Designing a Secure Device-to-Device File Transfer Protocol

By Chaitanya Rahalkar & Anushka Virgaonkar

Introduction

- Secure, reliable and fast transfer of files across the Internet is a problem attempted to be solved through many application layer protocols. None of them have managed to do it ideally!

- Requirements for an ideal file transfer mechanism -
  a. Secure
  b. Performance
  c. Private (involves no-third party, or putting trust in a third-party)
  d. Transparency (Open protocol and open source)

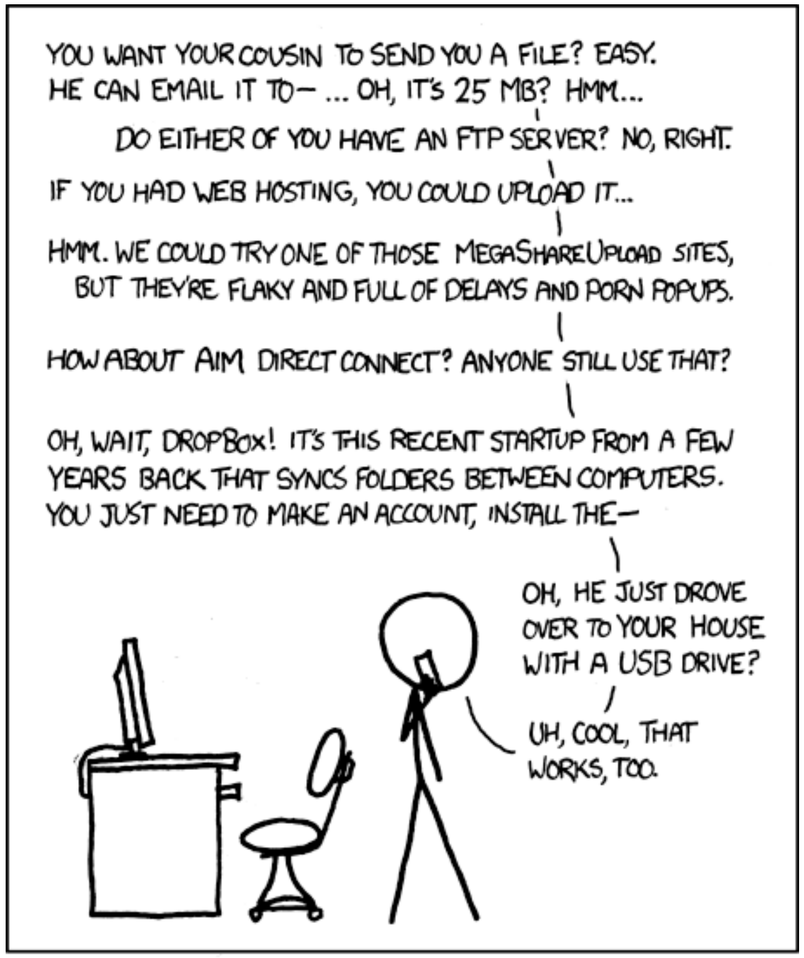
Introduction, con’t

Goals of this project

- Empirical analysis of security
- Comparative analysis with existing options (performance + security + privacy). Studying trade-offs between performance and security + privacy.

Motivation

There’s no keep it simple, stupid and secure way!



I LIKE HOW WE'VE HAD THE INTERNET FOR DECADES, YET "SENDING FILES" IS SOMETHING EARLY ADOPTERS ARE STILL FIGURING OUT HOW TO DO.

Motivation, con’t

- Decade old protocols still in existence! [FTP (File Transfer Protocol) just celebrated it’s 50th birthday!]
- Securely designed protocols like SCP (Copy over SSH) require port forwarding if parties are behind a NAT. No regulated access control.
- Cloud-based file transfer have storage + transfer limits. Inherent trust is placed in closed-source programs. Two stage tedious process - upload + download

Approach, con’t

Components

- Relay Server
- Password Authenticated Key Exchange
- IP Exchange Scheme
- Device Clients

Approach, con’t

Relay Server

- The two-computer file transfer approach relies on a relay server to relay the data packets from the sender to the receiver. As the name suggests, data is simply relayed, **not stored on the server**.

