Forensic Analysis

ZFONE

Detica Forensics

Document Reference:	DFA028.17.04.D004
Document Status:	Approved
Document Version:	1.00
Issue Date:	23 April 2008
Deliverability:	See section 1
Prepared by:	I H G Livingstone
Approved by:	A J Clark

Table of Contents

1 Introduction

1.1 Purpose

This document outlines the results of an analysis of Zfone-encrypted network traffic and a forensic analysis of a PC that has used the Zfone software to perform some VoIP calls.

1.2 Scope

This document is applicable to the current release version of Zfone (0.7, build 134) running under Windows XP. Zfone continues to evolve and be updated beyond this version. The analysis was performed by treating Zfone as a "black box"[1]. We only modified the source code to facilitate the checking of the cryptographic key generation and the remainder of the analysis was performed with no reference to the source code.

1.3 Amendment History

Issue	Date	Author(s)	Revisions
0.10	8 Apr 2008	IL	First Draft for Internal Review
0.90	23 Apr 2008	IL	Issued for review and Approval
1.00	23 Apr 2008	IL	Approved

1.4 References

Mnemonic	Document Details
[ZRTP_SPEC]	Title: ZRTP: Media Path Key Agreement for Secure RTP Doc Ref: draft-zimmermann-avt-zrtp-04 Version: n/a Date: July 9th 2007

1.5 Copyright Statement

[1] Black box in this sense refers to treating the Zfone product as a complete entity with no knowledge of how it works internally.

1.6 Disclaimer

1.7 Deliverability

Copy	Revisions
Original	Maintained in softcopy only
1	Philip Zimmermann

2 Executive summary

Zfone is a software product, available free to download on http://www.zfoneproject.com, which allows two people to make secure, encrypted, phone calls over the internet. It claims to ‘let you whisper in someone’s ear from a thousand miles away’.

Following discussions with the principal designer of Zfone Phil Zimmermann, and with his encouragement, we undertook a basic “black box” analysis of Zfone to see if it functioned in accordance with its description. We also checked to see if there was any obvious leakage of critical data that might compromise its security.

We carried out an analysis of the network traffic resulting from a VoIP call that had been encrypted using the Zfone software at either end.

We did this as a confidence test that the payload data had been correctly encrypted according to the specification and that no erroneous information was present that would allow a potential attacker a means of decrypting the data.

We independently performed the same cryptographic calculations that Zfone performed to generate key material and found them to match.

We did not detect any key material accidentally leaked during the call.

We also carried out a forensic residue analysis on Zfone, to check that it did not leave any artefacts on a machine upon which it had been running that could compromise the privacy of previous conversations.

Although we noted some sensitive values (previous shared secrets) were stored on the machine in an unencrypted format these were not sufficient to be able to decrypt a previously recorded encrypted conversation on their own.

The use of the log file to store debugging information needs to be closely monitored as it is possible to find key values stored in the log if a debug build has been used. We recommend that a debug version not be used in a production environment.

We did not perform a study of the Zfone source code to check for any potential security vulnerabilities in its implementation.

Our testing of Zfone has not been exhaustive, but has been designed to provide a measure of confidence that it operates in accordance with its published functionality.

3 Zfone

3.1 Introduction

Zfone is a software product, available free to download on http://www.zfoneproject.com, which allows two people to make secure, encrypted, phone calls over the internet. It claims to 'let you whisper in someone's ear from a thousand miles away'.

The software works in conjunction with Voice over Internet Protocol (VoIP) 'soft' phones by adding a layer of security around the standard soft phone's communications. It is described as a 'bump in the cord'.

Zfone is the reference implementation of the ZRTP key exchange protocol, as fully described in the proposed internet draft [ZRTP_SPEC].

ZRTP uses a Diffie-Hellman key exchange to agree on a session key and parameters for establishing Secure Real-time Transport Protocol (SRTP) sessions.

ZRTP does not assume any use of a Public Key Infrastructure (PKI) and so there is no reliance on certificates or central key management.

Zfone uses standard cryptographic algorithms, such as AES and RSA and so does not rely on any proprietary algorithms that have not undergone rigorous testing.

We used version 0.7 build 134 running under Windows XP for the work described in this report.

3.2 ZRTP

ZRTP provides an authentication mechanism, confidentiality and integrity between two parties communicating over VoIP.

It defines a mechanism to negotiate a key exchange that then allows the VoIP payload to be encrypted (using AES) and hashed (using SHA-1) with SRTP.

The basic outline of a ZRTP enabled call is shown in Figure 1.

Each installation of Zfone has a unique identifier known as a ZID, which is sent as part of the HELLO message. Once a HELLO has been received by the other participating party it can optionally return a HELLOACK message. Next, either party initiates the key exchange via a COMMIT message.

The payload of the COMMIT message contains the sender's ZID, a sequence of values indicating the versions of cryptographic and hashing algorithms that the sender supports and a Diffie-Hellman public value PVI). The party that sends the first COMMIT message is known as the Initiator (note that this does not necessarily have to be the party that initiated the call).

The party receiving the COMMIT message, known as the Responder, then sends a DHPART1 message, containing their Diffie-Hellman public value (PVR) and the hashes of any shared secrets that they have stored. The Initiator then sends a DHPART2 message containing their public Diffie-Hellman public value (PVI) and the hashes of any shared secrets that they have stored.

