Dominik Schürmann*, Fabian Kabus, Gregor Hildermeier, and Lars Wolf

Wiretapping End-to-End Encrypted VoIP Calls: Real-World Attacks on ZRTP

Abstract: Voice calls are still one of the most common use cases for smartphones. Often, sensitive personal information but also confidential business information is shared. End-to-end security is required to protect against wiretapping of voice calls. For such real-time communication, the ZRTP key-agreement protocol has been proposed. By verbally comparing a small number of on-screen characters or words, called Short Authentication Strings, the participants can be sure that no one is wiretapping the call. Since 2011, ZRTP is an IETF standard implemented in several VoIP clients.
In this paper, we analyzed attacks on real-world VoIP systems, in particular those implementing the ZRTP standard. We evaluate the protocol compliance, error handling, and user interfaces of the most common ZRTP-capable VoIP clients. Our extensive analysis uncovered a critical vulnerability that allows wiretapping even though Short Authentication Strings are compared correctly. We discuss shortcomings in the clients' error handling and design of security indicators potentially leading to insecure connections.

Keywords: ZRTP, VoIP, SIP, key exchange

DOI -
Received ..; revised ..; accepted ...

1 Introduction

Following Snowden's global surveillance revelations from 2013, public awareness of privacy and information security has increased and driven the demand for products aiming to protect their users. Thus, the design and implementation of end-to-end secure messaging protocols received a lot of attention [9, 35]. In 2016, these protocols have been adopted by mainstream messaging apps, such as WhatsApp and Facebook Messenger [10, 36]. As a result, mobile messaging, the most popular smartphone feature, finally includes end-to-end encryption for average users. Comparing their security features with that of voice calls shows a major imbalance. While making voice calls is the second most popular smartphone feature with 93% popularity [25], its security is often neglected. It is difficult to retrofit the traditional Public Switched Telephone Network with end-to-end security, but it is feasible to protect users of modern Voice over IP (VoIP) apps.

To protect real-time communication channels, the ZRTP key agreement protocol has been proposed. Based on the Diffie-Hellmann (DH) key exchange, it has been standardized in 2011 as RFC 6189 [38]. It can be implemented independently of the actual signaling protocol, however it is often used in conjunction with the Session Initiation Protocol (SIP) [16]. Instead of relying on a central Public Key Infrastructure (PKI), participants have to compare a few digits or words, called Short Authentication Strings (SASs). If done correctly, no one should be able to actively wiretap the call, i.e., perform an unnoticed Man-in-the-Middle (MitM) attack. The exchanged secrets are utilized to encrypt the stream end-to-end, usually using the Secure Real-Time Transport Protocol (SRTP).

Before its standardization, ZRTP has been formally verified by Bresciani et al. [6]. The authors analyzed the protocol with the "correctly verified SAS" assumption. In 2007/2008, Gupta and Smatikov [14] as well as Petraschek et al. [24] discuss practical attacks, in particular a flaw in the handling of ZRTP IDs (ZIDs). A recent study by Shirvanian and Saxena with 128 online-participants found that for a two-word SAS, an attacker can stay undetected with about 30% probability [33]. In 2016, two theoretical attacks against ZRTP have been published by Bhargavan et al. [3]. They discuss a version downgrade attack as well as a downgrade from DH to Preshared mode. To this day, no systematization of attacks has been done. Also, protocol attacks have only been discussed theoretically or applied to the abandoned Zfone desktop software.

In this paper, we analyze attacks against modern real-world ZRTP systems. It is known that an evil SIP operator can conduct MitM attacks, which can only be

***Corresponding Author: Dominik Schürmann:** TU Braunschweig, E-mail: schuermann@ibr.cs.tu-bs.de
Fabian Kabus: TU Braunschweig, E-mail: kabus@ibr.cs.tu-bs.de
Gregor Hildermeier: TU Braunschweig, E-mail: hilderme@ibr.cs.tu-bs.de
Lars Wolf: TU Braunschweig, E-mail: wolf@ibr.cs.tu-bs.de

detected by SAS comparison. We demonstrate the simplicity how to design a minimally invasive MitM attack that re-routes calls and records conversations in real-time. In the main part of our paper, we analyze attacks against specific ZRTP clients. Here, we assume that SASs are *correctly compared* by end users. We define a set of protocol test cases for verification of standard compliance as well as UI conformance tests. The most common ZRTP clients on major platforms, such as Android, iOS, Windows, and Linux, have been evaluated. Our findings include a critical vulnerability in Linphone (CVE-2016-6271) allowing wiretapping even though SASs have been compared correctly. We report about an issue in Jitsi, were a normal call is misinterpreted as an attack, resulting in a security warning that should have not been displayed. Furthermore, several weaknesses in the clients' user interfaces have been uncovered. By adapting our test cases and best practices we provide guidance on how to properly implement ZRTP.

After introducing ZRTP in Section 2, we explore the possibility of wiretapping encrypted VoIP calls in Section 3. Assuming the use of ZRTP and correct comparison of SASs, we provide protocol and non-protocol specific test cases in Section 4. Using these tests, we evaluate protocol compliance and usability of ZRTP clients in Section 5. In Section 6, we propose best practices for client developers on how to properly design a SAS verification UI. Previous studies and other related work are discussed in Section 7. In Section 8, we discuss the implications of our findings, before concluding the paper in Section 9.

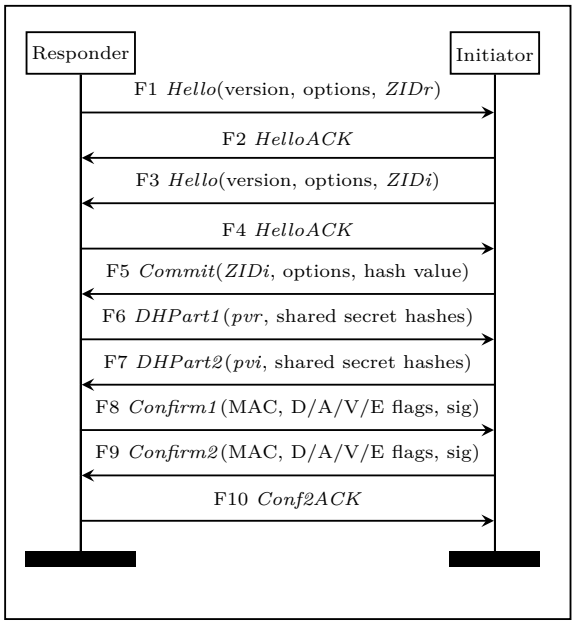


Fig. 1. ZRTP handshake in DH Mode between an Initiator (right) and a Responder (left). The protocol consist of three phases: *Discovery and Version/Algorithm Negotiation (F1-F4)*, *Key Agreement (F5-F7)*, and *Key Confirmation and Derivation (F8-F10)*.

2 ZRTP Fundamentals

The ZRTP key agreement protocol has been standardized in RFC 6189 [38] and uses SASs to detect MitM attacks. This agreement is transported over a Real-Time Transport Protocol (RTP) communication channel that has previously been established by a signaling protocol, such as SIP. The SASs are derived from the DH shared secrets and displayed on end users' displays. They need to be compared verbally by reading them out loud and verifying that the peer's SAS matches with the displayed one. In case of a MitM attack, the participants end up with different shared secrets and thus different SAS. The SASs are very short, e.g., 'bz4f' (B32 encoding) or 'spearhead Yucatan' (B256 encoding with PGP Wordlist [18, 19]), while still providing enough security due to the usage of a hash commitment [37]. This restricts a MitM to only one attempt to guess the correct key for generating the same SAS.

In this section, we provide an overview of ZRTP following the notation of RFC 6189 [38]. We focus on parts of the protocol relevant to our analysis in this paper. A representative call flow can be seen in Figure 1. During the exchange, one party ends up as the Initiator and the other as the Responder. The underlying transport layer protocol is most certainly UDP. Because errors of 16 bit UDP checksums cannot be distinguished from active MitM attacks, all ZRTP packets contain an additional Cyclic Redundancy Check (CRC) to detect errors. Additionally, two exponential backoff retransmission timers are utilized: one for *Hello* messages, the other for all messages sent after *HelloAck*. After the ZRTP handshake is complete, the SASs and keys for a SRTP session are derived and the SRTP session is established. The SAS then has to be compared verbally to ensure that no MitM was between the endpoints. If something goes wrong during the exchange an *Error* message is sent with a specific error code encoding what caused the handshake to fail. Following Figure 1, we will look into the three phases of the protocol, namely *Discovery and Version/Algorithm Negotiation (F1-F4)*, *Key Agreement (F5-F7)*, and finally *Key Confirmation and Derivation (F8-F10)*.

