Università degli Studi di Padova
Facoltà di Ingegneria

Corso di Laurea Magistrale
in Ingegneria delle Telecomunicazioni

SECURITY FOR THE SIGNALING PLANE OF THE SIP PROTOCOL

Candidato:
Gianluca Caparra
1012713

Relatore:
Nicola Laurenti

A.A. 2012/13

1 Summary

1 Summary 1
2 List of acronyms 3
3 Introduction 7
3.1 Outline 7
4 Description of underlying protocols 9
4.1 Session Initiation Protocol - SIP 9
4.2 IP security - IPsec 10
4.2.1 Encapsulating Security Payload - ESP 10
4.2.2 Authentication Header - AH 10
4.2.3 Operational modes 11
4.2.4 Internet Key Exchange - IKE 12
4.2.5 Standard requirements 13
4.3 Transport Layer Security - TLS 14
4.3.1 TLS cryptographic algorithms 14
4.3.2 Description of TLS handshake 15
5 Security provisions in SIP 17
5.1 Mandatory mechanisms 17
5.1.1 Application layer security approach 17
5.1.1.1 HTTP digest authentication 17
5.1.1.2 Secure Multipurpose Internet Mail Extensions - S/MIME 19
5.1.2 Lower layer security approach 20
5.1.3 SIPS URI 21
5.2 Proposed mechanisms 21
5.2.1 Datagram Transport Layer Security - DTLS 21
5.2.2 HTTP digest authentication with Authentication and Key Agreement 21
5.2.3 Certificateless public-key cryptography 22
5.2.3.1 Details of cerfiticateless public-key cryptography 23
5.2.3.2 Application of cerfiticateless public-key cryptography to SIP 24
5.3 Implemented mechanisms 25
5.4 Threats and known vulnerabilities 25
5.4.1 Main attacks 25
5.4.2 Weakness of HTTP Digest Authentication 28
5.4.3 IPsec and TLS 29
6 Description of the experiments 31

6.1 Environment ... 31
6.2 Measures ... 34
6.3 Note ... 35
7 Analysis ... 37
7.1 Code profiling ... 37
7.1.1 Kamailio percentage ... 37
7.1.2 Security percentage ... 38
7.1.3 Operating frequency ... 39
7.1.4 Profiling overhead ... 39
7.2 Power consumption ... 40
7.3 Load test ... 43
7.3.1 Achievable throughput ... 43
7.3.2 Response time ... 46
7.4 Computational power gain over times ... 48
7.5 IKE overhead ... 50
7.6 Comments ... 51
8 Conclusions ... 53
9 Bibliography ... 55
10 Appendix A: Kamailio configuration ... 57
10.1 Kamailio.cfg ... 57
10.2 Tls.cfg ... 59
11 Appendix B: SIPp configuration ... 60
11.1 Register.xml ... 60
11.2 Register-whith-HTTP-digest.xml ... 60
11.3 User.csv ... 61
12 Appendix C: IPsec-tools configuration ... 62
12.1 Racoon.conf ... 62
12.2 Ipsec-tools.conf ... 62

2 List of acronyms

AAA	Authentication, Authorization and Accounting: security architecture for distributed systems, which enables control of access to a service or a resource.
AES	Advanced Encryption Standard: block cipher standardized by the US NIST in 2001, based on the Rijndael cipher, with key size of 128, 192, or 256 bits.
AH	Authentication Header: a protocol of the IPsec suite, provides connectionless integrity and data origin authentication for IP datagrams and optionally an anti-replay service.
AKA	Authentication and Key Agreement: procedures for mutual authentication of the mobile station and serving system and session-key distribution in UMTS networks.
ARP	Address Resolution Protocol: protocol used for resolution of network layer addresses into link layer addresses.
ASCII	American Standard Code for Information Interchange: a character-encoding scheme.
CBC	Cipher Block Chaining: mode of operation for block ciphers that provides confidentiality but not message integrity. Each block of plaintext is XORed with the previous ciphertext block before being encrypted.
CL-PKC	CertificateLess Public-Key Cryptography.
CPU	Central Processing Unit.
CTR	CounTeR mode: mode of operation for block ciphers that turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter".
DES	Data Encryption Standard: block cipher standardized by the FIPS in 1977, based on the Feistel cipher, with key size of 56 bits.
DH	Diffie-Hellman: non-authenticated key agreement protocol, based on the discrete logarithm problem.
DHE	Diffie-Hellman Ephemeral: mode of operation for the DH key exchange.
DoS	Denial of Service: attacks that make a machine or network resource unavailable to its intended users.
DSA	Digital Signature Algorithm: FIPS standard for digital signatures, is a variant of the ElGamal Signature Scheme, based on the discrete logarithm problem.
DSS	Digital Signature Standard: FIPS standard for digital signature that uses the DSA.
DTLS	Datagram Transport Layer Security: datagram oriented counterpart of TLS.
DTMF	Dual-Tone MultiFrequency: technique for telecommunication signaling over analog telephone lines.
ESP	Encapsulating Security Payload: a protocol of the IPsec suite, provide confidentiality, data origin authentication, connectionless integrity and an anti-replay service for IP packets.
FIPS	Federal Information Processing Standard.
GPU	Graphics Processing Unit.

HMAC	keyed-Hash Message Authentication Code: construction for calculating a MAC using a cryptographic hash function and a secret key.
HTTP	HyperText Transfer Protocol.
HTTPS	HTTP Secure: protocol for secure communication over HTTP.
ICMP	Internet Control Message Protocol: part of the Internet Protocol Suite, designed for diagnostic or control purposes.
IETF	Internet Engineering Task Force.
IKE	Internet Key Exchange: protocol used to set up SA.
ISP	Internet Service Provider.
ITU	International Telecommunication Union.
KGC	Key Generating Centre.
MAC	Message Authentication Code: tag used to authenticate a message and to provide integrity and authenticity assurances.
MAC	Media Access Control address: unique identifier assigned to network interfaces.
MD5	Message-Digest Algorithm: cryptographic hash function that produces a 128 bits hash value, designed by Rivest in 1991.
MIME	Multipurpose Internet Mail Extensions: protocol for sending other kinds of information different from text, in email such as files containing images, sounds, videos.
NIC	Network Interface Card.
NIST	National Institute of Standards and Technology.
OS	Operating System.
PBX	Private Branch eXchange: telephone exchange that serve an office.
PKI	Public-Key Infrastructure: a system for the creation, storage, distribution and revocation of digital certificates.
POTS	Plain Old Telephone Service.
PSTN	Public Switched Telephone Network.
RADIUS	Remote Authentication Dial-In User Service: networking protocol that provides AAA management.
RC4	Rivest Cipher 4: stream cipher designed by Rivest in 1987, with variable key-length usually between 40 and 128 bits.
RFC	Request For Comments: memorandum published by the IETF describing methods, protocols or standards applicable to the working of the Internet.
RSA	Rivest Shamir Adleman: algorithm for public-key cryptography, based on the factoring large integers problem, proposed in 1977, with variable key-length usually between 1024 and 4096 bits.
RTP	Real-time Transport Protocol: transport layer protocol designed for end-to-end real-time data transfer.
S/MIME	Secure Multipurpose Internet Mail Extensions.
SA	Security Associations: shared security attributes between two network entities to support secure communication.
SADB	Security Association DataBase.
SAKA	Secure Authentication and Key Agreement.

SCTP	Stream Control Transmission Protocol: a transport layer protocol.
SDP	Session Description Protocol: format for describing streaming media initialization parameters.
SHA	Secure Hash Algorithm: family of cryptographic hash functions published by the NIST. According to the algorithm produces a 160-512 bits hash value.
SIP	Session Initiation Protocol.
SPD	Security Policy Database.
SRTP	Secure Real-time Transport Protocol: secure counterpart of RTP, provide encryption, message authentication, integrity and replay protection to the RTP stream.
SSL	Secure Sockets Layer: predecessor of TLS originally designed in 1995 by Netscape Communications.
TACACS+	Terminal Access Controller Access-Control System Plus: networking protocol that provides AAA management.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security: protocols that provide communication security over the Internet, first defined in 1999 and last updated in 2008.
Triple DES	variant of DES that increase the key size to 168 bits applying three times DES to each data block.
UA	User Agent.
UAC	User Agent Client.
UAS	User Agent Server.
UDP	User Datagram Protocol.
UMTS	Universal Mobile Telecommunications System.
URI	Uniform Resource Identifier.
VAS	Value-Added Service.
VOIP	Voice Over IP.

6 - List of acronyms

3 Introduction

When we think of the telephonic service we probably think of the traditional analogic telephonic service, the Plain Old Telephone Service (POTS), but in recent years service providers are changing their offers to a new kind of offer, the Voice Over IP (VOIP). This is a digital telephonic service whereby the voice is transported over a packet switched network based on the IP architecture, instead of the traditional system where the voice is transported in analogic mode over the Public Switched Telephone Network (PSTN), a circuit switched network.
VOIP systems are based mainly on two protocols: H.323 and Session Initiation Protocol (SIP). The former is a recommendation from the International Telecommunication Union (ITU), while the latter is a standard proposed from the Internet Engineering Task Force (IETF). I will focus on SIP, because at the moment its popularity is continuously growing mainly as a result of its lesser complexity and better integration with the IP stack.
The diffusion of the VOIP technology is rapidly increasing as a result of its many benefits, one of which is flexibility. With an inexpensive and small equipment, such as an old PC and a switch, it is possible to setup a Private Branch eXchange (PBX) and use many features that were once difficult to have with the traditional telephone service (such as pickup groups, conference calls, video calls, mobility of end-points).
A very attractive feature of VOIP is the fact that it is cost-effective: with an internet data contract and an Internet Service Provider (ISP) it is possible to make free calls with other VOIP users; this is one of the main reason as to why VOIP software like SKYPE is so successful. In addition service providers are changing their offers from PSTN to VOIP, especially on the business environment, making it possible for call centers for instance to simply use an internet connection instead of using several telephone lines, providing the same quality of the traditional phone system. Also according to the telecommunications company point of view the benefits are huge: for example, in the above-mentioned case of the call center, the use of single data stream in place of various copper cables is an enormous advantage.
However, there are also several disadvantages. In this thesis, I will focus on the security problem, more specifically on the security problem from the signaling point of view. The switch from POTS to VOIP creates many security concerns, mainly related to frauds and confidentiality/integrity of the voice transported. The biggest problem is that it is quite simple for an unauthorized person to make a call or impersonate someone else, because SIP can work without any kind of security. This thus makes it possible to perform several illegal actions such as intercepting calls, trying to defraud the service provider or even making calls using someone else's account. For this reason, SIP is equipped with various security mechanisms, but they have not often been used. In addition, many of the methods that have been used are outdated and are no longer considered sufficiently secure.

3.1 Outline

In chapter 4 I will present background information about SIP and IPsec and TLS. In chapter 5 I will describe the security in SIP. In chapter 6 I will expose my test environments and the scope of the test performed, and finally in chapter 7 I will I report and analyze the result of the measurement.

4 Description of underlying protocols

4.1 Session Initiation Protocol - SIP

SIP is an application layer signaling protocol for multimedia communication. It is a text-based protocol that has many elements in common with the Hypertext Transfer Protocol (HTTP), in which there is a request/response on text-based messages. Sip is used for the setup and the teardown of media session for example for audio or video calls. SIP only addresses the signaling part of the communication, while the audio/video traffic is carried over other protocol such as Real-time Transport Protocol (RTP) or its secure version, the Secure Real-time Transport Protocol (SRTP). For handling the media parameters, such as protocol or codecs, the Session Description Protocol (SDP) is used, which is carried in the SIP message bodies.
Every user is identified by a Uniform Resource Identifier (URI) like sip:username@host:port. SIP can run over various transport protocol: User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP) and Transport Layer Security (TLS). Commonly the 5060 port is used for the non-encrypted SIP traffic and the 5061 is used for encrypted traffic that runs over TLS.
SIP was first proposed as a standard with a Request For Comments (RFC) by the IETF in 1999 with RFC 2543 and was redefined in a 2.0 version in 2002 in the RFC 3261 [2].
SIP uses a set of text messages, mainly being:

- REGISTER: used by UAC to inform its current IP address of contact.
- INVITE: used to setup a media session.
- BYE: used to teardown a media session.
- ACK: used to confirm a message.
- CANCEL: used to delete a pending request.

Below are the definitions of the responses families:

- 1xx – Provisional: request received, processing the request. Typical message is the 100 – Trying, that was used by the proxy in response to an INVITE when forwarding the INVITE to another User Agent (UA).
- 2xx – Success: request received and accepted. Typical message is the 200 – OK, that the User Agent Client (UAC) of destination sent back to the sender when the user answered the call.
- 3xx – Redirection.
- 4xx – Client error: the request cannot be accepted by the server. Typical message is the 401 – Unauthorized, is used when an authorization is required in order to complete an action, such as an INVITE.
- 5xx – Server error: the server failed to fulfill an apparently valid request.
- 6xx – Global failure.

Two SIP agents can communicate directly, in a peer-to-peer mode, but usually between them there are one or more network elements. Typical elements are:

- Proxy Server: an intermediary server that receives request from User Agent Client (UAC) and makes request on behalf of them, the typical function performed are routing call from various domains.
- Registrar: a server that manages the REGISTER messages and keeps track of the binding of SIP URI and the socket where the UAC can be contacted.
- Redirect server: a server that generates 3xx responses to requests it receives, directing the client to contact an alternate set of URIs.

They are distinct logical elements but they can be co-located on the same machine.

4.2 IP security - IPsec

IPsec, abbreviation of IP security, is specified in the RFC 4301 [3] and is a suite of cryptographic services that can provide confidentiality and/or integrity/authentication.
It is composed of three main mechanisms: ESP, AH and IKE, that I will briefly describe.

4.2.1 Encapsulating Security Payload - ESP

Encapsulating Security Payload (ESP), defined in RFC 4303 [4], provide confidentiality, data origin authentication, connectionless integrity and an anti-replay service. The data encrypted and authenticated are showed in Figure 1.
The services provided depend on the configuration. For example, it can provide integrity and data origin authentication only, or confidentiality only without integrity. Note that the previous one is not a good practice, and without a strong independent integrity mechanism can be insecure. The anti-replay services can only be used if the integrity service is used. The RFC prescribes that the confidentiality-only mode may be supported, while the integrity only and the confidentiality plus integrity mode must be supported.

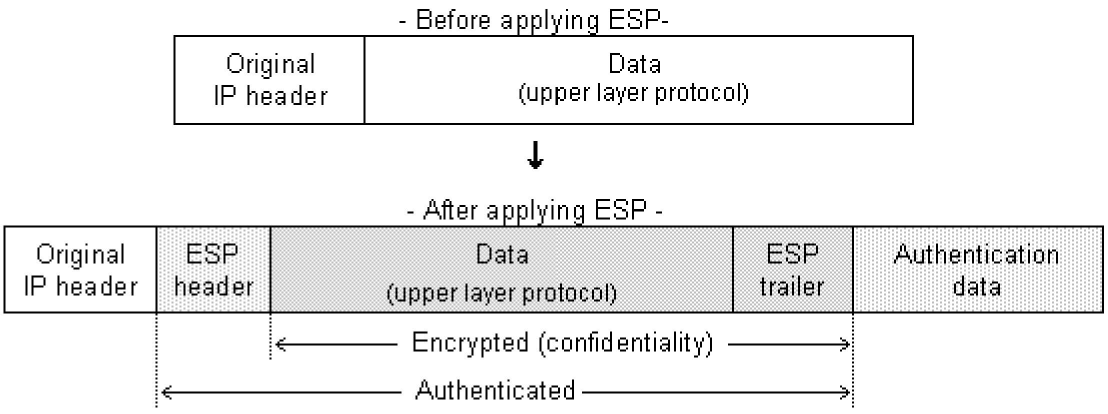


Figure 1: ESP in transport mode.

4.2.2 Authentication Header - AH

Authentication Header (AH), defined in RFC 4302 [5], provides connectionless integrity and data origin authentication for IP datagrams and optionally an anti-replay service. As showed in Figure 2, it can provide authentication for some fields of the IP header, as well as

for next level protocol data. It provides authentication only for some IP header fields because this may change in transit.
Against the integrity service provided by the ESP the main difference is that ESP does not cover anything of the IP header, unless this is encapsulated in another IP packets, instead of the AH that cover some fields.