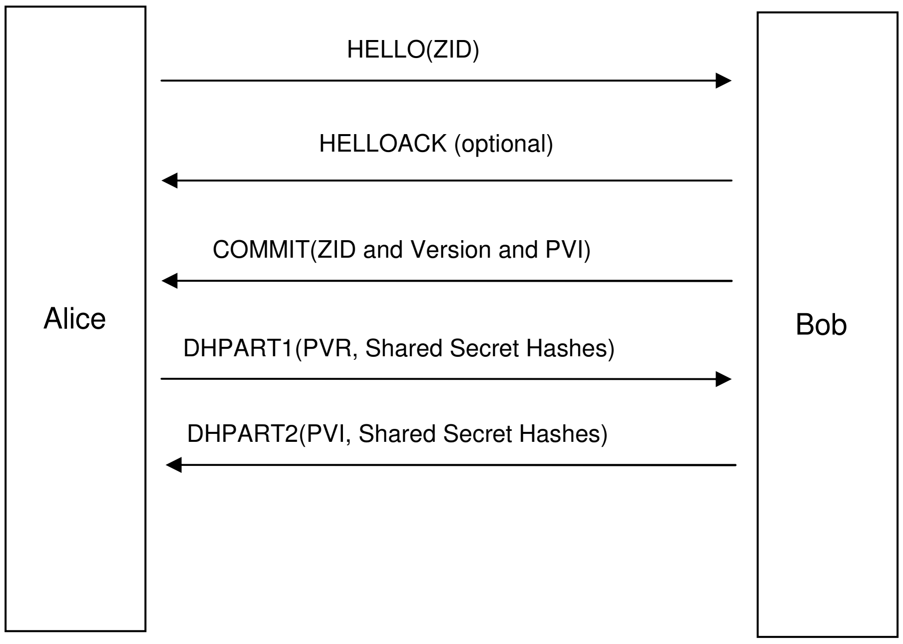
Alice
Bob
HELLO(ZID)
HELLOACK (optional)
COMMIT(ZID and Version and PVI)
DHPART1(PVR, Shared Secret Hashes)
DHPART2(PVI, Shared Secret Hashes)

Figure 1 ZRTP Call Structure

At this point, both parties can generate the SRTP session key that will then be used to encrypt all subsequent traffic.

Zfone also includes a verification mechanism, known as the Short Authentication String (SAS). This value is calculated from PVI and PVR and translates into a pair of words that both parties can say to each other.

The use of the shared secrets means that each call between the same parties is protected not only by the currently generated secret but by previous ones as well.

The shared secrets though, need to be stored at each end point.

3.3 Eavesdropping ZRTP

In order for somebody to successfully eavesdrop on a ZRTP protected call they need to know the following:

- The current shared secret;
- The retained shared secrets.

This allows the eavesdropper to recreate the SRTP session key and, therefore, decrypt the SRTP payload.

If the potential eavesdropper is not interested in decrypting the session in real time but, instead, relies on recording the encrypted conversation and then obtaining access to either parties machine at some point in the future then the ability to successfully obtain these shared secrets is essential.

4 Test environment

4.1 Hardware

In order to perform an analysis of the Zfone network traffic, we setup a representative test system. This involved two separate instances of Zfone end points (two separate workstations on the network with a soft phone and Zfone installed on them) and an internal SIP proxy server to handle the initiation of the call.

In addition, we monitored the network traffic using the Wireshark protocol analyzer.

This setup is shown in Figure 1.

Workstation 1 and Workstation 2 (Alice and Bob in standard cryptographic parlance) have Zfone (and an appropriate soft phone) installed on them. Workstation 3 (Eve) is used to monitor the traffic between Alice and Bob; in particular, the encrypted traffic once a secure session has been established.

For the test configuration, we also placed the SIP server on Workstation 3.