2.1 Discovery and Version/Algorithm Negotiation (F1-F4)

Both endpoints begin the exchange by sending a *Hello* message ensuring the peer also supports ZRTP. The endpoint is identified by a unique randomly generated 96 bit ZID. *Hello* includes the supported ZRTP version, which is used for the version negotiation: The highest version supported by both parties is used. At the time of this writing the ZRTP version is 1.10. The *Hello* message also includes supported hash and cipher algorithms, as well as authentication tag, key agreement and SAS types. The chosen key agreement type then is the fastest both parties have in common. For the remaining parameters, the Initiator may choose one mutually supported type. Received *Hello* messages are acknowledged by subsequent *HelloACK* messages.

2.2 Key Agreement (F5-F7)

After both *Hello* messages have been received, a *Commit* message begins the key agreement. The *Commit* message is sent by the Initiator containing her ZID as *ZIDi*. It is now possible to either proceed in Diffie-Hellmann mode (DH mode) or with existing cached shared secrets in Preshared mode.

2.2.1 DH Mode

In DH mode, the endpoints carry out a straightforward DH key exchange to derive a shared DH secret using the *DHPart1* and *DHPart1* messages (cf. Figure 1). To extend the DH in a way to protect against brute force attacks, the Initiator first commits to a public value *pvi* by including $hvi = hash(pvi)$, where SHA-256 is used for $hash()$ by default, in the *Commit* message. The Responder answers directly with her public key *pvr*. Only after receiving *pvr*, the Initiator sends her public key *pvi* she has committed to in the *Commit* message. This hash commitment provides a protection against pre-computation of hash collisions during the DH exchange [37]. Thus, an attacker can only guess with a chance of one out of 65536 when using a 16 bit SAS [38]. Finally, the DH secret is derived using the Initiator's or Responder's own secret key *svi* / *svr*:

$$DHResult = pvr^{svi} \mod p = pvi^{svr} \mod p$$

If both sides send a *Commit* at the same time, there are rules to break the tie: If one *Commit* is for DH mode and the other for Preshared mode, the DH mode wins. Otherwise, the one with the larger *hvi* value wins. The party with the winning *Commit* becomes the Initiator, the peer becomes the Responder.

2.2.2 Preshared Mode

In Preshared mode, the endpoints can skip the DH if they have shared secrets *rs1* / *rs2* from a previous session. Subsequent shared secrets are derived from the previous one that gives this mode similar properties like in DH mode, such as forward secrecy: If an attacker gains access to this secret, previous calls still can not be decrypted as old key material is immediately destroyed after use.

Shared secrets *rs1* / *rs2* are held in a long-term cache and associated with a ZID. An additional boolean flag is stored to indicate if the SAS has already been compared and verified. It is important to note that entries are not associated to the SIP address, only to the ZID. Furthermore, there is no mapping between ZIDs and SIP addresses in the ZRTP protocol. One can use many devices each with their own ZID but configured for the same SIP address. There can even be many SIP addresses configured using the same ZID. The RFC proposes to allow labeling of ZIDs to indicate the devices they are associated to, such as "Alice on her office phone".

Identifiers for the cached shared secrets are transmitted in the *DHPart* messages. By the Responder they are calculated as:

$$rs1IDr = MAC(rs1, \texttt{'Responder'})$$
$$rs2IDr = MAC(rs2, \texttt{'Responder'})$$

rs1IDi, *rs2IDi* on the Initiator's side are calculated analogously using '`Initiator`' as the second argument for the *MAC*. If a shared secret is not available, a random value is used instead. This hides from an eavesdropper that shared secrets are actually available.

2.3 Key Confirmation and Derivation (F8-F10)

First, a hash is calculated over the previous messages:

$$total\_hash = hash(Hello_{Responder} \parallel Commit \parallel DHPart1 \parallel DHPart2)$$

Both parties then continue to calculate $s0$ that dependents on the *Key Agreement Type*. In the following a simplified version of the protocol is presented[1].

2.3.1 DH Mode

For DH mode:

$$s0 = hash(counter \parallel DHResult \parallel \text{'ZRTP-HMAC-KDF'} \parallel ZIDi \parallel ZIDr \parallel total\_hash)$$

where $counter = 1$.

2.3.2 Preshared Mode

For Preshared mode, prior to sending the *Commit* message the Initiator calculated:

$$preshared\_key = hash(len(rs1) \parallel rs1)$$

where $len()$ denotes the length in octets. Finally, both participants proceed to calculate $s0$:

$$KDF\_Context = ZIDi \parallel ZIDr \parallel total\_hash$$

$$s0 = KDF(preshared\_key, \text{'ZRTP PSK'}, KDF\_Context, \text{negotiated hash length})$$

where the Key Derivation Function is defined as:

$$KDF(KI, Label, Context, L) = HMAC(KI, i \parallel Label \parallel 0x00 \parallel Context \parallel L)$$

2.3.3 Updating Shared Secret Cache

The cache is updated with a new derived $rs1$:

$$rs1 = KDF(s0, \text{'retained secret'}, KDF\_Context, 256)$$

2.3.4 SAS and SRTP Key Derivation

$s0$ has been derived regardless of the utilized mode. The following calculations are identical for both modes:

$$ZRTPSess = KDF(s0, \text{'ZRTP Session Key'}, KDF\_Context, hash\_len)$$

where $hash\_len$ is the negotiated hash algorithm length. The SAS is calculated by:

$$sashash = KDF(s0, \text{'SAS'}, KDF\_Context, 256)$$

where the leftmost 16 bit or 20 bit of the *sashash* are used for B32 or B256 encoding of the SAS respectively.

Finally, *srtpkey*, *srtpsalt*, *mackey*, and *zrtpkey* are calculated. The *srtpkey* and *srtpsalt* are used to encrypt the SRTP traffic, *mackey* is used by ZRTP as the key for subsequent HMAC calculations and *Confirm* messages are encrypted with the *zrtpkey*:

$$srtpkeyi = KDF(s0, \text{'Initiator SRTP master key'}, KDF\_Context, aes\_length)$$

$$srtpsalti = KDF(s0, \text{'Initiator SRTP master salt'}, KDF\_Context, 112)$$

$$mackeyi = KDF(s0, \text{'Initiator HMAC key'}, KDF\_Context, hash\_len)$$

$$zrtpkeyi = KDF(s0, \text{'Initiator ZRTP key'}, KDF\_Context, aes\_length)$$

where aes_length is the negotiated AES key length. The Responder keys are calculated analogously. Successful generation of keys is then confirmed via the *Confirm* messages, which signal that the SRTP session may now start.

2.3.5 PBX Enrollment

To support PBX (essentially a telephone system) in large companies, the ZRTP standard includes a PBX enrollment procedure. This feature is a critical component because it officially supports MitMs. The enrollment can be initiated by setting the PBX Enrollment flag (E) in the *Confirm* message. The client should provide a UI to let the user decide if she wants to allow this PBX for future communications. For this, a secret is calculated:

$$pbxsecret = KDF(ZRTPSess, \text{'Trusted MiTM key'}, ZIDi \parallel ZIDr, 256)$$

If a PBX has been accepted, *pbxsecret* is cached and the MitM flag (M) can be set in future *Hello* messages allowing the PBX forwarding of SAS via a special *SASRelay* message.

1 In contrast to RFC 6189 [38], we exclude *pbxsecret* and *auxsecret* in our calculations and omit the values $s1, s2, s3$, which are usually part of the $s0$ calculation.

