ZRTP Project - Introduction to ZRTP

Di Long Li (Darren) u5490127
Australian National University

September 1, 2017

Abstract

Encryption between two parties require shared key materials in order for the receiving party to decrypt and render the data to be useful. Ideally the communicating parties would share the keys used for encryption in an already secure environment. However, this is not always practical - said environment may be expensive to establish. Thus in practice, keys are often exchanged in a public and untrusted environment, and can be done via the use of key exchange algorithms such as Diffie-Hellman. While key exchange algorithms protects the parties from eavesdropping agents which are not attacking, the key exchange process is vulnerable to Man-in-The-Middle(MiTM) attacks. As a result, key exchange algorithms are normally used inside key exchange protocols, which defend against said attacks via use of authentication.

However, traditional key exchange protocols still require some form of shared and existing key materials, such as public-private keys, certificates, to provide such authentication. ZRTP key exchange protocol does not rely on long-term, static key materials to provide authentication, and still achieves high level of protection against MiTM attacks via use of hash commitment and Short Authentication String(SAS).

Acknowledgement

First and foremost, I would like to thank professor **Rajeev Gore**, leader of the ANU Logic and Computation Group for supervising me in this project. He has given me a lot of helpful advice in regards to scoping and directions of this project, and also feedback which helped me to improve my writings.

I would also like to thank the following people for their comments and suggestions for this writing : **Ming-Dao Chia** (from ANU), **Tsz Fung Kwong** (from Chinese University of Hong Kong), and **Ho Yin Or** (from ANU).

Contents

Glossary

Alice, Bob	-	legitimate communicating agents
Eve	-	eavesdropping agent/attacker
MiTM	-	Man-in-The-Middle

Notes

Interchangeable terms

In the following sections, the following terms may be used interchangably

Key agreement protocol	-	Key exchange protocol		
Key agreement algorithm	-	Key exchange algorithm		
Key materials	-	Cryptographic materials	-	Secrets
Plaintext	-	Clear text	-	Unencrypted data
Ciphertext	-	Encrypted data		

ZRTP name

ZRTP specification does not state the unabbreviated name of ZRTP, if any exists, thus the author of this report assumes ZRTP is the full name of the protocol.

The author of this report deduces Z may stand for Zimmermann, the last name of the author of ZRTP, and RTP may stand for Real-Time Protocol(which will be mentioned below).

This is only to address potential confusions, and the supposed "full" name is not used in any significant way in this report, if at all. Thus the reader should not be concerned and may treat ZRTP as a purely nominative term.

1 Key Agreement Protocol

1.1 Motivation

Encryption by itself does not necessarily require key exchanges if there is only one party involved, e.g. a user does an encrypted backup that it only intends to share with itself.

However, when encryption is used between two parties, e.g. HTTPS between browser and web server, encrypted voice chat, exchange of cryptographic key materials is required, as otherwise there is no way for the opposite side to decrypt the content.

The exchange of key materials or "secrets" can be in many forms, which we will illustrate using a concrete and (hopefully) approachable example.

Suppose Alice wants to send a file securely to Bob, but also wants to allow future exchanges of files with Bob, so Alice intends to establish a long term method. Alice first meets up with Bob, and both agree on a lengthy passphrase to be used in 7-zip(or similar archiving programs). Then later that day, Alice sends the file using the previously agreed passphrase to encrypt and archive the file, and send the archive over to Bob via email, or file sharing services(e.g. Dropbox, Google Drive). Bob downloads the archive, and decrypt using the previously agreed passphrase. Bob now has the file Alice wanted to share, and this completes the secure file exchange process.

In the above example, the meet up and agreement on a passphrase is the exchanging of key materials. It should also be obvious that without this stage, Bob has no way to decrypt Alice's archive, short of cracking the passphrase.

Similarly, for the encryption we use on a daily basis, in the form of HTTPS, SSL, SSH and so on, the encrypted traffic on one side can only be decrypted by the other side if a key exchange process has occured already.

However, one important thing we should recognise at this point is that Alice cannot always meet up with Bob - physically meeting up is not always possible, and can be very expensive even if possible.