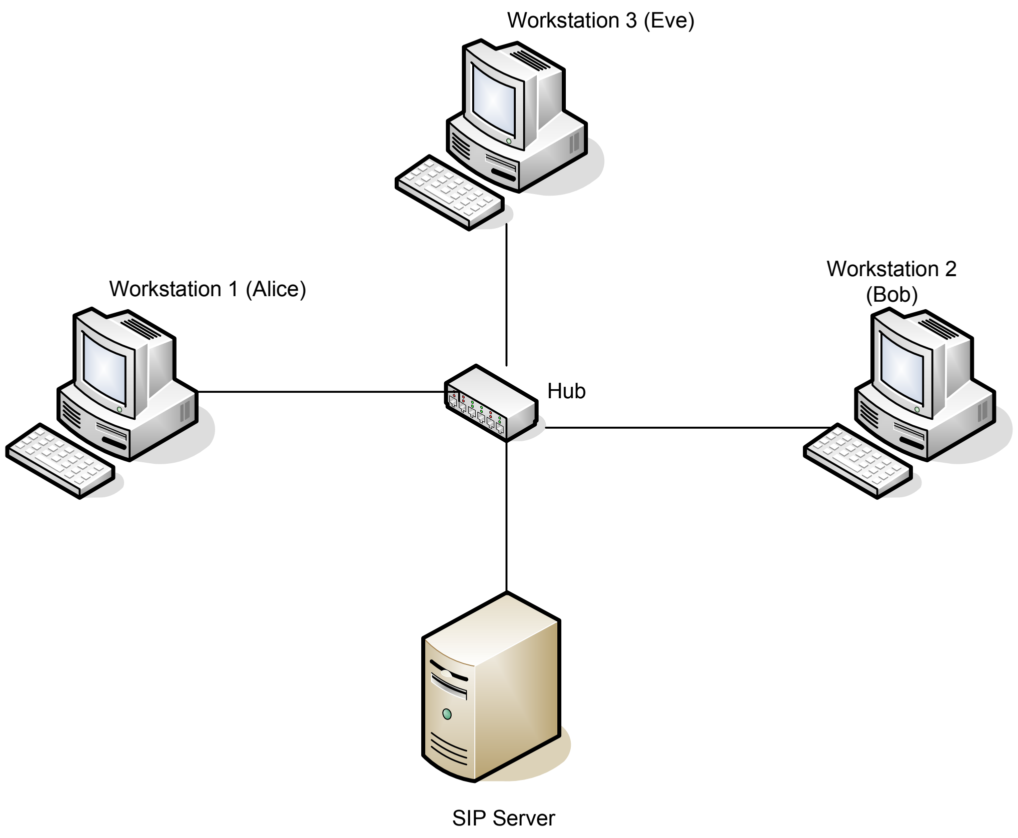

Figure 2

Once we installed Zfone, we made a note of both ZIDs. These are shown in Table 1.

Workstation	ZID (Hex-encoded)
Alice	`C1AC9564E0BE8544CF49D62D`
Bob	`3F166E17288FE81BF3A743E0`

Table 1 Workstation ZIDs

4.2 Software

4.2.1 Softphones

We used XLite (version 3.0), available free from www.counterpath.com, as the softphone for the tests.

4.2.2 Zfone

We used Version 0.7 build 134 of Zfone.

4.2.3 Wireshark

We used The Wireshark network protocol analyser (version 0.99.6a) to analyse the raw packets transferred between the two Zfone endpoints. A modified version that interprets the ZRTP packets was used.

4.2.4 Cain and Abel

We used the Cain and Abel program to monitor the network traffic between the two Zfone endpoints. Cain and Abel is a useful tool for detecting VoIP calls and automatically recording them in a format that can be played back (as .wav files).

5 Network traffic analysis

5.1 Unencrypted call

First of all, we placed an unencrypted call between Alice and Bob and monitored the network traffic using Wireshark and Cain and Abel.

Cain and Abel automatically detected the call and saved a recording of the call in a .wav file.

In Wireshark we output the raw packet data and then used a third party conversion package to translate the GSM encoded voice data into a .wav file that could be played back.

Both methods were relatively straightforward and highlight how easy it is to capture VoIP calls when they are sent in the clear.

5.2 Encrypted call

We then placed a Zfone encrypted call between Alice and Bob and monitored the network traffic using Wireshark and Cain.

Cain was unable to capture the data in a playable .wav file, as would be expected.

From within Wireshark, the RTP packet payload during the encrypted session could be easily observed to no longer follow the expected format of unencrypted GSM encoded data.

To fully check that the ZRTP specification was being followed, we built a debug version of Zfone from the provided source code that output information concerning the cryptographic exchange to the log file.

Full listings of the packet data we captured with Wireshark are provided in Appendix A.

For the analysis considered here, the call was placed from Alice to Bob. After the initial SIP exchange, both parties start exchanging standard RTP packets (unencrypted) and ZRTP HELLO packets.

The ZRTP key exchange is initialised by Bob through the sending of a COMMIT packet. Note that either party can send this and that some HELLOACK packets may have been sent in-between.

Within this COMMIT packet, Bob sends details of the cryptographic and hashing algorithms that he will be using. In addition, he sends a hash of his DHPART2 message (which he has not yet sent and includes PVI, the public Diffie-Hellman value that he calculated) and Alice's Hello message.

Alice then sends a DHPART1 message that contains her public Diffie-Hellman value.

Following that, Bob responds with the DHPART2 message containing his public Diffie-Hellman value.

We independently performed the cryptographic calculations, starting with the random numbers that Alice and Bob generated. Details of the calculations are given in Appendix B. We then compared these values with those that were captured in Wireshark and found that they were identical.

We also searched through the Wireshark traces for any evidence that the random number and the derived value S0 that is used as a seed for all further key generation had been mistakenly output, but we could find no trace of them.

We then checked that all of the master keys were generated as defined in the specifications.

6 Forensic residue analysis

6.1 Summary

We performed a forensic analysis of Zfone that consisted of the following steps:

- Create an image of a PC prior to installing Zfone
- Create an image of a PC directly after Zfone was installed
- Create an image of a PC after Zfone was used to make a call

All analysis was performed on a machine running Windows XP.

Once an image has been created its contents can be hashed (using the program MD5Deep). A comparison of the hashes of each of the files within each image indicates which have been changed, which have been created and which (if any) have been deleted.

In addition, we also monitored the registry by taking snapshots at each of the stages. This was done for convenience (the registries could have been recovered in their raw form from the images but it was easier to snapshot them whilst they were loaded).

The soft phone XLite was used to perform the actual calls on both endpoints.

Although the source code for the software was available, we performed the forensic analysis without reference to it.