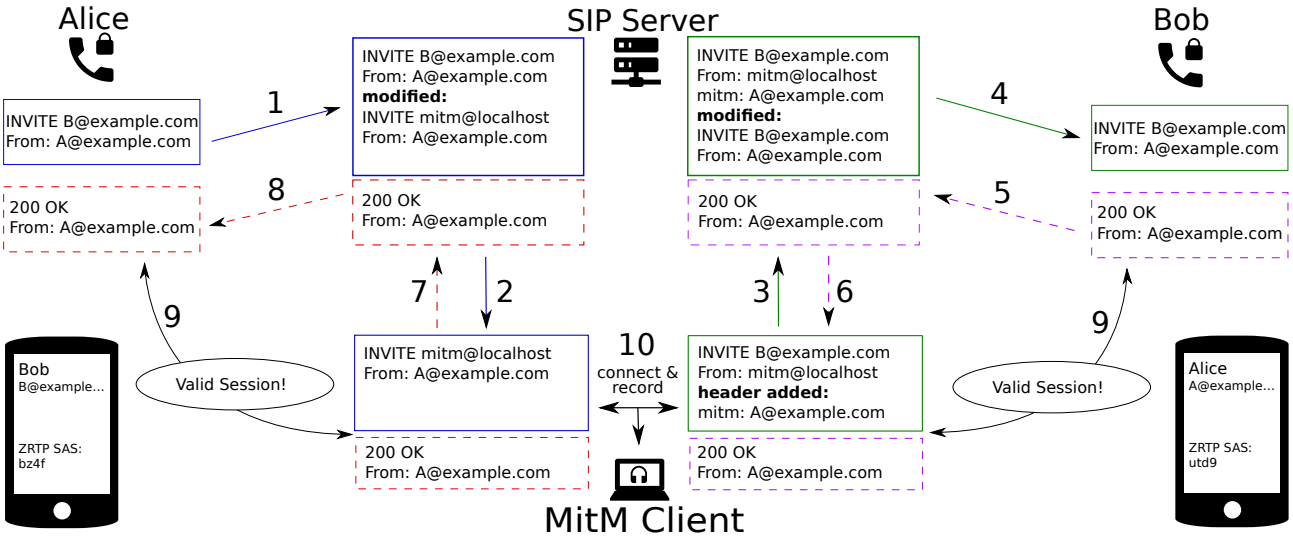


Fig. 2. Flow of minimally invasive wiretapping: The SIP server re-routes INVITE messages to a MitM client connecting the multimedia streams and records the conversation of Alice and Bob. Between the MitM client and the SIP server a 'mitm' header is introduced to pass-through the original 'From' header. As expected, the displayed SASs are different.

3 Wiretapping VoIP Calls

We motivate the importance of end-to-end encryption and authentication support in VoIP clients by showing the simplicity of wiretapping calls by an *evil operator*, i.e., someone having access to the central components of the VoIP network. Our implementation is designed to be as non-invasive as possible regarding to the original SIP flow between the caller Alice and callee Bob. In this section, we do not attempt to break or circumvent ZRTP. Instead we demonstrate the feasibility of wiretapping VoIP calls if SASs are *not* compared.

3.1 Design

To keep the interference and modifications to a minimum, we decided to implement the MitM by hooking into the SIP flow. An attack possibility is given by modifying incoming messages to forward calls to a recording MitM client. This leads to a tunnel through the MitM instead of a direct connection between Alice and Bob. The modification takes care that the header of the messages always contains the originally called SIP address(es) to cover up the attack. A special MitM SIP client accepts any incoming call automatically, starts a second call to the original callee Bob, and connects the incoming data stream from Alice with the new outgoing stream to Bob. Now, everything works according to the protocol but with a MitM recording the multimedia stream, i.e., the conversation. Following Figure 2, wiretapping works along these steps:

1. Alice initiates a call to Bob.
2. The server manipulates the INVITE message such that it is forwarded to the MitM client at 'mitm@localhost'. The information that the message should have been forwarded to Bob is not lost during the modification, because the 'To' header has not been changed.
3. The MitM client initiates, before accepting the incoming call, a second call to Bob. A new 'mitm' header is added to the outgoing call containing the original 'From' address from the incoming one.
4. The server manipulates the INVITE message from the MitM client such that it looks like it came from Alice.
5. Bob acknowledges the INVITE message.
6. This acknowledgment is forwarded as normal to the MitM client, because the MitM is the original caller.
7. The MitM client automatically accepts Alice's call.
8. This acceptance is forwarded as normal to Alice, because she is the caller.
9. Now, two valid connections have been established. Optionally securing these with ZRTP would now lead to divergent SASs.
10. The MitM client connects both multimedia channels and records everything.

We extended Kamailio [20] with the described MitM capabilities. Our patch forwards calls to a MitM client using PJSIP [26] and ZRTP4PJ [8]. Messages are modified

based on their type, e.g., INVITE, BYE, or OK. Multimedia streams are connected together using PJSIP's conference bridge framework. The whole call is recorded using PJSIP's recorder class. Our implementation has been published as open source software[2].

3.2 Summary

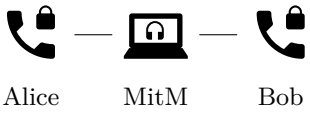
Our MitM implementation for the SIP allows wiretapping of VoIP calls. We were able to verify the correctness of our implementation by initiating calls between our accounts 'A@example.com' and 'B@example.com'. While the clients of Alice and Bob still show the expected SIP addresses, the calls were recorded by our MitM. These attacks can only be protected by end-to-end authentication methods. Enabling ZRTP allows to detect the attacker because the SAS displayed on Alice's client is unequal to Bob's SAS, e.g., "bz4f" vs. "utd9".

4 Attacking ZRTP Clients

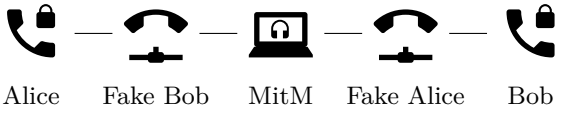
After motivating the importance of ZRTP by showing how MitM attacks stay undetected without it, we now focus on the security of ZRTP-capable VoIP clients. Assuming more cautious participants, who actually compare SASs by voice, an attacker may choose to employ specific attacks tailored towards the usability and correctness of specific software. It is important to note that we will not analyze the possibility of forging spoken SASs, which has already been done in several user studies [23, 32, 33]. Instead, we analyze specific issues of common ZRTP-capable VoIP clients.

4.1 Attack Methodology

We differentiate between two types of MitM attacks depending of the power of an *evil operator*: In the non-impersonating MitM attack (cf. Figure 3a), the attacker is a node on the route between Alice and Bob. While multiple SIP servers can be involved, only one server on the route needs to be behave maliciously for the whole call to be wiretapped. The attacker can read and modify forwarded ZRTP messages, as well as inject her own. The encrypted parts are opaque to the attacker, as she does not have the encryption key.



(a) Non-impersonating MitM: The attacker is a node in between and can forward as well as inject packets.



(b) Active MitM: The attacker maintains two separate calls to Alice and Bob and connects the streams.

Fig. 3. Types of MitM attacks

In contrast, if the attacker is an active MitM (cf. Figure 3b) she maintains two separate calls to Alice and Bob, impersonating the peers. In Section 3, this powerful MitM attacker has been implemented to be able to freely modify multimedia streams. ZRTP aims to detect these types of attacks by having the participants compare SASs by voice.

4.2 Protocol Test Cases

In the following, we provide functional test cases. These either follow the protocol specification to test basic implementation requirements or explicitly violate parts of RFC 6189 [38]. An app passes a test if it behaves according to the expected results defined per test. For a brief overview, our test cases are also summarized in Table 1.

[zrtpCall] Basic Call Functionality: Two calls are made: One should be secured using DH mode, the other using Preshared mode (if supported). To initiate DH mode, the first call uses a new random ZID. This simulates the scenario that either the participants never spoke before over ZRTP-secured VoIP or that one participant uses a new device or client (no shared secrets in cache). In malicious scenarios, an active MitM impersonating one SIP addresses can force a new ZID to bypass Preshared mode and fallback to DH mode.
Test: Conduct two consecutive ZRTP-protected calls.
Expected Results: The calls should succeed as expected.

[verDown] Version Downgrade (Non-Impersonating MitM): As analyzed by Bhargavan et al. the version negotiation in ZRTP is not protected against downgrade attacks [3]. In this test, an old version number is announced instead of the current version

2 https://gitlab.ibr.cs.tu-bs.de/groups/zrtp

Table 1. Overview of protocol and non-protocol test cases

Protocol Tests		
	Test	**Expected Result**
[zrtpCall]	Basic calls in DH and Preshared mode	Successful calls
[verDown]	Version downgrade to '1.0'	Abort key agreement
[weakDH]	DH public key set to '1'	Abort key agreement
[invSS]	Invalid shared secret in ZID cache	Inform user and re-execute SAS comparison
[invCom]	Invalid commit *hvi*	Abort key agreement
[sharedMitM]	Third person who shares secrets with victims acts as a MitM	ZID labeling / Association between SIP and ZID
[pbxEnroll]	Enrollment for PBX with *Confirm* message	Proper UI or abort key agreement if unsupported
Non-Protocol Tests		
	Expected Result	
[statusInd]	Security indicators distinguishing the provided security levels with icon and text	
[confSAS]	Confirmation dialog with button to confirm SAS	
[termError]	On protocol error, terminate the connection automatically	
[secDef]	Provide secure defaults for VoIP providers	

of 1.10. Previous versions are susceptible to an attack described by Gupta and Shmatikov [14], thus we expect that current implementations only provide 1.10.
Test: The version number inside the *Hello* message is set to '1.0'.
Expected Results: The ZRTP key agreement must be aborted. The standard specifies that a received *Hello* message must be ignored if an unsupported version number is received.