Thus, the most practical way of exchanging key materials would be to do it via public and untrusted channel, i.e. across the Internet or across adversaries' network. The difficulty should be obvious now, but we will state it explicitly in the following. In our previous example, Alice and Bob could rely on trust of the physical environment(e.g. no one could steal the key without them noticing). However, in the public environment, it is impossible to guarantee the lack of eavesdropping. The specific difficulty, stated explicitly, then is : How does Alice exchange key with Bob such that (1) both of them get the final key they can use for encryption, and (2) eavesdropping agent(no attacking), even with knowledge of all their communication, cannot deduce the key they are going to use?

The above two difficulties are what key exchange **algorithms** intend to address. Note that Alice and Bob are still vulnerable to MiTM(Man-in-The-Middle) attacks when Eve, outside of purely eavesdropping, also engages actively in sabotaging the communication. MiTM attacks are handled by key exchange **protocols**.

The vulnerability to MiTM attacks is analogous to a real life situation of meeting up with a person one has never seen before. Suppose Alice wishes to meet up with Bob, similar to the above situation, but with the added condition that they have never seen each other before, and has zero information about what each other looks like. Then it is entirely possible(and trivial) for Eve to pretend to be either of them - neither Alice nor Bob can tell. Furthermore, now it is also possible that Eve agrees on a passphrase with Alice, then also agrees on a passphrase with Bob. If done so, Eve then can decrypt files sent between Alice and Bob in either direction.

Also importantly, Eve can still eavesdrop silently. When Alice sends a file to Bob, Eve can decrypt it first using Alice's passphrase, and encrypt it using Bob's passphrase before passing onto Bob, and vice versa. This way, neither Alice nor Bob can notice their files are compromised(no longer secretive).

1.2 Principle

We start by discussing key agreement algorithms in general terms first, then we will reiterate the motivation of key agreement protocols.

Key agreement algorithms rely on mathematical properties of certain calculations, for example, the well-known Diffie-Hellman(DH), or finite-field DH, relies on modular arithmetic.

While DH is commonly used as reference for behaviour of key agreement algorithms, the process can be put in more general terms.

Below is a modified version of the *General Overview* section of Wikipedia page on *Diffie–Hellman key exchange*, specifically the paint analogy displayed[1]. The author of this report would like to note that, to the best of his knowledge, the section aligns with lecture notes such as Kirby A. Baker's[2].

The important steps put in general terms, are as follows (the order is not necessarily strict)

- A common parameter (not private) is shared and agreed on between Alice and Bob, denoted as c
- Alice generates a secret that ONLY Alice knows, denoted as sA
- Bob generates a secret that ONLY Bob knows, denoted as sB
- There exist irreversible (computationally infeasible to invert/reverse) functions, $mix1, mix2$, such that $mix2(mix1(c, sA), sB) = mix2(mix1(c, sB), sA)$ holds
- Alice computes $rA = mix1(c, sA)$, and sends rA to Bob
- Bob computes $rB = mix1(c, sB)$, and sends rB to Alice
- Alice computes $r = mix2(rB, sA)$
- Bob computes $r = mix2(rA, sB)$
- Note that due to the properties of $mix1, mix2$ as stated above, Alice's r = Bob's r, thus the key agreement process is complete, and Alice, Bob now share a common secret r

There are some important properties to be observed in the above process

- $mix1, mix2$ do not need to be different, as long as the above property holds - DH uses the same function
- sA, sB are never exposed in public, and also due to the irreversible property of the two functions, Eve cannot find out sA, sB from rA, rB
- Since sA, sB are never exposed, Eve has no way to compute r

However, the one last important property is what motivates key agreement protocol

- Neither Alice nor Bob can verify each other's identity in above steps - there is lack of authentication

The lack of authentication gives rise to Man-in-The-Middle(MiTM) attacks, thus key agreement algorithms are normally not used alone, but used within key agreement protocols which contain some form of authentication.

1.3 Authentication

Usually, the authentication is done by using existing, shared cryptographic materials, such as keys from a Public-Key-Infrastructure(PKI) setting, certificates issued by a Certificate Authority(CA), or shared secrets.

In the above cases, the verification of identity can be done for example in form of cryptographic challenges. Either side can issue a "cryptographic challenge", which is a computational problem that is only solvable if one has the private key, or shared secrets. This means, you only trust the other side if they have sucessfully solved your problem. A proof of identity test, so to speak.

The challenge in the simplest form may just be asking the other side to decrypt an encrypted message which you sent, and send the plaintext(or clear text, i.e. unencrypted) back to you.