6.2 Imaging

We used the FTK Imager tool to create raw (dd format) images of the machine's system disk (mounted as C:/). The images could then be later mounted onto another partition and the files within examined.

6.2.1 Differences due to Installation of Zfone

By analysing the pre and post installation images, we gained an understanding of what files Zfone created and/or modified.

The following files were created:

- C:\Documents and Settings\All Users\Application Data\Zfone\zfone.log
- C:\Documents and Settings\All Users\Start Menu\Programs\Zfone\Start Zfone Control Panel.lnk
- C:\Documents and Settings\All Users\Start Menu\Programs\Zfone\Uninstall Zfone.lnk
- C:\Program Files\Zfone\Credits.rtf
- C:\Program Files\Zfone\ReadMe.rtf
- C:\Program Files\Zfone\Uninstall.exe
- C:\Program Files\Zfone\Zfone Beta License.rtf

- C:\Program Files\Zfone\Zfone.exe
- C:\Program Files\Zfone\zrtp.chm
- C:\Program Files\Zfone\zrtp.sys
- C:\Windows\System32\drivers\zrtp.sys

An entry in the Prefetch directory for the install program and for Zfone.exe was also created.

In addition to this we noted the creation of several files in the C:\Windows\INF directory (for the installation we analysed these were oem9.inf, oem9.pnf, oem10.inf and oem10.pnf). Files of the same names were also created in the C:\Windows\LastGood\INF.

Files that were modified were as follows:

- C:\Windows\setupapi.log
- Registry files (see section below for more detail on the registry changes)

Several files in the C:\Windows\system32\wbem\Repository\FS directory were also modified.

6.2.1.1 Registry changes

Registry keys that we observed to be created by the installation of Zfone were as follows:

- HKEY_LOCAL_MACHINE\SOFTWARE\Zfone
  This key had three values: CacheVersion, ID and VersionBuild.
- HKEY_LOCAL_MACHINE\SOFTWARE\Zfone\cache
  This key had two values: count and mitmcount.
- HKEY_LOCAL_MACHINE\SOFTWARE\Zfone\Config
  This key had a single value: Pref.

Several other keys were created related to driver information (as Zfone is built around the zrtp.sys driver). Of interest is the HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\ZRTP key that contains several values, including log_path that points to the Zfone log file.

In addition to creating these keys, the installation also modified several existing keys. These include the key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\RNG that contains the value Seed.

6.2.2 Differences due to making a single call

By analysing the images made directly post installation and directly after a single call, we gained an understanding of what changes Zfone (and the soft phone) made during the call.

We placed a single call from the Zfone installation under examination to another on the same test network.

A new value, called ‘value’, was created under the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Zfone\cache and the value ‘count’ was modified from a binary value of 0x00000000 to 0x01000000.

The ‘cache’ value was a binary blob of 228 bytes.

The ‘Seed’ value in HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\RNG was also updated.

The Zfone log file was updated.

We also monitored the network traffic over which the call data was sent to check if any of the values that were stored in the cache related to those sent over the network.

Once these values were identified, we monitored them over the course of several calls.

6.3 Potential vulnerabilities

6.3.1 Zfone log

We assume that the Zfone log will not be present once the product has been finalised and is only created for debugging purposes.

We noted that when Zfone is built with an extra logging flag then this file contains a lot of information including the calculated shared secret. With this information it is possible to then generate all the keys required to decrypt the calls.

6.3.2 Cache

Zfone stores its shared secrets in a cache contained in the Windows Registry.

We did not know the exact format of the cache, but it is indexed according to the ZIDs of both parties and the stored secret values are stored unencrypted and can be retrieved easily.

This, however, is not in itself enough information for an attacker (who has access to the registry after a Zfone enabled call has been made) to decrypt the recorded call. If the attacker were able to obtain a registry snapshot just prior to and just after the call they would still be missing the random number that is used to generate the DHResult. This random number is seeded from the value stored in the registry, but as this value changes it would be very difficult for an attacker to correctly obtain the one that was used as a seed for the DHResult value.

It could be possible for an attacker to insert themselves between the two parties and exchange new DHResults with either party and then listen in on the conversation, but this would rely on the parties not explicitly verifying their SAS values (which would, of course, be different). Also, either party would generate a different shared secret that the attacker would need to reconcile and then update the registries of the targets (although only one would need to be updated).

6.3.3 Other residue

It is possible that some key material is left behind in, for example, the page file, but this would be very difficult to recognise and recover without specialist tools or applications.

Appendix A Detailed ZRTP packet listings

The packet listings here were taken directly from the modified version of Wireshark that can correctly interpret the ZRTP protocol.

