INTEGRATING SECURE RTP INTO THE OPEN SOURCE VOIP PBX ASTERISK

Bradley Clayton, Barry Irwin, Alfredo Terzoli

Computer Science Department
Rhodes University
Grahamstown

g01c2974@campus.ru.ac.za, b.irwin@ru.ac.za a.terzoli@ru.ac.za

ABSTRACT
Implementations of Voice over Internet Protocol (VoIP) have focused, up to now, mainly on the need to transport data in real-time, often at the expense of security. The neglect of secure VoIP is often intentional, as developers are striving to minimise overheads and delays. The Secure Real-Time Protocol (SRTP) has the potential to secure real-time streams without exacting too high a performance price. SRTP is the addition of security to the audio/video profile used in the Real-Time Transport Protocol (RTP). SRTP adds confidentiality, integrity and optionaly authenticity to RTP media streams. This paper focuses on the integration of SRTP into Asterisk, an open-source VoIP PBX. SRTP support has recently been added to Asterisk by Mikael Magnusson. This paper analyses Magnusson's implementation, contrasting it to a proof-of-concept implementation developed independently at Rhodes University. The interoperability of SRTP implementations cannot be taken for granted, given the relatively recent standardization of the protocol, and so Magnusson's implementation is tested against another SRTP implementation. Finally, the paper highlights a major shortcoming in Magnusson's implementation, namely that the exchange of encryption keys is done in the clear. It concludes by proposing possible solutions, such as TLS, IPSec and MIkey.

KEY WORDS
VoIP, Secure RTP, Multimedia Internet Key Exchange (MIKey) Asterisk

INTEGRATING SECURE RTP INTO THE OPEN SOURCE VOIP PBX, ASTERISK

1 INTRODUCTION

VoIP implementations focus on the need to transport data in real-time, concentrating on codecs and low latency, often at the expense of security. The neglect of security is intentional: to minimise overheads and delays, security is acknowledged but is deemed out the of the scope of the implementation or marked as future work.

In preparation for an investigation into the performance of VoIP security methods [1], secure VoIP implementations had to be written where none were available. The Secure Real-time Transport Protocol (SRTP) [2], had only been implemented in peer-to-peer VoIP clients, such as a SIP client called MiniSIP [3]. Our research required SRTP to be tested in a client-server architecture where the server also was SRTP-enabled. Therefore, we implemented SRTP into Asterisk, the Open Source PBX [4]. (Asterisk was chosen because of its compatibility over a wide range of VoIP protocols as well as our good knowlege of it.) Shortly after completing our implementation, Mikael Magnusson completed an equivalent implementation.

The structure of the paper is as follows. Section 2 introduces SRTP and explains its design and suitability for VoIP applications. Section 3 describes a library which provides SRTP functionality and was used for our integration of SRTP into Asterisk. Appropriate areas of the Real-time Transport Protocol (RTP) [5] structure within Asterisk, for SRTP integration, are discussed in section 4. Magnusson's contribution and our own local implementation are discussed in section 5. SRTP provides confidentiality, integrity and authentication, however, another mechanism for key exchange is required. An insecure key exchange compromises all of the security provided by SRTP. Appropriate methods of key exchanges are discussed in section 6. Finally, section 7 successfully tests the interoperability of Magnusson's implementation with existing SRTP clients.

2 THE SECURE REAL-TIME TRANSPORT PROTOCOL

Unless otherwise stated, this section is based on the SRTP RFC [2]. The SRTP protocol is merely a secure audio/video profile for RTP [6] that offers confidentiality, integrity and authentication for video and audio streams. The secure profile is designed to exist between the RTP application and underlying transport layers. Media packets moving down an RTP stack are intercepted and *secured*, into SRTP packets, before being passed to the transport layer. Conversely, SRTP packets moving up the stack are *unprotected*, into RTP packets, and passed to the application. Real-Time Control Protocol (RTCP) packets are converted into Secure Real-Time Control Protocol (SRTCP) packets in the same way.