[weakDH] Weak DH (Non-Impersonating MitM): This simulates a non-impersonating MitM attack, where the endpoints operate in DH mode. In this attack, a weak DH is enforced by using a *DHPart* (cf. Section 2) with a public value of 1. Since in finite field DH the result is calculated as $DHResult = pvr^{svi}$ on the Initiators side and $DHResult = pvi^{svi}$ on the Responders side, a received public value of 1 always leads to $DHResult = 1$. This effectively breaks encryption, as the encryption key is now known by the attacker. For ECDH, a public value of 1 also must not be accepted as defined in the standard.
Test: The public key of all *DHPart* messages (sent and received) is set to '1'.
Expected Results: The ZRTP key agreement must be aborted upon receipt of a *DHPart* with a public value of 1. This means that at least an indicator should display the insecure connection, even better, the connection should be terminated (cf. [termError]).

[invSS] Invalid Shared Secret (Active MitM): Invalid shared secrets *rs1 / rs2* are used in the following scenarios: 1) The client cache got corrupted and the shared secret is now invalid or deleted. 2) A shared secret has been established previously and the endpoints operate in Preshared mode. Then, a MitM impersonates one endpoint and tries to establish a connection with a wrong shared secret. The ZRTP authors consider this to be a more critical event in comparison to a call made with a new ZID (cf. [zrtpCall]). Thus, according to RFC 6189 [38], a cache mismatch must result in a security dialog explicitly stating that the SAS needs to be compared again indicating a higher risk of a MitM attack.
Test: The shared secret in the ZID cache is replaced with repeated '0xDEADBEEF'.
Expected Results: The user must be notified via a security dialog and the SAS comparison needs to be executed again.

[invCom] Invalid Commit (Active MitM): This simulates an active MitM attack, where the endpoints operate in DH mode. In case of a commit clash, that is both parties tried to commit, the higher *hvi* wins. *hvi* most certainly does not match the corresponding public key. This means the participant sending the commit is impersonated by a MitM who does not want to commit to a public key. This would allow him to receive the other public key and repeatedly generate a public key so that it results in a SAS collision. This is feasible since the SAS consists of 16 bit for a B256 SAS and 20 bit for a B32 SAS. Then Alice and Bob would end up with the same SAS, even though a MitM attack has been executed. It is essential to verify that the revealed public key actually corresponds to the *Commit*.
Test: In the *Commit* the *hvi* is set to repeated '0xFFFFFFFF'.

Expected Results: The ZRTP key agreement must be aborted when receiving such a commit.

[sharedMitM] Shared MitM (Active MitM): A third person Eve conducts a normal ZRTP-protected call with Alice and one with Bob at different times. All participants verify the SASs. Now, Alice as well as Bob have shared secrets associated to Eve's ZID in their cache. Eve can now act as an active MitM by announcing her own ZID instead of forwarding that of Alice and Bob. Because ZIDs are not associated to a participant's SIP address, this attack stays undetected. The RFC proposes to implement labeling of ZIDs (cf. Section 2). This would allow the detection of such attacks if the participants compare the SIP address with the displayed ZID label. It is important to note that SAS verification does not imply that someone trusts a conversation partner, e.g., Trump does not necessarily trust Clinton, but they will likely want to verify their SASs. Thus, it is valid to assume that users will establish shared secrets with persons outside their social community.
Test: An attacker conducts normal calls with Alice and Bob. Then she acts as a MitM and modifies the key agreement by sending her own ZID instead of forwarding Bob's and Alice's ZID.
Expected Results: Either a) ZID labeling is implemented, b) Preshared mode is not supported, or c) non-standard association between SIP address and ZIDs is available.

[pbxEnroll] PBX Enrollment: This test verifies that the client either provides an appropriate user interface for accepting PBX when *Confirm* is received with the E flag or that the PBX enrollment is not supported. In case this feature is unsupported, the client must not handle *Hello* with the M flag and *SASRelay* messages in any way.
Test: 1) *Confirm* with the E flag is sent to the client. 2) *Hello* with the M flag is sent. 3) *SASRelay* is sent.
Expected Results: Abort ZRTP key agreement if not supported or show proper UI for PBX enrollment.

4.3 Non-Protocol Test Cases

Besides verifying that ZRTP clients are compliant with RFC 6189 [38], we evaluated an additional set of requirements not specified in the RFC. These tests focus on usability, error handling, and defaults. A general proposal how to implement security indicators, SAS verification, and error handling is given in our 'Best Practices' in Section 6.

[statusInd] Clear Status Indicators: If a client supports different levels of security, these should be communicated via indicators to the user. According to "Rethinking Connection Security Indicators" [12], a clear status indicator should be a combination of an icon and corresponding text. As presented in their study, not all participants understood the presented icons in their intended way. Furthermore, displaying only icons means that visually impaired users, such as red-green or total color-blind individuals, must rely on the icon's shape. Thus, the used icons and texts must be clearly distinguishable from each other to separate the security levels.
Expected Results: Security indicators distinguishing the provided security levels with icon and text.

[confSAS] Explicit Confirmation of SAS: To encourage the user to perform the SAS comparison, we expect a dialog that explicitly asks for confirmation of the shared SAS. It should explain the process with at least a way to confirm the SAS with a button.
Expected Results: Confirmation dialog with button to confirm SAS.

[termError] Terminate on Protocol Error: In case of a ZRTP protocol failure, such as [weakDH] or [invCom], the call should not fallback to an insecure connection. It can be assumed that either the server is trying to intercept the connection or that the participant's client is severely broken. It has been shown that users prefer functionality over security and will use an insecure fallback mode if provided. For example, studies analyzing the effectiveness of SSL warnings showed that non-experts clicked through warnings with probability over 30% to be able to still access the website [1, 34]. In our scenario, we suspect that users will not terminate a call manually and instead continue a conversation. Also, when talking on the phone, users usually do not pay attention to the on-screen state and will miss the insecure fallback indicator. Thus, the connection should be terminated automatically.
Expected Results: On protocol error, terminate the connection automatically.

[secDef] Secure Defaults: For better usability of the client's security features, secure defaults should be provided. If the client is targeted at a specific service, this can easily be done by enabling all security features by default. If a client supports different SIP providers, a setup procedure should allow the selection of the service and then enable security features accordingly.
Expected Results: Provide secure defaults for VoIP providers.

Table 2. Evaluation results for the most common ZRTP-capable VoIP clients using our protocol and non-protocol tests (ascending alphabetical order by name).

Application	OS	Version	Library	Protocol Tests							Non-Protocol Tests			
				[zrtpCall]	[verDown]	[weakDH]	[invSS]	[invCom]	[sharedMitM]	[pbxEnroll]	[statusInd]	[confSAS]	[termError]	[secDef]
Acrobits Softphone	iOS	5.8.1	-	●	●	●	●	●	●	–	○	●	○	○
CSipSimple	Android	1.02.03	ZRTP4PJ	●	●	●	○	●	○	–	◐	●	○	●
Jitsi	Win, Lin, MacOS	2.9.0	ZRTP4J	◐	●	●	●	●	○	–	●	●	○	●
Linphone Android	Android	3.1.1	bzrtp	●	●	●	○	○[a]	○	–	○	●	○	○
Signal	Android	3.15.2	-	●	–	●	–[b]	●	–[b]	–	●	○	●	●
Signal	iOS	2.6.4	-	●	–	●	–[b]	●	–[b]	–	●	○	◐	●

● = pass, ◐ = partially, ○ = fail, – = not supported

[a] CVE-2016-6271

[b] Signal is a *cacheless implementation*. It does not support Preshared mode.