A.1 HELLO sent from Alice to Bob

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 50294
Magic Cookie: ZRTP
Source Identifier: 0xad5541f2
Message
   Signature: 0x505a
   Length: 18
   Type: Hello
   Data
      zrtp protocol version: 0.09
      Client Identifier: winZfone 0.7
      ZID: :c1:ac:95:64:e0:be:85:44:cf:49:d6:2d
      ..0. .... = Initiator: False
      ...0 .... = Passive: False
      Hash type count = 1
         Hash[0]: SHA-256 Hash
      Cipher type count = 1
         Cipher[0]: AES-CM with 128 bit Keys
      Auth tag count = 1
         Auth tag[0]: HMAC-SHA1 32 bit authentication tag
      Key agreement type count = 2
         Key agreement[0]: Preshared non-DH mode using shared secret
         Key agreement[1]: DH mode with p=3072 bit prime
      Sas type count = 2
         Sas type[0]: Short authentication string using base 256
         Sas type[1]: Short authentication string using base 32
```

A.2 HELLO sent from Bob to Alice

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 43215
Magic Cookie: ZRTP
Source Identifier: 0x52aabe0d
```

```
Message
   Signature: 0x505a
   Length: 18
   Type: Hello
   Data
      zrtp protocol version: 0.09
      Client Identifier: winZfone 0.7
      ZID: :3f:16:6e:17:28:8f:e8:1b:f3:a7:43:e0
      ..0. .... = Initiator: False
      ...0 .... = Passive: False
      Hash type count = 1
         Hash[0]: SHA-256 Hash
      Cipher type count = 1
         Cipher[0]: AES-CM with 128 bit Keys
      Auth tag count = 1
         Auth tag[0]: HMAC-SHA1 32 bit authentication tag
      Key agreement type count = 2
         Key agreement[0]: Preshared non-DH mode using shared secret
         Key agreement[1]: DH mode with p=3072 bit prime
      Sas type count = 2
         Sas type[0]: Short authentication string using base 256
         Sas type[1]: Short authentication string using base 32
```

A.3 COMMIT sent from Bob to Alice

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 47153
Magic Cookie: ZRTP
Source Identifier: 0x16898147
Message
   Signature: 0x505a
   Length: 19
   Type: Commit
   Data
      ZID: :3f:16:6e:17:28:8f:e8:1b:f3:a7:43:e0
      Hash: S256
      Cipher: AES1
      AT: HS32
      Key Agreement: DH3k
      SAS: B256
```

```
HVI Data
    27 cd c8 b0 af 7e 71 d1 2c f1 55 a1 be b0   56e5 d4 f7 f1 12 ef
    66 e2 d3 8422 68 ad 76 fc 75  c1 a6 fb 42 b0 61
```

A.4 DHPART1 sent from Alice to Bob

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 65269
Magic Cookie: ZRTP
Source Identifier: 0xe9767eb8
Message
   Signature: 0x505a
   Length: 109
   Type: DHPart1
   Data
      rs1ID: :87:17:ea:2b:82:4e:82:17
      rs2ID: :9e:05:e5:0c:3d:55:a8:f9
      sigsID: :38:45:1b:db:95:ad:e2:c7
      srtpsID: :ea:e3:f7:a0:54:95:c3:71
      other_secretID: :22:06:14:49:c4:56:2c:bc
      pvr/nouncer  Data
934B0ADF886AB166A57BDD6D6FCC14F7CCF185E9844C50EAA084173
1AAA73FBE7075C480CCE6BA0A7DE955D80E73C9944085B9E88F3A8D
CE1C23B0C68586DBD82A655334B33640FC2C306E9C74AF78A273D2A
0C563E91C593B4EAF56516A2164A9CDF330C7F2EE8E1076A6C9FDC9
59969BEEE1AE3BF8EAA535790CA015C087A250706B69DD5F1FFB367
728E2BBAFB990A83A89F1F78A9E399F94F6DA310B37769AA48B5A919
AB179871A1227C4EF23BE5655649C787103F084D7E8421B1EE0AFF6D
DB760CB2846323BA1F7EFCF54ECC2301AE72E0466091C7C6FF4E49C
280A2374EED47EF9FDA6A343B2813DA612E1424307DE0D20E1307FF
D79E5D051BCC1A1022F429F12F0F81938A5B8BC718B3A7D2D4EF79B
7A0B2965A1800C8B2DCB25E87A8F62E669252A6624D9282D6E887A0C
C0D3037B497BB8992470DB39B0C43D59223A76620A0AF5EF75CCA30
F0AADDFF3ABFA67A50B567056FEA4EFB027818F3985C0A4C9390E47
F7BDA6045E6B94F94346A8E0D9B94E2E120322BE5187A00E9A
```

A.5 DHPART2 sent from Bob to Alice

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 47154
Magic Cookie: ZRTP
Source Identifier: 0x16898147
Message
```

