Note:
this is a first raw version (draft only) with a lots of grammar and possibly errors !!! Not for public yet.

Torfone is a utility that works under Tor transporting compressed voice packets. Effective voice queue and use of duplicate circuits significantly reduces speech latency now is comparable to satellite phones. In addition users who do not care about anonymity can switch to direct connection mode by performing a NAT traversal using the established Tor connection. This functionality provides unique server-less VOIP solution works without any online registration and prevents the collection of metadata.

Torfone provides its own level of obfuscation, encryption and authentication using modern cryptography. Our protocol is not standard so you can ignore its presence fully trusting Tor. Nevertheless it provides reliable protection on primary direct connecting mode using the IP address and port as the destination of subscriber:

- Connecting is performed in two physical TCP sessions with random time interval (for some DPI protecting).
- Exchange Elligator2 key representations with random padding during first session (against censorship).
- Authenticate encrypting and random padding for all transcripts in second session with key derived in first session (for some DPI protecting)
- Proof of job required in second session (for protecting against scanners).
- Encrypting and decrypting symmetric keys derives with only DH and not depends from authentication (allowed unauthenticated sessions).
- DH secret is unpredictable due commitment (allowed comparing short secret's fingerprints: SAS)
- Zero knowledge mutual authentication by application public keys (Triple DH extended with SPEKE secret for zero knowledge property)
- KCI resistance (for leak of long term public keys)
- Deniability for using private key (in the case of other party communicate with Judge after session)
- Forward deniability (in the case of other party communicate with Judge before session to known participant)
- Protecting of originator's ID against passive and active attacks (interrupted probes and interceptions) with PFS.
- Automatic receiving any keys send by other party saved with generated names (for deniability of having the key).
- Zero knowledge comparing of pre-shared secrets during unauthenticated session (like SMP in OTR but uses SPEKE)
- Authentication of caller's onion address based on Hidden Service security (as in TorChat)

Torfone's structure

Torfone have 4 modules: transport (TR), cryptor (CR) including audio engine, storage (ST) and user (UI) including GUI.

Transport module must run under OS (Linux, Windows, Android) and provide communication with WAN and Tor SOCKS5 interface and NAT traversal for create p2p UDP connecting. Also this module provides encrypting level for obfuscation from some kinds of DPI. And also there is an audio queue needs in high-latency onion environment.
Cryptor module can be run on the same or different HW (including embedded Cortex M4 HW) and can communicate with transport module directly or over serial interface using specified packet format controlled by cryptor. Cryptor provide initial key exchange, symmetric encrypting layer and audio playing and grabbing depends of HW platform (alsa for Linux, wave for Windows, openSL for Android and ADC/DAC with DMA for Cortex M4 barrel metal).
Storage module can be run on the same or different HW (including embedded Cortex M1 HW) and can communicate with cryptor module directly or over serial interface using specified packet format controlled by storage and cryptor. Storage provide address book with contact's names, addresses and public keys, store own private key and provide scalar multiplication with it. Also there is true random number generator including Havege on OS or ADC based generator on embedded HW platforms.
User module provide main engine for work with address book, do calls etc. This module can be run under OS or on embedded HW (barrel metal) and also can communicate with cryptor module directly or over serial interface using specified packet format controlled by cryptor. User module has simple C-interface with GUI can be developing as an platform depend environment. For GUI we use old Wide Studio for Linux GUI, Borland C++ Builder 6.0 for Windows GUI, Embarcadero RAD Studio 10.2 for Android GUI and Arduino based code for 320*240 TFT + Touch display on HW platform (STM32F446RE Nucleo board).

There are some variants for developers can combine these modules:
- simplest but unsafe (due OS vulnerabilities) standalone applications for Linux, Windows or Android;
- external storage as an token connected to Desktop over USB CDC or to Mobile over Bluetooth SPP;
- most safe configuration with Tor + transport module runs on single-board device under Linux (RPi zero etc.) and all other modules runs on STM32F446RE Nucleo board with TFT + Touch screen in one thread and connected over UART.

Source files list:

1. **Transport (Linux, Windows, Android)**
   tr.c: sockets, connect/accept, SOCKS5 for Tor connect, timeouts, NAT traversal etc.
   local_if.c: platform-depend procedure for obtain local IP
   jit.c: audio queue for incoming audio and collector for outgoing audio
   tr_if.c: interface to other modules (memory or serial)
   obflib: folder for obfuscation library
    rnd.c: encrypting, decrypting, random padding etc.
    serpent.c: serpent-128 symmetric algo
    ocb.c: OCB cipher mode