In a PKI setting, the simplest challenge to be issued by Alice may just be Alice encrypt a randomly generated text using Bob's public key. The encrypted message can only be decrypted by Bob's private key(the cryptographc explanation is beyond the scope of this report). Thus if the other side can decrypt successfully and send the original text back to Alice, then Alice has evidence to believe the other side is indeed Bob. Certificates are dealt with similarly.

In a symmetric setting, that is, some form of shared secrets are used symmetrically - encrypt using the same key, decrypt using the same key, then the challenge can be as follows. Alice encrypts a randomly generated text using the symmetric key, if the other side can decrypt and send the text back to Alice, then Alice has evidence to believe the other side shares that particular key with Alice, and that implies the other side is Bob.

In the above settings, by relying on existing cryptographic materials, the authentication process is fairly straightforward - you test the knowledge of the other person, and you gain resistance to attacks by having longer and higher quality keys etc.

However, ZRTP does not rely on existing cryptographic materials and yet still exhibits strong resistance to MiTM attacks in the authentication phase by usage of Short Authentication String(SAS) and hash commitment, which will be covered in the following sections.

2 ZRTP

2.1 Introduction

ZRTP is a "key agreement protocol that performs a Diffie-Hellman key exchange during call setup". A call refers to a voice/video communication session in the Voice over IP(VoIP), or IP telephony, sense. Examples of VoIP applications would be Skype, and other mobile applications(or apps) where one calls someone via the Internet(the call does not rely on the mobile network, but on the data network or WiFi instead).

As mentioned in the **Key Agreement Protocol** section, key agreement protocols address the need of cryptographic materials exchange/agreement before the actual encryption takes place. In this context, the encryption would be the Secure Real-Time Protocol(SRTP) traffic.

SRTP provides security features on top of Real-Time Protocol(RTP), such as "confidentiality, message authentication, and replay protection to the RTP traffic and to the control traffic" [3, pg. 3].

RTP is the core part of delivery of content, it "provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video" [4, pg. 1].

ZRTP "generates a shared secret, which is then used to generate keys and salt for a Secure RTP (SRTP) [RFC3711] session"[5, pg. 4].

To summarise, **SRTP**, which is based on **RTP**, provides the encryption of traffic (and also some other security features) between two communicating parties, and **ZRTP** handles the key exchange process before the encryption takes place.

Both SRTP and RTP are beyond the scope of this report, but they are mentioned in above paragraphs to hopefully provide sufficient context for the reader to understand the role of ZRTP during a call.

2.2 Overview

There are roughly four stages to ZRTP before the key exchange is complete and SRTP session is established. The stages are : **Discovery**, **Commitment**, **Key Generation**, and **Confirmation**.

There are three modes of ZRTP, which differ at the **Commitment** stage, and the **Key Generation** stage. The modes are : **Diffie-Hellman mode**, **Preshared mode**, **Multi-stream mode**. Since in the latter two modes, Diffie-Hellman is not used, the standard also refers them as **non-Diffie-Hellman modes**.

2.3 Stages

2.3.1 Discovery

This stage is similar to what usually occurs in other protocols, the two parties attempt to establish whether the other supports ZRTP. And if so, then begin to negotiate the parameters used in remainder of the ZRTP process.

The parameters include the protocol version, and key exchange algorithms.

Note that while only two types of Diffie-Hellman algorithms are used : finite-field Diffie-Hellman(DH) and Elliptic Curve Diffie-Hellman(ECDH), there are multiple variants in each type which differ by sizes and/or elliptic curve parameters.

2.3.2 Commitment

Commit messages mainly act to confirm parameters such as hash, cipher algorithm choices. The messages also serve important purpose in DH mode, which will be explained in detail in **Core security components of ZRTP** section below.

2.3.3 Key Generation

All three modes apply key derivation function(KDF) on the final shared secret(whether derived via DH or not) to generate **ZRTP session key**(which is used for either Multi-stream mode to generate additional shared secrets, or to generate PBX secrets), **Short Authentication String(SAS)**, **SRTP keys and salts**, **message authentication code keys(MAC keys)**, and **ZRTP keys**(which are used to encrypt confirmation messages described below).

"A Key derivation function (**KDF**) is a basic and essential component of cryptographic systems: Its goal is to take a source of initial keying material, usually containing some good amount of randomness, but not distributed uniformly or for which an attacker has some partial knowledge, and derive from it one or more cryptographically strong secret keys. We associate the notion of "cryptographically strong" keys with that of pseudorandom keys, namely, indistinguishable by feasible computation from a random uniform string of the same length. In particular, knowledge of part of the bits, or keys, output by the KDF should not leak information on the other generated bits. " [6].