- Uploading and downloading can be done simultaneously due to full duplex communication.

- Relaying is **much faster** than uploading.

- Bandwidth charges for cloud-hosted relay servers are high! Data centres charge for ingress and egress bandwidth consumed.

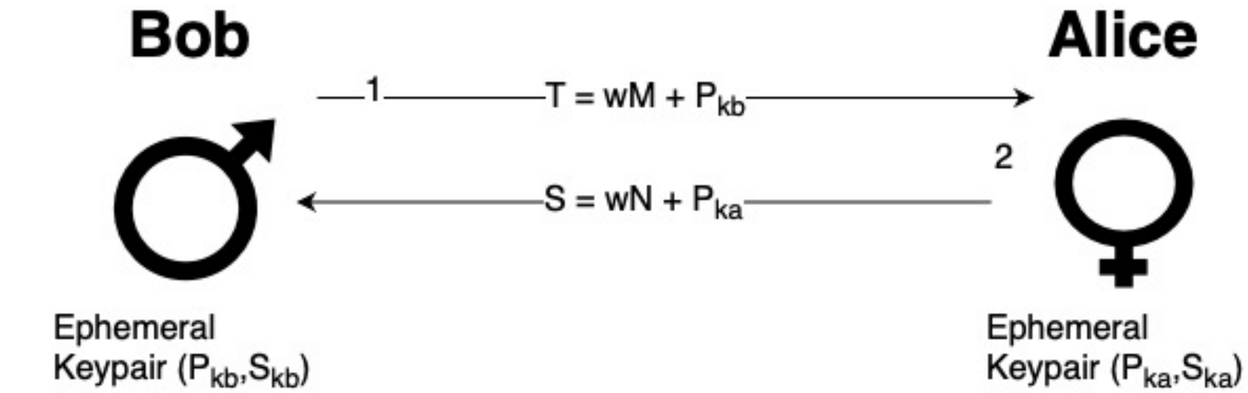
Approach, con’t

Password Authenticated Key Exchange

- Interactive method for two parties (can be more than two) to establish cryptographic keys based on one or more party’s knowledge of a password.
- Shared session key is established from the secret passphrase. This session key is used to further encrypt communication between the two parties.
- Without interaction with parties, eavesdropper cannot have enough information to brute-force!
- Strong security even from weak passwords. Security does not depend on the strength of the password.
- Handled by another program running on the same relay server.

Approach, con’t

sPAKE2 (symmetric PAKE)



$w = H(\text{weak-passphrase})$

3

$K = h * S_{kb} * (S - wM)$

3

$K = h * S_{ka} * (T - wN)$

4

$\text{Shared Key} = H(ID_{Alice} \,||\, ID_{Bob} \,||\, S \,||\, T \,||\, K \,||\, w)$

Further key confirmation step not included

Approach, con’t

IP Exchange Scheme

- In order to establish a communication channel, the relay server must know the public IP addresses of both the parties.
- One or both of the parties can be behind a NAT router.

Approach, con’t

Device Client

- Use TCP sockets for reliable transport
- sPAKE2-based key authentication
- Works even if the client is behind a NAT (not heavily restricted NATs or hardened firewalls)