2. **Cryptor (Linux, Windows, Android, Cortex M4)**
   cr.c: main engine of torfone crypto
   cr_if.c: interface to other modules
   ike.c: initial key exchange
   pke.c: SPEKE and protecting of contact exchange
   crplib: folder for primitives
    KeccakP800.c: sponge permutation optimized for 32 bits platform
    shake.c: shake-128 XOF
   ecc: folder for X25519 math
    add.c: field addition
    copy.c: copies data

cswap.c: constant time swap data
from_bytes.c: convert string to data
invert.c: invert scalar
mul.c: multiple scalars
mul121665.c: multiple with constant
sqr.c: square scalar
sub.c: subtracts
to_bytes.c: convert data to string
scalarmult.c: multiple points
r2p.c: elligator2 and point representation
snd: forlder for voice processing procedures
audio.c: high-layer voice procedures
amr: folder for AMR 4750 encoding/decoding
npp: folder for noise protecting procedures
dev: folder for platform-depend low-level audio processing
audio_alsa.c: Linux
audio_wave.c: Windows
audio_sles.c: Android
aecm.c: echo suppression for all platforms

3. **Storage: (Linux, Windows, Android, Cortex M1 and M4)**
   st.c: contacts and private key storage engine
   st_if.c: interface to other modules
   book.c: platform-depend address book engine (files on OS, Flash on HW).
   tndlib: library for book encrypting end TRNG
   chacha20.c: cipher for address book protecting
   curve25519donna.c: fast realization of X25519 point multiplication
   trng.c: true random number generation and check statistic
   havege.c: entropy collector for OS
   timing.c: platform-depend timing for entropy collector

4. **User: (Linux, Windows, Android, Cortex M4)**
   ui.c: main operation with contacts, calls etc.
   ui_if.h: interface to other modules
   whereami.c helper for get current path of executable

5. **GUI: platform-depend realization of GUI input/output**
   In separated thread uses thread-safe C-interface to ui.c
   (Linux, Windows, Android, Cortex M4)

KeccakP800.c, chacha20.c and all math for X25519 (curve25519donna.c and ecc folder) replaces with volatile assembler codes developed specially for Cortex M1 or Cortex M4 platforms.

Torfone Android user manual:

Installation:

Download *http://torfone.org/download/Torfone.apk*, install and run.
Input password 1-31 chars (one or two words separated by space) in blue field. Press New button for apply.
Now Torfone ready for use: Tor private key (hidden service) and application private key were created.

Note: on first run 'torfone' folder will be created in Shared Documents path. Files into this folder are: 'tf.ini' is settings, 'bb' is encrypted address book, 'bb.sec' is key material encrypted by application password, 'hs_backup' is unencrypted copy of Tor hidden service private key. Store 'tf.ini', 'bb', 'bb.sec' and 'hs_backup' in a safe place and delete 'hs_backup' from your device (but left other files on his places!). For restoring on new device get last copy of 'torfone' folder from old device, add 'hs_backup' into it and put this folder to Shared Documents path of the new device. Install and run Torfone: your onion address, application private key, contacts and other setting will be restored on the new device. Old password will be required on the start.

Note: password protects application key material and address book. By default password will be saved in private file space of Torfone. If you want a more safety check Password checkbox on the Settings panel and enter password every time Torfone runs.

Setings panel:

pull left border of the screen for drive Settings panel. See Our onion address field displays your onion address:port (number for calls) generated by Tor. You will send this address to you friends.
See detailed manual about other settings.

Contacts list:

pull right border of the screen for drive Contacts list.
'*Myself' item on the top of list is your address alias.
Contacts can be adds to your list manually (by you) or automatically (by remote party during call).
The last functionality can be disabled on Settings panel.

Create new contact:

Press New button and enter name in blue field. Press New button again. New contact with specified name and empty address will be created.

Select contact:

Open contact list (see above) and click to contact. The name will be copied to name field and the address – to address field on the main panel.

Change contact:

Edit name and/or address field after contact was selected. Press Change button to apply new values.

Delete contact:

Clear name field after contact was selected. Press Change button to delete this contact.

Coping contact:

Edit name and/or address field after contact was selected. The first char of new name must be '='. Press Change button to create alias of selected contact (with same key but different name/address).

Outgoing call:

Select contact (see above) and press Call button. Wait for connecting. After connecting success the 4 Hex digits of the session code will be showed and long beeps will be heard until the subscriber answers.

Cancel call:
Press Cancel button for cancelling outgoing or incoming calls in any stage.

Answer incoming call:
After incoming connecting the 4 Hex digits session code will be showed and ring will be heard. You will also receive a system notification from your application (this option can be disabled on the Settings panel).
For authenticated call (if you already have contact of caller and caller also has your contact) caller's name and address will be showed in corresponds fields. Press Call button for answer.

Talk/Mute and change audio path:
During active call press Call button to switch Talk/Mute mode. Also during the call you can change voice output path (loudspeaker or earphone) on Settings panel and re-change Talk/Mute to apply new settings.