In the context of this report, it suffices to keep in mind that KDF tranforms some already random materials to actual cryptographic materials to be used by encryption algorithms. This is necessary because outside of the algorithmic choice, the strength of encryption also depends on "quality"(uniformly random, unpredictable, etc) of the key materials used, and KDF standardises and raises the quality of the key materials one provides, so to speak informally.

PBX refers to the private branch exchange, and is part of a business phone system(a possibly internal phone network) [7]. A PBX may serve as a information relay or a proxy in VOIP communication. However, PBX is beyond the scope of this report, and it sufficies to know that some part of ZRTP messages are dedicated to negotiating PBX parameters, and PBX is used for underlying systems that facilitate use of ZRTP, and is not a core component of ZRTP.

SAS simply refers to a short piece of alphanumerical text that is displayed to both communicating users and needs to be verified to be match at the start of a call [5, p. 77]. This will be further explained in later sections.

SRTP keys and salts are used for the actual encryption that takes place after key exchange, and the actual usage procedures are beyond the scope of this report [5, p. 33-34].

MAC addresses the need of defending against message forgery by providing assurance about the source and integrity of an object [8]. In other words, to authenticate a message. MAC are attached in some ZRTP messages [5, p. 55-63], but is not a core part of the key exchange protocol, thus any further explanation is beyond the scope of this report. It suffices to know it allows either party to detect forgery of messages, and while MAC depends on the content of message, MAC itself is not confidential - adversary cannot forge messages with legitimate MAC even with knowledge of previous MACs [9].

2.3.4 Confirmation

Encrypted confirmation messages are exchanged as the final part of the key exchange process. The encryption uses the **ZRTP keys** derived in the **Key Generation** stage [5, p. 33-34].

2.4 Modes

The three modes differ by behaviour in **Commit** and **Key Generation** stage.

2.4.1 Diffie-Hellman mode

This mode is the most critical mode as it establishes the initial key materials from scratch, and the other two modes rely on the key materials derived in this mode. Thus DH mode will be the focus of this report.

In other words, given two parties with no prior communication, Diffie-Hellman mode is the only mode that can establish shared key materials, as key exchange only happens in this mode.

This mode uses either finite-field Diffie-Hellman(DH) or Elliptic Curve Diffie-Hellman(ECDH) as the core algorithm. Both algorithms are based on the same principle described in the **Key Agreement Protocol** section. They differ by the mathematics used and the cryptographic explanations are beyond the scope of this report.

In this mode, the initiator also performs **hash commitment** (which will be explained in detail in later section) to reduce probability of a successful attack.

Note that if this mode is used when there are cached shared secrets, the cached secrets may be used as part of the computation of the final new shared secret.

2.4.2 Preshared mode

This mode relies on cached secrets from previous session, and does not rely on DH to compute the shared secret.

This mode is useful for quick re-establishment of communication, as recomputing via DH is expensive and can take a long time. This also benefits low-power devices by reducing number of expensive computations required.

2.4.3 Multistream mode

This mode is mainly used to establish additional media streams between two parties which already have an active SRTP session.

The ZRTP session keys generated previously can be used to derive the keys and salts for the new media streams without performing DH computations.

3 Core security components of ZRTP

3.1 Hash commitment and SAS

This section is dedicated to DH mode, the other two modes are not considered as key exchanges do not happen in the non-DH modes. Furthermore, this section is concerned largely with **Commitment** stage

The following text is based on a paragraph from the standard as shown below, which contains a short description of security properties related to hash commitment and SAS.

"The use of hash commitment in the DH exchange constrains the attacker to only one guess to generate the correct Short Authentication String (SAS) (Section 7) in his attack, which means the SAS can be quite short. A 16-bit SAS, for example, provides the attacker only one chance out of 65536 of not being detected. Without this hash commitment feature, a MiTM attacker would acquire both the pvi and pvr public values from the two parties before having to choose his own two DH public values for his MiTM attack. He could then use that information to quickly perform a bunch of trial DH calculations for both sides until he finds two with a matching SAS. To raise the cost of this birthday attack, the SAS would have to be much longer. The Short Authentication String would have to become a Long Authentication String, which would be unacceptable to the user. A hash commitment precludes this attack by forcing the MiTM to choose his own two DH public values before learning the public values of either of the two parties." [5, pg. 21].