```
    Signature: 0x505a
    Length: 109
    Type: DHPart2
    Data
        rs1ID: :ae:8d:c3:d7:eb:e6:7f:11
        rs2ID: :40:44:d7:2e:10:48:84:82
        sigsID: :f4:96:34:de:b0:30:81:17
        srtpsID: :8a:f7:af:eb:93:2f:1f:c2
        other_secretID: :ba:46:bd:05:6e:d8:38:66
        pvr/nouncer  Data
89D48D93FF2276135B4DB7A4BD6A07B6D04228FDBCDBC4C5C97BFF
6A4436D118B9DDF26E21E2AF921B703489CA38FB4A56DB429D183A9
11712836BE87D8D0FD402317060BF1AF806D2EE315635EB0AF295D40
7C4CADE4AA3667C74C161E692E4B2EABA27F3D39AA715B0B3796EB
D73D6A5BF74550C31E0ED19D30BB1FB6E9A25F523E95AEFC7598BA0
17C81D760BA2C1A58828488AD0F0AAC7C588EE2F6F16AAB723E97D6
C14D17FFC02C80BED478EA0046A65EBA0839A7E86B49C78AB31D9A1
3147E8282AE2493FB0CAAF81871520C26BFFA2E6F58DD4FDDE00851
43D4E772D4EFBCEBF20F19B48492E92DE9303B0CAC12D6F33A83CC
DB96C162929AA87E45C0B562765451F8000B41C2475290DA5DD62887
E5DDD99B9C12ABB8F169584D62C181467674384E5A6BD50B51F7050
E237CE322051F8406062995591F3CB4A90F146DB3152766B4B16D0E18
927D275F47C3035BCE39701587C13198966083A0D816285DE8A9CC54
198CA30D40EB2ED88F9314156F07998340EBB2D54D1B32090FA
```

A.6 CONFIRM1 sent from Alice to Bob

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 65270
Magic Cookie: ZRTP
Source Identifier: 0xe9767eb8
Message
    Signature: 0x505a
    Length: 13
    Type: Confirm1
    Data
        HMAC: :55:55:c3:76:91:3c:4d:e6
        CFB: :34:53:9c:76:b9:d5:9f:2d:e9:44:83:20:8e:3c:11:90
        encrypted  Data
            2f db 67 e8 2a f2  b9 d7 25 c5 fd 39 59 da 35 5a
```

A.7 CONFIRM2 sent from Bob to Alice

```
00.. .... = RTP Version: 0
```

```
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 47155
Magic Cookie: ZRTP
Source Identifier: 0x16898147
Message
   Signature: 0x505a
   Length: 13
   Type: Confirm2
   Data
      HMAC: :6b:74:4e:62:25:9d:c3:e2
      CFB: :72:b1:d3:cb:a1:54:d2:d4:5a:aa:cf:47:12:7d:7b:39
      encrypted  Data
           74 ad 1a af 77 4a 60 c5 e2 b0 63 95 80 a5 ca 29
```

A.8 CONF2ACK sent from Alice to Bob

```
00.. .... = RTP Version: 0
..0. .... = RTP padding: False
...1 .... = RTP Extension: True
Sequence: 65271
Magic Cookie: ZRTP
Source Identifier: 0xe9767eb8
Message
   Signature: 0x505a
   Length: 3
   Type: Conf2ACK
```

Appendix B Cryptographic calculations

This appendix details the cryptographic and hashing calculations that take place in the ZRTP protocol. All values are printed here as hex-encoded strings.

B.1 Diffie-Hellman key exchange

For the analysis considered here, the random number (SVI) generated by Alice was:

c39cf79ead488e62723a0ac530fd65c02543472ead91faab0cf0e0e460cef65
2

Using Diffie-Hellman in 3072 bit mode, we can calculate PVI = g^{SVI} mod p, where g = 2 and p is (as given in RFC3526):

FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E0
88A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A43
1B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42
E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1
FE649286651ECE45B3DC2007CB8A163BF0598DA48361C55D39A69163
FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED5290770969
66D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E7
72C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF69558
17183995497CEA956AE515D2261898FA051015728E5A8AAAC42DAD33
170D04507A33A85521ABDF1CBA64ECFB850458DBEF0A8AEA71575D0
60C7DB3970F85A6E1E4C7ABF5AE8CDB0933D71E8C94E04A25619DC
EE3D2261AD2EE6BF12FFA06D98A0864D87602733EC86A64521F2B181
77B200CBBE117577A615D6C770988C0BAD946E208E24FA074E5AB314
3DB5BFCE0FD108E4B82D120A93AD2CAFFFFFFFFFFFFFFFFF

Therefore, we calculated PVI as

934B0ADF886AB166A57BDD6D6FCC14F7CCF185E9844C50EAA084173
1AAA73FBE7075C480CCE6BA0A7DE955D80E73C9944085B9E88F3A8D
CE1C23B0C68586DBD82A655334B33640FC2C306E9C74AF78A273D2A
0C563E91C593B4EAF56516A2164A9CDF330C7F2EE8E1076A6C9FDC9
59969BEEE1AE3BF8EAA535790CA015C087A250706B69DD5F1FFB367
728E2BBAFB990A83A89F1F78A9E399F94F6DA310B37769AA48B5A919
AB179871A1227C4EF23BE5655649C787103F084D7E8421B1EE0AFF6D
DB760CB2846323BA1F7EFCF54ECC2301AE72E0466091C7C6FF4E49C
280A2374EED47EF9FDA6A343B2813DA612E1424307DE0D20E1307FF
D79E5D051BCC1A1022F429F12F0F81938A5B8BC718B3A7D2D4EF79B
7A0B2965A1800C8B2DCB25E87A8F62E669252A6624D9282D6E887A0C
C0D3037B497BB8992470DB39B0C43D59223A76620A0AF5EF75CCA30
F0AADDFF3ABFA67A50B567056FEA4EFB027818F3985C0A4C9390E47
F7BDA6045E6B94F94346A8E0D9B94E2E120322BE5187A00E9A