Application authentication:
If you already have contact of other party and other party also have your contact the name icon will be green on both sides. Caller's name and address will be showed in corresponds fields of calee. This connecting is trusted.

Onion authentication:
If you provide onion call that you are sure than onion address of remote party is exactly you specified on address field (while you trust Tor). Address icon will be green on caller side. If parties are already authenticated (see above) calee provide parallel call to caller ensures that caller address is also valid. In this case Address icon will be green on callee side too.

Authentication by shared secret:
Torfone can perform unauthenticated calls (both parties haven't each other contacts in address books, caller only have address of calee). In this case parties can ensure link is safe by comparing pre-agreed secret phrase. Press New button during active call, enter phrase in blue field and press again. New button on other side will be blue (request). Now other participant must do the same. New buttons will be green (secrets the same) or red (secrets not matches) on both sides.

Authentication by session ID (SAS):
After call is established 4 Hex digits of SAS will be displayed and must be the same on both sides. You can remember value and compare it later (on direct contact or using other reliable communication channel) to make sure that this session was safe.

Sending contacts to remote:
During the call participants can send each other contact from his address books. Open contact list and click contact you want send to remote. It can be '*Myself' or each other contact presented in your address book. Contact will be automatically add to address book of you party (automatic receiving can be disabled on Settings panel, in this case Change icon will be blue indicating request). The name which the contact will be added to address book will be generate automatically as an string of '+', 4 hex digits of current session ID and 8 Hex

digits of key fingerprint of received contact. Later parties can compare session ID in names of exchanged contacts must be the same (if session was safe - received contacts are trusted). Or any time after call you can rename received contacts to any unique names suitable for you.

Switch to UDP direct connecting:

Tor connecting is anonymous and safe but has significant latency (comparable to satellite phones). If users do not care about anonymity they can try to establish a direct UDP connection by performing the NAT traversal using an existing Tor connection and external STUN servers. Safety still provided by the Torfone internal encrypting layers with session keys shared under Tor. During Tor call press Direct button for try switch to direct connecting. On the other side button will be blue (request). You participant also must press Direct buttons for start NAT traversal procedure.
If NAN traversal complete (not always possible) Direct buttons on both sides will be green. Any time later you can return back to Tor connection (it still active). For this any party must press the Direct button again.

Initial TCP direct connecting:

Instead of using Tor to route your calls you can use IP address (domain name):port to make a direct TCP connection. To receive incoming calls you may have to open the corresponding TCP port on your router. The number of listening port is a part of your address and is accessible on the Settings panel. Checkbox WAN must also be set to receive connections directly from Internet.

tDH extended with SPEKE

Recently I chose the Initial Key Exchange procedure for this project and compare the properties of known protocols. I found that tDH has certain minor flaws that can be fixed.

The one flaw is the weak PFS of sender's ID protection: Eve intercepts Alice's connection to Bob and claims as Bob. She follows the tDH protocol, receives the authenticator and aborts.
Later, by revealing Bob's private key, Eve can check Alice participation in this past session.

Many other protocols also inherit this flaw.
For example, see the Noise Structure Document:
http://www.noiseprotocol.org/noise.html chapter 7.8. "Personality hiding"
Authors are going to use signatures in the future. But to use signatures we need the full format of the points (both x and y coordinates or at least a sign of y), which is incompatible with X25519 Montgomery format.

There is an elegant way to fix this flaw by extending tDH using the SPEKE protocol performed in parallel.
SPEKE will only require a Hash2Point implementation based on Elligator2. This is easy to implement with elementary field math of X25519.

SPEKE base point is derived from the value B * x == X * b. Both sides compute SPEKE public keys using random private keys independent of DH private keys (x and y). The parties exchange the DH and SPEKE keys simultaneously. SPEKE shared secret can be used to explicitly protect the sender's ID and is also included into tDH hash (there will now be four elements) to provide PFS of implicit ID protecting.

SPEKE has only recently become free so perhaps other effective combinations can be used to develop protocols with good properties.

Initial key exchange (IKE) features:
- Connecting is performed in two physical TCP sessions with random time interval (for some DPI protecting)
- Exchange Elligator2 key representations with random padding during first connecting
- Authenticate encrypting and random padding for all transcripts in second session with key derived in first session
- Proof of job required in second session (scanning protecting)
- Encrypting and decrypting symmetric keys derives with only DH and not depends from authentication (allowed unauthenticated sessions)
- DH secret is unpredictable due commitment (allowed comparing short secret's fingerprints: SAS)
- Zero knowledge mutual authentication by application public keys (Triple DH extended with SPEKE secret for zero knowledge property)
- KCI resistance (for leak of long term public keys)
- Deniability for using private key (in the case of other party communicate with Judge after session)
- Forward deniability (in the case of other party communicate with Judge before session to known participant)
- Protecting of originator's ID against passive and active attacks (interrupted probes and interceptions) with PFS

Alice is originator, have long-term key pair a and A and also have Bob's public key B in address book.
Bob have long-term key pair b and B and also have Alice's public key A in address book.
Alice know that she originate call to Bob (key B) but Bob not know caller before IKE: Alice must send his key A to Bob in safe way.
H is Hash, MAC and PKDF use Shake-128 XOF with Keccak-800 permutation.
Elliptic curve X25519 is used. E is Hash-To-Point used Elligator2 algorithm. For detailed transcript see diagram.