The aim of SRTP is to ensure the confidentiality, integrity and authenticity of RTP and RTCP payloads, hence addressing the security needs for real-time multimedia applications. This is achieved through a framework that allows for the upgrade to new cryptographic algorithms, while maintaining a minimal overhead. The SRTP RFC lists the Advanced Encryption Standard (AES) and NULL (or

no encryption) ciphers as being appropriate . Scoping the selection of cryptography algorithms to ones which exhibit low computational costs and a small footprint make SRTP ideal for mobile or small embedded devices, such as, telephone handsets.

SRTP currently uses HMAC-SHA1 [7] for authentication and integrity. SRTP is vulnerable to payload adjustment and source spoofing when message authentication is not utilized. For this reason, SRTP message authentication must always be enabled. SRTP should also be protected with a strong authentication code. However, the sole use of integrity protection will not protect communications from replay attacks.

SRTP counters replay attacks through the use of a sliding window and Replay List. The Replay List contains an index of all packets which have been received and authenticated. Upon receiving a packet, its index is compared to a list of recent packet indexes. The packet is rejected if its index is smaller then the index of the last received packet, less the size of the sliding window. The sliding window allows the protocol to use a fixed amount of memory for replay.

SRTP uses stream ciphers which are vulnerable to statistical attacks if an attacker assumes the presence of formatting bits encrypted within an intercepted payload. The length of a stream cipher payload is always known, making it possible to use the known formatting bits and their position to derive the corresponding bit of the key-stream. However, it is claimed in RFC 3711 that an attacker will not be able to use these known bits to deduce the rest of the stream if the cipher is secure.

SRTP avoids denial or service attacks by using seekable stream ciphers. This means that a cipher is able to address any position in its key-stream. This feature enables the encryption or decryption of a packet without having to depend on preceding packets.

3 AN SRTP LIBRARY

The integration of SRTP into Asterisk, discussed in section 5, utilised a library called libSRTP [8].

The library was selected for the following reasons:

- Like Asterisk, the library is open source, making its inner workings easily accessible.
- Also, like Asterisk, the library is written in C. This allows us to perform the integration without the need for language wrapping.

libSRTP [8] is maintained by David McGrew (also co-author of RFC3711) from Cisco Systems. The library is licensed in the open source domain with its aim being to promote the use of SRTP in various applications. The library supports all mandatory features defined in the SRTP RFC [2]. However, some of the optional features, including the list below, are not supported as of version 1.4:

- The Master Key Index (MKI).
- Key derivation rate other than zero.
- The F8 mode of AES.
- Anti-replay lists with sizes other than 128 packets.
- The use of the packet index to select between master keys.

When securing RTP and RTCP data, the data is categorized into sessions and streams: more than one concurrent stream is termed a session. Cryptographic options, keys, and stream addressing are configured through policies. A stream is configured by the instantiation of a policy which is uniquely identified by a Synchronization Source Identifier (SSRC). The SSRC field is already used in the RTP protocol to identify multiple streams on a single host. This enables the SRTP engine to match incoming or outgoing RTP packets with stream policies. libSRTP allows the developer to create multiple streams within a single session. A session is created when more than one stream and stream policy exists (libSRTP represents a session as a linked list of stream policies [8]). Policies are defined in the library by a data structure containing the following fields:

- CIPHER_TYPE, an integer representing the type of cipher that should be used for confidentiality.
- CIPHER_KEY_LEN, the length of the cipher key in octets.
- AUTH_TYPE, denoting the authentication function to be used.
- AUTH_KEY_LEN, length of the authentication function key in octets.
- AUTH_TAG_LEN, length of the authentication tag in octets.
- SEC_SERV, a flag representing security services to be applied.

Linked list nodes are described by the following structure:

```
Type struct
srtp_policy_t {
ssrc_t SSRC
crypto_policy_t RTP
crypto_policy_t RTCP
octet_t *KEY
srtp_polity_t *NEXT
}
```

The SSRC value, as mentioned, associates a policy to a secure stream. The RTP and RTCP values hold the policies for the real-time payload and real-time control data. KEY points to memory which contains the cryptography key for encrypting or decrypting the associated RTP and/or RTCP data. Lastly, the NEXT pointer addresses the next policy instance, or NULL to denote the end of the list.

libSRTP is initialized when the SRTP_INIT() function is called. To create a new SRTP session, SRTP_CREATE() is called and passed a session pointer and a populated policy structure. Once a session is created, streams can be added to the session by calling the SRTP_ADD_STREAM() function, passing it the session pointer and new stream policy. A session or stream is de-allocated when the SRTP_REMOVE_STREAM() and SRTP_DEALLOC() functions are called and passed a session pointer and SSRC value.