The following text is an attempt of a clearer and more explicit explanation of the above statements by the author of this report.

Note that MiTM attacks of key exchange process without authentication happens easily if the attacker(Eve) can intercept the traffic between the two parties(Alice, Bob), as then Eve can negotiate a separate key with Alice, and a separate key with Bob, allowing itself to decrypt and reencrypt later traffic sent in either direction. This concept has already been explained in **Key Agreement Protocol** section. The following cases argue for how a half-complete authentication(that is, only using SAS, but no hash commitment) still fails.

We first start by clarifying the terms used in following paragraphs, then we will follow with a description of the sequential ordering of events.

- We use the word **"settle"** in the following sections to refer to the act of making some value constant in the process(neither party can change the value later on in the process)
- Commit message is a message that makes one settles on a public value(which in turn implies settling on the corresponding secret as well). This is sent by the Initiator, and contains the hash of the DHPart2 message
- Initiator is whichever party that sends the commit message
- Responder is whichever party that receives the commit message
- A DH message is a message that contains hashes of cached shared secrets of the sending party, and the result of mixing(application of mix functions mentioned in section 1.1) between sending party's secret and public common parameter
  - The cached shared secrets are ignored in the following paragraphs to simplify explanations, as they do not affect the hash commitment and SAS mechanism
  - The standard refers the <u>i</u>nitiator's secret as **sv<u>i</u>**, and the result of mixing as public value **pv<u>i</u>**
  - The standard refers the <u>r</u>esponder's secret as **sv<u>r</u>**, and the result of mixing as public value **pv<u>r</u>**

- The private and private values above map to the names in **Principle** section as follows, with Alice being the Initiator(using **svi**, **pvi**) and Bob being the Responder(using **svr**, **pvr**). The common parameter c is agreed on via Hello messages that are exchanged before everything in this section.

svi	–	$sA =$ secret that ONLY Alice knows
pvi	–	$rA = mix1(c, sA)$
svr	–	$sB =$ secret that ONLY Bob knows
pvr	–	$rB = mix1(c, sB)$

- DHPart1 is the DH message sent from **responder** to **initiator**, which contains **pvr**
- DHPart2 is the DH message sent from **initiator** to **responder**, which contains **pvi**
- Hash commitment refers to the action of sending the hash of entire DHPart2 in the commit message. This is done by the initiator, and forces the initiator to settle on its public value **pvi**. Since if the initiator decides to change the **pvi** later, responder can detect it due to the hash of the new DHPart2 message being different from the one contained in the commit message
- Short Authentication String(SAS) refers to a short piece of text that the two communicating parties need to confirm (verbally) to be matching at the start of SRTP session(a voice call). The text is displayed on whatever device the user may be using. SAS can also be confirmed via use of cryptographic signatures, but overall is still simply an action to make sure they match, thus we will just consider the case where users check them manually

The strict sequential ordering of events are as follows

1. Initiator settles on its public value **pvi**, and generates the DHPart2 message. Then embed the hash of the DHPart2 message into the commit message
2. Initiator sends the commit message, completing hash commitment in process of doing so
3. Responder receives the commit message
4. Responder sends DHPart1
5. Initiator receives DHPart1
6. Initiator sends DHPart2
7. Responder receives DHPart2
8. Responder makes sure the hash of the received DHPart2 message matches the hash embedded in the commit message

Above process contains the hash commitment mechanism in ZRTP, which is part of the core defense against MiTM attacks, and the following paragraphs will only focus on the above steps.

An additional detail reader should keep in mind is that the final secret produced from the above key exchange process will be used to generate the SAS, thus if the users notice their SASs are different, they know that they no longer share the same final secret, which entails that someone has interfered with the key exchange process. However, this is a one way implication : it is possible that they no longer share the same final secret, but still have the same SAS - this is exactly what the attacker wants to achieve.

Thus, we define a successful MiTM attack as follows : the SAS displayed to both users are same, and as a result neither user knows MiTM attack has occured. But the attacker shares the same final secret with them separately, allowing silent eavesdropping by decryption and re-encryption(similar to the example given in **Principle** section.

We now explain the importance of above process by going through all potential scenarios(similar to proof by exhaustion, albeit informal), and how MiTM attacks may be carried out successfully in each of them. Note that we ignore timeouts(we simply assume nothing fails due to timeout), and network specifics etc.