So session symmetric keys produced by DH, any parties can contact even without knowing each other application public keys.
The ways for alternatively authentication (MitM protection) are:
- Comparing SAS during or after session (like in ZRTP);
- Zero-knowledge comparing short common secret (PIN-code) shared before session (like in OTR but use SPEKE instead of SMP).
- Authentication by onion addresses (like in TorChat) uses Martin Abadi protocol with public key encryption (PKE) with hidden service RSA key provided by Tor.

After unknown participants was contacted and ensures link is safe they can exchange his public keys and save in address book.
Technically we send contact in two pass: address then public key (both protected from passive observer).
Address linked to key (protecting from active attacker) and automatically save in address book with name generated as SAS and key's fingerprint string. Later participants can compare SAS for ensure session was safe and keys are valid.
So any unauthorized caller can call and send any key will be saved in book automatically the deniability for having the key is provided. Address book encrypts with key derives from main application password applies once on the start. Other part of this password protects long-term private key.

Normally address book and curve multiplication with long-term private key doing into hardware device (storage token) as external module connected via serial interface: physical UART, USB (CDC device) or Bluetooth (SPP profile).
Other modules is GUI with human interface (safe), Cryptor with audio interface (safe) and Transport with Tor (partially safe).
This modules can be realized on same or different hardware and communicate each other with strictly specified commands using safe interface (UART etc.). So if Linux HW with Tor + Transport will be compromised (for example due SPECTRE/MELTDOWN) then one-thread Storage, Cryptor and GUI on separated HW (Cortex M4) still safe. Developer can build your own system depend different safety and usability requirements.

Transport module has additional own protection layer. Connecting procedure split into two sessions. During a first session participants exchange Elligator2 representations of X25519 ephemerals with random padding and close connecting. After a few seconds second new connecting establishes. All transcripts are encrypts with Serpent128 in OCB mode with random padding. In addition encrypted key updated with initial data using PKDF. This is require some "proof of job" and can protect from active scanning based on data content (DPI scanning by timings still available). After IKE finished the transport encrypting key will be updated with IKE shared secret and transport layer will provide real encrypting and MAC instead obfuscation.

Detail transcript of IKE:

H is XOF Shake-128 with Keccak-800, curve is X25519, E is HashToPoint with Elligator2

Alice(originator): have a[32], A[32]=G * a, B[32]		Bob: have b[32], B[32]=G * b
pick x[32], u[32] at random Tb[32] = B * x (their tag) P[32] = E(Tb) (hash to point) U[32] = P * u (SPEKE key) X[32] = G * x (DH key) X'[32] = X ^ H(U) (mask DH key: kind of commitment)	X' -->	
	<-- Y	pick y[32], v[32] at random store masked DH key X' Y[32] = G * y (DH key)
S[32] = Y * x (DH secret)	U -->	
	<-- V	D[32] = U * v (SPEKE secret) X[32] = X' ^ H(U) (unmask X) S[32] = X * y (DH secret) Tb[32] = X * b (compute our tag in storage) P[32] = E(Tb) (hash to point) V[32] = P * v (SPEKE key)
Check V[32] != U[32] (prevent reflecting attack) D[32] = V * u (SPEKE secret) A'[32] = A ^ H1(D) (hide our ID) F[32] = H2[D]; (SPEKE tag for extended TripleDH)	A'-->	
	<-- C	D[32] = U * v (SPEKE secret) A[32] = A' ^ H1(D) (unhide their ID) F[32]=H2(D) (SPEKE tag for extended TripleDH) Ta = A * y (their tag) C[32] = H1(S) (convince know y for Forward deniability)
C[32] ?= H1(S) (check Bob know his y, abort) Ta[32] = Y * a (compute out tag in storage) Ke[16]|Kd[16]|Mx[16]|My'[16]|Kt[32]|SAS[2] = H(S) Ma[16] = H(F | S | Ta | Tb) (TripleDH output) Mb'[16] = H(F | S | Tb | Ta) (TripleDH expected)	Ma | Mx -->	
	<-- Mb | My	Kd[16]|Ke[16]|Mx[16]'|My[16]|Kt[32]|SAS[2] = H(S) Ma'[16] = H(F | S | Ta | Tb) (TripleDH expected) Mb[16] = H(F | S | Tb | Ta) (TripleDH output) Mx ?= Mx' : abort (shared secret not valid) Ma ?= Ma' : authenticate A (replace Mb with random if fail)
My ?= My' : abort (shared secret not valid) Mb ?= Mb' : authenticate B		

Both parties authenticate each other by public keys A and B, produce encrypting and decrypting keys Ke, Kd, shared secret Kt (for updating transport layer) and SAS. Symmetric encrypting on transport layer: Serpent-128 in OCB mode, on main layer: Shake-128 with Keccak-800 in CTR mode with one-way packet counters.