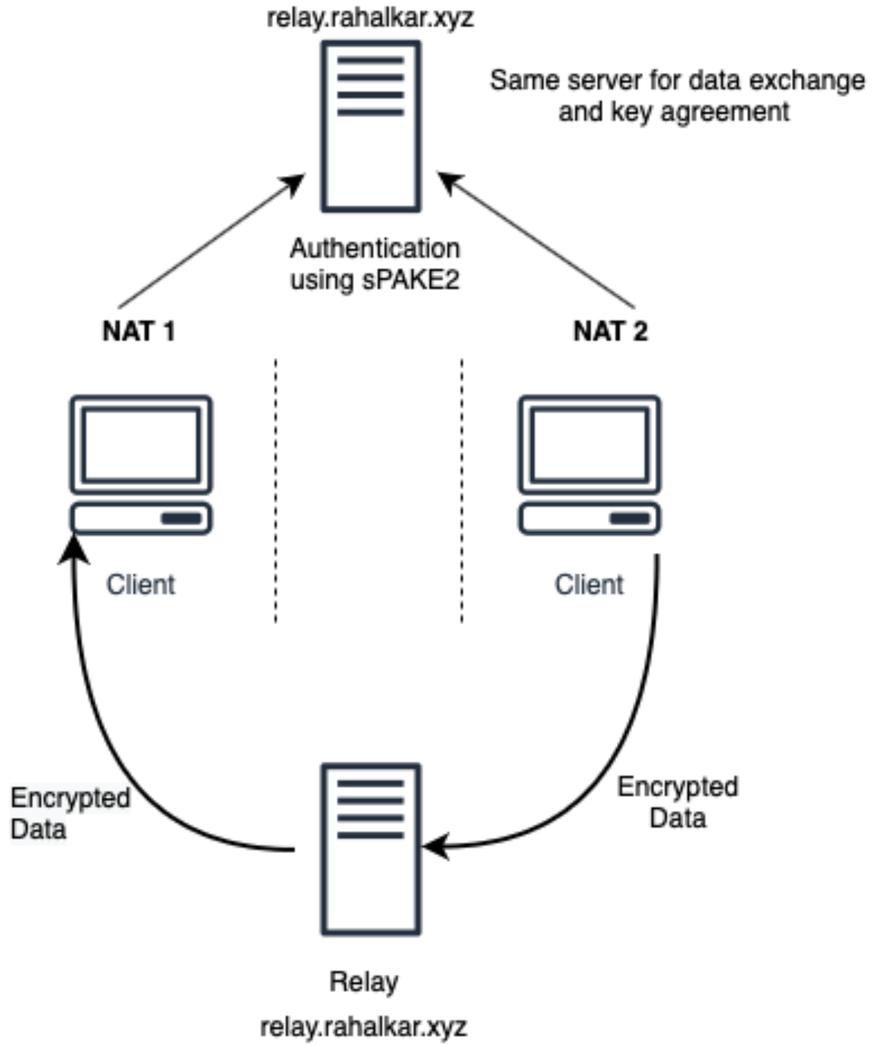
Approach, con’t

Putting it all together

- Both the clients exchange messages to establish secure shared key using sPAKE2 based on the generated passphrase.
- Once the secure channel is established, data is encrypted (Secretbox Go Library - NaCl crypto) and sent in chunks of 16384 bytes.
- Receiver gets a confirmation once the correct passphrase is provided.
- Data is relayed and transferred to the receiver. MAC is used to verify integrity + authenticity on the receiver end.

Approach, con’t

Putting it all together



Approach, con’t

Transfer in Action!

```
chaitanya@linux in ~/benchmarks via  v3.9.2 took 2ms
λ ~/fsend send 1MB.bin
On the other computer, please run: fsend receive (or fsend recv)
Secret code is: 2-enrollment-python
1.05 MB / 1.05 MB [---------------------------------------] 100.00% 538.13 kB p/s 2.1s
file sent
```

Sender

```
recon@ubuntu-s-1vcpu-1gb-blr1-01:~$ ./fsend recv
Enter receive secret code: 2-enrollment-python
Receiving file (1.0 MB) into: 1MB.bin
ok? (y/N):y
1.05 MB / 1.05 MB [---------------------------------------] 100.00% 104.16 kB p/s 10s
recon@ubuntu-s-1vcpu-1gb-blr1-01:~$
```