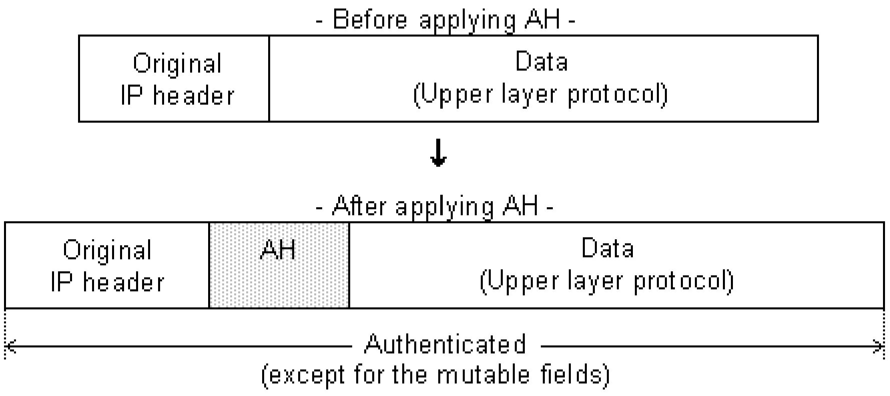

- Before applying AH -
Original IP header
Data (Upper layer protocol)
- After applying AH -
Original IP header
AH
Data (Upper layer protocol)
Authenticated (except for the mutable fields)

Figure 2: AH in transport mode.

4.2.3 Operational modes

The ESP and AH protocols can work in opposition or can be used in composition; both can operate in two modes:

- Transport: in this mode the ESP header or AH header are inserted between the original IP header and the payload, so that they protect only the next layer protocols. It is typically employed between a pair of hosts to provide end-to-end security services. It is the operational mode represented in Figure 1 and Figure 2.
- Tunnel: in this mode the original IP packet is encapsulated in a new IP packet and the IPsec protects the whole original datagram. This mode is typically employed between a pair of intermediate hosts, such as security gateways, in order to provide security services between them. They are represented in Figure 3 and Figure 4.

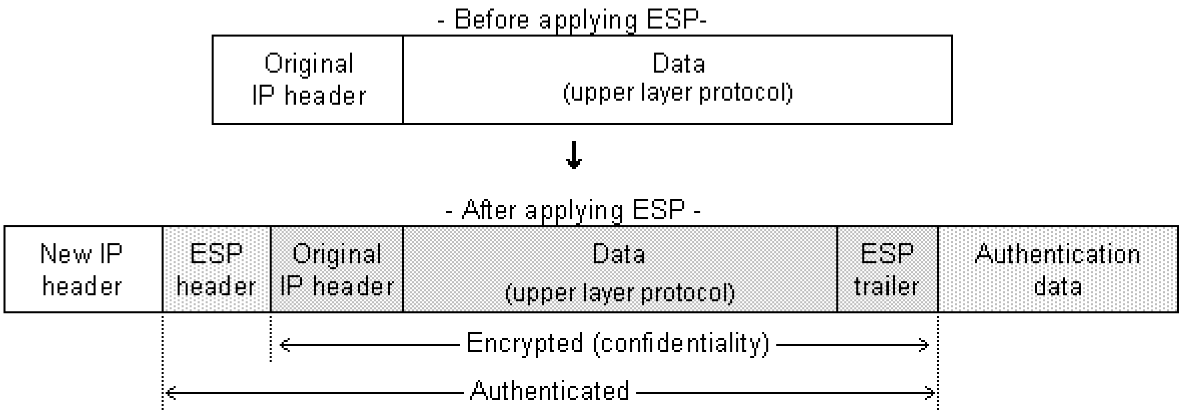


Figure 3: ESP in tunnel mode.

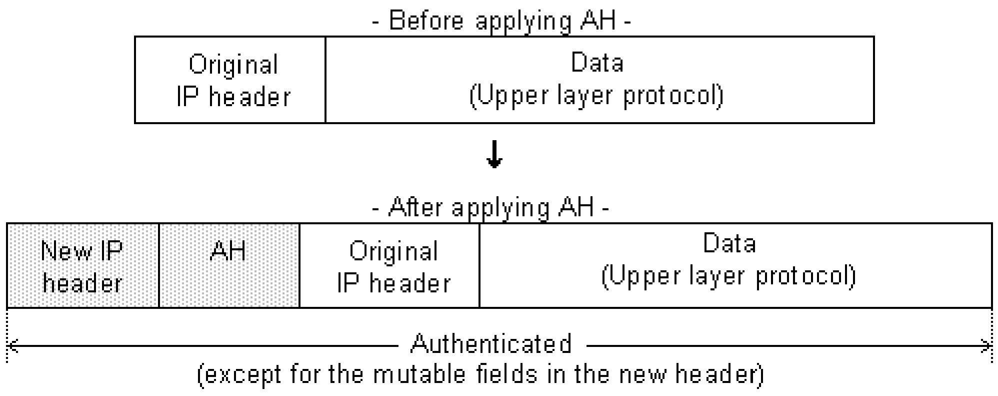


Figure 4: AH in tunnel mode.

4.2.4 Internet Key Exchange - IKE

Both ESP and AH operate based on requirements defined by a Security Policy Database (SPD), and make use of Security Associations (SA). An SA is a simplex connection that affords security services to the traffic carried by it. In order to secure bi-directional communication a pair of SAs is required, one in each direction. SAs are created and maintained by IKE.

Internet Key Exchange (IKE), defined in RFC 5996 [6], performs mutual authentication between two parties and establishes an IKE security association that includes shared secret information that can be used to efficiently establish ESP or AH. After a couple of reviews, it has reached to version 2 in 2005, named IKEv2. Figure 5 shows an IKEv2 exchange. There are four kinds of exchanges defined, which are as follows:

- IKE_SA_INIT: It performs three functions in the setup of the IKE-SA.
    - Negotiates security parameters for the IKE-SA.
    - Sends nonces.
    - Exchange Diffie-Hellman (DH) values.

  This is the first exchange that establishes the IKE-SA and must be completed before any further exchanges can take place.
- IKE_AUTH: It performs three required functions:
    - Transmits identities.
    - Proves knowledge of the secrets related to those identities.
    - Establishes the first, and usually the only, AH and/or ESP CHILD-SA.

  This is the second exchange and must be completed before any further exchanges can take place.
- CREATE_CHILD_SA: This is simply used to create additional CHILD-SAs as needed.
- INFORMATIONAL: This is a maintenance exchange that performs a variety of functions to maintain the SAs. Some of these functions include:
    - Delete SAs as needed.
    - Report error conditions.
    - Check SA liveliness.
    - Other SA housekeeping functions.

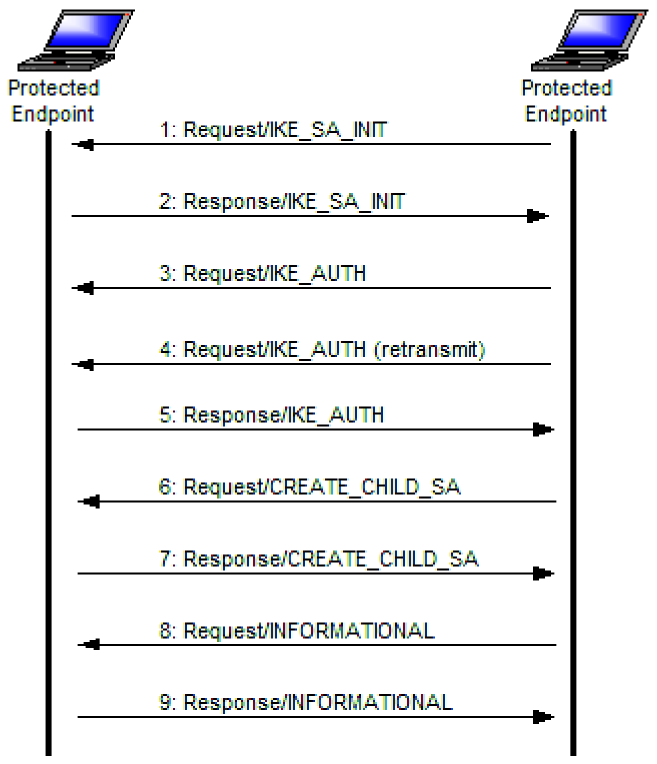


Figure 5: IKEv2 exchange.

4.2.5 Standard requirements

The RFC 4301 prescribes for every kind of machine what protocols and operational modes must be supported. Requirements are summarized in Table 1, Table 2 and Table 3. In these and in all the following tables the keywords MUST, MUST NOT, SHOULD, SHOULD NOT, MAY are used and should be interpreted as described in RFC 2119 [1].

Requirement	Protocol
MUST	ESP
MAY	AH

Table 1: IPsec protocol requirements.

Requirement	Services
MUST	Confidentiality and integrity
MAY	Confidentiality only

Table 2: Services requirements for ESP.

Type of machine	Requirement	Operational mode
Host	MUST	Tunnel
	MUST	Transport
Security gateway	MUST	Tunnel
	MAY	Transport

Table 3: IPsec operational modes requirements.

The IPsec suites is designed to be independent from the underlying cryptographic algorithms, so it is possible to select various cryptographic suites or modify it without modifying the entire IPsec protocols.

At the moment of writing the cryptographic algorithms for ESP and AH are defined in RFC 4835 [7]. The Table 4 show the algorithms that must be supported for the confidentiality in ESP.

Requirement	Algorithm
MUST	NULL
MUST	AES128-CBC
MUST-	TripleDES-CBC
SHOULD	AES-CTR
SHOULD NOT	DES-CBC

Table 4: Encryption algorithms for ESP.

The Table 5 show the algorithms that must be supported for the authentication and integrity in ESP and in AH. Note that the NULL authentication is suitable only for ESP.

Requirement	Algorithm
MUST	HMAC-SHA1-96
SHOULD+	AES-XCBC-MAC-96
MAY	NULL
MAY	HMAC-MD5-96

Table 5: Authentication algorithms for ESP and AH.

4.3 Transport Layer Security - TLS

The Transport Layer Security (TLS), descendant of the Secure Sockets Layer (SSL), is a cryptographic protocol that aims to provide privacy and data integrity between two applications. It operates on the top of a reliable transport protocol, normally TCP. The current release of the TLS protocol are the 1.2, described in the RFC 5246 [8].
Two layer composes the TLS:

- TLS Record Protocol: is the lower layer, provides confidentiality and integrity of the message. It encapsulates the traffics of the higher-level protocols.
- TLS Handshake Protocol: operates above TLS Record Protocol, provides mutual authentication and negotiate encryption algorithms and secret key before the connection setup.

4.3.1 TLS cryptographic algorithms

TLS makes use of public-key cryptography for the authentication, symmetric key cryptography for the encryption of the data flow, with key generated uniquely for any connections, and keyed Message Authentication Code (MAC) for the integrity, they are summarized in Table 6.

Service	Algorithm
Encryption	AES256-CBC
	AES128-CBC
	3DES-EDE-CBC
	RC4-128
	NULL
Integrity	HMAC-SHA256
	HMAC-SHA1
	HMAC-MD5
	NULL
Authentication/key exchange	RSA
	DH-DSS
	DH-RSA
	DHE-DSS
	DHE-RSA
	DH-anon
	NULL

Table 6: TLS cipher suites.

4.3.2 Description of TLS handshake

The TLS handshake is composed by a set of messages exchanged by the client and the server. The sequence of the exchange is:

- The client sends a ClientHello message with a list of the supported cipher suites, random number, the supported TLS versions and the compression methods.
- The server sends a ServerHello message with the TLS version, a random number, the cipher suite and a compression method chosen from the client's list.
- The server may sends its own certificate, depending on the cipher suite.
- The server may send a certificate request and a server key to the client.
- The server sends a ServerHelloDone message.
- The client, if requested, sends its own certificate.
- The client sends a key depending on the cipher selected, and then begins computing the master secret.
- The client sends a CertificateVerify message which is a signature of the previous message using the client's certificate.
- The client sends the ChangeCipherSpec message informing the server that encryption and authentication starts.
- The client sends its Finished message, which the server decrypts and verifies. If the verification fails the handshake is considered failed.
- The server sends a ChangeCipherSpec message informing the client that encryption and authentication starts..
- The server sends a Finished message, which the client decrypts and verifies, after that, if the verification is successful the handshake ends.

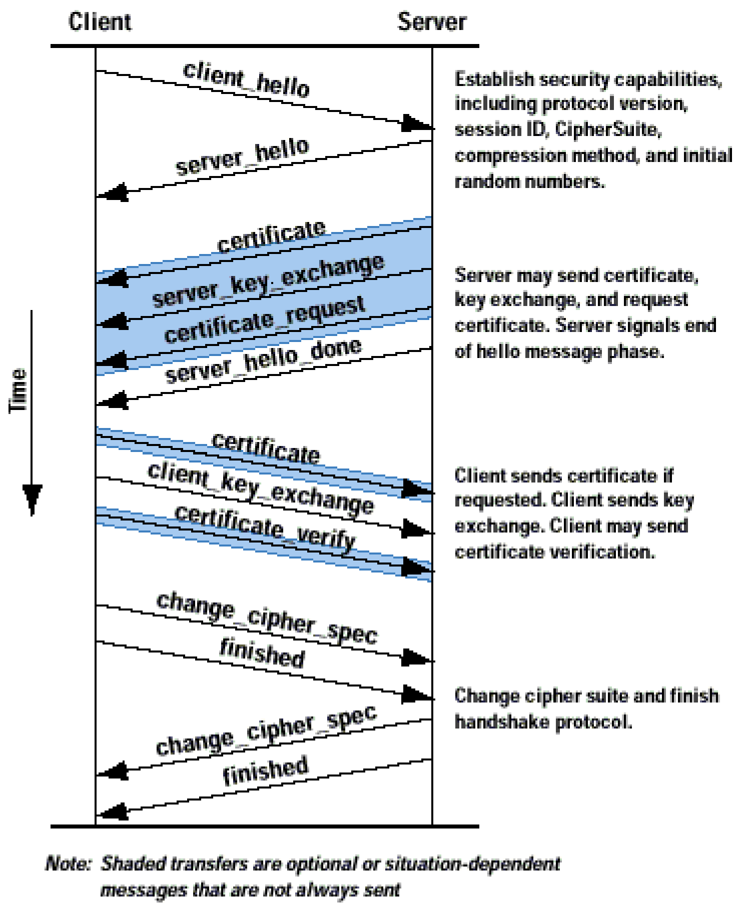

Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, CipherSuite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent

Figure 6: TLS handshake.

5 Security provisions in SIP

As previously mentioned, SIP is a protocol that make it possible to create media-session between endpoint, even though it only cares about the signaling part of the session. The actual multimedia data flow can be transported in several ways, independent from SIP. For these reasons, the RFC 3261 defines a set of security mechanism for protecting only the signaling layer of SIP sessions, for the media layer there are many possibilities, like using the secure version of RTP, the SRTP. In the next session, I will expose the security mechanism that can be used for the signaling level, divided in those that are prescribed by the standard of SIP and those that have been proposed in literature. After that, I will present what the main vulnerabilities of SIP are and of the presented solutions, explaining in detail why I have focused on the signaling plain.

5.1 Mandatory mechanisms

The following is a list and a description of the security mechanisms that are described in the RFC of SIP.

5.1.1 Application layer security approach

5.1.1.1 HTTP digest authentication

The first, and more basic, mechanism supported from SIP is derived from the HTTP Authentication, described in RFC 2617 [9], and is a stateless challenge-based mechanism. The use of basic scheme is deprecated; SIP provides only the use of HTTP Digest authentication scheme.
When a User Agent Server (UAS) receives a request can use the 401 - UNAUTHORIZED response to challenge the UAC, containing a WWW-Authenticate header, that indicates the authentication scheme and the parameters of at least one challenge, like the realm and the nonce. An example, taken from the RFC, is the following:

```
WWW-Authenticate: Digest
     realm="biloxi.com",
     qop="auth,auth-int",
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

The client response to the challenge resending the request including in it an authorization header, like:

```
Authorization: Digest username="bob",
     realm="biloxi.com",
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     uri="sip:bob@biloxi.com",
     qop=auth,
     nc=00000001,
     cnonce="0a4f113b",
     response="6629fae49393a05397450978507c4ef1",
     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

The realm is a string to indicate to the client what host/service are requiring and processing the authentication.
The response string is calculated as follows:

```
response = <"> < KD ( H(A1), unq(nonce-value)
        ":" nc-value
        ":" unq(cnonce-value)
        ":" unq(qop-value)
        ":" H(A2)
        ) <">
```

Where:

- A1 = unq(username-value) ":" unq(realm-value) ":" passwd
- A2 = Method ":" digest-uri-value

And:

- KD(secret, data) = the string obtained by applying the digest algorithm to the data "data" with secret "secret"
- H(data) = the string obtained by applying the checksum algorithm to the data data"
- unq(X) = the value of the quoted-string X without the surrounding quotes

When using MD5 as digest they became:

- KD(secret, data) = H(concat(secret, ":", data))
- H(data) = MD5(data)

The call-flow of an unauthorized REGISTER request is shown in Figure 7, instead Figure 8 show the call-flow when using digest authorization.

