this command is useful if the originator at the first time did not give agreed to onion-verification (does not have own onion-address in the configuration file), but then decided to declare own onion-address to the opposite side. In this case First, the originator, and then the recipient must perform this command.
If parameter omitted onion-address specified in configuration file will not be uses. This is prevents unwanted verification (so, the recipient does not know the onion-address of the originator and the originator remains completely anonymous).

-T [interval] sets the interval to reconnect the slower of the two onion-connections (to reduce the overall latency of the connection). If parameter omitted the default value is used in the configuration file. If the parameter is **'0'**, the counter connection is not uses for data exchanging and breaks immediately after verifying the onion-address. If **'-1'** the counter connection is not established (onion-address verification is not possible). If the parameter is too large (e.g., **'1000000000'**), the counter-connection is used, but never reconnected.

Switch to direct connection:

-S [stun] Initiates switching from onion-connection to direct UDP connection and specifies STUN-server for the NAT traversal. If parameter omitted the default STUN specified in configuration file will be used. If **'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.

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

<u>Extended call command:</u>

-N[name] destination_commad [options_commands]
Initiates a connection with the subscriber ‘name’ (the subscriber’s public key must be added to 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 the attacker). The authentication by using *‘guest’* key is impossible and there is a danger MitM-attack in the case of installation of direct connection . In the case of over Tor connection the key agreement is going on inside Tor so MitM like impossible (while you trust Tor).
destination_commad determines the type of connection (UDP, TCP, Onion) and the address of the recipient (IP or hostname and option port number separated by *‘:’*), for example:
-U178.95.218.29:17447
-Talica.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 (only for **-O**) 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.
Further separates by spaces can follow subcommands **-P [password]** (see above) and **-I [our_name]**
‘our_name’ parameter specifies the name of the originator appears to the recipient. If this option is completely omitted the proper name is defaults as defined in the configuration file. If **-I** option is used without a parameter the default name *‘guest’* will be uses (see above). If parameter is specified will use the specified name as proper name for 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 his adress book, otherwise the connection will not be established.
If subscribers are connected first with each other they can be assured only in the presence of guest key on both sides. Thus the first connection must be initiated as untrusted *guest -> guest*:
-N –I –Oaddress
Such this connection is still encrypted with the session key but can resist MitM just so so you trust Tor. In addition the parties can not 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 (it denied because this keys are public and attacker can not be sure that customers really declare own keys: 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 of the PGP-signatures users can set the desired level of trust by editing contacts entries in the address book file. Later users can re-install already trusted connection:
-Nalice -Or4kxspnzpnsel4fu -Ibob

Key managment

OnionPhone uses public keys to authenticate the subscribers to each other using PGP. Also the keys in this format will be used in other projects in the future (audio chat group, encryption radio and GSM-audio channels, etc.). Console utility '*addkey*' usefull for key managment.

Step 1: To generate a new key pair use:
addkey -Aname [options]
'*name*' is your identifier and options will be passed to other participants. The most common use is option -Oour_onion_address to indicate own address in the key. Upon receipt of your key this option will be automatically copied in address books of other participants. For example, after the
./addkey -Galice -Or4kxspnzpnsel4fu
files *'alice.sec'* (private key) and *'alice'* (the public key) will be created in the folder 'KEYS'. Private key is not protected so it is recommended to store OnionPhone folder in a protected place (TrueCrypt container etc.).

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 (rename 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 parameter **'Our_name'** as keys name:
Our_name = alica

Step 4: Now you can send the key to other users. For transmission of the key establish un-authenticated call *'guest -> guest'* with the other party (see above) and execute
-Kalica (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 lower level of trust. After manual checking of PGP-signature in the key file user can set the desired trust level by editing **-L** parameter to the corresponding entries in the address book file *'KEYS / contacts.txt'*, for example:
[alica] {FNrjEjGmlZtvKXzBQkNIDA ==} #alica -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

OnionPhone Cryptography

Primitives:

- **Assymetric cryptography** is ECDH25519 (curve25519_donna C-code) [1];
- **Symmetric cryptography** is Keccak Sponge Dulpexing encryption [2];
- **SPRNG** is Keccack Sponge Duplexing RNG [2] with Havege reseeding.

Key formats:

1. **Public key** is a text file contains: Info, Key, Sig(Info, Key)
- Info-string with first symbol ‘#’, name and options.
  Option is: ‘-‘, followed by option type (capital letter), followed by parameter (chars). Options separated by spaces. All options passes to addrees book for further processing by application.
- Key string is base64 representation of ECDH 25519 Montgomery 256 bits public key (in figure brackets).
- Some other string are optional comments etc.
- PGP signature is a part of key file: info-string and key string must be signed by PGP.