Receiver

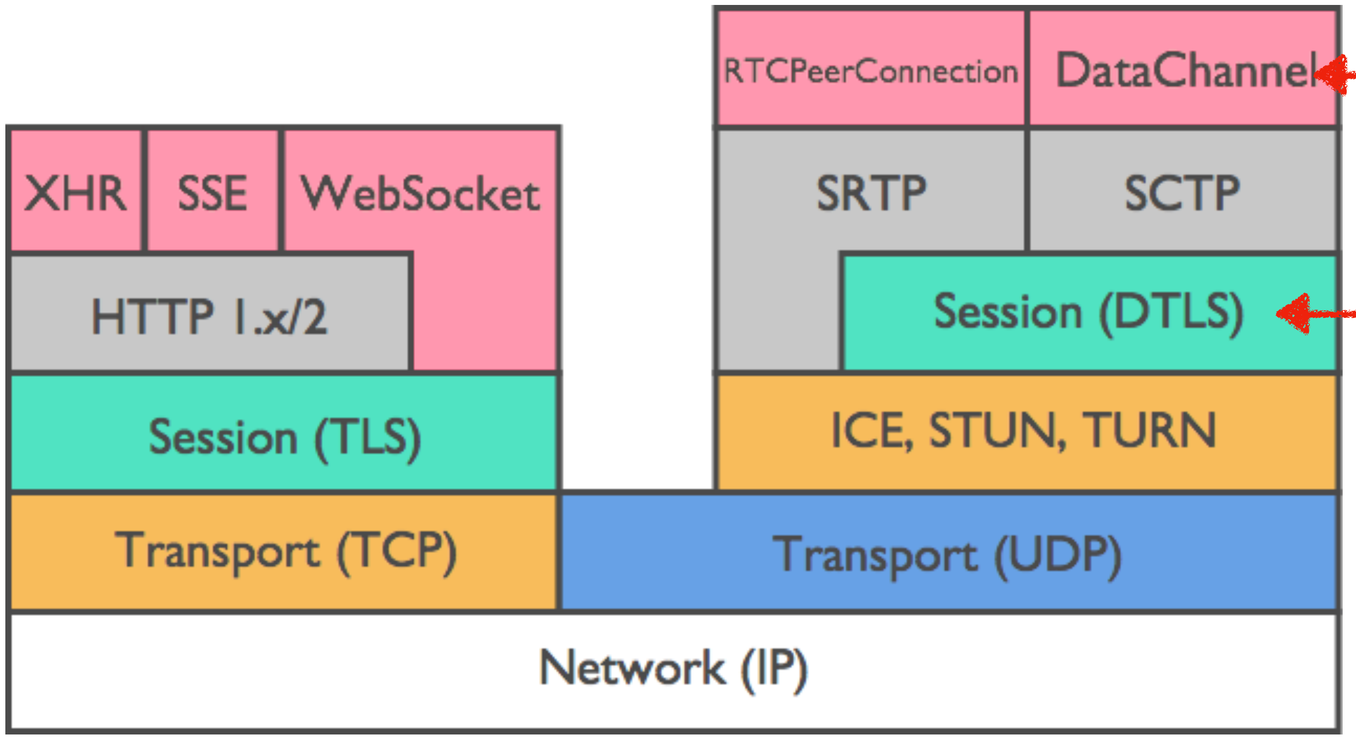
Approach, con’t

When there’s no relay!
WebRTC Data Channels

- WebRTC offers peer-to-peer and real-time communication. (Signaling server for metadata exchange)
- Traditionally used for VoIP and media communication (audio + video) (BlueJeans / Kaltura is using WebRTC to stream this!)
- WebRTC has the *RTCDataChannel* API. It can be used to transfer arbitrary data between two parties.
- Comes with baked-in security!