We also introduce a new function to further abstract away the technical details :

$$genSAS(\text{finalsecret}) = \text{SAS to be displayed to user}$$

We also introduce a few notations :

$\underline{\text{var}}$	-	the variable is still under control by someone, not settled yet
~~var~~	-	the actual value of the variable is not known by a certain person

The exact person the notation refers to will be mentioned along with the usage of it.

Without the hash commitment (no commit messages)

case 1 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Initiator		Responder | Initiator		Responder

The events may unfold in the following way which results in Eve's success

1. Eve picks its pvr and svr, denoted as **eapvr** and **easvr**, where the **eapvr** is shared with Alice
2. Eve sends **eapvr** to Alice via DHPart1 message
3. Alice picks its pvi and svi, denote as **apvi** and **asvi**, where **apvi** is shared with Eve
4. Alice sends **apvi** to Eve via DHPart2 message
5. Bob picks its pvr and svr, denoted as **bpvr** and **bsvr**, where **bpvr** is shared with Eve
6. Bob sends **bpvr** to Eve via DHPart1 message
7. Eve now still has <u>**ebpvi**</u> and <u>**ebsvi**</u> under its control, so it can bruteforce the two values such that the following holds, where aSAS = Alice's SAS and bSAS = Bob's SAS

$$
\begin{aligned}
bSAS &= genSAS(mix2(\underline{\mathbf{ebpvi}}, \mathbf{bsvr})) && \text{(Bob's calculation)}\\
&= genSAS(mix2(\mathbf{bpvr}, \underline{\mathbf{ebsvi}})) && \text{(Eve's calculation)}\\
&= genSAS(mix2(\mathbf{apvi}, \mathbf{easvr})) && \text{(Eve's calculation)}\\
&= genSAS(mix2(\mathbf{eapvr}, \mathbf{asvi})) && \text{(Alice's calculation)}\\
&= aSAS
\end{aligned}
$$

8. Eve sends **ebpvi** to Bob, and this results in Alice and Bob seeing the same SAS

Above concludes Eve's successful MiTM attack, and now Eve can eavesdrop or modify Alice's and Bob's traffic silently.

case 2 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Responder		Initiator | Reciever		Initiator

Notice this case is symmetrically identical to our first case, with Alice and Bob swapped, so in a fashion similar to our first case, we argue that Eve can successfully accomplish a MiTM attack.

case 3 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Initiator		Responder | Responder		Initiator

Eve has no way to succeed in this case, which we will show below

1. Eve picks its first pvr and svr, denoted as **eapvr** and **easvr**, where the **eapvr** is shared with Alice
2. Eve sends **eapvr** to Alice via DHPart1 message
3. Eve picks its second pvr and svr, denoted as **ebpvr** and **ebsvr**, where the **ebpvr** is shared with Bob
4. Eve sends **ebpvr** to Bob via DHPart1 message
5. Eve now has no control over any of the remaining variables, thus no attacks are possible

Above concludes Eve's failed MiTM attack case 4 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Responder		Initiator | Initiator		Responder

1. Alice picks its pvr and svr, denoted as **apvr** and **asvr**, where **apvr** is shared with Eve
2. Alice sends **apvr** to Eve via DHPart1 message
3. Bob picks its pvr and svr, denoted as **bpvr** and **bsvr**, where the **bpvr** is shared with Eve
4. Bob sends **bpvr** to Eve via DHPart1 message
5. Eve now has control over $\underline{\textbf{eapvi}}$, $\underline{\textbf{easvi}}$, $\underline{\textbf{ebpvi}}$, and $\underline{\textbf{ebsvi}}$, so it can bruteforce the four values such that the following holds, where aSAS = Alice's SAS and bSAS = Bob's SAS

$$\begin{aligned}
bSAS &= genSAS(mix2(\underline{\mathbf{ebpvi}}, \mathbf{bsvr})) && \text{(Bob's calculation)}\\
&= genSAS(mix2(\mathbf{bpvr}, \underline{\mathbf{ebsvi}})) && \text{(Eve's calculation)}\\
&= genSAS(mix2(\mathbf{apvr}, \underline{\mathbf{easvi}})) && \text{(Eve's calculation)}\\
&= genSAS(mix2(\underline{\mathbf{eapvi}}, \mathbf{asvr})) && \text{(Alice's calculation)}\\
&= aSAS
\end{aligned}$$

6. Eve sends **eapvi** to Alice
7. Eve sends **ebpvi** to Bob
8. This results in Alice and Bob seeing the same SAS

Above concludes Eve's successful MiTM attack, and now Eve can eavesdrop or modify Alice's and Bob's traffic silently.