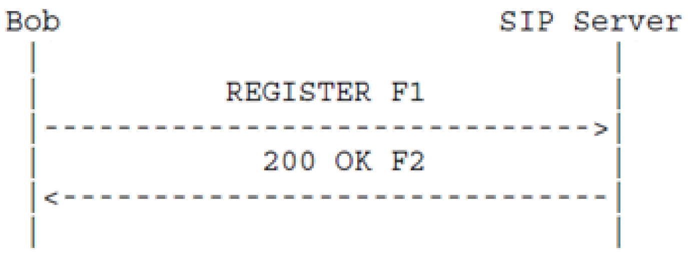


Figure 7: unauthenticated call-flow for a REGISTER request.

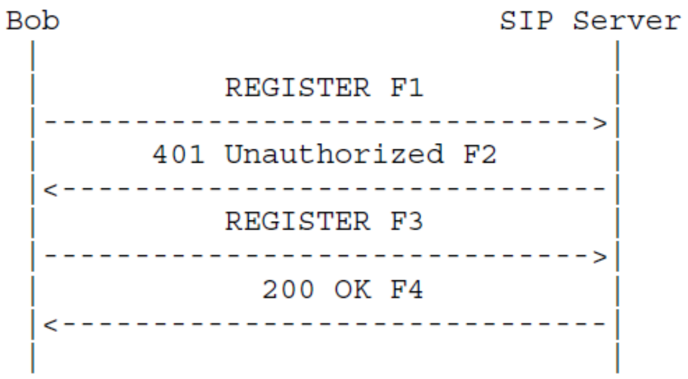


Figure 8: authenticated call-flow for a REGISTER request.

It is possible to see that HTTP digest authentication introduce a high overhead, requiring the retransmission of some messages. For example, this means doubling the number of messages required for the management of a REGISTER.
For the RFC proxy servers, redirect servers, registrars, and UAs must implement Digest Authorization.

5.1.1.2 Secure Multipurpose Internet Mail Extensions - S/MIME

For the correct working SIP needs that some field of any message, like the Request-URI or the Route header, to be visible to the intermediate nodes, and also these intermediates nodes should be able to modify the message, like inserting a Via header. This is related to the routing and accounting features. However it is possible to encapsulate the original SIP packet into a Multipurpose Internet Mail Extensions (MIME) body that will be secured with Secure MIME (S/MIME, defined in version 3.2 in RFC 5751 [10]). Thus, the original packet is protected and authenticated end-to-end, but this does not provide any security advantage to the network layer, because it does not have access to the S/MIME body. Thus, this mechanism must be used in combination with some other security services.
S/MIME provides confidentiality through encryption and provide authentication, message integrity, and non-repudiation with proof of origin by using digital signatures.
The standard specify the use of the algorithms reported in Table 7 for the digital signature of the message and those reported in Table 9 for encryption. In Table 8 is reported the key-length requirements for the public-key operations. We can see that it makes use of secure and well-known algorithms.

Requirement	Algorithm
MUST	RSA with SHA256
SHOULD+	DSA with SHA256
SHOULD+	RSASSA-PSS with SHA256
SHOULD-	RSA with SHA1
SHOULD-	DSA with SHA1
SHOULD-	RSA with MD5

Table 7: Digital signature algorithms for S/MIME.

Key-length	Algorithm
MUST	$1024 \leq l \leq 2048$
MAY	$l < 1024$
MAY	$l > 2048$

Table 8: Key-length requirement for public-key operation in S/MIME, in bits.

Requirement	Algorithm
MUST	AES128-CBC
SHOULD+	AES192-CBC
SHOULD-	3DES

Table 9: Encryption algorithms for S/MIME.

The main advantage is that encapsulating the payload S/MIME can provide confidentiality and digital signature end-to-end between host, that other methods cannot provide. However, the advantages can also be a disadvantage, because encapsulating the original packets in the actual one means overhead, doubling up the dimension of the message. This overhead can be negligible if seen from the clients side, but it would double the bandwidth usage on the proxy side. However, the main disadvantage is related to the usage of digital signature, which requires a large-scale Public-Key Infrastructure (PKI) that permits clients to verify the original message, also when the original sender is part of another domain.
Again, for RFC UAs could support the signing and encrypting of MIME bodies, and transference of credentials with S/MIME.
It is noteworthy that, quoting the RFC, "future security extensions may upgrade the normative strength associated with S/MIME as S/MIME implementations appear". This has been written in 2002, and until now very few SIP software that support S/MIME have been released.

5.1.2 Lower layer security approach

Encrypting end-to-end a message is not a good solution, because it cannot be routed to destination, like mentioned above. For this reason, a SIP message cannot be encrypted end-to-end, but it must be encrypted on a hop-by-hop basis. This implies that there must be a chain of trust between the sender and the receiver. For this motivation, the SIP RFC encourage using security mechanism at lower level. More precisely the RFC 3261 encourage using security mechanism at network or transporting layer, like IPsec and TLS. It suggests the use of IPsec when there are pre-shared keying relationship or when an association between the hosts can be setup, instead TLS is suggested when there are no pre-existing trust associations.
The RFC does not specify any profile to be used in IPSEC, instead for the TLS set the requirements listed in Table 10, where the first is the minimum requirement and the second is for compatibility; in addition to these, any other TLS cipher suite can be used.

Requirement	Algorithm
MUST	TLS_RSA_WITH_AES_128_CBC_SHA
SHOULD	TLS_RSA_WITH_3DES_EDE_CBC_SHA
MAY	Any other

Table 10: Cipher suites requirements for TLS in SIP.

For the RFC the requirements are that proxy servers, redirect servers, and registrars must implement TLS, and must support both mutual and one-way authentication, in addition it is strongly recommended that UAs are capable initiating TLS.
Proxy servers, redirect servers, registrars, and UAs could also implement IPsec or other lower-layer security protocols.

5.1.3 SIPS URI

Both IPsec and TLS must be used in a hop-by-hop basis, so a host has the certainty of the method used only in the first hop, but does not have any assurance of the complete path. For allowing the host to have control, an equivalent of the HTTPS URI scheme has to be introduced, named SIPS URI scheme.

When used as the Request-URI, each hop over which the request is forwarded, until the request reaches the SIP entity responsible for the domain portion of the Request-URI, must be secured with TLS. Once it reaches the domain in question it is handled in accordance with local security and routing policy. It is however possible to use TLS for any last hop to a UAS.

All SIP elements that support TLS must also support the SIPS URI scheme.

5.2 Proposed mechanisms

Beyond the mechanism discussed there are many other methods proposed but not standardized yet, or not incorporated into the standard of SIP yet. These mechanisms can have some advantages, but at the moment they are not used. In the following section, I will present three different approaches to the problem, to give an idea of what the futures developments could be.

5.2.1 Datagram Transport Layer Security - DTLS

The RFC 6347 [11] defines the Datagram Transport Layer Security (DTLS) 1.2 standard. It is a derivation of TLS protocol, which does not require reliable transport level, so it can run over datagram transport protocols like UDP. The reason as to why TLS cannot operate over an unreliable transport layer is because TLS does not allow independent decryption of individual records because if a record was lost the decryption of the next record will fail; moreover, the handshake layer is designed assuming that the handshake messages are delivered reliably, and breaks if those messages are lost. The changes introduced in DTLS from TLS are mainly for solve these problems.

The advantages of DTLS is that operating over UDP the transmission requires much less overhead, and therefore has a lower performance impact. Furthermore, it is suitable for real-time traffics, like other unreliable transport protocols, and it is also proposed to transport media streams, so if adopted for the media path it can also be easily used for the signaling level. Instead, the main disadvantages of DTLS is that it is more suitable than TLS to DoS attacks. The RFC describes a mechanism for protecting the system from this kind of attacks, but leaves to the implementers the choice to implement.

5.2.2 HTTP digest authentication with Authentication and Key Agreement

In 2002, Niemi, Arkko and Torvinen, with the RFC 3310 [12] proposed the use of the Authentication and Key Agreement (AKA) protocols within the HTTP Digest authentication. The AKA is a challenge-response based mechanism that uses symmetric cryptography, that performs user authentication and session-key distribution in Universal Mobile Telecommunications System (UMTS) networks. It is typically run in a UMTS IM Services Identity Module (ISIM), which resides on a smart card like device that also provides tamper resistant storage of shared secrets.

The RFC maps the AKA parameters onto HTTP Digest authentication, enabling the usage of AKA as a one-time password generation mechanism for Digest authentication. Moreover, because the SIP Authentication closely follows the HTTP Authentication, Digest AKA is directly applicable to SIP.
If a password is chosen by a user, or must be memorized and typed by a user, it must be "simple", like 8-10 printable characters maximum and usually this is not truly random, making simple the offline password guessing described in 5.4.2. If the cryptographic secret is stored safely in a memory, it can be more complex and more random. This implies that the HTTP digest with AKA could require more trials to guess the correct password, and therefore makes the dictionary attacks ineffective.
Moreover, this scheme introduce a mechanism whereby a server may allow each nonce value to be used only once by sending a next-nonce directive in the Authentication-Info header field of every response and tracking the request using a SQN sequence number, protecting the system from replay-attacks. This also protects the client, as he is able to detect an old request that has been resent from an attacker.
Furthermore, the AKA is able to generate additional session-keys for integrity and confidentiality protection that can be used for some other security mechanism, for example for securing the media stream.
The main disadvantage of this scheme is that it requires every host on which the AKA must be performed to have a SIM. This is a strong requirement, but now the diffusion of smartphone that can run VOIP services is in a phase of strong growth and customers are also getting used to the idea of internet keys. Thus the diffusion of a method like this should not be seen as a problem but rather an opportunity to increase the number of mobile subscribers, perhaps selling traditional-mobile integrated phone services.
A note is that, like the original HTTP digest authentication, it relies on a MD5 digest. This solution should be replaced by other stronger digest algorithm like a SHA2 digest, because as we will see later, the differences in performance are not dramatically high but it could greatly improve the security for two reasons: one is that the digest algorithm is more robust, and the second one is that they produces longer digest.

5.2.3 Certificateless public-key cryptography

In [13] Wang and Zhang proposed a new mechanism named Secure Authentication and Key Agreement (SAKA), based on CertificateLess Public-Key Cryptography (CL-PKC) that was originally proposed by Al-Riyami and Paterson in [14] . CL-PKC is an intermediate solution between ID based PKC and traditional PKC, which otherwise ID-PKC does not suffer from the key escrow problem. The operation of a CL-PKC is based on the idea that, using a different construction of the public/private key, the key of an entity can be verified without a certificate. For the key generation, CL-PKC makes use of a Key Generating Centre (KGC) that computes from the identifiers ID_A of the entity A a partial private-key D_A. A combines D_A whit some secrets to generate the private-key S_A, and combining it to public parameters provided by the KGC, compute the public-key P_A. This way the private-key is available to A only, and the public-key is not computable by any other entity. To verify a signature it is only needed the knowing of P_A and ID_A only.

5.2.3.1 Details of cerfiticateless public-key cryptography

Al-Riyami and Paterson in [14] explain in detail the operation of a certificateless public-key system, that I summarized in the following.

In the following, $\mathbb{G}_1$ denotes an additive group of prime order q and $\mathbb{G}_2$ a multiplicative group of the same order. We let P denote a generator of $\mathbb{G}_1$. A pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ with the following properties:

- The map e is bilinear: given $Q, W, Z \in \mathbb{G}_1$, we have $e(Q, W + Z) = e(Q, W) \cdot e(Q, Z)$ and $e(Q + W, Z) = e(Q, Z) \cdot e(W, Z)$
- The map e is non-degenerate: $e(P, P) \neq 1_{\mathbb{G}_2}$
- The map e is efficiently computable.

The Bi-linear Diffie-Hellman Problem in $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$ is as follows: Given $\langle P, aP, bP, cP \rangle$ with uniformly random choices of $a, b, c \in \mathbb{Z}_q^*$, compute $e(Q, Z)^{abc} \in \mathbb{G}_2$.

Description of the scheme:

- Setup:

  Choose an arbitrary generator $P \in \mathbb{G}_1$. Select a master-key s uniformly at random from $\mathbb{Z}_q^*$ and set $P_0 = sP$. Choose cryptographic hash functions:
$H_1: \{0,1\}^* \to \mathbb{G}_1^*, H_2: \mathbb{G}_2 \to \{0,1\}^n, H_3: \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and $H_4: \{0,1\}^n \to \{0,1\}^n$.
Where n will be the bit-length of plaintexts.
The system parameters are params= $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2, H_3, H_4 \rangle$, the master-key is $s \in \mathbb{Z}_q^*$ and message space is $\mathcal{M} = \{0,1\}^n$, the ciphertext space is $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^n$.
- Partial-Private-Key-Extract:

  This algorithm takes as input an identifier $ID_A \in \{0,1\}^*$ and carries out the following steps to construct the partial private-key for entity A with identifier ID_A:
  - Compute $Q_A = H_1(ID_A) \in \mathbb{G}_1^*$.
  - Output the partial private-key $D_A = sQ_A \in \mathbb{G}_1^*$.
- Set-Secret-Value:

  This algorithm takes as inputs params and an entity A's identifier ID_A as inputs. It selects $x_A \in \mathbb{Z}_q^*$ at random and outputs x_A as A's secret value.
- Set-Private-Key:

  This algorithm takes as inputs params, an entity A's partial private-key D_A and A's secret value $x_A \in \mathbb{Z}_q^*$. It transforms partial private-key D_A to private-key S_A by computing $S_A = x_A D_A = x_A s Q_A \in \mathbb{G}_1^*$.
- Set-Public-Key:

  This algorithm takes params and entity A's secret value $x_A \in \mathbb{Z}_q^*$ as inputs and constructs A's public-key as $P_A = \langle X_A, Y_A \rangle$ where $X_A = x_A P$ and $Y_A = x_A P_0 = x_A sP$.
- Encrypt

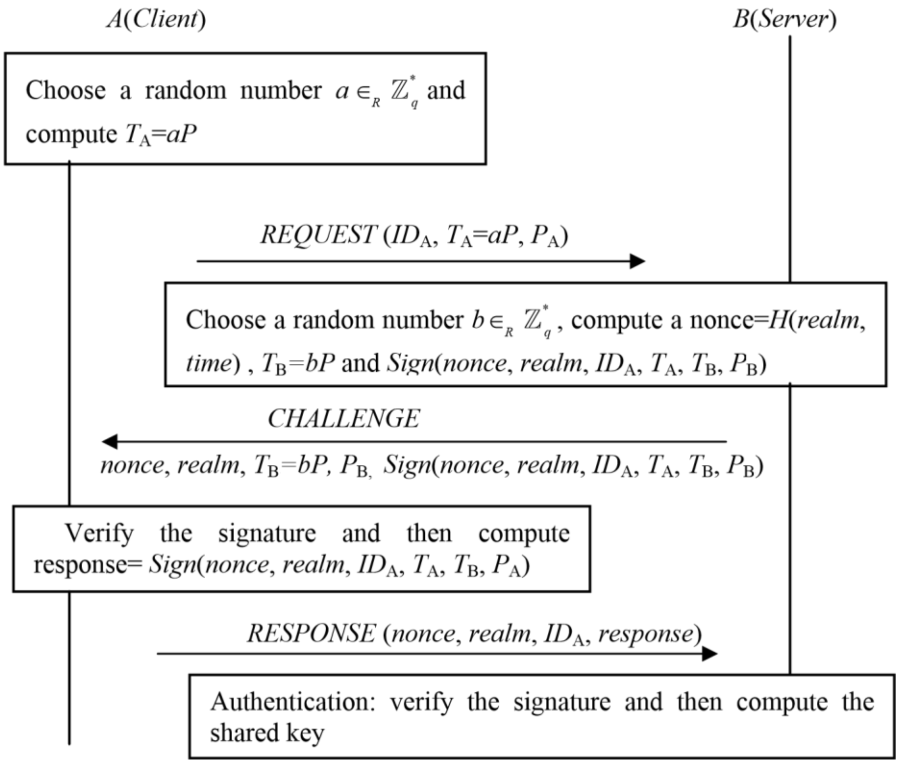
  To encrypt $M \in \mathcal{M}$ for entity A with identifier $ID_A \in \{0,1\}^*$ and public-key $P_A = \langle X_A, Y_A \rangle$, perform the following steps:
  - Check that $X_A, Y_A \epsilon\ \mathbb{G}_1^*$ and that the equality $e(X_A, P_0) = e(Y_A, P)$ holds. If not abort encryption.
  - Compute $Q_A = H_1(ID_A)\ \epsilon\ \mathbb{G}_1^*$.
  - Choose a random $\sigma\ \epsilon\ \{0,1\}^n$.
  - Set $r = H_3(\sigma, M)$.