Detail transcript of send contact:

Alice(sender): have session symmetric encrypting and decrypting keys Ke[16], Kd[16] send key A[32] and onion-adress O[32] to Bob		Bob (receiver): have session symmetric encrypting and decrypting keys Ke[16], Kd[16]
O' = O ^ H(Ke | A)	O' -->	
	<-- q | Q	q[16] at random Q[16] = H(q | Ke | O')
Q'[16] = H(q | Kd | O') Q' ?= Q :abort A' = A ^ H(q | Q | Ke)	A' -->	
		A = A' ^ H(q | Q | Kd) O = O' ^ H(Kd | A) Check O is valid onion address Bob receive contact A | O

Party can send any contact from his address book (including himself) to remote side during authenticated or not-authenticated sessions. If remote party must allow receiving of contacts the key will be added to address book automatically. New contact name will be generated as SAS + received key's fingerprint as a string with first char '+'. Parties can compare SAS later to be ensured this session was safe and can trust received contacts. And later received contacts can be renamed to user specified names. So anyone can send any contact will be automatically saved in address book we have some deniability to having the keys sensitive for forensics.

Zero-knowledge comparing pre-shared secret using SPEKE (like SMP in OTR):

Alice(initiator): have session symmetric encrypting and decrypting keys Ke[16], Kd[16], password W[32]		Bob: have session symmetric encrypting and decrypting keys Ke[16], Kd[16], password W[32]
u[32] at random W'[32] = PKDF(W) (multiple invoke Shake-128 with salt) P[32] = E(W') (use Elligator2) U = P * u (SPEKE key)	U -->	
	<-- V	v[32] at random W'[32] = PKDF(W) (multiple invoke Shake-128 with salt) P[32] = E(W') (use Elligator2) V = P * v (SPEKE key)
D[32] = V * u (SPEKE secret) Ma[32] = H(D | Ke) (Alice's authenticator) Mb'[32] = H(D | Kd) (expected Bob's authenticator)	Ma -->	
	<-- Mb	D[32] = U * v (SPEKE secret) Mb[32] = H(D | Ke) (Bob'a authenticator) Ma'[32] = H(D | Kd)(Expected Alice's authenticator) Ma ?= Ma' : set Mb at random in constant time if fail Report result to user
Mb ?= Mb' Report result to user		

Torfone inter-module communications

Application Init:

User	Storage	Cryptor	Transport
FormCreate **CR_PTH >**			
		< ST_PTH cr_ui_pth	
	st_storage_init **CR_RND >**		
		cr_st_R **TR_SEED >**	
			< EVENT_SEED_OK tr_seed
		< UI_INI cr_event_ROK	
GetPSW **CR_PSW >**			
		< ST_PSW cr_ui_psw	
	st_set_password **CR_PUB >**		
		< UI_FGP cr_st_A	
SetFGP **TR_INIT >**			
			< **EVENT_INIT_OK** tr_init
		< UI_TCP cr_event_net	
SetNET **ST_REQ >**			
	< **UI_LST** st_get_next_name		
AddLST **ST_REQ >**			
	< **UI_LST** st_get_next_name		
AddLST **CR_AUD >**			
		cr_au	

Reset:

User	Storage	Cryptor	Transport
User: **CR_RST >**			
		cr_rst **TR_SET >**	
			< EVENT_TERMINATED call_terminate
		< ST_RST cr_term	
SetRST	**<UI_RST** st_reset		

Set contact:

User	Storage	Cryptor	Transport
ButtonSelect **ST_GETADR >**			
	st_get_addr_by_name **< UI_ADR**		
SetADR			

Add contact:

User	Storage	Cryptor	Transport
ButtonNew **ST_NEW >**			
	< UI_NEWOK st_new_contact		
AddNEW			