We can then calculate the Diffie-Hellman result, DHResult = PVI^{SVR} mod p as

E81797A67CA9807B16E0D3CB6B138F2590137882EF2B8E87F634E109
8EC508A77C0BF1AC66FF96298D8A94B331A11B3FD81AB6BEEA10F6D
3AD01C11AC70D0A1283BDAD74CFBBF7CA2375DB36C75ACA4809ED1
CDD66E9B5491B7C1D0F7DFA9A2979F9B83244D03C90DBB8D4F0C3B
C5FA89A8DE3FA47096EC4992295C8EEA6DC63D13818085856B34C86
8A93DE25850C8724E103432991B446D97229D4DC38047247C262642D4
1B1643B2B262E3F24C8734BA870BD3079819BA552E5E246888C467826
3E3CB422B8B6885D5A5552F3A77A4E841118DFC169C55A8F8E4A044
DE4F589B07E94271CD0B23BCEB855AE29307832B5D3E5A0F77512601
66F57E826018C0840BA436F10CD3445212BFA5D884349F323B65EEEF
D7F070C67A9987A94C286DC3A4F3360B72B07525037BC14E40F60B6A
203BAF3705E61BE8B2C6D9A061B7D9EA722067E899452057939AA76E
A66BE22775B3F236F679920B2A51CAE968DC3C79589A0FB7042F0AB3
1ED189EFEB0B671D385DCD3A87669CD1E04E7C0C2D313

The random number generated by Bob, SVR, was:

7941fc0737f69411a942c5bd1f3ab363b7dd9c9390b6d8c9b27048adfac946
a8

PVI can then be calculated as
89D48D93FF2276135B4DB7A4BD6A07B6D04228FDBCDBC4C5C97BFF
6A4436D118B9DDF26E21E2AF921B703489CA38FB4A56DB429D183A9
11712836BE87D8D0FD402317060BF1AF806D2EE315635EB0AF295D40
7C4CADE4AA3667C74C161E692E4B2EABA27F3D39AA715B0B3796EB
D73D6A5BF74550C31E0ED19D30BB1FB6E9A25F523E95AEFC7598BA0
17C81D760BA2C1A58828488AD0F0AAC7C588EE2F6F16AAB723E97D6
C14D17FFC02C80BED478EA0046A65EBA0839A7E86B49C78AB31D9A1
3147E8282AE2493FB0CAAF81871520C26BFFA2E6F58DD4FDDE00851
43D4E772D4EFBCEBF20F19B48492E92DE9303B0CAC12D6F33A83CC
DB96C162929AA87E45C0B562765451F8000B41C2475290DA5DD62887
E5DDD99B9C12ABB8F169584D62C181467674384E5A6BD50B51F7050
E237CE322051F8406062995591F3CB4A90F146DB3152766B4B16D0E18
927D275F47C3035BCE39701587C13198966083A0D816285DE8A9CC54
198CA30D40EB2ED88F9314156F07998340EBB2D54D1B32090FA

Therefore, for Bob, DH Result = PVR^{SVI} mod p is calculated as

E81797A67CA9807B16E0D3CB6B138F2590137882EF2B8E87F634E109
8EC508A77C0BF1AC66FF96298D8A94B331A11B3FD81AB6BEEA10F6D
3AD01C11AC70D0A1283BDAD74CFBBF7CA2375DB36C75ACA4809ED1
CDD66E9B5491B7C1D0F7DFA9A2979F9B83244D03C90DBB8D4F0C3B
C5FA89A8DE3FA47096EC4992295C8EEA6DC63D13818085856B34C86
8A93DE25850C8724E103432991B446D97229D4DC38047247C262642D4
1B1643B2B262E3F24C8734BA870BD3079819BA552E5E246888C467826

3E3CB422B8B6885D5A5552F3A77A4E841118DFC169C55A8F8E4A044
DE4F589B07E94271CD0B23BCEB855AE29307832B5D3E5A0F77512601
66F57E826018C0840BA436F10CD3445212BFA5D884349F323B65EEEF
D7F070C67A9987A94C286DC3A4F3360B72B07525037BC14E40F60B6A
203BAF3705E61BE8B2C6D9A061B7D9EA722067E899452057939AA76E
A66BE22775B3F236F679920B2A51CAE968DC3C79589A0FB7042F0AB3
1ED189EFEB0B671D385DCD3A87669CD1E04E7C0C2D313

As can be seen, this matched the DH Result value as calculated by Alice.

The value HVI is the hash of the initiator's DHPART2 message and the responder's HELLO message. Note that the messages do not include the checksums.

We calculated this value to be (using SHA1)
27CDC8B0AF7E71D12CF155A1BEB0E5D4F7F112EF66E2D3842268AD7
6FC75C1A6

We matched this value with that sent in the COMMIT message (see A.3).

The total hash can then be calculated as the hash of the responder's HELLO, the COMMIT and the DHPART2 message.

We calculated this value to be

15D5E2279B5FC48368FCB0BA32A62FF3979359AFA1566BAB57D990F2
154FAD9D

The value S0 can then be calculated. This is the value that is used as a seed for all the master keys.