5 Evaluation

For our evaluation, we selected common ZRTP-capable VoIP clients. Our selection criteria was as follows:

1. We must be able to execute our test cases:
   - If the client supports federated SIP and does **not operate in a closed network**, we can test it by conducting a call to a special Jitsi client that has been modified to execute our test cases against the calling client.
   - If the client is **operating in a closed network**, we need to implement our test cases directly into this client. Thus, the client's **source code must be available**.
2. The client should be relevant:
   - The client should be **actively used**, i.e., with a user base of at least 100,000 installations.
   - The implementation should be under **active development**, i.e., new versions have been released in 2016.

ZRTP + SIP: While ZRTP can be deployed independently of the signaling protocol, RFC 6189 [38] mainly focuses on its usage with SIP. The developer collective *Guardian Project* provides the Open Secure Telephony Network (OSTN) specification, a de-facto standard to deploy secure federated VoIP services based on SIP [13]. Using their testbed, we analyzed the following SIP clients:

Acrobits Softphone Closed source but standard-conform SIP client with ZRTP extension for iOS. Recommended on the OSTN website.

CSipSimple Free Software for Android with an active user base of over 1M users. Several proprietary forks have been released on Google Play.

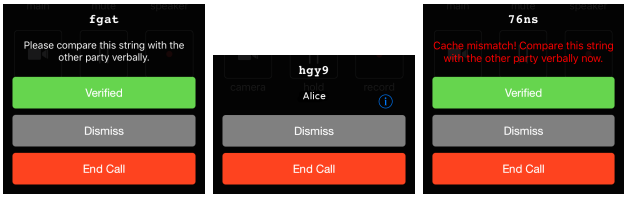
Jitsi Open Source software for desktop operating systems that is actively developed with at least one new git tag per month.

Linphone Free Software for Android with an active user base of over 100,000 users.

We excluded clients that are no longer available, such as Zfone (abandoned since 2011-01-29), Qutecom (abandoned since 2015-03), and SFLPhone (successor app named *Ring* no longer uses ZRTP). We excluded PrivateWave because we were not able to execute our test cases due to its operation in a closed network and because no source code is available. Silent Phone has been excluded because it operates in a closed network and we were not able to compile its source code[3], which is available on GitHub.

ZRTP + XMPP: Besides SIP, ZRTP can also be integrated with the Extensible Messaging and Presence Protocol (XMPP). XMPP has been extended by Jingle in XEP-166/XEP-167 [30, 31] to support media sessions between peers, primarily used for voice communication. For end-to-end security, ZRTP has been standardized as an extension in XEP-0262 [28]. To the best of the authors' knowledge, only Jitsi supports ZRTP over Jingle. Other XMPP clients, such as Empathy and Pidgin, do not support XEP-0262.

3 https://github.com/SilentCircle/silent-phone-android/issues/11



(a) [zrtpCall] in DH mode

(b) [zrtpCall] in Preshared mode with ZID label

(c) Warning message for [invSS]

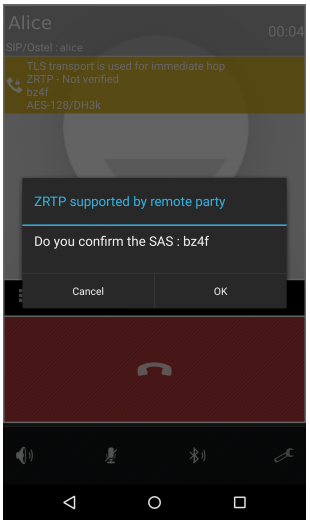
Fig. 4. Acrobits Softphone: Dialog containing instructions, SAS verification, and ZID label

(a) no security

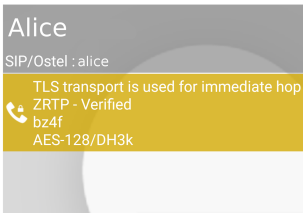
(b) full ZRTP

(c) One ZRTP-protected and one unprotected stream

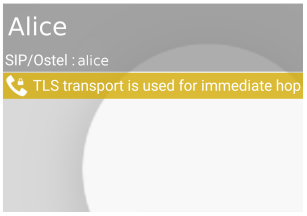
Fig. 5. Acrobits Softphone: States of security indicators



(a) SAS verification UI shown for [zrtpCall] in DH mode, no warning for [invSS]



(b) Security indicator for [zrtpCall] in Preshared mode



(c) Security indicator for non-ZRTP connections

Fig. 6. CSipSimple: UI of interesting test cases

ZRTP + HTTP: Due to design objectives, such as high asynchronicity in mobile scenarios, Open Whisper Systems decided to design their own minimal signaling protocol with a RESTful HTTP API [21]. This has been deployed in conjunction with ZRTP in their messaging and VoIP app *Signal*. While Signal does not support federation, we were able to implement our test cases based on their source code for Android and iOS and thus selected it for our evaluation:

Signal Free Software for Android and iOS with an active user base of over 1M users.

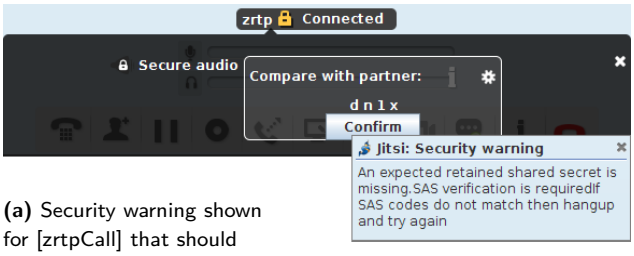
Our results are summarized in Table 2. In the following we discuss them in detail for each app individually.

5.1 Acrobits Softphone

As shown in Figure 4, Acrobits Softphone behaved perfectly in all protocol tests. It is the only implementation that implemented the labeling of ZIDs and thus provides protection against [sharedMitM]. For [verDown], [weakDH], [invSS], and [invCom] the key agreement was aborted falling back to non-ZRTP. Unfortunately, the connection was not terminated ([termError]). It lacks a proper wizard to setup an account with the OSTN, thus users are required to configure a secure server connection by themselves ([secDef]). What negatively stood out in our analysis is the use of security indicators. As shown in Figure 5, there is no icon for insecure connections making it difficult to assess the current security for an end user. Furthermore, we were able to establish connections where two indicators were displayed: Unfortunately, it was not clear for us what these meant just by looking at them. Acrobit's support explained that the icon labeled with 'CLEAR' indicates an insecure video stream besides the ZRTP-protected voice channel.

5.2 CSipSimple

No serious protocol issues have been encountered when testing for [verDown], [weakDH], and [invCom]. As expected for [confSAS], a SAS confirmation dialog is shown (cf. Figure 6a). No warning message is shown for [invSS], thus this test does not pass. Also, there is no way to label ZIDs, thus [sharedMitM] cannot be detected. Regarding [statusInd]: Detailed security information is displayed (cf. Figure 6b): The underlying network layer (TLS), ZRTP verification status, the SAS, the block cipher algorithm (AES 128), and the key agreement type (finite field DH with 3072 bit modular exponentiation group). Users with a security background will be well informed, but other users will probably mistake the small lock icon in the insecure fallback for guaranteed confidentiality (cf. Figure 6c). As seen in Figure 6c, CSipSimple does not terminate the call on errors but falls back to non-ZRTP ([termError]). This behavior can easily go unnoticed by users, especially because the different statuses are not differentiated by icon or color.

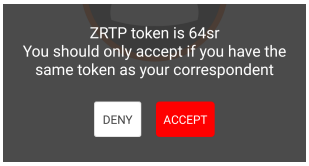


(a) Security warning shown for [zrtpCall] that should only be shown for [invSS].