Key stamp (**ID**) is a 128 bits hash of keyfile. After keyfile was generated and signed you can’t modify it but you can rename it. You must add own public key to your addressbook before using. All other parties also must add this key before using.

2. **Private key** is a binary file contains 256 bits ECDH25519 secret.

Deniability authenticated Key agreement:

Alice is initiator and connects to Bob:

A:
- generates two ECDH pairs **Xx**, **Pp**;
- computes 128 bits modificator $\mathbf{R}$=H($\mathbf{p}$);
- computes 128 bits commitment $\mathbf{C}$=H($\mathbf{R}$);
- computes 32 bits identificator $\mathbf{N_A}$=H($\mathbf{ID_A \mid X \mid B^p}$),
  where $\mathbf{ID_A}$ is Alica’s key-stamp and **B** is Bob’s ECDH public key;
- sends *REQUEST* to Bob: $\mathbf{N_A}$**, X, P, C**.

B:
- receives *REQUEST* from unknown: $\mathbf{N_A}$**, X, P, C**;
- precomputes $\mathbf{P^b}$, where **b** is Bob’s private key;
- Computes expected identificators $\mathbf{N_i}$=H($\mathbf{ID_i \mid X \mid P^b}$) for each contact ID extracted from address book;
- finds $\mathbf{N_A}$= =$\mathbf{N_i}$ matched for sender (for Alica’s $\mathbf{ID_A}$);
- generates two ECDH pairs **Yy, Qq**;
- computes two 128 bits session keys: $\mathbf{S_a|S_k}$=H($\mathbf{X^y}$);
- computes two 128 bits authenticators:
  $\mathbf{M_A}$=H($\mathbf{A^q \mid P^b \mid P^q \mid X \mid Y}$)

$\mathbf{M_B}$=H($\mathbf{A^q}$ | $\mathbf{P^b}$ | $\mathbf{P^q}$ | **Y** | **X**);
- sends *ANSWER* to Alice: **Y, Q,** $\mathbf{M_B}$.

Note:
- this is a fully deniable [4] SKEME autentification protocol with difference that original SKEME [3] is used Abadi [6] protocol for signing parameters X, Y but our variant is used TripleDH (slightly modified KEA+ [5]) protocol;
- Bob also may compute identificator $\mathbf{N_B}$=H($\mathbf{ID_B}$ | **Y** | $\mathbf{A^q}$) for presentation his identity for Alice. But this application has connection style of communication instead of using the common environment. Thats why Alica can detect Bob by transport and $\mathbf{N_B}$ can be ommited.

A:
- receives *ANSWER* from Bob (sender detects by transport): **Y, Q,** $\mathbf{M_B}$;
- computes session keys: $\mathbf{S_a}$|$\mathbf{S_k}$=H($\mathbf{X^y}$);
- computes authenticators:
  $\mathbf{M_A}$=H($\mathbf{Q^a}$ | $\mathbf{B^p}$ | $\mathbf{Q^p}$ | **X** | **Y**)
  $\mathbf{M_B}$=H($\mathbf{Q^a}$ | $\mathbf{B^p}$ | $\mathbf{Q^p}$ | **Y** | **X**), where **a** is Alice private key;
- checks $\mathbf{M_B}$ is matched;
- computes 32 bits control value **L**=H(**R** | $\mathbf{S_a}$) and prints words for voice autentification;
- sends *ACK* to Bob: **R,** $\mathbf{M_A}$.

B:
- receives *ACK* from Alice: **R,** $\mathbf{M_A}$;
- check $\mathbf{M_A}$ is matched;
- check commitment is matched: **C**= =H(**R**);
- computes contol value **L**=H(**R** | $\mathbf{S_a}$) and prints words for voice autentification.

Optional authentication by pre-shared password:

Password must be identical in both parties except last two chars: one of the parties swaps them with each other. Defines that Alice's password is **pass|p|q** and Bob's password is **pass|q|p**

The participant under pressing must change last char of own password on any other. Autentification will look like correct from it's side but other party will be notified.
Each party can initiate autentification independently, for example, Alice is the originator of call, and initiates autentification using pra-shared password:

A:
- computes $\mathbf{U1_A}$=H(**O** | $\mathbf{S_a}$ | **pass**), where **O** is char type 0x00 for originator of call (Alice) and 0x01 for acceptor of call (Bob);
- sends *AU_REQ* to Bob: $\mathbf{U1_A}$.