Once a session with one or more stream policy is in place, RTP and RTCP data can be protected. The SRTP_PROTECT() function is responsible for the authentication and encryption of RTP and RTCP packets streams. For each RTP or RTCP packet, the function is passed a session pointer, RTP data pointer (addressing the packet to be protected) and the length of the packet. The function returns

the RTP pointer addressing the authenticated and encrypted packet. Should the function fail, an appropriate error code is returned which can be looked up in a table [8].

Similarly, at the receiving end of an RTP or RTCP stream, the SRTP_UNPROTECT() function is used to authenticate and decrypt a packet. For each RTP or RTCP packet, the function is passed a session pointer, RTP data pointer addressing the packet to be unprotected and a pointer to the length of the packet. The function returns the RTP pointer addressing the authenticated and encrypted packet. The un-protect function will authenticate each packet and return an ERR_STATUS_AUTH_FAIL code if the packet has been tampered with. Packet replay is detected when the ERR_STATUS_REPLAY_FAIL code is returned.

4 ASTERISK

Asterisk performs all RTP and RTCP processing in the RTP.C file, while Session Initiation Protocol (SIP) [9] and Session Description Protocol (SDP) [10] handshakes are defined in CHAN_SIP.C. The following were marked as important functions in chan_sip.c:

- SIP_CALL() when a SIP call is to be initiated from the PBX, this function starts the SIP and SDP handshakes.
- ADD_SDP() adds fields to the SDP handshake. SDP fields are used to negotiate specifics such as the audio codec, video codec and, IP and port addressing to be used during a call.
- PROCESS_SDP() processes SDP handshake. The function attends to incoming SDP fields and creates the RTP session appropriately.
- SIP_REGISTRY_DESTROY() un-registers a sip device from the PBX.
- SIP_REGISTER() registers a SIP device on the PBX.

The following were found to be important functions within the asterisk RTP stack:

- AST_RTP_INIT() initializes the RTP system and calls the AST_RTP_RELOAD() function. (This function is called when Asterisk is started.)
- AST_RTP_RELOAD() is called when the RTP stack within Asterisk is reloaded.
- AST_RTP_NEW_WITH_BINDADDR() is called and passed the destination's IP address when creating a new RTP session. The SIP and SDP systems in Asterisk are responsible for obtaining the destination's IP address and RTP ports. This function creates an RTP session.
- AST_RTP_STOP() is executed when an RTP session is not needed any more.
- AST_RTP_RESET() sets all the default RTP session attributes to their initial values, effectively recreating the RTP session.
- AST_RTP_RAW_WRITE() is called when there is an RTP packet that needs to be sent to an RTP receiving node. This function is responsible for passing RTP packets to the computer's network stack.

- AST_RTP_READ() is called when the RTP stack has received a packet. The read function accepts RTP packets, checks for basic data corruption and passes their contents to Asterisk where a decoder converts RTP data into an encoded format for retransmission to the next node.

These functions were marked as areas where libSRTP could be used to create or destroy secure sessions and protect or un-protect RTP streams.

5 INTEGRATING SRTP INTO ASTERISK

In the following discussion, all comments are related to the Rhodes University implementation unless otherwise stated. Implementation begun by making libSRTP available to Asterisk. This caused conflicts with the Inter-Asterisk Exchange (IAX) [11] system within Asterisk, due to cryptography function name clashes in IAX and libSRTP. To resolve this, the convention of appending Asterisk specific functions with *ast_* was used to rename the functions within the IAX system. While this resolved the issue, the solution makes re-implementation of libSRTP into a newer version of Asterisk complicated. Each implementation would require multiple patches to rename the IAX cryptography functions and update areas of code where these functions are called. Magnusson's created a resource module, RES/RES_SRTP.C, which contained all the calls to libSRTP, avoiding the common name problem. (Our implementation is intended to be a proof-of-concept and temporary implementation of the SRTP protocol for Asterisk, while Magnusson's is suitable for redistribution. The use of static cryptography keys is sufficient for our application as we are, ultimately, merely interested in the performance cost of the security addition. Magnusson implemented a key exchange within the Session Description Protocol (SDP) which is discussed in section 6.)