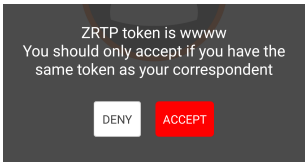
(b) Security indicator for [zrtpCall] in Preshared mode after SAS confirmation

(c) Security indicator for non-ZRTP connections

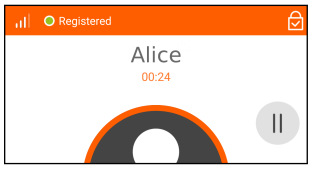
Fig. 7. Jitsi: UI of interesting test cases



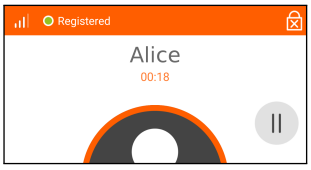
(a) SAS verification UI shown for [zrtpCall] in DH mode



(b) Succeeded SAS attack



(c) Security indicator for [zrtpCall]



(d) Security indicator for non-ZRTP connections

Fig. 8. Linphone: UI of interesting test cases

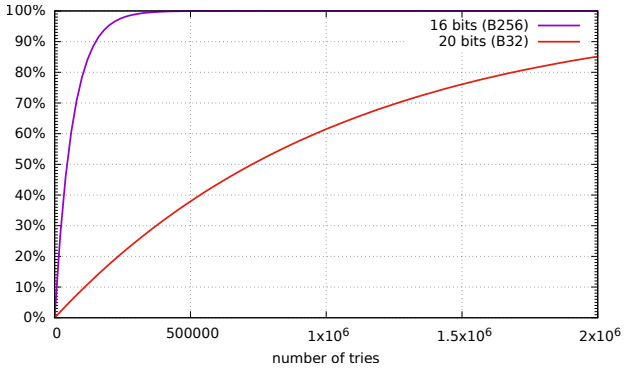


Fig. 9. Linphone: Probability of hitting a targeted SAS when exploiting CVE-2016-6271

5.3 Jitsi

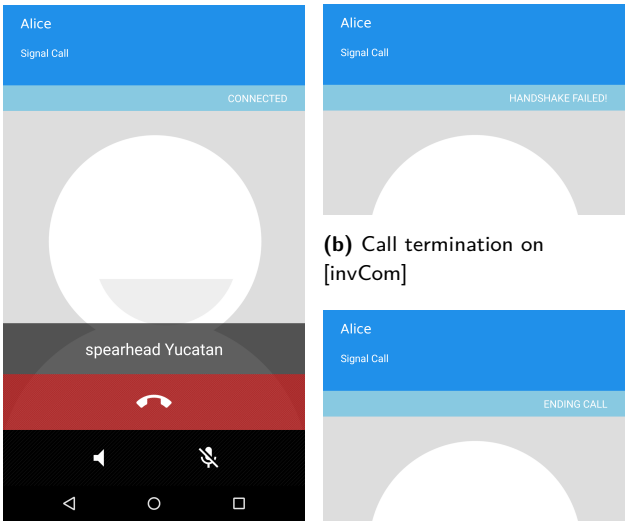
No protocol issues have been encountered when testing for [verDown], [weakDH], [invSS], and [invCom]. ZID labeling is not implemented, thus [sharedMitM] cannot be detected. Two strong visual cues are used to convey the security status (cf. Figure 7b/7c): an closed or opened lock ([statusInd]). Additionally, the states between verified and unverified SAS are differentiated by green and yellow. The corresponding text says "zrtp Connected" for ZRTP connections or "Connected" for other connections. While this can still be improved as described in Section 6, Jitsi provides the best representation for end users compared to other analyzed SIP-based clients. The warning message for [invSS] is a little bit misleading, but the client still responds correctly (cf. Figure 7a). However this warning is also shown for [zrtpCall] after two other calls have been made. We analyzed this problem in detail and fixed the issue: Instead of generating a completely new ZID entry in-memory, the last read entry from the ZID cache was used for a call with a new participant. This happened because a variable was not resetted properly. As seen in Figure 7c, Jitsi does not terminate the call on errors but falls back to non-ZRTP ([termError]).

5.4 Linphone Android

[verDown] and [weakDH] succeeded as expected. [invSS] did not pass as no warning message is shown here. ZID labeling is not implemented, thus [sharedMitM] cannot be detected. The test case [invCom] ended *fatal*: We uncovered a critical security vulnerability that gives an attacker full control over the displayed SAS. We implemented a fully working exploit using a patched Jitsi client that simulates a MitM. There are two variants to this attack, when taking the perspective of an active MitM:

a) Only one client is vulnerable: The SAS of the other client is random and the attacker forces the newly generated SAS to collide with the already established SAS. If b is the number of bits in the SAS, finding a collision after k trials is a Bernoulli experiment and the probability is $1-(\frac{2^b-1}{2^b})^k$, where b is the number of bits (cf. Figure 9).

b) Both clients are vulnerable: The search for an SAS collision becomes a lot easier. The attacker is not required to collide with one certain SAS, but any SAS that can be forced on the other side suffices. Note however, that this is not a *birthday attack*. In theory, the attacker could hit a SAS twice on one side without reaching a collision between both sides. There is no

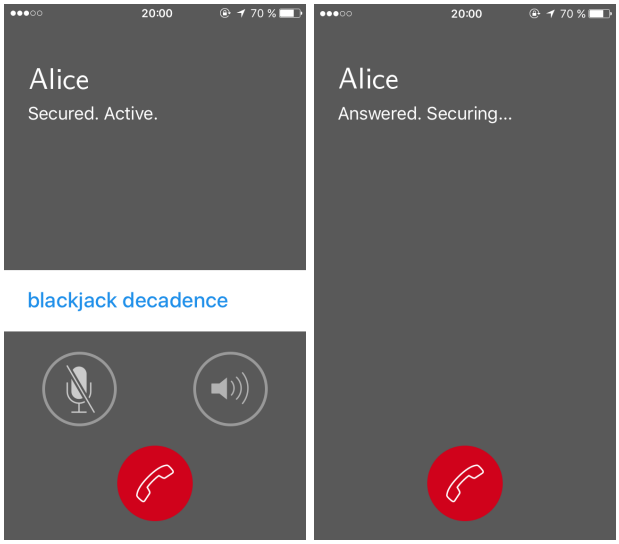


(a) Only DH mode is supported for [zrtpCall].

(b) Call termination on [invCom]

(c) Call termination on [weakDH]

Fig. 10. Signal on Android: UI of interesting test cases



(a) [zrtpCall]

(b) Unsuccessfull call termination for [weakDH], [invCom]

Fig. 11. Signal on iOS: UI of interesting test cases

limit to when a collision is inevitable, as there is when performing a birthday attack.

As we simulate a MitM without actually having two targets, our attack tackles variant a), the more challenging case. In Figure 8b, Linphone was exploited to display a SAS of 4 matching digits. Imagine the SAS being a 4-letter word (like 'fake' or 'okay'), then the SAS blends in with the message and can alter its meaning. We responsibly disclosed this vulnerability on 07/05/2016 to Belledonne Communications and got CVE-2016-6271 assigned.

The UI for SAS verification is implemented properly as seen in Figure 8a. A small indicator in the top right displays the security status (cf. Figure 8c). Unfortunately, the insecure fallback mode is very difficult to detect, as seen in Figure 8d. Instead of a check mark, a 'X' is displayed inside the lock placed in the top right corner. We consider this as insufficient for [statusInd] to pass. More clearer icons combined with texts and strong colors displayed in a focus area are required.

5.5 Signal Android

No protocol issues have been encountered when testing for [weakDH] and [invCom]. Figure 10a shows Signal's call screen without errors. Notice that in the [weakDH] case in Figure 10b the call is ended directly while in the [invCom] case in Figure 10c the error message "Handshake failed!" is displayed on screen before ending the call. Signal ignores the version field, because it uses a closed network, where Signal clients can only communicate among each other. Thus, even when set to 'NOPE', nothing happens ([verDown]). Because Signal is a *cacheless implementation* [38], the [invSS] and [sharedMitM] tests are ignored.

In Signal, no security indicators are displayed ([statusInd]). We interpret this positively, because only ZRTP calls are supported in its closed network and thus no indicators to differentiate between secure and non-secure calls are required. Due to the fact that the verification status of communication partners cannot be stored for future calls, Signal does not pass [confSAS].

5.6 Signal iOS

Signal on iOS behaves similar to the Android implementation. A simple [zrtpCall] is shown in Figure 11a. Its behaviour only differs for [termError]: While the Android client successfully terminates the connection, the iOS client hangs at the screen indicating that the key agreement is still in progress, as shown in Figure 11b.

6 Best Practices

As shown in our analysis in Section 5, most apps comply with the protocol, but greatly differ regarding their SAS verification UI and use of security indicators. Other publications focusing on voice forgability of SAS found

Table 3. Our proposal for security indicators and actions corresponding to specific ZRTP states. For federated SIP clients we propose *Base Configuration*. A configuration for *High-Sensitive Communication* is proposed for scenarios with higher security requirements. Our extension adds an additional error state in *SIP-ZID Binding*.