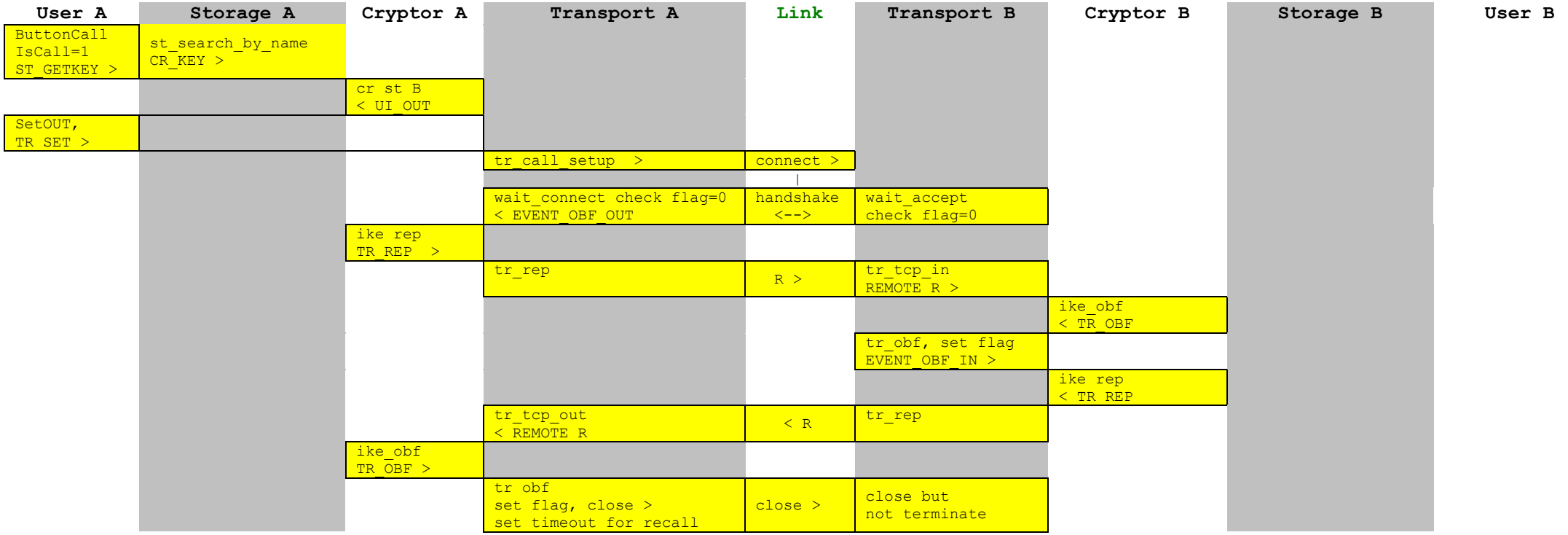
Change/delete contact:

User	Storage	Cryptor	Transport
ButtonSave **ST_ADR >**			
	< UI_ADROK st_set_address		
AddADR **ST_SAVE >**			
	< UI_SAVEOK st_save_contact		
SetUPD			

Mute:

User	Storage	Cryptor	Transport
ButtonMute **CR_MUTE >**		cr_ui_mute TR_MUTE >	**jt_set_talk**

Call: A > B (pre-IKE session)



In this session parties exchange X25519 ephemeral public keys as an Elligator2 pseudo-random representation in random-padded packets, share secret for obfuscating second session and close connection. Originator will reconnect after a random time of few seconds.

Call: A > B (IKE and work session)

User A	Storage A	Cryptor A	Transport A	Link	Transport B	Cryptor B	Storage B	User B
			reconnect >	connect >				
SetAcc, goto Call mode		cr event accept start audio Call < UI_ACCEPT	< EVENT_CALL_TCP	TCP layer	EVENT_ACCEPT_TCP >	cr event accept start audio Ring UI_ACCEPT >		SetAcc,goto Call mode
			wait_connect check flag=1 < EVENT READY OUT	handshake <-->	wait_accept check flag=1 EVENT READY IN >	cr_event_run ike run acceptor UI INC >		SetINC IsCall=2(TCP) or 3(Tor)
		cr_event_run ike_run_originator TR_Q (X') >						
						ike_r, ike_2 < TR_Q (Y)		
		ike r, ike 3 TR_Q (U) >						
						ike_r, ike_4 ST_SEC (X) >		
							< CR_SEC st_get_sec	
						ike r, ike 6 < TR_Q (V)		
		ike_r, ike_5 TR_Q (A) >						
						ike_r, ike_8 < TR_Q (C)		
		ike r, ike 7 < ST_SEC (Y)						
	st_get_sec CR_SEC >							
		ike r, ike 9 TR_Q (Ma) >						
						ike r, ike 10 ST GETNAME >		
							st_search_by_key < CR_NAME	
						cr_st_N < TR_Q (Mb)		
		ike r, ike 11 TR SEC >						
			tr_set_secret < EVENT SET CONTROL SECRET >		wait_tcp_in EVENT_REQ >			
		ike_r, ike_13				ike_r, ike_12 < TR_SEC		
			wait_tcp_out < EVENT_SEC		tr set secret EVENT_SEC > < CONTROL_SECRET			
SetCALL		ike r, ike 15 < UI_CALL				ike r, ike 14 UI_NAME >		
								SetNAME < ST GETADR
							st_get_addr_by_name UI_ADR >	
								< TR SET SetADR
				< connect				
		cr_event_run	wait accept < EVENT_READY_IN	handshake <-->	wait accept EVENT_READY_OUT >	cr event run UI_DBL >		SetDBL