B:
- receives *AU_REQ* from Alice: $\mathbf{U1_A}$;
- recomputes and checks it's validity;
- computes $\mathbf{U1_B}$ = H(**O+0x10** | $\mathbf{S_a}$ | **pass**);
- computes $U2_B$ = H(**O+0x10** | $\mathbf{S_a}$ | **pass** | **p**);
- sends *AU_ANS* to Alice: $\mathbf{U1_B}$, $\mathbf{U2_B}$.

A:
- receives *AU_REQ* from Bob: $\mathbf{U1_B}$, $\mathbf{U2_B}$;
- recomputes and check validity of $\mathbf{U1_B}$;
- recomputes and check validity of $\mathbf{U2_B}$: if wrong than notes: 'Bob under pressing';
- computes $\mathbf{U2_A} = H(\mathbf{O} \mid \mathbf{S_a} \mid \mathbf{pass} \mid \mathbf{q})$;
- sends AU_ACK to Bob: $\mathbf{U2_A}$.

B:
- receives *AU_ACK* from Alice: $\mathbf{U2_A}$;
- recomputes and check validity of $\mathbf{U2_A}$: if wrong than notes: 'Alice under pressing'.

<u>Note:</u> the participant never sends **U2** before then will be ensured that received **U1** is valid. An attacker can not verify the validity of the last character or during the execution of the protocol or later, even if it has the old sessions dumps and makes the test sessions with the parties both before and after.

Originator's onion address verification:

After initiator was connects to acceptor's Hidden Service he is sure of acceptors address (while he trust Tor). But acceptor can't know initiator's adress: initiator must present it to acceptor. Acceptor can trust it only after it's verification. Like Abadi protocol uses for this (similar to TorChat verification).
Alica is initiator and Bob accepts the connection to its Hidden Service:

A: connects to Bob's HS using their onion address. Parties agree keys as describe above.

B: requests for Alice's onion address.

A: sends own onion address to Bob.

B:
- connects to this adress;
- computes invite $\mathbf{I_{req}}=H(\mathbf{K_a})$. Uses the second half (128 bits) of 256-bit hash;
- sends $\mathbf{I_{req}}$ to Bob.

A:
- receives I_{req} from Bob;
- checks its validity;
- computes invite $\mathbf{I_{ans}}=H(\mathbf{K_a})$. Uses the first half (128 bits) of 256-bit hash;
- sends $\mathbf{I_{ans}}$ to Bob.

B:
- receives $\mathbf{I_{req}}$ from Bob;
- checks its validity. Now Bob is sure that Alice onion adress is valid.

Encryption:

Both parties have synchronizing 32 bits counters **CTR** separately for sended and received packets. TCP transport is lossless and provide counters synchronization himself. For UDP each packet's header contains last 7 bit of sender's outgoing counter value for synchronization incoming couter on receiver's side.
For each packet Keccak Sponge initiates and absorbs ($\mathbf{S_k}$ **| CTR | O**), where **O** is 0x00 for an originator of the call and 0x01 for an acceptor of the call, then squeezes gamma and XORed with packet's bytes except first byte (header **HD**).
After packet was encrypted 32 bits **MAC** computed:
MAC=H($\mathbf{S_k}$ **| CTR | O | PKT**),
where PKT is encrypted packet including first header byte. After this CTR increments by 1. There is no way to return **CTR** counter back
Encrypted packet is **HD, BODY, MAC**,
where **HD** is unencrypted header (1 byte), **BODY** is encrypted data and **MAC** is autentification of (**HD | BODY**).
Decryption is similar to encryption.

Knowledge base:

1. Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records http://cr.yp.to/ecdh/curve25519-20060209.pdf
2. Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications http://eprint.iacr.org/2011/499.pdf
3. Hugo Krawczyk. SKEME: A Versatile Secure Key Excha Mechanism for Internet http://www.di.unisa.it/~ads/corso-security/www/CORSO-9900/oracle/skeme.pdf
4. Mario Di Raimondo, Rosario Gennaro, Hugo Krawczyk. Deniable Authentication and Key Exchange https://www.dmi.unict.it/diraimondo/web/wp-content/uploads/papers/deniability-ake.pdf
5. Kristin Lauter, Anton Mityagin. Security Analysis of KEA Authenticated Key Exchange Protocol http://research.microsoft.com/en-us/um/people/klauter/security_of_kea_ake_protocol.pdf
6. Martin Abadi, C'edric Fournet. Private Authentication http://users.soe.ucsc.edu/~abadi/Papers/tcs-private-authentication.pdf