ZRTP State	Indicator	Action
Base Configuration		
SAS Verified	Secure	-
SAS Unverified	Not Secure	SAS Verification
No ZRTP	Not Secure	-
Errors, e.g., [invCom]	-	Terminate
Error: [invSS]	Not Secure	SAS Verification+Warn
High-Sensitive Communication		
SAS Unverified	Not Secure	-
No ZRTP	Not Secure	-
SIP-ZID Binding		
Error: Mismatch	Not Secure	SAS Verification+Warn

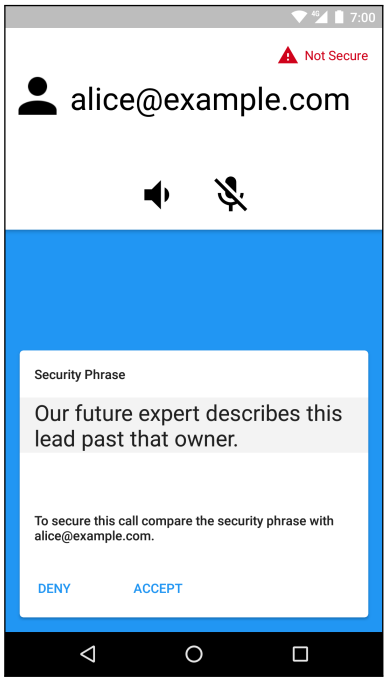


Fig. 12. Our proposed SAS verification UI for high-sensitive communication. The security indicator is chosen in accordance with Table 3. Instead of four characters (B32) or two words (B256), we propose the usage of sentences generated from the SAS hash. The buttons are design in a neutral way to prevent priming the end user for a specific choice.

that users often did not detect forged voices or dismiss security warnings [23, 32, 33]. To improve the user experience, better convey the current security status, and assist end users' decision making during the SAS verification, we propose a set of improvements for ZRTP clients. These improvements are based on results from other publications and justified individually.

The following design criteria are primarily written for ZRTP client developers. The SIP-ZID binding and SAS SENTENCE encoding could be published as an IETF Internet Draft if accepted by the community.

Sentences for SAS Verification: We propose a SENTENCE encoding [2] that uses the leftmost 50-90 bit from *sashash* to deterministically generate sentences as depicted in Figure 12. It has been shown that deviations in sentences are more easily detected [7]. This also protects against forged SAS as synthesized sentences can be better distinguished from human-spoken sentences [22]. While single words are spoken separately, the tone of words as part of the sentence depend on each other. Single words can easily be synthesized into voice samples for a specific victim and then stored in a lookup table for the actual attack (size: $256 + 256$). In comparison, the cost to synthesize all possible sentences is too high (2^{50} to 2^{90} depending on the algorithm).

SAS Verification UI: The comparison of SAS should be provided in a way that provides clear guidance for users and does not habituate users to always accept SAS. Thus, our card-like design in Figure 12 provides guidelines and buttons that are tinted with a neutral color to prevent users from automatically clicking 'accept'. In our future work, this UI definitely needs to be evaluated in larger user study.

Security Indicators: For apps providing ZRTP alongside insecure communication, non-ambiguous security indicators should be implemented. As depicted in Table 3, we propose the usage of 3 different indicators based on the recommendations by Porter Felt et al. [12] and adapted for ZRTP. For federated SIP clients we propose the usage of grey 'Not Secure' indicators for non-ZRTP calls that happen quite often. For clients configured for *High-Sensitive Communication*, these should be replaced with red indicators in addition to a warning icon (cf. Table 3).

Terminate on Error: ZRTP errors should lead to call termination (in contrast to insecure fallback as in Acrobits Softphone, CSipSimple, Jitsi, and Linphone). Error messages should be displayed in full screen for a short time to be recognized, not inside the call screen (like in Signal).

Warning Message for [invSS]: Together with the red security indicator in Table 3, we propose the follow-

ing warning message for [invSS] in red: "The security phrase of alice@example.com changed. To verify that no one is wiretapping the conversation, compare the new security phrase with alice@example.com."

Shared Secret Cache: We propose to implement a cache for shared secrets and not just use DH mode for every connection. End users should be annoyed very rarely by SAS verification to not get habituated to clicking 'accept'.

6.1 Extension: SIP-ZID Binding

As discussed in Section 2, RFC 6189 [38] proposes to let the user write a label for each encountered ZID to describe its usage, such as "Alice on her office phone". This has been implemented in Acrobits Softphone and protects against [sharedMitM] attacks. Without redesigning the whole protocol, we propose to use SIP addresses as ZID labels without requiring user input. In this way the [sharedMitM] attack can be detected automatically.

Because the same ZID can be used for many SIP accounts, a mismatch can happen. Then, the following warning message should be shown: "Your participant uses a new address. To secure this call compare the security phrase again." After the SAS is verified again, the new SIP address should be added to this ZID. Thus, our extension requires that the client-side cache allows to a save a set of labels (SIP addresses) associated to each ZID. Conclusively, the extension provides a way to pin SIP addresses to specific ZIDs.

7 Related Work

ZRTP has been verified formally by Bresciani et al. [4–6]. It was shown that it is a secure key agreement protocol under the Dolev-Yao model. For their verification, the authors assume the SAS comparison to be able to detect MitM attacks. They have not analyzed voice forgery and similiar attacks. Gupta and Smatikov discovered a flaw in one of the previous versions of ZRTP before its standardization [14]: ZIDs were not authenticated early enough in the protocol exchange. Because they are used to look up shared secrets from the cache, an attacker could spoof a known ZID to conduct a MitM attack. Petraschek et al. analyze theoretical and practical attacks against ZRTP [24]. On the one hand, they focus on bypassing the SAS comparison by tampering with the audio signal once a MitM has been established. On the other hand, practical attacks are discussed, similar to ours. They show how to get into the media path for a MitM attack and analyze the behaviour of the—now abandoned—ZRTP client Zfone. They recognize that if an attacker uses a new ZID and spoofs the SIP address of the target, it is easily dismissed by inattentive users that this new session now longer uses shared secrets from the cache and instead should be verified again to detect the attacker. Recently, two theoretical attacks have been published by Bhargavan et al. [3]. They discuss a version downgrade attack as well as a downgrade from DH to preshared mode. To this day, uncovered attacks have not been systematized or applied for testing modern ZRTP clients.

Petraschek's attacks on SAS could be combined with recent work showing the feasibility of crowdsourcing voice imitation [23]. Similarly, imitating the voice of a participant to forge the SAS has been researched by Shirvanian et al. [32]. Two approaches were investigated: the *short voice reordering attack* takes prerecorded SAS strings of the target and uses them to forge the SAS, the *short voice morphing attack* generates arbitrary strings in the victim's voice given just a few minutes of eavesdropped sentences. The effectiveness is demonstrated by testing against manual detection as well as automatic detection. In the user study with 30 participants, about 50% of morphing attacks and 80% of reordering attacks were undetected. In a subsequent study with 128 online-participants, they found that for a two-word SAS, an attacker succeeds with about 30% probability [33]. This is due to human errors, such as failed speaker identification or wrong checksum comparison. While we do not provide an overview over the large amount of research in the area of voice synthesis, newer results, such as DeepMind's WaveNet [22], can drastically decrease the detectability of these attacks. All discussed results show the feasibility of replacing spoken voice with imitated recordings. In their SoK paper, Unger et al. compare the verification via SAS with others trust establishment approaches [35]. In particular, they classify SAS as not being *inattentive user resistant* because users are often required to manually end the call on failed verification.

Mechanisms to mitigate MitM attacks other than SAS that do not require the parties' active participation have also been proposed by Hlavacs et al. [15]. The first one assumes that Alice and Bob do not know each other (and therefore the attacker is not likely to know Bob either and can not predict that Alice is going to call Bob in the future), a situation in which the confidentiality is most threatened by attacks on SAS. Bob is obliged to send Alice the solution of computational

puzzle within a small timeframe (10 seconds for example) that involves: A period of validity, Bob's URI, a temporary public key that is used to create a VPN. An improvement by associating ZIDs to SIP addresses is proposed by Petraschek et al. to protect against previously discussed voice forgery attacks [17]. A different idea has been investigated by Schürmann et al. by utilizing audio fingerprinting to replace the manual comparison of SAS [29]. This enables the use of devices without displays and hands-free equipment. However, to the best of the authors knowledge, no research specifically investigated implementation aspects in which security might go wrong in regard to ZRTP. This includes errors and UI weaknesses in real-world clients that secure their communications with ZRTP.

A lot of non-ZRTP-specific research exists related to the verification of public keys via fingerprints and hash commitments. In a 1000-participant large usability study, Dechand et al. evaluate fingerprint representations [7]. They recommend a sentence-based encoding, which achieves the highest attack detection rate and best usability perception. A hash commitment protocol with up to 10 peers is proposed by Farb et al. in SafeSlinger [11]. Their implementation includes interesting new UI concepts for verifying SAS by displaying radio buttons with three possible SAS choices where only one has been generated from the shared secret.