  - Compute and output: $C = \langle rP, \sigma \oplus H_2(e(Q_A, Y_A)^r), M \oplus H_4(\sigma) \rangle$.
- Decrypt

  Suppose the ciphertext $C = \langle U, V, W \rangle \in \mathcal{C}$. To decrypt this ciphertext using the private-key S_A:
  - Compute $\sigma' = V \oplus H_2\big(e(S_A, U)\big)$.
  - Compute $M' = W \oplus H_4(\sigma')$.
  - Set $r' = H_3(\sigma', M')$ and test if $U = r'P$. If not and reject C.
  - Output M' as the decryption of C.

5.2.3.2 Application of cerfiticateless public-key cryptography to SIP

Going back to the application on SIP, in [13] the authors prove the security of the scheme, that result provably secure in the CK security model and provides mutual authentication, perfect forward secrecy and key confirm. The scheme proposed makes use of the challenge response handshake of the digest authentication, so it can operate on systems that follows the RFC 2617 directive. The handshake for a request like an INVITE is represented in Figure 9.



$A(Client)$
$B(Server)$
Choose a random number $a \in_R \mathbb{Z}_q^*$ and compute T_A=aP
$REQUEST\ (ID_A, T_A{=}aP, P_A)$
Choose a random number $b \in_R \mathbb{Z}_q^*$, compute a nonce=$H(realm, time)$, T_B=bP and $Sign(nonce, realm, ID_A, T_A, T_B, P_B)$
$CHALLENGE$
$nonce, realm, T_B{=}bP, P_B, Sign(nonce, realm, ID_A, T_A, T_B, P_B)$
Verify the signature and then compute response= $Sign(nonce, realm, ID_A, T_A, T_B, P_A)$
$RESPONSE\ (nonce, realm, ID_A, response)$
Authentication: verify the signature and then compute the shared key

Figure 9: SAKA.

As showed, the handshake allows mutual authentication and it provides a key agreement using a Bilinear Diffie-Hellman procedure. If the request is like a REGISTER, that does not require the agreement of a session-key, then the computes and the exchange of the keying materials T_A and T_B can be omitted. Note that the nonce is defined like function of the realm and time, and can be used only once, therefore further protecting from replay attacks.

The authors have suggested that every SIP domain has its own KGC, this way they strictly bind the generation of a partial private-key only to known peers. They also assume that the distribution of these key in an intra-domain scenario take places in a secure manner.

This mechanism has the advantages of taking the benefits of public-key cryptography, without the drawback of a large-scale public-key infrastructure. Additionally, it does not need a pre-shared key associations between client and server, or between host in peer-to-peer mode. Instead, the disadvantage is that it requires a completely new infrastructure: every SIP domain needs its own KGC, and software or hardware must implement this new scheme, so it could be difficult for a large-scale adoption in the near future.

5.3 Implemented mechanisms

The commonly used security mechanism used for securing the signaling level of SIP is the HTTP digest authentication. It can be used in a standalone version or used in combination with some Authentication Authorization and Accounting (AAA) protocols; such as Remote Authentication Dial-In User Service (RADIUS) or Terminal Access Controller Access-Control System Plus (TACACS+). If used in combination with AAA service, the benefits are that the shared secret are stored in another machine rather than SIP proxy, usually not accessible from internet but only from the local domain. This mechanism is often the one used by the service providers.
For the transport and network layer solutions, both TLS and IPsec is widely supported.
In addition, also the network elements commonly supports both IPsec and TLS. For example the Acme Packet's Net-Net 4500, a service provider class SIP router, supports up to 200000 IPsec tunnels or up to 200000 TLS session with hardware acceleration [15].

5.4 Threats and known vulnerabilities

The SIP protocol is exposed to huge number of attacks; the more common threats are registration hijacking, tampering messages and Denial of Service (DoS).
The media traffic is affected by some particular security problems not covered by this thesis, but several security issues that affect the media streams are related to the SIP signaling. For example, if the SIP INVITE message, that carries the RTP socket in which the client is listening in the SDP payload, is not authenticated an attacker could intercept the INVITE and modify the RTP socket putting a new socket over he is listening. So the called host sends his media traffics directly to the attackers, that can rely the flow to the original sender, making a man in the middle attacks.

5.4.1 Main attacks

- Impersonating someone else: an attacker can try to use the identity of another client or can try to operate like a server, identifying itself like the wanted entity, and without a strong authentication mechanism a user cannot notice it.
- Registration hijacking: an enemy can try to use valid credential of another use for making calls and putting it in the account of another user or may even try to receive the calls that are directed to another user. Moreover, it is possible to de-register a valid user from a server, leaving it without the service.
- Closing session: an adversary can forge request like CANCEL or BYE and cancel the processing of the REQUEST of a legitimate user or tear down a valid session successfully established. Even in this case the user cannot use the service.

- DoS: an opponent can try to render a network machine unavailable to its user. Usually this is made overloading a server with request. SIP is very sensitive to this kind of attacks because of its nature it is normally exposed to internet and there are some particular elements of SIP that can be used against SIP itself. For example if an attacker is able to spoof some field of a request, like the VIA or Record-Route, it can redirect the valid traffic generated by normal UAs to a particular machine. Or it can also try to change the registration of a huge amount of user in a registrar server, pointing all to the target's address. All the messages directed to the user will be redirected to target, amplifying the attacks.
- Eavesdrop: an attacker can try to eavesdrop the media flow between the endpoints, this can give him access to confidential information, that could be something that the user said or other kinds of media that the user has exchanged, for instance, text, image or maybe Dual-Tone MultiFrequency (DTMF) that could carry personal information, like the credit card number.
- Call redirection: an opponent can try to redirect a legitimate call to a different URI from the intended one, this way he can try impersonate someone or can simply route a call to a Value-Added Service (VAS), and make a fraud.

In Table 11, taken from [16], Butcher et al. have summarized the various potential attacks to a VOIP system, it is possible to see that they are related to different levels and have different purposes. For example, an ICMP flood has the only scope of disturbing the normal service behavior, leaving some users without service, creating a kind of DoS attack. Instead, a registration hijacking can have many objectives, attacking also the confidentiality and the integrity of the messages. Table 12 shows the possible security mechanisms that can be used to protect from various attacks.
Many of the problems reported are common in all IT scenarios, such as the MAC spoofing that is an attack related to the Ethernet protocol suite, and it affects VOIP only because SIP operates above it. The reason as to why I focused on the security of the signaling plane of the SIP protocol, is because as shown in Table 12, securing the signaling plane of SIP makes it possible to protect the system from a wide spread of attacks. Furthermore, I consider that many other mechanism, such as OS protection, physical access control or the firewall configuration are just good practice in security.
Another consideration that I have made is that on the traditional phone system, the PSTN, there is no protection to the voice traffic, but there is only an intrinsic security for the signaling data, due to the physical copper line and the number associated to them. This is because on the PSTN network in order to make a call impersonating someone else or charging the call to another user, one must have access to the physical cable. This does not happen anymore when using VOIP and anyone, from any place in the world, with an internet connection can try to make free calls or even worse frauds. For this reason I think that the first step that has to be made, in the switch to a VOIP system, is to introduce a security mechanism for the signaling plane and only after that securing also the media traffic.

Instead, the decision to analyze the behavior of standardized and implemented mechanisms is due to the desire to give a concrete solution to the problem, with what that is already in commerce, so that it can be implemented today. Describing the proposed mechanisms, instead, I want to give an overview of what the future developments could be.

Layer	Attack Mechanism	Confidentiality	Integrity	Availability
Physical	Physical attack	X		x
Data link	ARP cache poison	X	X	X
	MAC spoofing	X	X	X
Network	Device IP spoofing	X	X	X
	Malformed packets			X
Transport	TCP or UDP floods			X
	TCP or UDP replay	X	X	
Application	TFTP server insertion		X	
	DHCP starvation			X
	ICMP floods			X
	Buffer overflow	X	X	X
	Operating system	X	X	X
	Viruses and malware	X	X	X
	Database attacks	X	X	
SIP	Registration hijacking	X	X	X
	Message modification	X	X	
	Cancel/bye attack			X
	Malformed command			X
	Redirect	X		X
RTP	RTP payload			X
	RTP tampering	X	X	X

Table 11: Potential attacks to a VOIP system.

Defense mechanism	Attack prevented																		
	Physical attack	ARP cache poison	MAC spoofing	IP spoofing	Malformed packets	TCP or UDP floods	TCP or UDP replay	TFTP server insertion	DHCP starvation	ICMP floods	Buffer overflow	Operating system	Viruses and malware	SIP registration hijacking	SIP message modification	SIP cancel/bye attack	SIP malformed command	SIP redirect	RTP payload
Physical access control	X																		
Dynamic ARP Inspection		X																	
OS protection											X	X	X						
Port authentication		X	X	X	X	X	X	X	X	X	X			X	X	X	X	X	X
Router configuration				X						X									
Firewall configuration					X	X													
Separate VOIP and data traffic		X	X	X	X	X	X	X	X	X	X	X			X	X	X	X	X
Configuration authentication								X											
Signaling authentication							X							X	X	X	X	X	
Media encryption							X												X

Table 12: Security mechanisms.

5.4.2 Weakness of HTTP Digest Authentication

I would like to begin this section quoting the RFC 2617, the one that defines the HTTP digest authentication, which says "By modern cryptographic standards digest authentication is weak."

There are two main security issues: the replay attacks and a cryptographic weakness. The first problem is that an eavesdropper that intercept a message can try to resend the message after some time to the destination. This could be a problem if the message is for example an INVITE because it setup an unwanted call. In order to protect the system from this type of attacks the server can implement some particular structures of the nonce. For example, some implementations may be willing to accept the server overhead of one-time nonces to eliminate the possibility of replay, or can use nonce-count, that is a count of the number of request, like retransmission, for that a client uses the same nonce.

Another problem is that if an attacker eavesdrops a request that contain a challenge-response, he knows the response, and username, nonce, realm and URI-request. Therefore, it can compute the response guessing the password and if the digest is equal to the one contained in the response he will have found the user password.
In recent years, it has been possible to use the computational power of the GPUs for highly parallelized process, with an enormous advantage over running the same process on a traditional CPU. It is the case of trying all possible password. In December 2010, Marc Bevand released a tool for cracking an MD5 password, at the speed of 33.1 billion password per second on a 2700 $ PC with 4 ATI HD 5970 (dual GPU) [17]. If modified to be usable on HTTP digest authentication, the author estimated a speed of about 2 billion password per second, due to the fact that one check requires the computation of two hashes, one inner an one outer. Moreover, for Moore's law the performance of the actual PC in these two years should be about doubled, and in addition the software could be optimized, so it is reasonable to believe that at a similar price it is possible to check about 4 or more billion password per second. This means that a true random 6 characters alphanumeric password could be exhaustive searched in about 15 seconds, that grows to 180 seconds if considering all the 95 printable ASCII characters. This estimation is made thinking of the silliest attacks, that try all possible combinations, but the actual time could be much lower if most sophisticated attacks are used, such as dictionary based attack. This off-line guessing of the password becomes not feasible if longer and more random password are used.
I would like to point out that this esteem is based on a single high-end PC, but nothing prevents it to run on a cluster of PC or on a cloud based server, however there must be a tradeoff between the cost of the machine and the revenue of attack.
It is noteworthy that this kind of attack requires that the enemy to eavesdrop a request containing the authenticate header, and that this header is removed from the message once it has been authorized by the server that has challenged the host. Thus, the attacker needs to eavesdrop the request in the first hop, the one between the user and the proxy server, and this could not be easy, especially if the user is directly connected to the service provider.

5.4.3 IPsec and TLS

The IPsec does not have any real flaws itself, but there are various concerns regarding the implementation of the standard. This is because it is a complex standard, that defines two protocols (ESP and AH) for protecting the data, that can work in two mode of operation (tunnel and transport), with many options and many degrees of freedom. This leaves the implementers the possibility of choice as of weather to implement or not, and this can introduce several security problems, if not correctly addressed. Moreover, several difficulties could be issued from the local policy configured in the SPD, that should be correctly setup from the administrator, and a wrong configuration could arise some security issues.
There are also various issues concerning some contour elements: if IPsec protects a data stream between two networks but there are some malicious elements in the networks they can have direct access to unsecured traffics, or can try to corrupt the actual unprotected stream. This is not a problem in TLS, because TLS is tightly coupled to the application and

encryption/authentication is made on the source machines. Other attacks, like those showed from Degabriele and Paterson in [18], regards the using of encryption only mode. They also show that many open source IPsec implementation have some security issues, but this derives from a wrong implementation of the RFC. Indeed, they state that all the attacks presented in their paper works only because the implementers do not follow the RFC.
Instead Gajek in [19] and Paulson in [20] have deep inspected TLS showing that it is well-written protocols that realize true secure communication sessions, not finding any effective attacks. Although, in the time was discovered and fixed some vulnerabilities, such as the recently published Lucky Thirteen attack published by AlFardan and Paterson in [21], that apply also to DTLS. The authors demonstrate that an incorrect implementation of low-level details can lead to a timing attack that can be used for plaintext recovery. This attack was discovered in November 2012 and when published, at the end of February 2013, almost all of the implementation have already addressed the problem.

6 Description of the experiments

6.1 Environment

All my measures were carried out using two PC:

- PC #1: it is a laptop from 2008, with a dual core CPU, precisely an Intel Core 2 Duo T9300 that work at 2.50GHz and 4 GB of DDR2 RAM memory that work at 333 Mhz. This was used as SIP proxy.
- PC #2: it is a laptop from 2009, which was used as load generator.

The two PC were connected using a direct Ethernet cable that connects directly the two Ethernet NIC, without any hub, switch or router. This was a decision made for not introducing network delay of variables that is not under my control, like a processing delay of an economic switch that can be very different from a professional router.
On the two PC Debian Linux in the release Wheezy (7.0) in the 64-bits version were installed with all the update released until the 1 March 2013.
As SIP registrar server I used Kamailio, the result of the merging of SIP Express Router (SER) and OpenSER, at the release 4.0. The choice of Kamailio is because it is only a SIP router without any media functionality, rather than other software like Asterisk that also integrate the media proxy. In this way, I have focused on a specialized software on SIP signaling, rather than using a multipurpose software.
For the setting up of IPsec, I used Racoon, an IKEv1 protocol implementation, and IPSEC-TOOLS, an IPsec stack. The fact that Racoon is an IKEv1 protocols does not influence the results of the test, because the SAs was created and the key agreement phase was accomplished before the test began. Furthermore, the analysis of the IPsec setup phase is beyond the scope of this thesis. Racoon was setup to authenticate the remote host using a 1024-bits RSA certificate.
The details of the environment is summarized in Table 13.

Purpose	Software
OS	Debian 7.0 64-bits
Kernel	Linux 3.2.0-4-amd64 #1 SMP Debian 3.2.35-2 x86_64 GNU/Linux
SIP router	Kamailio 4.0.0
SIP load generator	SIPp v3.3-TLS
Network	Direct CAT-5 Ethernet cable
TLS/Crypto library	OpenSSL 1.0.1e
IPsec	Ipsec-tools 0.8.0
Database	MySQL Server version: 5.5.28-1 (Debian)
Profiling tool	Sysprof 1.1.8

Table 13: Environment specification.