We see that Eve cannot attack successfully in the 3rd case, but Eve can just force renegotiation whenever it encounters the 3rd case(the exact messages to be sent are beyond the scope of this report), thus Eve always has a chance of successfully attacking Alice and Bob.

Obviously we are assuming Eve can bruteforce the values within timeout period, thus we can potentially void the assumption by making SAS very lengthy. However, this invalidates the original intention of having SAS reasonably short and yet still effective.

With hash commitment

case 1 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Initiator		Responder | Initiator		Responder

1. Alice settles on its pvi and svi, denote as **apvi** and **asvi**, where **apvi** is shared with Eve
2. Alice sends hash of **apvi** to Eve via commit message
3. Eve settles on its first pvi and svi, denoted as **ebpvi** and **ebsvi**, where **ebpvi** is shared with Bob
4. Eve sends hash of **ebpvi** to Bob via commit message
5. Bob picks its pvr and svr, denoted as **bpvr** and **bsvr**, where **bpvr** is shared with Eve
6. Bob sends **bpvr** to Eve via DHPart1 message
7. Eve now still has $\underline{\textbf{eapvr}}$ and $\underline{\textbf{easvr}}$ under its control. but it does not have the actual value of **apvi**, only the hash of it. We denote the variables which Eve lacks knowledge of by crossing them out, and we reiterate the equality Eve needs to uphold

$$\begin{aligned} bSAS &= genSAS(mix2(\textbf{ebpvi}, \textbf{bsvr})) && \text{(Bob's calculation)} \\ &= genSAS(mix2(\textbf{bpvr}, \textbf{ebsvi})) && \text{(Eve's calculation)} \\ &= genSAS(mix2(\cancel{\textbf{apvi}}, \underline{\textbf{easvr}})) && \text{(Eve's calculation)} \\ &= genSAS(mix2(\underline{\textbf{eapvr}}, \textbf{asvi})) && \text{(Alice's calculation)} \\ &= aSAS \end{aligned}$$

8. Now the only way Eve can obtain **apvi** is by sending **eapvr** to Alice via DHPart1 message, but this means Eve has only one chance of picking the correct **eapvr** such that the above equality holds

The standard uses 16-bit SAS as an example, which we will use as well. For a 16-bit SAS, Eve then only has one chance out of 65536 of not being detected, which translates to probability of 0.00153% (rounded to 3 most significant digit).
We consider this to be sufficiently low chance of success, and above concludes Eve's failed attack.

case 2 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Responder		Initiator | Responder		Initiator

Notice this case is symmetrically identical to our first case, with Alice and Bob swapped, so in a fashion similar to our first case, we argue that Eve failed to accomplish a MiTM attack.

case 3 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Initiator		Responder | Responder		Initiator

Since we have concluded Eve has no way to succeed in this case even with no hash commitments, we simply state that Eve has no way to succeed in this case either.

case 4 :

Alice	$\longleftrightarrow$	Eve	$\longleftrightarrow$	Bob
Responder		Initiator | Initiator		Responder

1. Eve settles on its first pvi and svi, denoted as **eapvi** and **easvi**, where **eapvi** is shared with Alice
2. Eve sends hash of **eapvi** to Alice via commit message
3. Alice picks its pvr and svr, denoted as **apvr** and **asvr**, where **apvr** is shared with Eve
4. Alice sends **apvr** to Eve via DHPart1 message
5. Eve settles on its second pvi and svi, denoted as **ebpvi** and **ebsvi**, where **ebpvi** is shared with Bob
6. Eve sends hash of **ebpvi** to Bob via commit message
7. Bob picks its pvr and svr, denoted as **bpvr** and **bsvr**, where the **bpvr** is shared with Eve
8. Bob sends **bpvr** to Eve via DHPart1 message
9. Eve now has no control over any variables as it has already settled down on **eapvi**, **easvi**, **ebpvi**, and **ebsvi** via hash commitment. Reiterating the equality that needs to be upheld

$$\begin{aligned} bSAS &= genSAS(mix2(\mathbf{ebpvi}, \mathbf{bsvr})) && \text{(Bob's calculation)} \\ &= genSAS(mix2(\mathbf{bpvr}, \mathbf{ebsvi})) && \text{(Eve's calculation)} \\ &= genSAS(mix2(\mathbf{apvr}, \mathbf{easvi})) && \text{(Eve's calculation)} \\ &= genSAS(mix2(\mathbf{eapvi}, \mathbf{asvr})) && \text{(Alice's calculation)} \\ &= aSAS \end{aligned}$$

   We see that Eve now can only bet on **apvr** and **bpvr** to be of certain values such that the above equality holds. Similar to the first case, we know that the chance is extremely slim.