8 Ethics and Follow-Up

We hope that our findings contribute to the security of the VoIP ecosystem by having an impact on protocol designers, developers, and subsequently the end users.

We provide a MitM implementation to show wiretapping of unprotected VoIP calls. Our intention is not to harm end users, but to demonstrate the simplicity of interception software.

The security vulnerability CVE-2016-6271 in Linphone has been responsibly disclosed on 07/05/2016 to Belledonne Communications and fixed in Linphone 3.2.0[4]. We directly fixed the issue that a MitM warning is shown in Jitsi for normal calls due to erroneously reading the last entry from the ZID cache[5]. CSipSimple and Linphone did not implement a warning dialog for [invSS]. While RFC 6189 [38] requires this, it is not a fatal protocol error and its usefulness is limited. No tested client except Acrobits Softphone is protected against [sharedMitM]. RFC 6189 proposes ZID labeling to provide users a way to detect this attack. We suspect that the adoption of ZID labeling is hindered by its UI complexity. Here, we want to encourage a broader discussion how to prevent this attack automatically, e.g., by SIP-ZID binding.

We encountered different status indicators, which were not optimal and easily dismissed. Developers should follow our best practices and use indicators from Table 3. To prevent accidental insecure usage, clients should terminate on errors and provide secure defaults for SIP accounts. We hope that our proposed best practices encourages a discussion about usability and UI elements in the ZRTP developer community.

9 Conclusion

In this paper, we analyzed how VoIP calls can be wiretapped despite end-to-end protection by the ZRTP key exchange protocol. We motivated the importance of ZRTP by showing that active MitM attacks are easy to deploy for operators of VoIP infrastructure. As the main part of our research, we evaluated six of the most used open-source ZRTP clients available. We found one critical vulnerability (CVE-2016-6271) where Linphone on Android does not follow the standard and implements no verification of the hash commitments. This vulnerability allows successful wiretapping, even when comparing SASs. In Jitsi, a normal call was misinterpreted as an attack resulting in a false security warning. We also found that most implementations fall back to insecure non-ZRTP connections on ZRTP errors, which is hard to see for end users, who do not observe their screen when calling. This is made worse by bad UI practices, where security indicators are difficult to differentiate or not placed in central UI areas. Finally, we proposed best practices on how to overcome the deficiencies related to the way ZRTP has been integrated in VoIP user interfaces.

4 https://github.com/BelledonneCommunications/bzrtp/commit/bbb1e6e2f467ee4bd7b9a8c800e4f07343d7d99b

5 https://github.com/wernerd/ZRTP4J/pull/6
https://github.com/jitsi/jitsi/issues/303

References

[1] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 257–272, Washington, D.C., 2013. USENIX.

[2] akwizgran. basic-english. https://github.com/akwizgran/basic-english. (Accessed: 10/2016).

[3] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, and S. Zanella-Béguelin. Downgrade resilience in key-exchange protocols. In *IEEE Symposium on Security and Privacy (SP)*, pages 506–525, May 2016.

[4] R. Bresciani and A. Butterfield. A formal security proof for the ZRTP protocol. In *International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–6, Nov 2009.

[5] Riccardo Bresciani. The ZRTP protocol analysis on the diffie-hellman mode. *Computer Science Department Technical Report TCD-CS-2009-13, Trinity College Dublin*, 2009.

[6] Riccardo Bresciani and Andrew Butterfield. ProVerif analysis of the ZRTP protocol. *International Journal for Infonomics (IJI)*, 3(3), 2010.

[7] Sergej Dechand, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. An Empirical Study of Textual Key-Fingerprint Representations. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 193–208, Austin, TX, August 2016. USENIX.

[8] Werner Dittmann. ZRTP4PJ README. https://github.com/wernerd/ZRTP4PJ/tree/develop, 2015. (Accessed: 10/2016).

[9] Electronic Frontier Foundation. Secure Messaging Scorecard. https://www.eff.org/secure-messaging-scorecard, November 2014.

[10] Facebook. Messenger Secret Conversations. https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf, July 2016.

[11] Michael Farb, Yue-Hsun Lin, Tiffany Hyun-Jin Kim, Jonathan McCune, and Adrian Perrig. SafeSlinger: easy-to-use and secure public-key exchange. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 417–428. ACM, 2013.

[12] Adrienne Porter Felt, Robert W. Reeder, Alex Ainslie, Helen Harris, Max Walker, Christopher Thompson, Mustafa Embre Acer, Elisabeth Morant, and Sunny Consolvo. Rethinking Connection Security Indicators. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 1–14, Denver, CO, June 2016. USENIX.

[13] Guardianproject. Open {Secure,Source,Standards} Telephony Network (OSTN). https://dev.guardianproject.info/projects/ostn/wiki/Wiki, April 2016.

[14] Prateek Gupta and Vitaly Shmatikov. Security Analysis of Voice-over-IP Protocols. In *20th IEEE Computer Security Foundations Symposium (CSF 2007)*, pages 49–63, Venice, Italy, July 2007.

[15] Helmut Hlavacs, Wilfried Gansterer, Hannes Schabauer, Joachim, Martin Petraschek, Thomas Hoeher, and Oliver Jung. Enhancing ZRTP by using Computational Puzzles. *Journal of Universal Computer Science*, 14(5), 2008.

[16] IETF. SIP Working Group. https://datatracker.ietf.org/wg/sip/, July 2009.

[17] O. Jung, M. Petraschek, T. Hoeher, and I. Gojmerac. Using sip identity to prevent man-in-the-middle attacks on zrtp. In *2008 1st IFIP Wireless Days*, pages 1–5, November 2008.

[18] Patrick Juola. Isolated-Word Confusion Metrics and the PGPfone Alphabet. In *International Conference on New Methods in Natural Language Processing*, 1996.

[19] Patrick Juola. Whole-word phonetic distances and the PGPfone alphabet. In *Fourth International Conference on Spoken Language (ICSLP 96)*, volume 1, pages 98–101 vol.1, October 1996.

[20] Kamailio. Kamailio SIP-Server. https://www.kamailio.org. (Accessed: 10/2016).

[21] Moxie Marlinspike. Creating a low-latency calling network. https://whispersystems.org/blog/low-latency-switching/, January 2013.

[22] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[23] Saurabh Panjwani and Achintya Prakash. Crowdsourcing Attacks on Biometric Systems. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 257–269, Menlo Park, CA, July 2014. USENIX.

[24] Martin Petraschek, Thomas Hoeher, Oliver Jung, Helmut Hlavacs, and Wilfried Gansterer. Security and Usability Aspects of Man-in-the-Middle Attacks on ZRTP. *Journal of Universal Computer Science*, 14(5):673–692, 2008.

[25] Pew Research Center. The Smartphone Difference. http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/, April 2015.

[26] PjProject. PjProject. http://pjsip.org/, 2015. (Accessed: 10/2016).

[27] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002.

[28] Peter Saint-Andre. Use of ZRTP in Jingle RTP Sessions. XEP-0262, June 2011.

[29] Dominik Schürmann and Stephan Sigg. Poster: Handsfree ZRTP - A Novel Key Agreement for RTP, Protected by Voice Commitments. In *Symposium On Usable Privacy and Security (SOUPS)*, July 2013.

[30] Joe Beda Peter Saint-Andre Robert McQueen Sean Egan Scott Ludwig and Joe Hildebrand. Jingle. XEP-0166, May 2016.

[31] Peter Saint-Andre Sean Egan Robert McQueen Scott Ludwig and Diana Cionoiu. Jingle RTP Sessions. XEP-0167, July 2016.

[32] Maliheh Shirvanian and Nitesh Saxena. Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones. In *Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 14)*, pages 868–879, New York, NY, USA, 2014. ACM.

[33] Maliheh Shirvanian and Nitesh Saxena. On the security and usability of crypto phones. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 21–30, New York, NY, USA, 2015. ACM.

[34] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 399–416, Berkeley, CA, USA, 2009. USENIX.

[35] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure Messaging. In *IEEE Symposium on Security and Privacy*, pages 232–249, May 2015.

[36] WhatsApp. WhatsApp Encryption Overview. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf, April 2016.

[37] Wikipedia. Commitment scheme. http://en.wikipedia.org/wiki/Commitment_scheme. (Accessed: 10/2016).

[38] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media Path Key Agreement for Unicast Secure RTP. RFC 6189 (Informational), April 2011.