Approach, con’t

When there’s no relay!
WebRTC Data Channels



WebRTC Stack
Source: webrtc-security.github.io

Approach, con’t

Sender SDP

Receiver SDP

```
chaitanya@linux in ~/benchmarks via v3.9.2 took 2ms
λ ./webrtc send 1MB.bin
-- Please send this message to the remote party --
{"sdp": "v=0\r\no=- 3827798933 3827798933 IN IP4 0.0.0.0\r\ns=-\r\nt=0 0\r\na=group:BUNDLE 0\r\na=msid-semantic:WMS *\r\nm=application 40146 DTLS/SCTP 5000\r\nc=IN IP4 192.168.1.7\r\na=mid:0\r\na=sctpmap:5000 webrtc-datachannel 65535\r\na=max-message-size:65536\r\na=candidate:d3528481068ed87c99af586f1926906f 1 udp 2130706431 192.168.1.7 40146 typ host\r\na=candidate:4730076f24ccad9e63cf512158753073 1 udp 1694498815 117.217.44.94 40146 typ srflx raddr 192.168.1.7 rport 40146\r\na=end-of-candidates\r\na=ice-ufrag:m66H\r\na=ice-pwd:S169Z4gRfghMDvoijZjhuM\r\na=fingerprint:sha-256 BF:CE:BB:81:FC:87:36:7F:2C:2C:69:8E:EE:4D:76:3C:96:F4:9A:E9:2E:27:72:73:7D:05:73:E8:08:8A:5F:03\r\na=setup:actpass\r\n", "type": "offer"}

-- Please enter a message from remote party --
{"sdp": "v=0\r\no=- 3827798965 3827798965 IN IP4 0.0.0.0\r\ns=-\r\nt=0 0\r\na=group:BUNDLE 0\r\na=msid-semantic:WMS *\r\nm=application 55790 DTLS/SCTP 5000\r\nc=IN IP4 167.71.227.140\r\na=mid:0\r\na=sctpmap:5000 webrtc-datachannel 65535\r\na=max-message-size:65536\r\na=candidate:a6b4587524eba60a01fad27e18b9af7d 1 udp 2130706431 167.71.227.140 55790 typ host\r\na=candidate:5a92684a1645efe73f013f2702f453a2 1 udp 2130706431 10.47.0.5 36165 typ host\r\na=candidate:175307e69deee0a9c7e5325223080e35 1 udp 2130706431 10.122.0.2 58803 typ host\r\na=candidate:580b5f32035da7f14a30ea7a8d826c67 1 udp 2130706431 172.17.0.1 43565 typ host\r\na=candidate:be678389bed9f023c266ce57a3305e63 1 udp 1694498815 64.225.87.196 36165 typ srflx raddr 10.47.0.5 rport 36165\r\na=candidate:4258052a13b002fef1e69a749de097f5 1 udp 1694498815 167.71.227.140 55790 typ srflx raddr 167.71.227.140 rport 55790\r\na=end-of-candidates\r\na=ice-ufrag:D9D2\r\na=ice-pwd:HxbrJ5oil5qvirjg9iTDVU\r\na=fingerprint:sha-256 87:7E:1E:50:FE:6B:D6:BB:6D:46:CC:53:A0:CD:A3:46:06:5C:01:10:A3:FD:11:A7:00:E6:57:D5:AC:78:D3:D5\r\na=setup:active\r\n", "type": "answer"}

-- Please enter a message from remote party --
{"type": "bye"}

Exiting
-- Please send this message to the remote party --
{"type": "bye"}

chaitanya@linux in ~/benchmarks via v3.9.2 took 50s
λ
```

```
recon@ubuntu-s-1vcpu-1gb-blr1-01:~$ ./webrtc receive 1MB.bin
-- Please enter a message from remote party --
{"sdp": "v=0\r\no=- 3827798933 3827798933 IN IP4 0.0.0.0\r\ns=-\r\nt=0 0\r\na=group:BUNDLE 0\r\na=msid-semantic:WMS *\r\nm=application 40146 DTLS/SCTP 5000\r\nc=IN IP4 192.168.1.7\r\na=mid:0\r\na=sctpmap:5000 webrtc-datachannel 65535\r\na=max-message-size:65536\r\na=candidate:d3528481068ed87c99af586f1926906f 1 udp 2130706431 192.168.1.7 40146 typ host\r\na=candidate:4730076f24ccad9e63cf512158753073 1 udp 1694498815 117.217.44.94 40146 typ srflx raddr 192.168.1.7 rport 40146\r\na=end-of-candidates\r\na=ice-ufrag:m66H\r\na=ice-pwd:S169Z4gRfghMDvoijZjhuM\r\na=fingerprint:sha-256 BF:CE:BB:81:FC:87:36:7F:2C:2C:69:8E:EE:4D:76:3C:96:F4:9A:E9:2E:27:72:73:7D:05:73:E8:08:8A:5F:03\r\na=setup:actpass\r\n", "type": "offer"}

-- Please send this message to the remote party --
{"sdp": "v=0\r\no=- 3827798965 3827798965 IN IP4 0.0.0.0\r\ns=-\r\nt=0 0\r\na=group:BUNDLE 0\r\na=msid-semantic:WMS *\r\nm=application 55790 DTLS/SCTP 5000\r\nc=IN IP4 167.71.227.140\r\na=mid:0\r\na=sctpmap:5000 webrtc-datachannel 65535\r\na=max-message-size:65536\r\na=candidate:a6b4587524eba60a01fad27e18b9af7d 1 udp 2130706431 167.71.227.140 55790 typ host\r\na=candidate:5a92684a1645efe73f013f2702f453a2 1 udp 2130706431 10.47.0.5 36165 typ host\r\na=candidate:175307e69deee0a9c7e5325223080e35 1 udp 2130706431 10.122.0.2 58803 typ host\r\na=candidate:580b5f32035da7f14a30ea7a8d826c67 1 udp 2130706431 172.17.0.1 43565 typ host\r\na=candidate:be678389bed9f023c266ce57a3305e63 1 udp 1694498815 64.225.87.196 36165 typ srflx raddr 10.47.0.5 rport 36165\r\na=candidate:4258052a13b002fef1e69a749de097f5 1 udp 1694498815 167.71.227.140 55790 typ srflx raddr 167.71.227.140 rport 55790\r\na=end-of-candidates\r\na=ice-ufrag:D9D2\r\na=ice-pwd:HxbrJ5oil5qvirjg9iTDVU\r\na=fingerprint:sha-256 87:7E:1E:50:FE:6B:D6:BB:6D:46:CC:53:A0:CD:A3:46:06:5C:01:10:A3:FD:11:A7:00:E6:57:D5:AC:78:D3:D5\r\na=setup:active\r\n", "type": "answer"}

-- Please enter a message from remote party --
received 1048576 bytes in 0.5 s (16.946 Mbps)
-- Please send this message to the remote party --
{"type": "bye"}

{"type": "bye"}

Exiting
-- Please send this message to the remote party --
{"type": "bye"}

recon@ubuntu-s-1vcpu-1gb-blr1-01:~$
```