From the default Kamailio configuration file the unused modules were removed, while from the Route routine the processing related to other purpose were removed. The initial sanity check was left because it is a form of security mechanism, preventing the processing of malformed messages that can be dangerous or may cause performance degradation and DoS. The actual configuration file is reported in Appendix A. Instead, the SIPp's scenario

xml and an example of the Comma-Separated Values (CSV) file used is reported in Appendix B. Finally, Appendix C show the configuration used for IPSEC-TOOLS.
In order to have an idea of the performance of the two machines and an indication on the performance of the various algorithms, I run the OpenSSL integrated benchmark (OpenSSL speed) on the PC. The result is shown in Figure 10, Figure 11 and in Figure 12. This gives an idea of how heavy the various algorithms are.

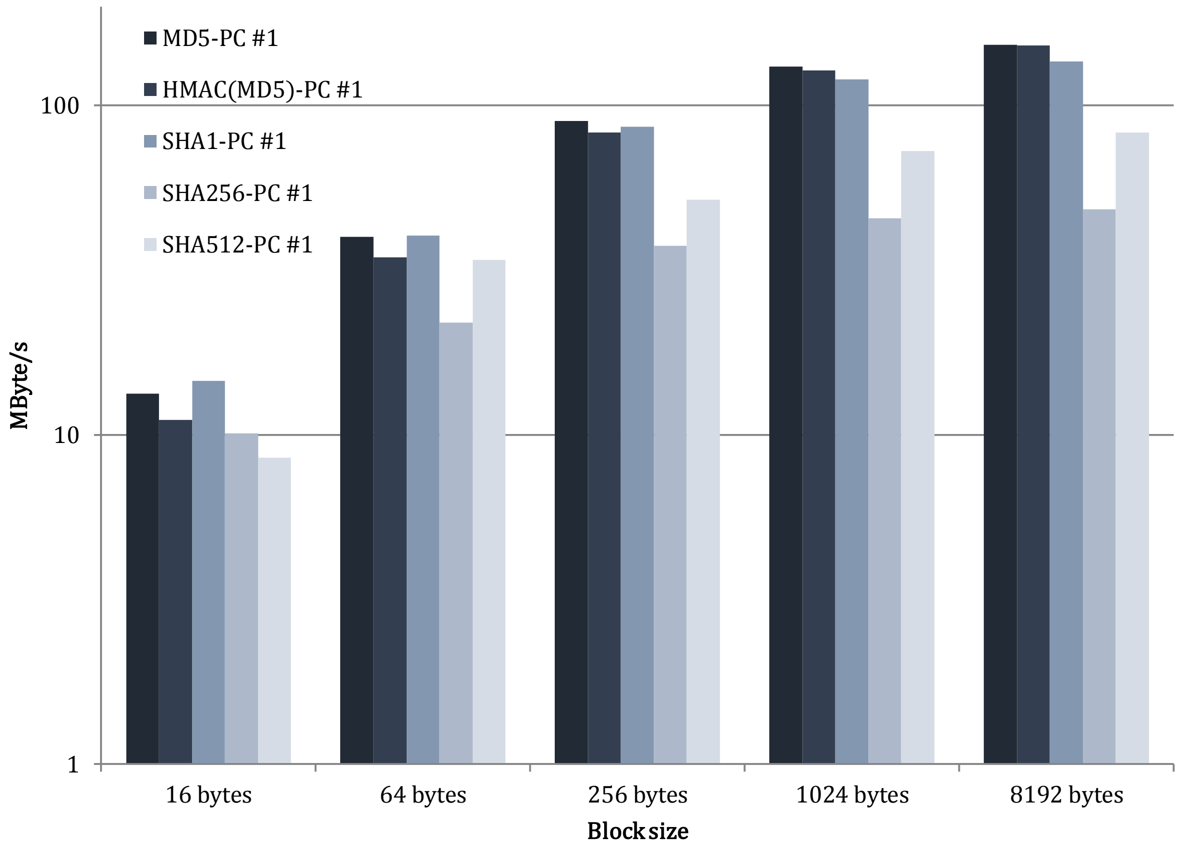


Figure 10: OpenSSL benchmark – digest algorithms.

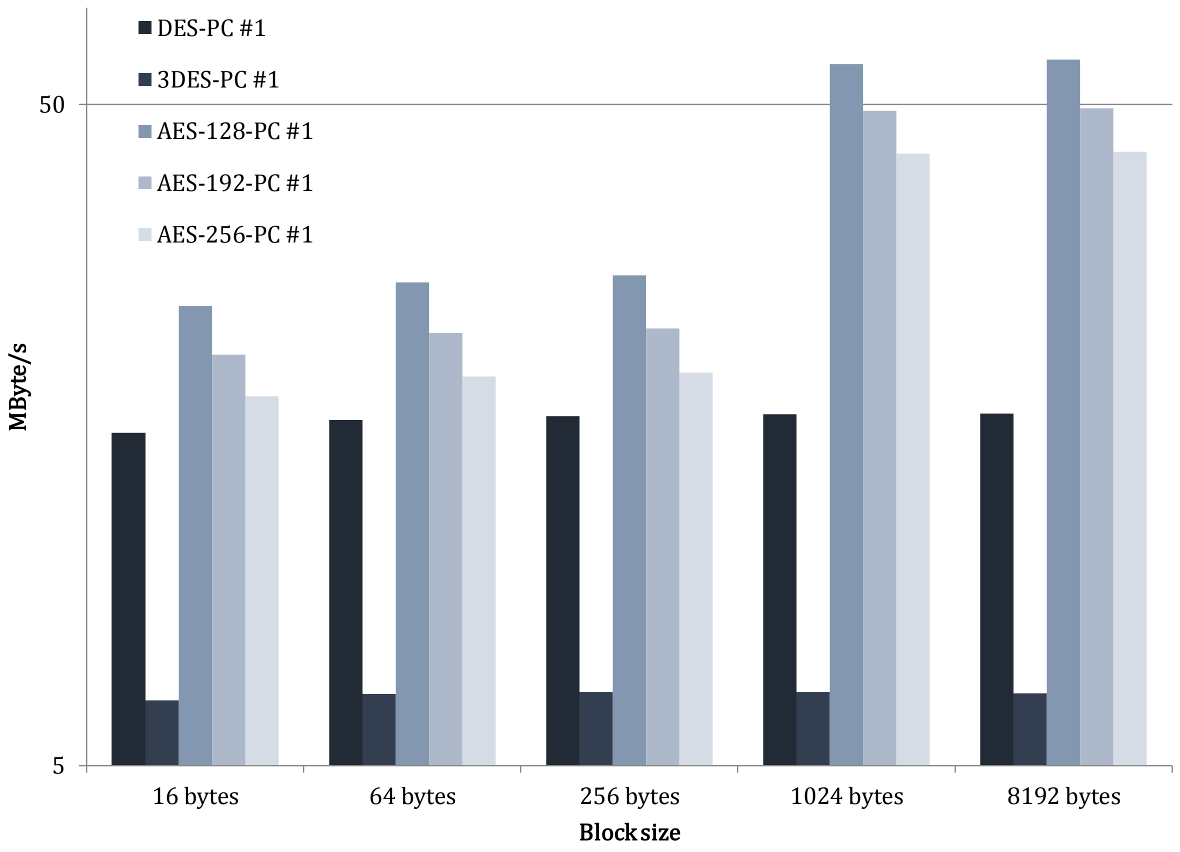


Figure 11: OpenSSL benchmark – encryption algorithms.

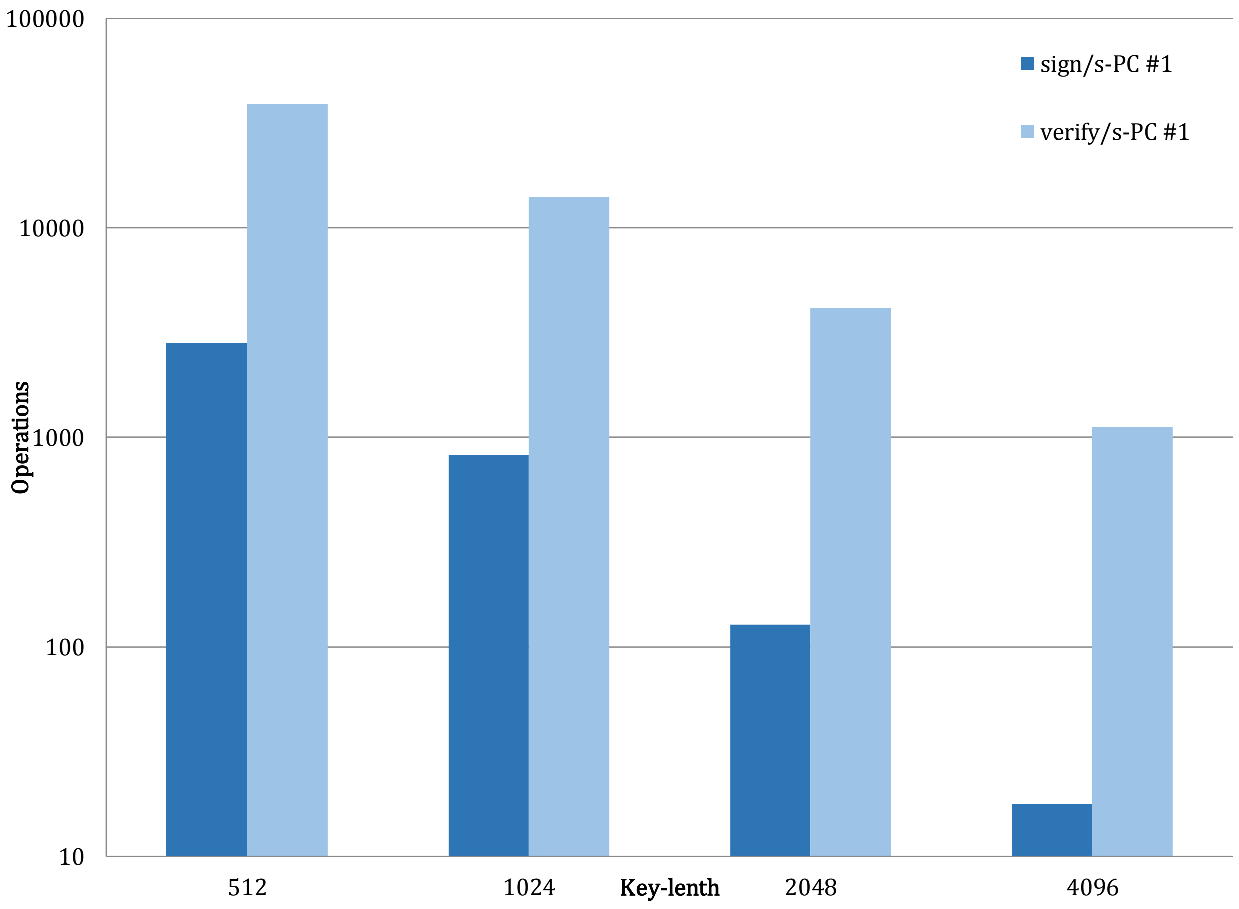


Figure 12: OpenSSL benchmark – RSA operations.

We can see that 3DES is the encryption algorithm that has the lowest throughput, while AES 128 the higher one. Thus, in my test I will use these two cases as lower and upper bound for the encryption side.
For the digest algorithms, MD5 and SHA1 are the one that performs better, while SHA512 is worse. Furthermore, according to the block size, HMAC(MD5) performs slightly worse than plain MD5, thus I will use HMAC(MD5) and HMAC(SHA512) as bounds.
Instead, for the RSA operations, it is possible to see just how heavy the operations with a 4096-bits key are, so I will use 1024 and 2048-bits key, because 512-bits is considered unsecure.

6.2 Measures

All my tests are about the processing of a REGISTER request; this is because it is a kind of SIP message that requires the server to parse the message, verify the authorization credentials, register the client and confirm the registration to the client. It is noteworthy that this kind of scenario does not introduce variables like routing, inter-domains policy, but is rather intended to measure the impact of security methods on the processing of the request in a more accurate way.
I have ran three main kinds of test:

- Code Profiling: I profiled the system while running Kamailio on the PC and loading it by the other PC whit 200 request per second per 10 minutes. This test shows in details how the security affects the performance, focusing on where the system spends more time and computational power. I will analyze the results of the profiling of code under three main aspects: Kamailio percentage, Security percentage and Operating frequency.
- Power consumption: In this experiment, I have measured the power consumption of the laptop running Kamailio. I loaded Kamailio with a number of REGISTER per second and I measured the absorbed power. For measuring power consumption, I used a power-meter attached to the alimentation of the notebook, in order to obtain stable and precise results I removed the battery, turned off the Wi-Fi, and set the LCD backlight to the minimum. The scope of this measure is to have a metrics that can be a concrete parameter for making economic considerations. If you consider that a server runs 24 hours a day, 365 days a year, increasing the power consumption of some percent points can be very expensive, and if you are a service provider and you have hundreds of servers then this increase can be a deal-breaker.
- Load test: I flood the Kamailio to see how many request per second the system can handle and what is the mean response time when varying the number of requests per second. The scope of measuring the response time is to have another metrics that in processing of the REGISTER is not so important, but can be of interest in processing other request like INVITE, because it directly affects the call-setup time. The statistic is collected using SIPp with the option "-trace_stats".

Every test was conducted in the following scenario: UDP without security mechanism, HTTP Digest Authentication over UDP, TLS, and UDP over IPsec. The first is used for have a benchmark of how much the security influences the normal SIP behavior. Table 14 shows what modules are loaded on every scenario, Table 15 and Table 16 shows what algorithms are used respectively in the IPsec and TLS scenario.

Scenario/Modules loaded	MYSQL+AUTHDB modules	TLS module
UDP without security	No	No
HTTP Digest Authentication over UDP	Yes	No
TLS	No	Yes
UDP over IPsec	No	No

Table 14: Modules loaded on each scenario.

Purpose	Algorithm		
Encryption	Null	AES-128	
Authentication	HMAC-MD5	HMAC-SHA1	HMAC-SHA512

Table 15: Algorithms used in the IPsec scenario.

Purpose	Algorithm		
Key-length	1024	2048	
Encryption	3DES	AES-128	
Authentication	HMAC-MD5	HMAC-SHA1	HMAC-SHA512

Table 16: Algorithms and key-length used in the TLS scenario.

6.3 Note

- I noted that SIPp when used to generate TLS traffics could be a bottleneck: if I use a single instance, I cannot generate more than about 80 REGISTER per second using a 2048-bits key, but if I use four instance of SIPp I can reach, 50 request per second on each instance, so 200 cumulative REGISTER per second. Using a 1024-bits key, I can generate up to 320 cumulative request per second using 4 process. Above this limit, the system is not stable anymore, and for example, the sampling period oscillate, not giving an accurate measure anymore. For this reason, on the TLS side, I usually launch multiple SIPp instance and stay under these limits.
- When carrying out tests on IPSEC mode I used only two security associations between the two PC, one in each direction. I agree that this does not represent a real environment, because in the real word in order to allow N host to connect to a server, it is required to setup 2N SAs. I made this decision because in order to simulate this on only two PCs, I should setup many IP addresses or socket, and over each one I will have to run a single SIPp instance and set Kamailio for listen on each socket. Furthermore, once an SA pair is established, this can be left in memory, also if not in use, for sometime without the need of rekeying. So my choice was made thinking that in steady state, the Security Association DataBase (SADB) was populated and the client who contacts the server does not need to make the IKE process. Note that the RFC specifies two kinds of processes for the renewal of the SAs: rekeying and reauthentication. In the first case the system considers the SA to

still be valid, but it needs to regenerate the session-key, while in the second one the parties want to verify each other, in this case the process is equal to a generation of a new SA. Rekeying is more frequent than reauthentication. Another notable thing is that rekeying and reauthentication can be done independently from the requests, so when the system is idle it can initiate the rekeying of a SA, also if not receive any request from this host. This way the host does not have to authenticate in the moment in which it is sends a request. This cannot be done in TLS.
Thus, I think that the authentications of new users, mixed to rekeying and reauthentication of existing user have for sure an impact on the performance of the systems, but this is not dramatically high like the one of the TLS. I will consider this when analyzing data in the next section.

- When carrying out tests on TLS mode, instead, I used the TLS multi-sockets option of SIPp (-t ln), this way I simulated that every host that wants to register open an independent TLS session with the server.
- For more precision also if the environment is the same the code profiling and power consumption test was run in two different moments. Therefore, the overhead introduced by the profiling tools does not affect the measure.