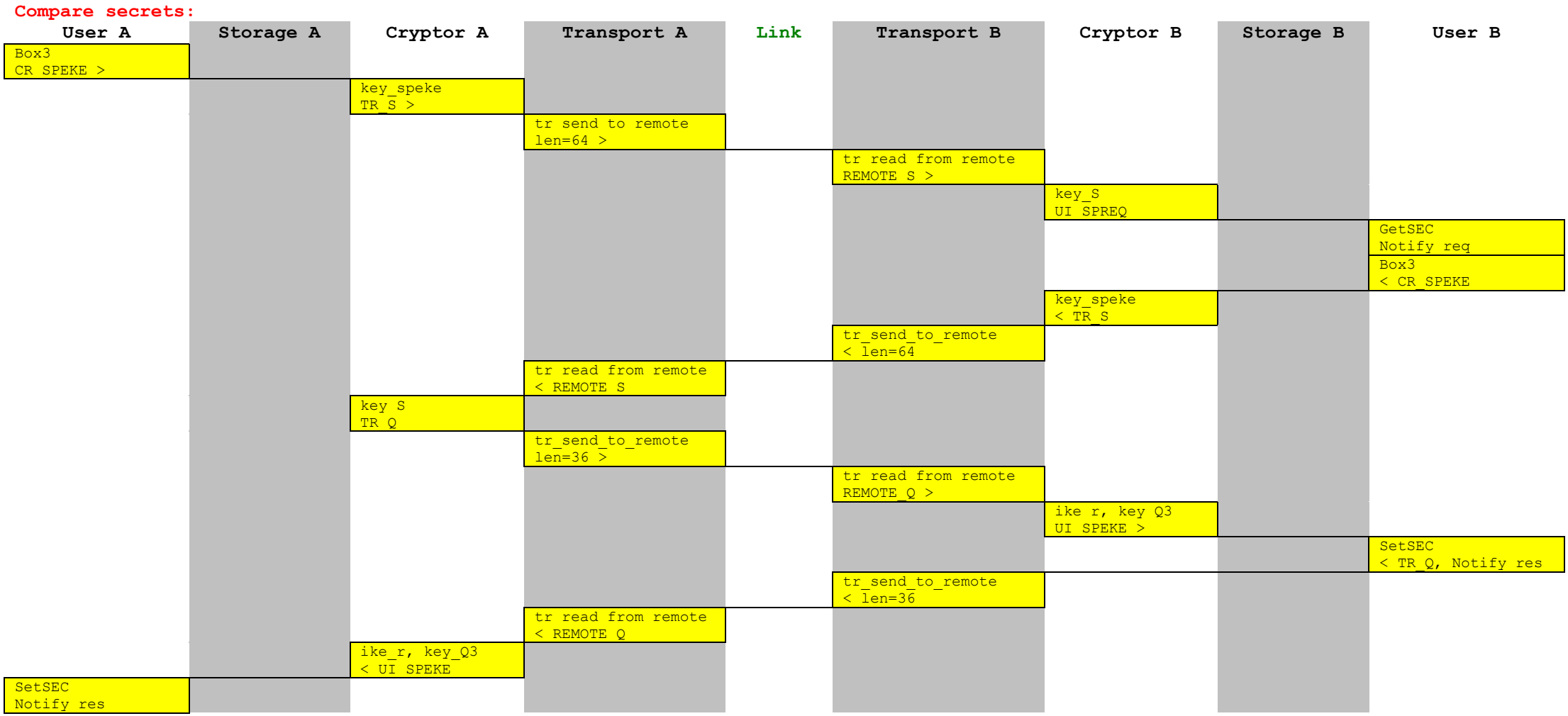
In this session all transcript protected by secret shared in first session (looks like pseudorandom).
Parties provides session key exchange with commitment, originator send his ID (public key) in protected way provides mutual authentication by public keys and proceed to talk mode

Answer (call from A to B):