The library initialization and policy configuration of each implementation take place at different times during the setup of a call. Our implementation configures the library and policies when the RTP stack in Asterisk is initialised. Magnusson does the initialisation and policy creation within the Session Description Protocol (SDP) handshake. Firstly, the AST_RTP_RELOAD() function is used to initialize libSRTP. This placement ensures that libSRTP is initialized when Asterisk is started and when the RTP system is reloaded. Policy creation is placed in the AST_RTP_NEW_WITH_BINDADDR() function. Each SRTP session is created with the following attributes:

- The SSRC value assigned by Asterisk is used to address the new SRTP session.
- The cryptography key is assigned statically or after a key exchange. (This is explained further in section 6.)
- The cypher type is set to AES_128_ICM, currently the strongest cipher available to libSRTP.
- For authentication, the function is set to HMAC_SHA1.
- Lastly, the services for the policy are set to perform confidentiality and authentication.

The AST_RTP_STOP() function is a convenient place where SRTP sessions can be removed once they are no longer needed. Likewise, the AST_RTP_RESET() function is used to recreate SRTP sessions.

Both implementations perform the protection of RTP packets in a similar fashion. Our implementation intercepts the RTP packet within the AST_RTP_RAW_WRITE() function. Immediately before

an RTP packet is sent to the network stack, its contents are moved to a larger buffer. The buffer is passed to the SRTP_PROTECT() function where it is encrypted and an authentication header is added. The larger buffer allows SRTP_PROTECT() to increase the size of the RTP payload without running out of memory. The buffer is converted from an RTP payload into an SRTP payload. This buffer is then passed to the network stack to be sent to its destination. Magnusson makes the same call to SRTP_PROTECT(), except within the RTP_SENDTO() function which is called by AST_RTP_RAW_WRITE().

On the receiving end of an SRTP stream, the AST_RTP_READ() function is called and passed a pointer to an SRTP packet. Before any processing, the packet is again copied into a buffer. The SRTP_UNPROTECT() function is called and passed the buffer which is decrypted, converting the SRTP payload into an RTP payload. SRTP_UNPROTECT() also checks the authentication header of the SRTP payload to ensure its integrity. Any authentication error is then logged and the packet is discarded. Finally, a pointer is addressed to the contents of the new buffer, which is then used by Asterisk for further processing. Magnusson performs the same call to SRTP_UNPROTECT(), from the RTP_RECVFROM() which is called by AST_RTP_READ().