7 Analysis

In this section, I will review and comment the result of the measures.
Note that for simplicity in the following I will omit HMAC(), so when describing the algorithm used I will write SHA1 for HMAC(SHA1).

7.1 Code profiling

7.1.1 Kamailio percentage

In Figure 13, I report the percentage of sample that derives from the works of Kamailio. This value is the one that Sysprof indicates like "total" and includes all the time spent from the Kamailio executable and all the time spent in the external function called from it. The last row is a run of Kamailio without any security that is used to have a benchmark. TLS is used whit a 2048-bit key.
It is possible to see that when IPsec is used, the percentage of time worked by Kamailio is normally a bit smaller than the no security case. This is because the management of IPsec was made from the OS. The percentage of sample worked by Kamailio is smaller when the overhead introduced by the mechanism is higher, indeed is lower when using ESP than when using AH. Instead, when TLS is used, all the management of the security is made by Kamailio itself so the percentage grows. A different analysis is required for the HTTP digest authentication case, because it makes use of the MySQL server to check the credential, so I have reported in blue the Kamailio percentage, and in red I added the MySQL server percentage.

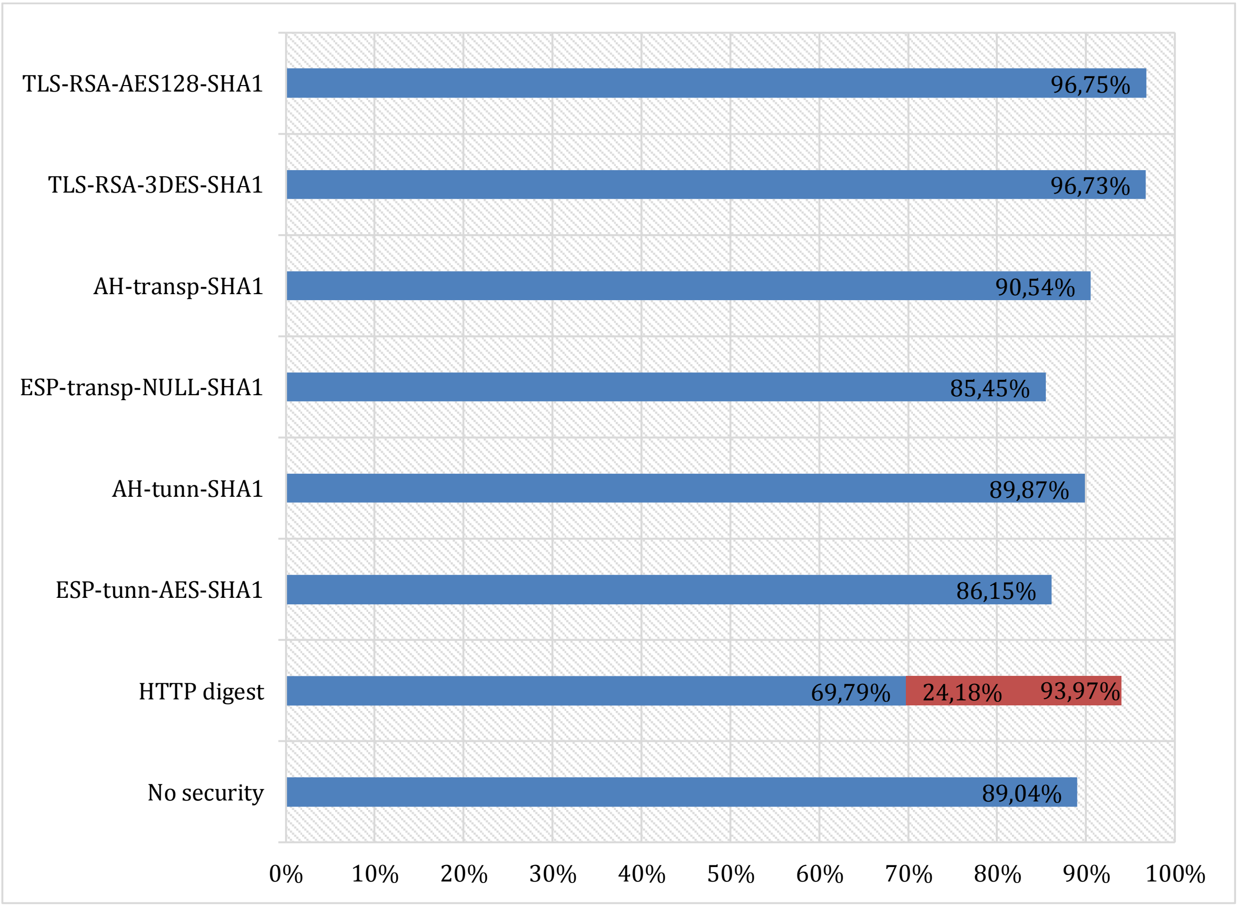


Figure 13: Time percentage spent in Kamailio.

7.1.2 Security percentage

In order to obtain a measure of what is the overhead required from the various solutions, I summed each symbols that regards the security, like the OpenSSL functions or the internal function of Kamailio that regards the digest authentication. In Figure 14, is reported the percentage of the sum of the "self-time" of security function over the total number of sample. The no security case is omitted because the number of security samples is 0, instead in the case of HTTP digest the security includes the sample of the SQL server, because it is used for checking the credentials of the user when the authentication header is received.

Furthermore, we can see that the TLS uses a huge quantity of resource. Instead, in the IPsec scenario we can see that this has about a 3 % of overhead when encryption is not used, that grows to about 5 % when encrypting the ESP with AES. In the HTTP digest authentication case, instead, adding the number of sample of Kamailio's function related to security and the sample related to the MySQL server, the percentage grows to about 27 %.

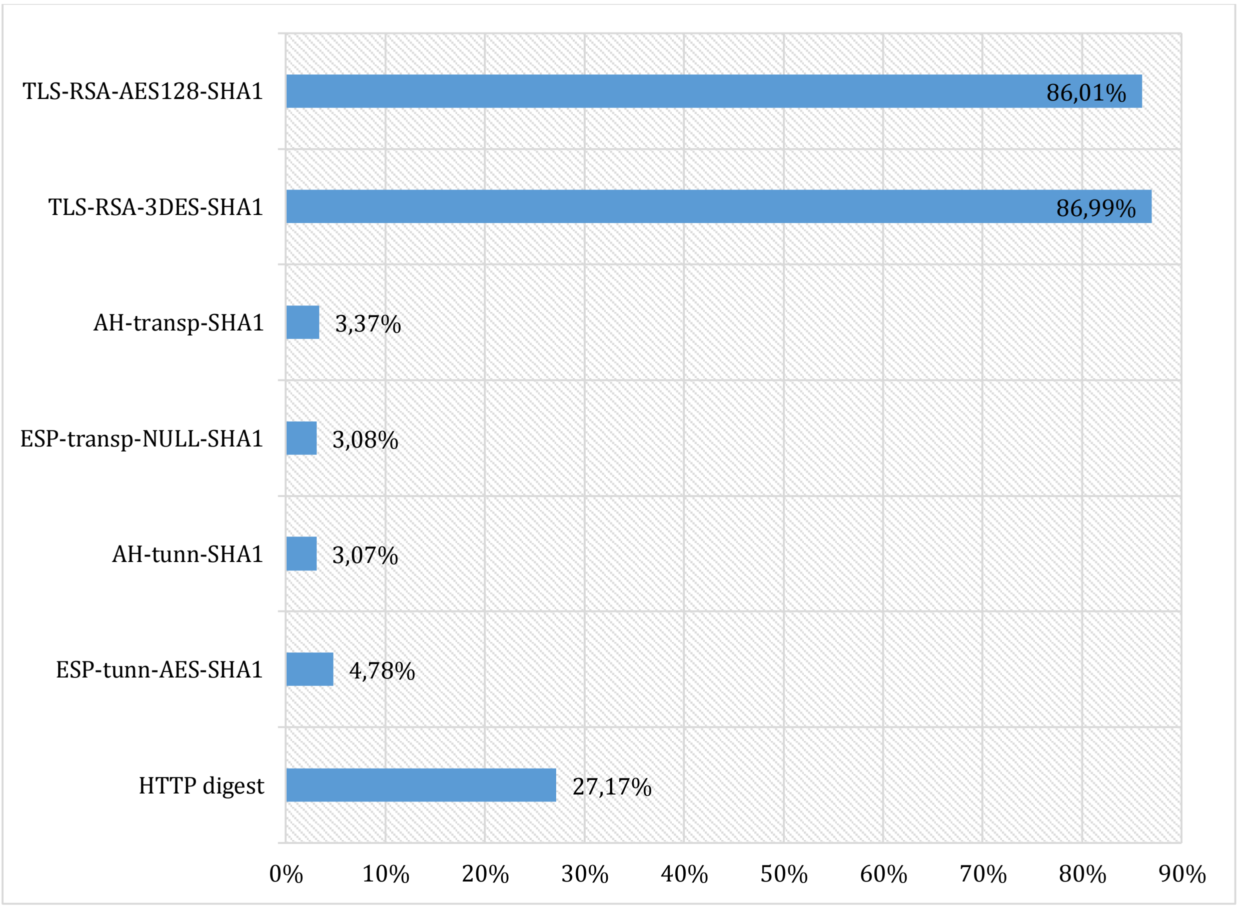


Figure 14: Time percentage spent in security operations in the whole system.

7.1.3 Operating frequency

Another analysis that could be carried out from the profiling of code is about the total number of samples. This is related to the operating frequency, because Sysprof makes use of hardware performance counters. Therefore, it is possible to see that when TLS is used the system works at higher frequency, instead when IPsec or the HTTP digest is used the frequency of operation is lower. This means that the system is less busy, and so a lower power consumption is expected. The result is showed in Figure 15.

7.1.4 Profiling overhead

The last analysis that I want to perform is just a check to verify that the profiling tools itself does not influence too much the measured value. For this reason, I have summed each symbol that is related to the profiling, and I have reported in Figure 16. As we see Sysprof introduces a small overhead, that vary between about 0,8% and 2%. This means that it is possible to neglect the variations to true system behavior.

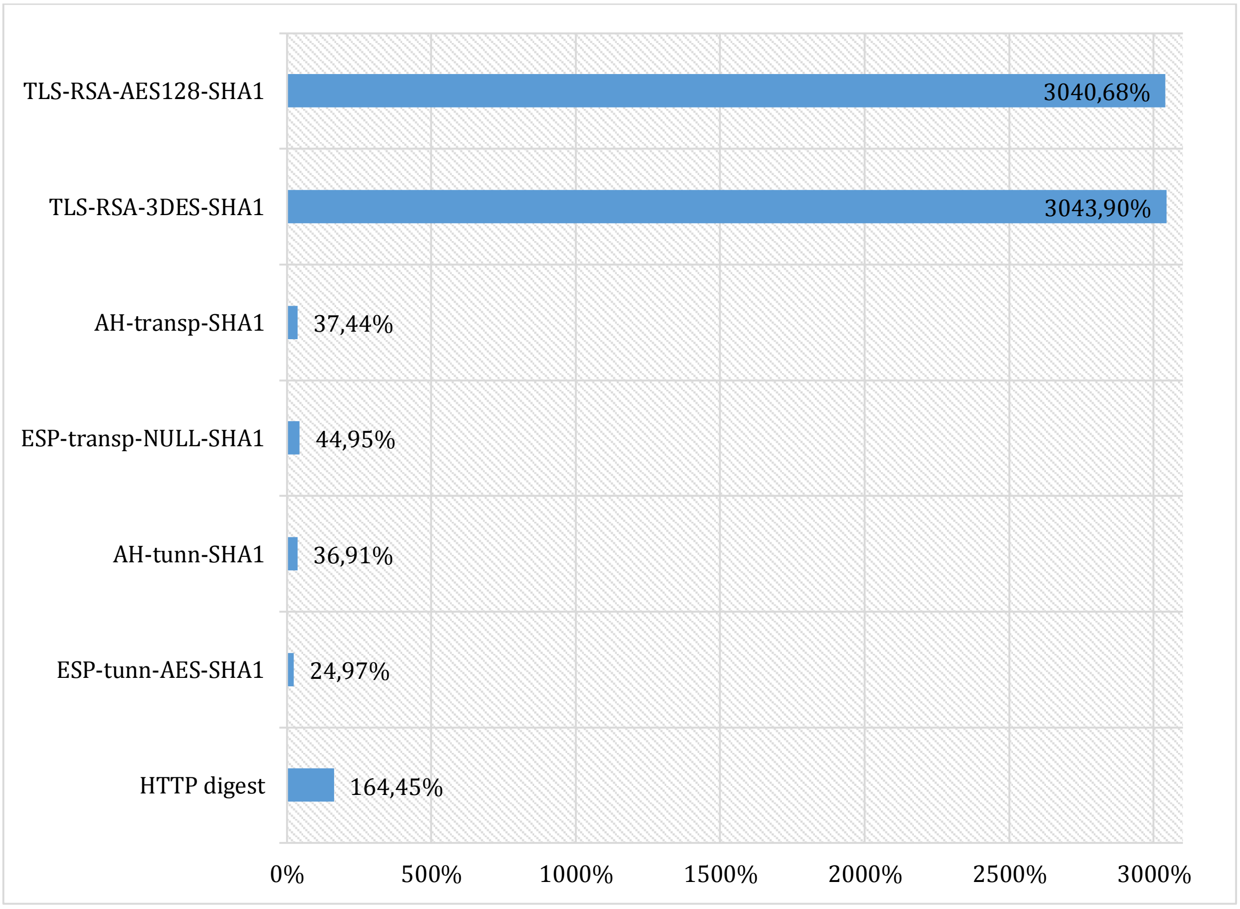


Figure 15: Total number of sample collected.

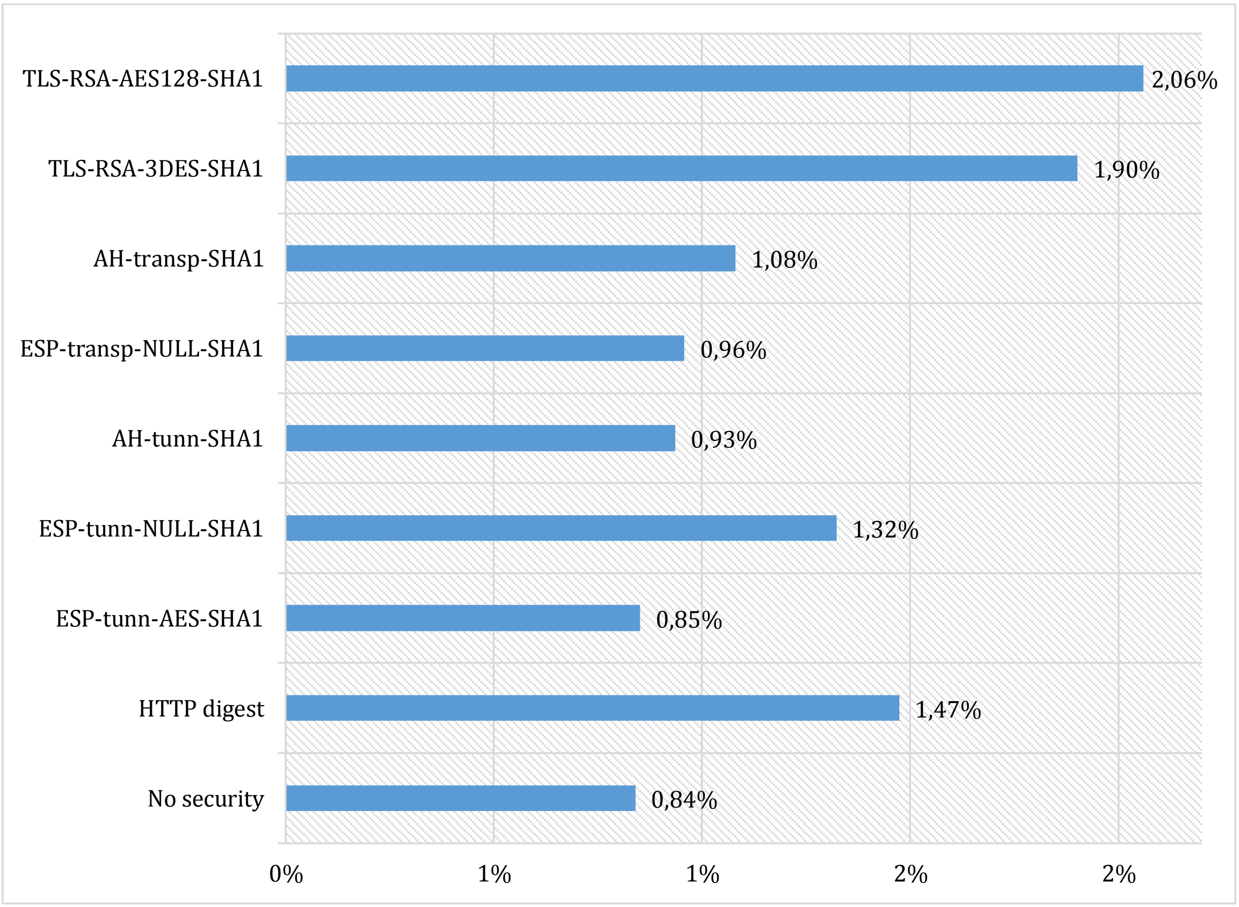


Figure 16: Profiling overhead.