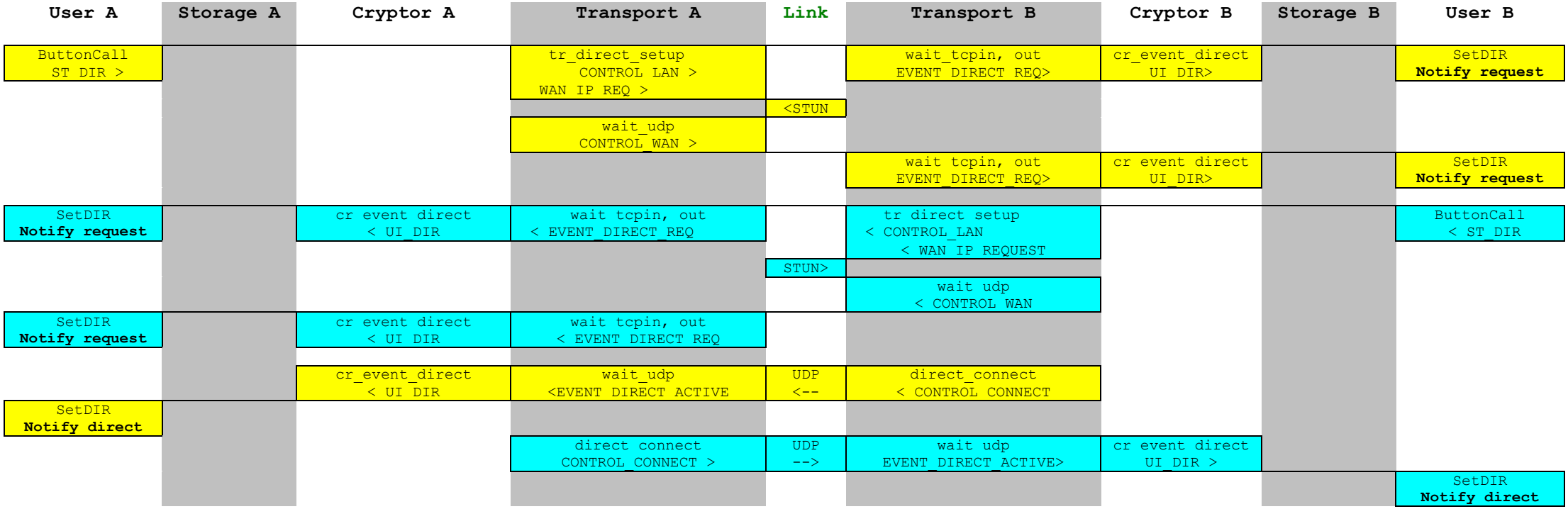
User A	Storage A	Cryptor A	Transport A	Link	Transport B	Cryptor B	Storage B	User B
ud_setans		< UI_ANSWER cr_ans		< REMOTE_A		cr_ui_mute < TR A		ButtonCallClick **< CR_MUTE**

Send key:

User A	Storage A	Cryptor A	Transport A	Link	Transport B	Cryptor B	Storage B	User B
ButtonSelect **ST_GETADR >**								
	st get addr by name **< UI_ADR**							
SetADR CR_REQK >								
		cr_set_key **< UI_REQ**						
SetKEY ST_GETKEY >								
	st search by name CR_KEY >							
		cr_st_B, key_run TR_K (Ta) >						
			tr_send_to_remote len=52 >					
					tr read from remote REMOTE_K >			
						key_K < TR Q (Tr)		
					tr_send_to_remote < len=36			
			tr read from remote < REMOTE_Q					
		ike r, key Q1 < TR Q (Tk)						
			tr_send_to_remote < len=36					
					tr_read_from_remote REMOTE_Q >			
						ike r, key Q2 ST_ADR >		
							st_set_address UI_ADROK >	
								AddADR **< CR_ADDK**
						cr add key ST_ADD >		
							st_save_key UI_KEY >	
								GetKEY


Compare secrets:
User A
Storage A
Cryptor A
Transport A
Link
Transport B
Cryptor B
Storage B
User B
Box3
CR_SPEKE >
key_speke
TR_S >
tr send to remote
len=64 >
tr read from remote
REMOTE_S >
key_S
UI_SPREQ
GetSEC
Notify req
Box3
< CR_SPEKE
key_speke
< TR_S
tr_send_to_remote
< len=64
tr read from remote
< REMOTE_S
key S
TR_Q
tr_send_to_remote
len=36 >
tr read from remote
REMOTE_Q >
ike r, key Q3
UI_SPEKE >
SetSEC
< TR_Q, Notify res
tr_send_to_remote
< len=36
tr read from remote
< REMOTE_Q
ike_r, key_Q3
< UI_SPEKE
SetSEC
Notify res

Nat traversal (set UDP direct connection using Tor connection):



After call is established parties can try to connect directly using UDP with NAT traversal. For this both parties must allow direct connecting then obtain his local IP:port and send request to specified STUN server for obtain his external IP:port. After values will be known parties sends each other control packet over TCP with his internal/external IP:ports. After packets was received parties periodically sends each other control packets over UDP to specified IP:ports. After at least one party receive valid packet over UDP hi will send data back to IP:port of sender. Now UDP connecting is active and parties sends all data over UDP instead TCP/TOR (except ping packets). For returns back to TCP connecting at least one party send deny packet over TCP.