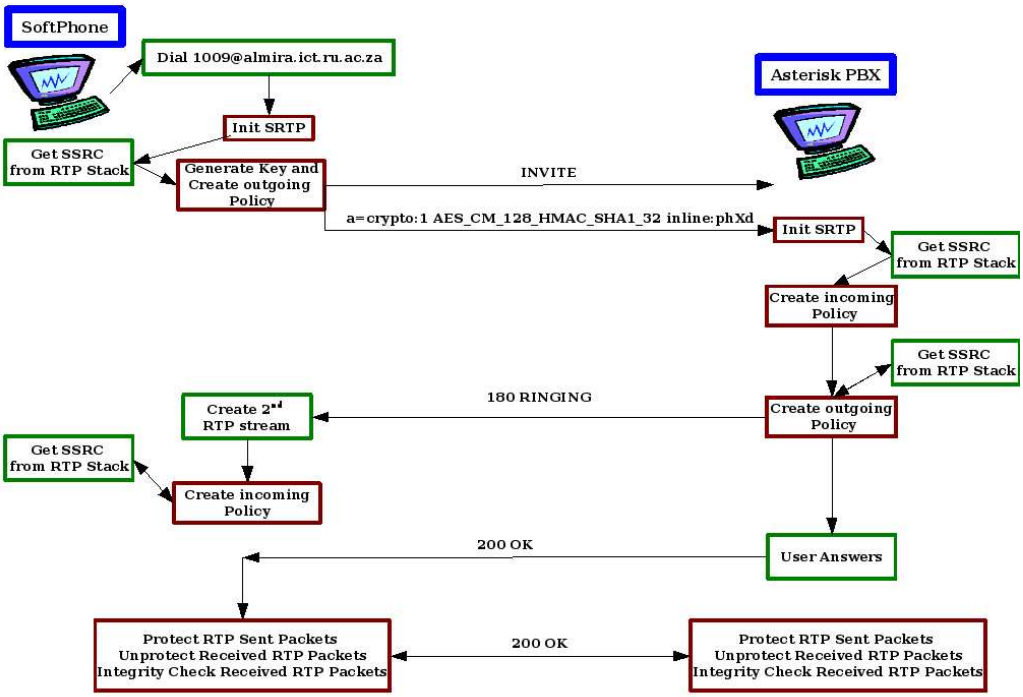

Figure 1: Simple flow of events when creating SRTP streams.
Green - Asterisk events
Red - SRTP events.

6 SECURING THE KEY EXCHANGE

As already mentioned in the previous section, our integration of SRTP into Asterisk makes use of static cryptography key assignment. This method is secure as long as the key is never transmitted in an insecure channel, but is not feasible should we wish to periodically change the key. Ideal key management for our implementation would involve a key exchange while a call is being setup. In this section we focus exclusively on Magnusson's implementation, which allows for such a key exchange to take place. Magnusson has implemented Secure Descriptions (sdescriptions) [12], a standard that defines how cryptography information can be shared during the SIP/SDP handshake of a VoIP session in order to instantiate SRTP communication. The encryption type, message authentication method and cryptographic key is sent by the caller to the callee during the SDP handshake.

```
rtpmap=0 PCMU/8000
rtpmap=97 iLBC/8000
rtpmap=3 GSM/8000
rtpmap=8 PCMA/8000
rtpmap=101 telephone-event/8000
silencesupp=off
crypto=1 AES_CM_128_HMAC_SHA1_80 inline:+dHGQ4ngRj7oz3kAnH0PmCx.......
```

The segment above is taken from an SDP packet. The RTPMAP lines describe a list of audio codecs supported by the caller. SILENCESUPP=OFF declares that the caller will not make use of silence suppression. The CRYPTO line tells the callee that the caller will be using 128bit AES to encrypt RTP payloads. It also defines HMAC SHA1 as the algorithm to authenticate RTP payloads. Finally, it defines the encryption key as "`dHGQ4ngRj7oz3kAnH0PmCxFO7VtgJIqHcieXFyf`". The sdescriptions protocol allows us to change encryption methods and, more importantly, it transports fresh encryption keys for every call. However, SIP and SDP communication is done in clear text, exposing the method and key we exchanging. This introduces a third and final requirement: a method to secure the key exchange.

This requirement could be addressed by encapsulating the SIP and SDP handshake into another secure protocol, of which many are available but none is particularly appropriate. The Encapsulated Security Protocol (ESP), a protocol used within IPSec VPN's [7], could be used. Instantiating an IPSec session requires its own key exchange which, in this case, is excessive as we merely wish to exchange a second key for SRTP. This method would increase the time taken from dialing a number to the remote telephone ringing. Transport Layer Security (TLS) [13] could be used instead. TLS requires certificates and, when properly implemented, will perform a third party authentication check. This adds an additional delay to the call setup. Another drawback of TLS is that, currently, TLS can only be transported within the TCP protocol. The TCP protocol does not suit real-time applications as it has built-in retransmission mechanisms and large over-heads. However, a draft RFC has been written [14] which proposes Datagram Transport Layer Security (DTLS) for UDP. Should DTLS become a reality, SDP handshakes could be transported over UDP with transport layer security.

To successfully solve the key exchange problem, we need a protocol that is able to securely exchange a key with minimal or no overhead. Working in a client-server environment, one could employ

a method where the key is exchanged at client registration. This method would allow for the key to be available, with no delay, before any call is created. However, this method would need to expire and re-create keys, possibly on SIP re-registrations.