Above concludes Eve's failed MiTM attack.

Conclusion

From above, we see that hash commitment forces Eve to have exactly only one chance to pick the correct public values to accomplish a successful MiTM attack. Thus even picking a relatively short SAS length(16-bit in above case) still allows Alice and Bob to be pracitcally immune to MiTM attacks, given that they do verify the SAS.

4 Aspects which are not of primary concerns

4.1 Algorithm selection

Algorithm selection is limited to the intersection of the sets of supported/accepted algorithms provided by the two parties[5, pg. 13-14].

Thus either party can make the acceptance criteria as strict as the security policy requires without compromise.

In other words, there is no possibility of downgrade attacks even with message forgery.

4.2 Rejection of false messages

The following text is based on section 9 of the standard[5, pg. 95-96].

This is mainly to deal with situation where the attacker does not lie in the middle of the media path(no MiTM attacks), but sends bogus/false messages to either side with the intention to disrupt, i.e. denial-of-service(DoS) attack.

While rejection of false messages can be done after detection of mismatching shared secret, this has already passed the computationally expensive DH calculations.

In order to reject false messages with lower cost, all messages contain a hash image, which is a hash truncated to 256-bit.

The hash images H0, H1, H2, H3 are generated as follows

- H0 is generated randomly by the sending party
- H1 = hash (H0)
- H2 = hash (H1)
- H3 = hash (H2)

Note that above forms a one-way hash chain.

The above hash images are used in reverse order, that is, Hello message contains H3, Commit message contain H2, DH message contains H1, and the final confirm message contains H0.

Since H0 is unpredictable due to being randomly generated, there is no way to pre-determine H1-H3 either by the attacker.

Note that any missing hash image in the chain can be inferred given the previous hash image is known. For example, in non-DH modes, DH messages are never sent, thus H1 is never actually received, but since H0 is received in the confirm messages, H1 can be generated by hashing H0, and then used to verify H2, completing the chain.

Again, since this tackles DoS attacks rather than attacks that undermines secrecy of traffic, it is not the focus of this research project.

Similarly, tampering of messages will likely also lead to requirement of resending traffic or renegotiation, but this is again DoS attack rather than an attack on the protocol itself, thus is beyond the scope of this project.

Furthermore, in general, DoS attacks cannot be prevented entirely.

References

[1] *Diffie–hellman key exchange*, in. [Online]. Available: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange (visited on 08/31/2017).

[2] K. A. Baker, *Diffie-hellman key exchange*, 1999. [Online]. Available: http://www.math.ucla.edu/~baker/40/handouts/rev_DH/node1.html.

[3] M. Baugher, D. A. McGrew, M. Naslund, E. Carrara, and K. Norrman, *The secure real-time transport protocol (SRTP)*, Mar. 2004. [Online]. Available: https://tools.ietf.org/html/rfc3711.

[4] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, *RTP: A transport protocol for real-time applications*, Jul. 2003. [Online]. Available: https://tools.ietf.org/html/rfc3550.

[5] P. Zimmermann, A. Johnston, and J. Callas, *ZRTP: Media path key agreement for unicast secure RTP*, Apr. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6189.

[6] H. Krawczyk, *Cryptographic extraction and key derivation: The HKDF scheme*, 2010. [Online]. Available: http://eprint.iacr.org/2010/264.

[7] K. Bartley, *What is a PBX phone system?*, Jul. 31, 2017. [Online]. Available: https://www.onsip.com/blog/what-is-pbx-phone-system.

[8] F. B. Schneider and T. Roeder, *Hashes and message digests*, 2005. [Online]. Available: http://www.cs.cornell.edu/courses/cs513/2005fa/NL20.hashing.html.

[9] I. Koren, *Introduction to cryptography ECE 597xx/697xx part 12 message authentication codes (MACs)*. [Online]. Available: http://euler.ecs.umass.edu/ece597/pdf/Crypto-Part12-MAC.pdf.