The input for S0 consists of the following data items:
Counter = 00000001
DHResult (as given above)
The string "ZRTP-HMAC-KDF" = 5A5254502D484D41432D4B4446
ZIDI (Initiator's ZID) = 3f166e17288fe81bf3a743e0
ZIDR (Responder's ZID) = c1ac9564e0be8544cf49d62d
Total Hash (as given above)
The shared secrets s1 – s5 are prepended by a 32 bit length value. If they are null, then the length will be zero and is part of the hash calculation.

s1 (retained shared secret 1) =
a7619b0412c54eaab8bf6c5fca055b104b51da7d54d3b24cc5a63df7560f8b
95

s2 (retained shared secret 2) =
86fff569a1c9e2eab48af8eaff1758329b7ec5ecbc27832ceb8a3c67e573fb49

s3 (retained shared secret 2) = null

s4 (retained shared secret 2) = null

s5(retained shared secret 2) = null

So input for s0 hash calculation is as follows:

```
00000001
E81797A67CA9807B16E0D3CB6B138F2590137882EF2B8E87F634E109
8EC508A77C0BF1AC66FF96298D8A94B331A11B3FD81AB6BEEA10F6D
3AD01C11AC70D0A1283BDAD74CFBBF7CA2375DB36C75ACA4809ED1
CDD66E9B5491B7C1D0F7DFA9A2979F9B83244D03C90DBB8D4F0C3B
C5FA89A8DE3FA47096EC4992295C8EEA6DC63D13818085856B34C86
8A93DE25850C8724E103432991B446D97229D4DC38047247C262642D4
1B1643B2B262E3F24C8734BA870BD3079819BA552E5E246888C467826
3E3CB422B8B6885D5A5552F3A77A4E841118DFC169C55A8F8E4A044
DE4F589B07E94271CD0B23BCEB855AE29307832B5D3E5A0F77512601
66F57E826018C0840BA436F10CD3445212BFA5D884349F323B65EEEF
D7F070C67A9987A94C286DC3A4F3360B72B07525037BC14E40F60B6A
203BAF3705E61BE8B2C6D9A061B7D9EA722067E899452057939AA76E
A66BE22775B3F236F679920B2A51CAE968DC3C79589A0FB7042F0AB3
1ED189EFEB0B671D385DCD3A87669CD1E04E7C0C2D313
5A5254502D484D41432D4B4446
3f166e17288fe81bf3a743e0
c1ac9564e0be8544cf49d62d
15D5E2279B5FC48368FCB0BA32A62FF3979359AFA1566BAB57D990F2
154FAD9D
00000020
a7619b0412c54eaab8bf6c5fca055b104b51da7d54d3b24cc5a63df7560f8b
95
00000020
86fff569a1c9e2eab48af8eaff1758329b7ec5ecbc27832ceb8a3c67e573fb49
00000000
00000000
00000000
```

Therefore, S0 is

```
F84EA3C314249D934E8CEEBBAEEBA0EE3F95ECC8B1C089B9C1D276
F04E201083
```

We can then calculate the retained shared secret hashes.

rs1IDr = HMAC(rs1, "Responder") =
8717EA2B824E8217077FB89923F45640FB7AD9B8431991A60F69498ED
C500D10

rs2IDr = HMAC(rs2, "Responder") =
9E05E50C3D55A8F91FF3E01187EFFE8E4A6D05DA52A16A7B12B5B50
D83030EE1

These can be matched with the values that were sent in the DHPART1 message (A.4)

rs1IDi = HMAC(rs1, "Initiator") =
AE8DC3D7EBE67F1110482366CC78E6E3E33D2242E98138383DBE79B
2F8845D95

rs2IDi = HMAC(rs2, "Initiator") =
4044D72E10488482BAA5D08EE4E8C56C2A4BCD75819CDD6AF1F1B2
A4D56E83EC

These can be matched with the values that were sent in the DHPART2 message (A.5).

The new retained shared secret, rs1, can then be calculated.

rs1 = HMAC(s0,"retained secret") =
60B74E707CDAFC7A10F73908F06C0373BFF876F7FC24DDA0DDEBB7
FAAA76DD46

This value was matched with that stored locally in the cache at both Bob and Alice (for the Windows version, this cache exists in the Windows Registry).

The keys can then be calculated as follows:

srtpkeyi = HMAC(s0, "Initiator SRTP master key") =
6554AA93B5A2D9C7583B05723231E8D0C6F0F31616079A397F4692716
39D33CA

srtpsalti = HMAC(s0, "Initiator SRTP master salt") =
0578211B57D75E7C0AA499CA75CA8CBAC755E8B4C04851D6C598B3B
68212BB67

srtpkeyr = HMAC(s0, "Responder SRTP master key") =
FB9A7FB99A23866F9D58D866D8C21DA7E2983CB32386EE3E090A792
7176E38BA

srtpsaltr = HMAC(s0, "Responder SRTP master salt") =

1D53E521B6C896324632C88DF7FA68E4171768B7AAB8714FC03EBDC
C9A07A492

hmackeyi = HMAC(s0, "Initiator HMAC key") =
83F5D7999116D500C216DEE63007A43DC29A6E634894D497E264CE6D
6C6AF307

hmackeyr = HMAC(s0, "Responder HMAC key") =
29193E8BA2EDAA779B4668BF4BD1597094722621EB32990C734F9119
1095F9BB

zrtpkeyi = HMAC(s0, "Initiator ZRTP key") =
C085A37ABB39BA4C5EB80C1615E58F4391415C50A8FD538FC1F146F
EE46A82E9

zrtpkeyr = HMAC(s0, "Responder ZRTP key") =
F39E954C8F0D6BDEB5A51CEC5ECD34286874307C2DCAE5F1382B4B
A4864C5E8D

These values were matched with those that Zfone outputs in its log file when built in debug mode.