A better solution is to use the Multimedia Internet Keying (MIKey) protocol (RFC 3830) [15] is designed to securely exchange cryptography keys for multimedia sessions. MIKey makes use of the Diffie-Hellman key exchange [7], which allows us to derive a symmetric key between two parties. The key is never transmitted in the clear and the use of intercepted information to generate the key is nearly impossible [7]. It is important to note that, without authentication, Diffie-Hellman is vulnerable to man-in-the-middle attacks. Diffie-Hellman is able to derive a key, between two parties, within three exchanges [16]. This allows us to perform a key exchange during a SIP/SDP handshake, which also requires three messages, eliminating any delay in call setup. The next step in the integration of SRTP into Asterisk would be to implement MIKey instead of exchanging keys within SDP.

7 INTEROPERABILITY OF MAGNUSSON'S IMPLEMENTATION

To evaluate the interoperability of Magnusson's integration, calls were made to and from a third party implementation of SRTP and the SDP key exchange. For this test we used the Snom360 softphone [17]. The Snom softphone allows us to create a secure call without encrypting the SDP key exchange. Other phones, for example MiniSIP [3], require a certificate to secure the SDP key exchange. The Asterisk source code was edited in order to generate a log of all library calls made to libSRTP. The segment below was generated when a call was made from Asterisk to the Snom softphone.

```
Apr 24 07:52:07 NOTICE[32234]:  rtp.c:459 ast_srtp_policy_set_master_key:  SRTP
Setting master key
Apr 24 07:52:07 NOTICE[32234]:  rtp.c:459 ast_srtp_policy_set_master_key:  SRTP
Setting master key
Apr 24 07:52:07 NOTICE[32234]:  rtp.c:413 ast_rtp_add_srtp_policy:  Registering SRTP
Adding a Policy
Apr 24 07:52:07 NOTICE[32234]:  rtp.c:413 ast_rtp_add_srtp_policy:  Registering SRTP
Adding a Policy
Apr 24 07:52:07 NOTICE[32234]:  rtp.c:439 ast_srtp_policy_destroy:  SRTP destroying
policy
Apr 24 07:52:07 NOTICE[32234]:  rtp.c:439 ast_srtp_policy_destroy:  SRTP destroying
policy
-- Executing Ringing("SIP/1009-7b71", "") in new stack
-- Executing Dial("SIP/1009-7b71", "SIP/7534@sip.ict.ru.ac.za") in new stack
Apr 24 07:52:07 NOTICE[9096]:  rtp.c:1214 ast_rtcp_new:  RTCP Init
-- SIP/sip.ict.ru.ac.za-e131 is ringing
-- SIP/sip.ict.ru.ac.za-e131 answered SIP/1009-7b71
```

To further confirm the interoperability of Asterisk and Snom, the network was sniffed while the call was setup. These excerpts were collected when a call was made from the Snom softphone to Asterisk. The following table shows the successful SIP negotiation. The segment after the table shows the key exchange within the SDP handshake.

Source	Destination	Protocol	Description
Snom	Asterisk	SIP/SDP	Request: INVITE sip:7534@almira.ict.ru.ac.za;user=phone, with session description
Asterisk	Snom	SIP	407 Proxy Authentication Required
Snom	Asterisk	SIP	ACK sip:7534@almira.ict.ru.ac.za;user=phone
Snom	Asterisk	SIP/SDP	INVITE sip:7534@almira.ict.ru.ac.za;user=phone, with session description
Asterisk	Snom	SIP	100 Trying
Asterisk	Snom	SIP	180 Ringing

```
62 2.473182 146.231.121.208 146.231.123.45 SIP/SDP Request:  INVITE
Media Attribute (a):  crypto:1 AES_CM_128_HMAC_SHA1_32 inline:pHHprnxpQZPqdJLoQVa...
Media Attribute Fieldname:  crypto
Media Attribute Value:  1 AES_CM_128_HMAC_SHA1_32 inline:pHHprnxpQZPqdJLoQVaDw...
```

From these tests we can conclude that Magnusson's integration of SRTP into Asterisk is compatible with at least one other implementation using SDP headers for the key exchange and SRTP to ensure confidentiality and integrity.

