CS259 Final Project

Analysis of ZRTP

By Jeremy Robin and Andrew Schwartz
March 13, 2006

1. Protocol Overview

ZRTP, as implemented by ZFone, defines a key establishment protocol for use with SIP (Session Initiation Protocol) and SRTP (Secure Real-time Transport Protocol). It is designed to provide authentication between parties, secrecy, and perfect forward secrecy between sessions. The motivation behind ZFone's novel form of authentication is that PKIs have proven to be unwieldy for common use, and instead voice authentication digests can be used to verify identities. It is interesting to note that, according to the protocol description, it is not required that parties know each others' voices in advance, only that they can distinguish one voice from another.

Sessions using ZRTP will begin with SIP, which provides familiar telephone-like features over the Internet. Next, the ZRTP extensions to SRTP would be engaged by the initiator. If the responder does not have ZRTP available, then the protocol will default back to RTP. ZRTP negotiates master key material, which is passed on to the SRTP layer. The SRTP layer provides a secure transport for the VOIP standard RTP. It handles deriving session keys from the master key material and handles replay attack protection via authenticated sequence numbers. It also ensures that neither master keys nor session keys encrypt enough traffic to allow an adversary to mount any cryptanalytic attack. The payload is encrypted using AES and the entire SRTP packet including headers is authenticated using SHA-1.

First some notations are listed:
vi: represents versioning information, which includes information about encryption algorithms, public key modulus, hash algorithms, etc.
zid: represents a unique identifier for an installation of ZFone
pvi: public Diffie-Hellman value sent to initiator by responder
pvr: public Diffie-Hellman value sent to responder by initiator
hvi: hash(A's vi, B's vi, pvi)
s1: shared secret (if one is available)
rs1r: HMAC(s1, "responder")
rs1i: HMAC(s1, "initiator")

The ZRTP protocol is as follows:

```
A -> B:   HELLO (A's zid, A's vi)
B -> A:   COMMIT (B's Zid, hvi, B's vi)
A -> B:   DHPart1 (pvr, rs1r)
B -> A:   DHPart2 (pvi, rs1i)

----------traffic below secured by SRTP using master_key, master_salt----------
-----where master_key = HMAC(HASH(HASH(DHResult) | s1), "SRTP master key")-----
-----and master_salt = HMAC(HASH(HASH(DHResult) | s1), "SRTP master salt")-----

SAS=hex(hash(pvi | pvr | "voice authentication digest"))
A -> B:   speaks bytes1and2(SAS)
B -> A:   speaks bytes3and4(SAS)
A <-> B:  normal conversation
```

A and B will only be able to verify the other's hex values if they have negotiated the same key material, which would not be the case if an adversary had executed a trivial man in the middle attack. Potentially, since only 4 bytes are verified between the initiator and the responder, an attacker could find a hash collision on the first 32 bits of the SAS hash. Thus, even though the initiator and responder would not share master key material, they would be fooled into believing that they did. This is guarded against in the protocol by requiring the responder to send the hvi value in the responder's COMMIT message. Thus, even though the initiator sends his pvr before the responder sends his pvi, the responder has already committed to his pvi. Hence, neither party can deterministically influence the value of the SAS hash.

Once A and B have established a session successfully (including verifying SAS values), they store an evolving shared secret from the session, indexed by their counter-party's ZID. Thus, subsequent sessions can proceed directly to conversation from the Diffie-Hellman exchange, provided that both parties have retained their shared secret. There is no need to perform SAS in that case.

2. Objectives

ZRTP is a new standard, just released to the public by Phil Zimmermann at the beginning of March 2006. We were able to procure a preprint copy from the authors before the general release. In this project, we want to analyze the protocol with Murphi, and make clear what the appropriate and inappropriate use-scenarios are for ZRTP.

3. Modeling the Protocol

We model ZRTP in Murphi using the Dolev Yao Intruder Model. We simplified the message exchange, to remove messages that only provide cryptographic enhancement, and hence are irrelevant to our model. We also leave out messages that are only relevant to a bid-down attack, because it seems clear that a bid-down attack is not possible, given the hash commit of the version fields.

To attack the protocol, an attacker first must have successfully taken control over the network. He then must be able to execute a man-in-the-middle attack and fool the parties involved into either not performing SAS, or personally performing SAS, such that the parties involved believe that they share a key. The ZRTP protocol itself is not designed to handle cases where a powerful attacker simply poses as some party. It is assumed that though Alice and Bob might not know each other's voices, they will be able to figure out if they are talking to the correct person. As a result, the only invariant we found compelling reasons to check for was that of confidentiality - if the two correct parties are in fact engaging in a conversation, can an attacker listen in?

In modeling the protocol, we often had to make decisions about various features of the initiator and responder, as well as the strength of an attacker. Instead of making arbitrary decisions, we left the choices as constants. In our final revision, we were left with the following 7 constants:

-Attacker can convert a voice heard to his own voice in real time
-Initiator forgets voice from one session to the next
-Responder forgets voice from one session to the next
-Attacker can forge Initiator's voice
-Attacker can forge Responder's voice
-Initiator does not know Responder's voice in advance
-Responder does not know Initiator's voice in advance

4. Attacks on ZRTP

We wanted to explore all 2^7 combinations of parameters. We did this by constructing a bash script, which executed our model 2^7 times, using all possible values for the constants. Of the 128 trials, 61 of the trials yielded attacks against the model. Many of these attacks, however, were not independent of each other. So, we constructed the minimal set that entailed all of the other attacks. This minimal set contained only 7 unique assignments to the constants. Of these 7 "attacks", 2 pairs of assignments were symmetric between the initiator and the responder. This left us with 5 entirely independent attacks. Below are the results, including the symmetric pairs 6,7 and 2,3:

	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Attacker can convert voice to his own in real time	0	0	0	0	1	1	1
Initiator forgets voice between sessions	0	0	1	1	0	0	0
Responder forgets voice between sessions	0	1	0	1	0	0	0
Attacker can forge Initiator's voice	1	0	1	0	0	0	1
Attacker can forge Responder's voice	1	1	0	0	0	1	0
Initiator does not know Responder's voice in advance	0	0	1	1	1	0	1
Responder does not know Initiator's voice in advance	0	1	0	1	1	1	0

We refer to the columns as follows:

1. Pure Voice Forgery Attack
2. Bill Clinton Attack
3. (Symmetric) Bill Clinton Attack

4. 6 Month Attack
5. Court Reporter Attack
6. Hybrid Clinton-Court Reporter Attack
7. (Symmetric) Hybrid Clinton-Court Reporter Attack

The Pure Voice Forgery Attack works as follows: The intruder intercepts the packets sent from the initiator to the responder and poses as the responder. Similarly, the intruder initiates a session with the responder, posing as the initiator. Then, at the SAS phase with the initiator, the intruder says his SAS string using his imitation of the responder's voice. Similarly, the intruder says his other SAS string to the responder using the initiator's voice. Both parties accept the SAS, and the intruder now only needs to decrypt and re-encrypt the voice traffic between the initiator to the responder.

The Court Reporter Attack works as follows: The initiator and the responder do not know each others’ voices in advance (note that this is a mode that is explicitly allowed in the ZRTP paper). The successful man-in-the-middle uses his own voice to say the SAS strings. Then, when the intruder receives voice traffic from the initiator, he (very) quickly relays the traffic to the responder in his own voice, and visa-versa. This is more easily achievable with four "men-in-the-middle" -- two court reporters, and two talkers.

The 6 Month Attack works as follows: The initiator and the responder forget the voice of an unfamiliar counter-party from one session to the next (this can be imagined as a brief conversation that occurs 6 months before the conversation of interest). The intruder initiates and completes conversations with both parties, each time posing as the other, such that both parties think they share a secret with the other. Then, 6 months later, when the parties actually desire to have a conversation, they will each believe that they share a secret with the other, but in reality, they share a secret with the intruder. They will not execute SAS, due to this shared secret. The intruder can then decipher and relay the entire conversation between the parties.

The Bill Clinton Attack works as follows: A celebrity will not remember a random person's voice, however the random person will remember the celebrity's voice. The intruder executes SAS with the celebrity some time in advance, to create a false shared secret, as in the 6 Month Attack. When the initiator then later on wants to contact the celebrity, the man-in-the-middle forges the celebrity's voice for the SAS string, then establishes a session with the celebrity. Finally, the intruder relays and records all traffic. Note that for this attack, one does not need to know who the random person that wants to contact the celebrity is -- one only needs the celebrity to forget voices between sessions.

The Hybrid Clinton-Court Reporter Attack essentially substitutes a "Court Reporter"-capable intruder for the in advance contacting of the celebrity in the Bill Clinton Attack.

We propose a modification to the ZRTP protocol, which attempts to address all of the attacks we discovered. The premise of this solution is that two conversations that start at different times are difficult to interleave. We do this by having the initiator and the responder start a timer at the very beginning of the protocol. In the HELLO and COMMIT messages, we add two new pieces of versioning information: minN and maxN, which specify the minimum and maximum values for the value N, which the party is willing to allow. If the ranges [A's minN, A's maxN], [B's minN, B's maxN] have no overlap, then the party's cannot agree on a value for N and should not proceed. Otherwise, N is chosen as average(max(A's minN, B's minN), min(A's maxN, B's maxN)).

In standard ZRTP, we have a shared random value, which cannot be influenced by either party, namely Hash(Master Key). We interpolate that value, which should be uniform random over its range, to [0,N]. Some constant number of seconds is then added to this value, so that network latency and time for the SAS are taken into account (if desired, this could also be determined dynamically, in the same way that N is determined). Finally, the timer that began at the very beginning of the protocol (sending the Hello message for the initiator, and receiving the Hello message for the responder -- note that these two times are not identical, but with small network latency should be insignificant) will let the conversation participants know that after the elapsed amount of time, they must begin conversation. Furthermore, users of the protocol should be advised to get to the "meat" of their conversation early on, so that an attacker cannot interleave conversations by manipulating "small-talk."

This modification foils all attacks we mention earlier, because with good probability depending on the negotiated value of N, the Initiator will have a different delay value with the Intruder than will the Responder. Hence the Intruder now must stall one conversation while the other is still unavailable. We do acknowledge that the probabilistic nature of this scheme leaves it open to attack. However, two parties who desire arbitrarily high probability of foiling any attack can set their minimum values very high. Parties who are unconcerned by these types of difficult man in the middle attacks may choose not to execute the timing scheme at all.

5. Conclusions

The ZRTP protocol is cryptographically well designed and achieves its objectives under normal use cases. Certainly, in the case of no active attacker, the ZRTP protocol appears secure. There are, however, reasonable use cases (some of which are discussed in the protocol paper) that leave Alice and Bob vulnerable to adversaries with strong capabilities. There is a compelling argument for modifying the protocol to include a randomized start-time for the conversation if one were concerned about powerful adversaries.