Results

	Security	End-to-End Encryption	Source	Intermediary Involved	Eavesdroppers
Google Drive	Account-based (other details unknown)	No. Keys held by Google	Closed source	Yes (Google Servers)	Google, CAs
WebRTC	DTLS	Yes	Open protocol design. FOSS libs.	No (STUN server to get public IP)	None. Peer-to-peer design
Our implementation	sPAKE, NaCl Crypto	Yes	Open protocol design	Yes (Relay server)	None. Property of PAKE
FTP	Password-based access to server.	No	Open protocol design FOSS libs.	Yes (FTP server)	FTP server host
Email	TLS	Only in PGP / S-MIME	Open protocol design	Yes (Mail server)	Mail servers (None in PGP/S-MIME)
SCP	AES, RSA / ECC, HMAC	Yes	Open protocol design FOSS libs.	No	None

Declared outdated by OpenSSH

Results, con’t

- For all intermediary involved tests, a DigitalOcean VPS at a data centre in US (Portland, Oregon) [12,714 km from client] was used.
- Benchmarks were taken for binary files (random bytes) of size 1MB, 100MB, 512MB and 1GB.
- For all peer-to-peer tests, one of the peer was the DigitalOcean VPS.
- The relay was hosted at a Google Cloud VPS at a data centre in Council Bluffs, Iowa [13275 km from one of the client] was used.
- Tests for all implementations were conducted in a similar environment.

Results, con’t

	1MB	100MB	512MB	1GB
SSH	3.28s	49.69s	4m 29s	9m 13s
Our Implementation	10s (receiving)	1m 39s (receiving)	6m 12s (receiving)	10m 28s (Receiving)
WebRTC	1.80s	41.32s	4m 13s	7m 2s
FTP	0.13s	27s	3m 26s	8m 16s

Conclusion & Observations

- Even though relay transfer is not as performant as direct transfer (WebRTC), it works in almost all cases.

- Sometimes, WebRTC has problems sending big files. (failure observed on file.pizza and our implementation using aiortc for files bigger than 1GB). WebRTC Data Channels are not recommended for large files. (Firefox caps limit at 1GiB).

- WebRTC data channel packet size is capped at 16KB. Fragmentation required for larger files. Porting from Javascript to other languages is complex.

- Trade-offs between performance and security + privacy are considerable.