7.2 Power consumption

Watching the testbed from this point of view it is possible to see how the various security methods affect the power consumption of the PC. The difference of consumption derives from the numbers of access into the memory, the read/write operations in the HarD Disk (HDD) and the frequency scaling of CPU: according to the workload, for reducing the power consumption the OS can reduce the CPU frequency. The HDD is another thing that absorb much power: it is used greatly when working in HTTP digest authentication mode, instead when using other methods the HDD access considerably lower; also in this case when not used the system can slow down or stop the rotation speed of the HDD. I measured that, when idle, in the conditions described in section 6.2, the PC absorbs 23 W. Comparing the measured value when running SIP without any security with the actual one it is possible to have a measure of the overhead introduced by the method. I divided the results in two figures; in Figure 17 I reported only the measures of TLS, HTTP digest and no security cases only. This because the throughput is too small compared to the other solutions. Instead, in Figure 18 I reported the full operating range of other methods than TLS.

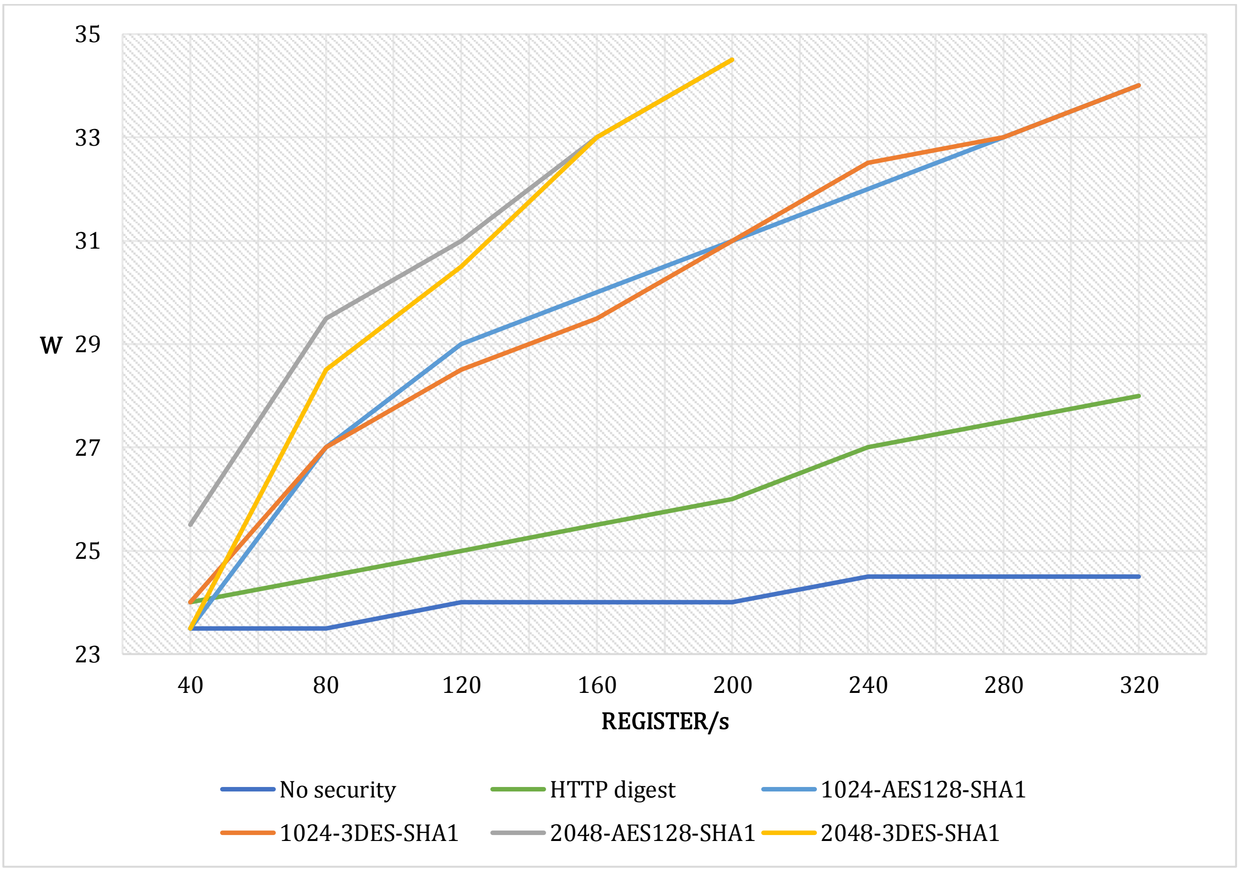


Figure 17: Absorbed power for no security case, HTTP digest authentication and various TLS configuration, when changing the request rate.

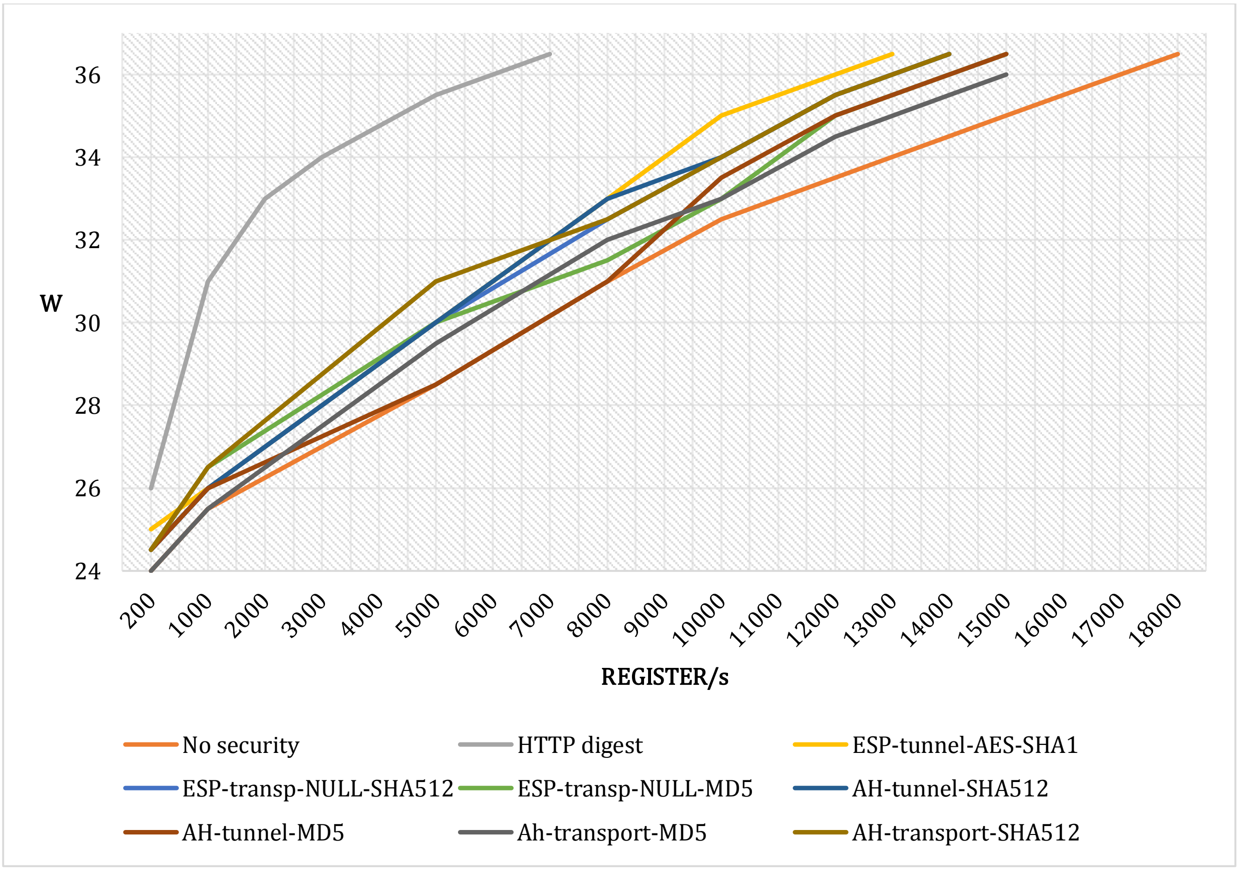


Figure 18: Absorbed power for no security case, HTTP digest authentication and various IPsec configuration, when changing the request rate.

We can see that the TLS is the most absorbing method and HTTP digest absorb a high quantity of power if compared to IPsec. So also for this point of view, the IPsec is the best performing solution.

In Table 17, I reported the measured power consumption of the system, and in Table 18, I summarized the increase of absorbed power. In both tables, the values are relative to three different request rate, precisely at 200, 5000 and 12000 REGISTER per second. We can see that at 200 request per second according to the methods used the system can absorb from 24 W of the no security case, to 34.5 W of the 2048-bits key TLS. This means an increase of about 30%. At this rate the HTTP digest authentication absorb about 8% more than the no security case; and all the IPsec configuration is between the no security case and the HTTP digest, whit the peak of about 4% reached by the ESP using AES and HMAC(SHA1) in tunnel mode and a mean of about 2%. At 5000 request per second, the TLS has no measure because as previously shown it cannot reach this throughput on my PC. At this rate, the IPsec overhead increases slightly, with a mean of about 5%, but the interesting thing is that the HTTP digest authentication grows exponentially, reaching about a 25%. Lastly, at 12000 REGISTER per second, the IPsec continues to introduce a mean overhead of about 5%.

	200	5000	12000
No security	24	28,5	33,5
HTTP digest	26	35,5	ND
AH-transport-MD5	24	29,5	34,5
AH-transport-SHA512	24,5	31	35,5
AH-tunnel-MD5	24,5	28,5	35
AH-tunnel-SHA512	24,5	30	35,5
ESP-transp-NULL-MD5	24,5	30	35
ESP-transp-NULL-SHA512	24,5	30	35,5
ESP-tunnel-AES-SHA1	25	30	36
TLS 1024-bit AES128-SHA1	31	ND	ND
TLS 1024-bit 3DES-SHA1	31	ND	ND
TLS 2048-bit AES128-SHA1	34,5	ND	ND
TLS 2048-bit 3DES-SHA1	34,5	ND	ND

Table 17: Power consumption of various security methods at 200, 5000, and 12000 request per second.

	200	5000	12000
HTTP digest	8,33%	24,56%	ND
AH-transport-MD5	0,00%	3,51%	2,99%
AH-transport-SHA512	2,08%	8,77%	5,97%
AH-tunnel-MD5	2,08%	0,00%	4,48%
AH-tunnel-SHA512	2,08%	5,26%	5,97%
ESP-transp-NULL-MD5	2,08%	5,26%	4,48%
ESP-transp-NULL-SHA512	2,08%	5,26%	5,97%
ESP-tunnel-AES-SHA1	4,17%	5,26%	7,46%
TLS 1024-bit AES128-SHA1	29,17%	ND	ND
TLS 1024-bit 3DES-SHA1	29,17%	ND	ND
TLS 2048-bit AES128-SHA1	43,75%	ND	ND
TLS 2048-bit 3DES-SHA1	43,75%	ND	ND

Table 18: Absorbed power overhead relative to the no security case at 200, 5000, and 12000 request per second.

The last analysis that I can perform is to estimate the number of requests per second that the system can handle using the same power. I take the value of 33 W, for which normally the system is at about 70-80 % of the maximum throughput, and compute the ratio between the rate of the actual method and the one that can be reached with no security. The results are shown in Table 19.
It is possible to see that with the same power consumption the TLS handles about the 2% only of the throughput of the no security case, also the HTTP digest does not perform very well reaching only the 18%; instead, in mean, the IPsec can manage about the 70%.

	Request per second	% of no security case throughput
No security	11000	100%
HTTP digest	2000	18,18%
TLS 1024-bit	280	2,55%
TLS 2048-bit	160	1,45%
IPsec	9000	81,82%

Table 19: Comparison of the achievable throughput for the various methods.

7.3 Load test

7.3.1 Achievable throughput

I will start analyzing the Figure 19, that is a graphical representation of the statistics collected using SIPp. I have reported the no security case as a system benchmark, to which I have overlapped the HTTP digest authentication and the IPsec cases. The TLS cases is not reported because, as previously mentioned, with SIPp I cannot reach the actual system limit, so I just collected a series of points in which the success rate is equal to the output rate. However, from the analysis of TLS made in the other test I can say that the limit that I have fixed are quite close to the maximum system throughput. This is because looking at the Figure 13 we can see that in the case of TLS with a 2048-bit key, already with 200 requests per second the number of samples relative to Kamailio represents about 97% of the samples of the entire system, so we are close to the maximum of what the server can handle. Furthermore, in section 7.2 the power absorbed for the system in the TLS configuration at the limit of 200 request per second with a 2048-bits key is 34 W and at 320 request per second with the 1024-bits key is 34,5 W. In other cases, we can see that the maximum power absorbed from my PC is of 36,5 W but sometimes the retransmission and the failure starts under this limits. For this reason, watching the trend reported in Figure 17 it is possible to see that the stability limit that I have used is close to the maximum throughput of the system; hence, in the following I will treat the rate reached using TLS as a lower bound.

Now, watching Figure 19, it is possible to see that also in this test HTTP digest authentication does not perform very well, with a maximum throughput of 2600 request per second, that means that with this methods my server can handle about 17% of the throughput that can be achieved if no security is used. Instead IPsec, according to the configuration, can reach really good result, with a throughput included in the range 12000-13990 request per second, whit a mean value of about 85% of the no security case. The detailed maximum throughput are reported in Table 20.