8 CONCLUSION

The computational cost is not too high when providing confidentiality, integrity and authenticity between two endpoints in a peer-to-peer VoIP system. On the other hand, a client-server topology requires the server to manage at least two streams per client, the performance impact of security is significant on a server and potentially problematic when transferring data in real-time.

This paper introduces SRTP, a security protocol designed to suit the real-time and resource intensive needs of VoIP. The integration of SRTP into Asterisk is described and its successful implementation is proved through inspection. Interoperability is confirmed by testing the integration with another implementation of SRTP. However, future work is needed as the encryption keys are exchanged via an SDP header, therefore, keys are exchanged in the clear and can be captured easily. MIKey is a promising secure key exchange protocol which, like SRTP, is designed with real-time-multimedia applications in mind. The addition of MIKey to the SRTP-Asterisk integration is recommended.

ACKNOWLEDGEMENTS

This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA, Business Connexion, Comverce, Verso Technologies, THRIP, the National Research Foundation and the German Academic Exchange Service.

References

[1] B. Clayton, A. Terzoli, and B. Irwin, "Performance Cost in Securing Confidentiality, Integrity and Authenticity of VoIP Communications," in *Proceedings of SATNAC 2005 - Convergence - Can technology Deliver*, September 2005.

[2] D. McGrew, E. Carrara, M. Baugher, M. Naslund, and K. Norrman, "RFC 3711: The Secure Real-time Transport Protocol (SRTP)," tech. rep., Cisco Systems, Inc and Ericsson Research, March 2004.

[3] I. Abad, "Secure mobile voip," Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, June 2003.

[4] J. Penton and A. Terzoli, "Asterisk: A Converged TCM and Packet-based Communications System," in *Proceedings of SATNAC 2003 - Next Generation Networks*, September 2003.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RFC 1889 - RTP: A Transport Protocol for Real-Time Applications," tech. rep., GMD Fokus, Precept Software Inc, Xerox Palo Alto Research Center, Lawrence Berkeley National Laboratory, January 1996.

[6] S. Casner and H. Schulzrinne, "RFC 3551: RTP profile for Audio and Video Conferences with Minimal Control," tech. rep., Columbia University, Packet Design, July 2003.

[7] C. R. Davis, *IPSec. Securing VPNs*. Berkeley, California: Osborne/McGraw-Hill, 2001.

[8] D. McGrew, "libSRTP 1.4 Overview and Reference Manual," tech. rep., Cisco Systems, Inc, 2001.

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "RFC 3261: SIP Session Initiation Protocol," tech. rep., Dynamicsoft, Columbia U., Ericsson, WorldCom, Neustar, ICIR, AT&T, June 2002.

[10] M. Handley and V. Jacobson, "RFC 2327: SDP: Session Description Protocol," tech. rep., ISI, LBNL, April 1998.

[11] M. Spencer and F. Miller, "Inter-Asterisk EXchange (IAX) Version 2," tech. rep., Digium (Inc), Cornfed Systems (LLC), July 2005.

[12] F. Andreasen, M. Baugher, and D. Wing, "Session Description Protocol Security Descriptions for Media Streams ," tech. rep., Cisco Systems, Inc, September 2005.

[13] K. Ono and S. Tachimoto, "SIP signaling security for end-to-end communication," *Communications*, vol. 3, pp. 1042–1046, September 2003.

[14] J. Fischl and H. Tschofenig, "Session Description Protocol (SDP) Indicators for Datagram Transport Layer Security (DTLS)," tech. rep., CounterPath Solutions, Inc. and Siemens, Feburary 2006.

[15] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman, "MIKEY: Multimedia Internet KEYing," tech. rep., Ericsson Research, August 2004.

[16] M. Steiner, G. Tsudik, and M. Waidner, “Refinement and extension of encrypted key exchange,” *SIGOPS Oper. Syst. Rev.*, vol. 29, no. 3, pp. 22–30, 1995.

[17] “Snom VoIP Phones Available - http://www.snom.com,” 2006.