	Max throughput	% of no security case
No security	15420	
HTTP digest	2600	16,86%
ESP-tunn-AES-SHA1	12000	77,82%
ESP-tunn-NULL-MD5	13990	90,73%
ESP-tunn-NULL-SHA512	12682	82,24%
ESP-transp-NULL-SHA512	13199	85,60%
AH-tunn-SHA512	12823	83,16%
AH-transp-SHA512	13226	85,77%
AH-transp-MD5	13798	89,48%
AH-tunn-MD5	13854	89,84%
TLS 1024-bit*	320	2,08%
TLS 2048-bit*	200	1,30%

Table 20: Maximum throughput achievable by the system.
**: the values for TLS are lower bound.*

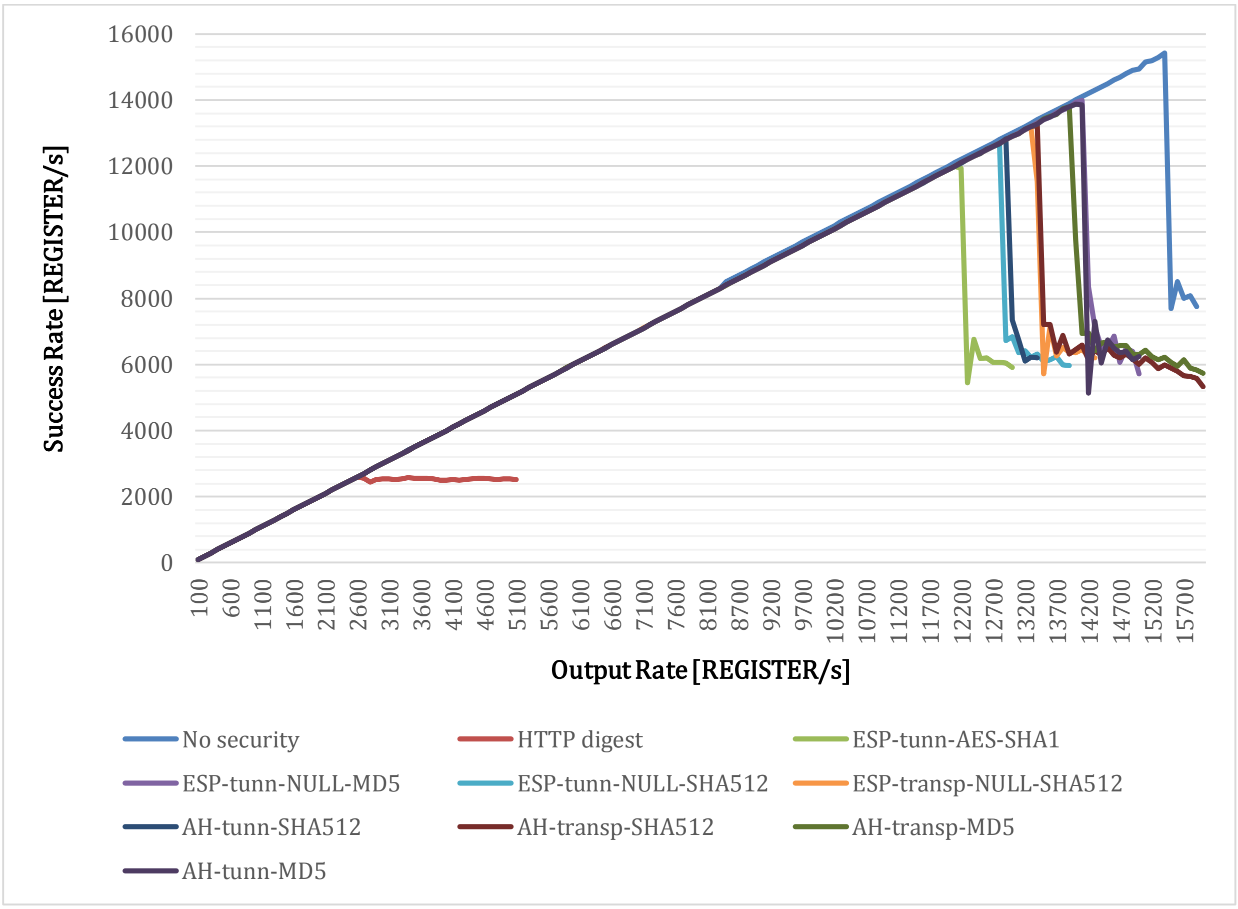


Figure 19: Measured throughput.

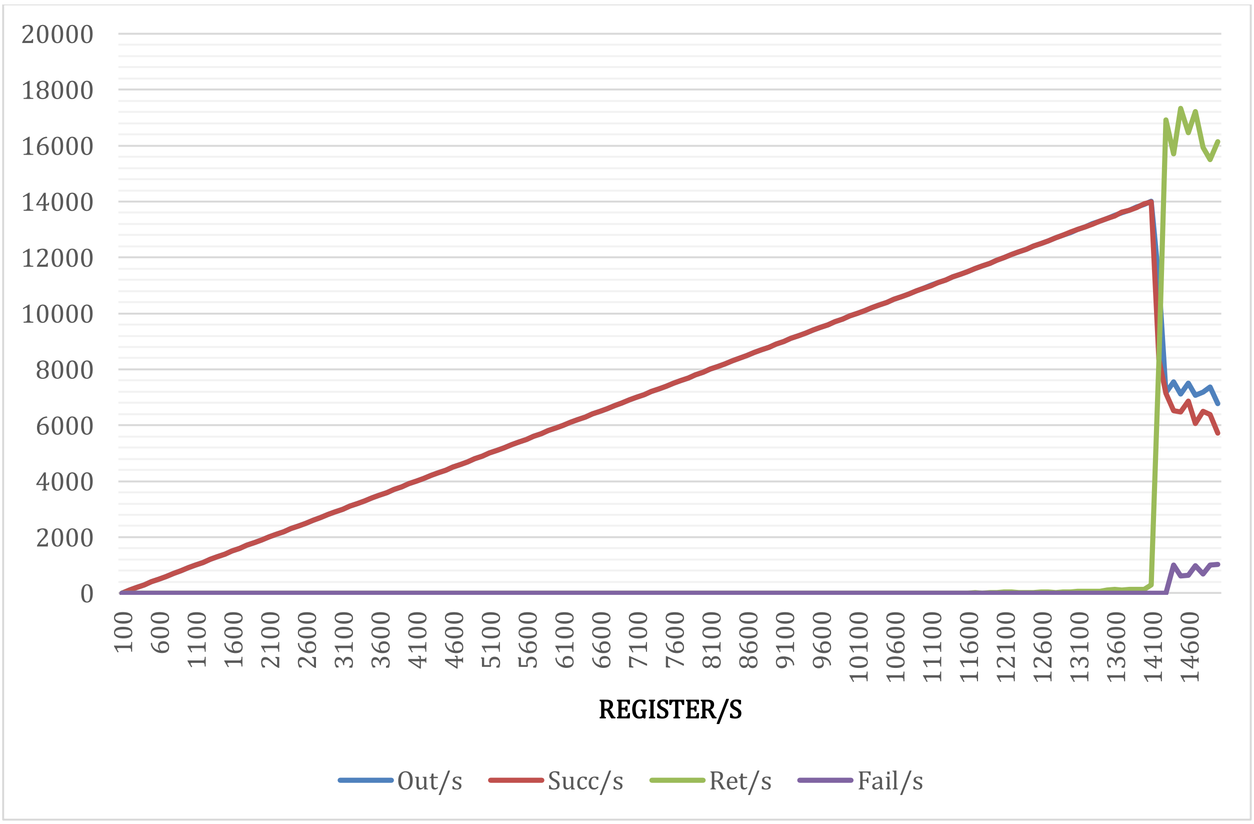


Figure 20: Detailed analysis of the ESP using NULL encryption and HMAC(MD5) in tunnel mode.

In Figure 20 is reported a detailed analysis an experiment, the one in question is on the ESP using no encryption and HMAC(MD5) for authentication. It is easy to see that the success rate (red line) grows linearly, following the imposed request rate, that is reported in the x-axis, until it reaches its maximum and after it drops down. The reason for this fall is that the server cannot fulfill the request in before the retransmission timer expires, thus every request not processed will be retransmitted generating even more workload (green line). After some retransmission, the client will finally mark the request as failed (purple line). This is the worst situation, because the user remains without the service.

7.3.2 Response time

The second analysis that can be done from this load test is to evaluate the response time, that is the time elapsed between the moment in which the client sent a REGISTER request to the server and the moment in which the client get the 200 (OK) response. When analyzed the HTTP digest authentication scheme, it still refers at the time from the first request to the receiving of the OK response. Thus, the response time reported contains both the unauthorized response and the retransmissions. In order to have more stable data, without too many peaks, I filtered the result with a moving average filter of forth order. In Figure 21 are reported the measured values for HTTP digest and some IPsec cases of interest. I do not represent all the IPsec cases because with all the series the diagrams would be difficult to read, thus I reported the IPsec lower and upper bound and two mean cases. Furthermore, also in this analysis, I treated the TLS separately and reported it in Figure 22.

Starting from Figure 21, as expected we can see that HTTP digest before the breaking point has a response time of about 1 ms, that is higher than the one of the other methods, and at the breaking point, the same seen before (2600 request per second), this time grows up vertically, due to the start of retransmission.

The no security and the IPsec cases, instead, have a slightly different behavior, slightly increasing their response time before growing up vertically. Among those tested, the ESP with AES-CBC for encryption and SHA1 for authentication in tunnel mode is the one with the worse performance, and the ESP with no encryption and HMAC(MD5) for authentication is the one that performs better. Before the breaking point all IPsec configurations, behave approximately as the no security cases.

For the TLS cases instead, we note that the key-length is the most influencing thing, because we have a couple of similar lines for every length of key. As expected, a longer key means more connection setup time, due to the certificate exchange, computation of session-key, etc., and this results in a greater response time. The last note is that TLS is the method with higher response time, with a mean of 5 ms for a 1024-bits key that grows over 10 ms for the 2048-bits key. This is negligible for a request like the REGISTER, which is usually handled by the first hop server, but if you think of an INVITE this could be a problem, since it has to traverse many nodes each one protected using TLS as enforced by the SIPS scheme. This means that at every hop will be added a delay and the user can notice it, because it has to wait longer time before the session is successfully created.

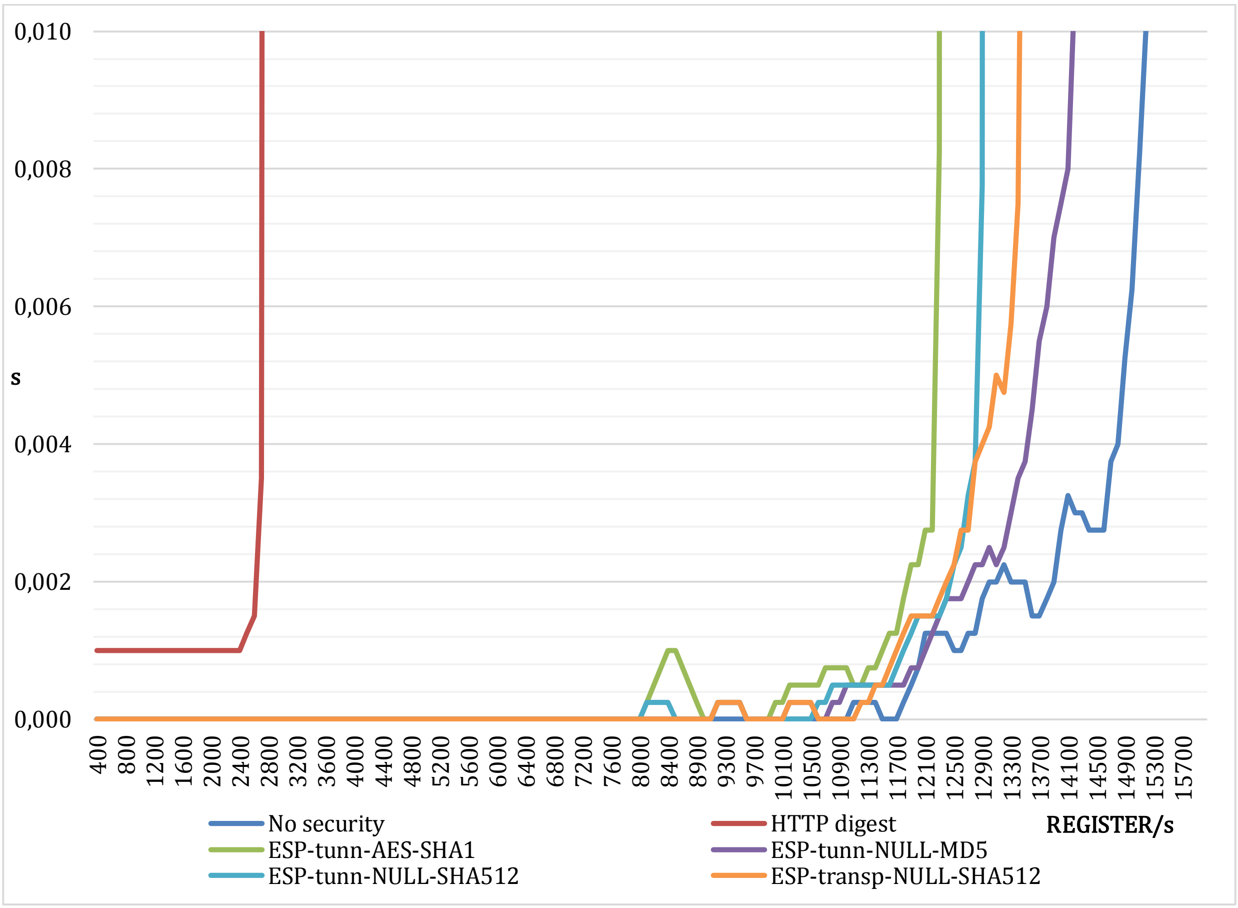


Figure 21: Response time measured for HTTP digest and IPsec scenario.

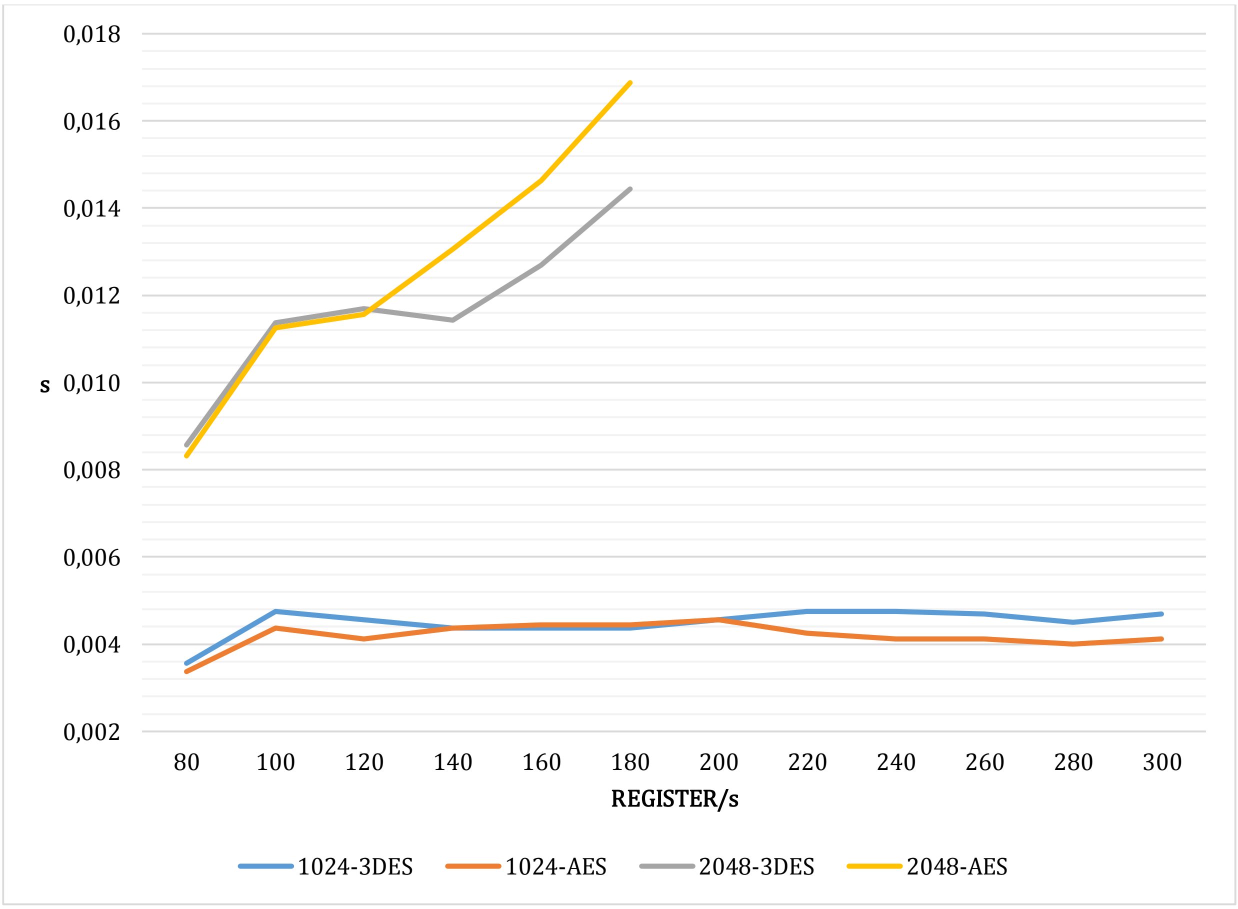


Figure 22: Response time measured in the TLS scenario.

7.4 Computational power gain over times

In section 6.1 I have reported the results of the benchmark of OpenSSL on my PC. To have an idea of what the future could be, I have made the same test on another PC, a laptop of the 2009, with a quad core CPU, precisely an Intel Core i7 Q820 @ 1.73GHz and 4 GB of RAM. I reported the results in Figure 23, Figure 24 and Figure 25. It is possible to see that by just changing the CPU it becomes possible to achieve about four times the performance achievable on PC #1. I want to remark these results because this demonstrates that even though at the moment of writing a security mechanism may have appeared too heavy, it could become usable in a few years.

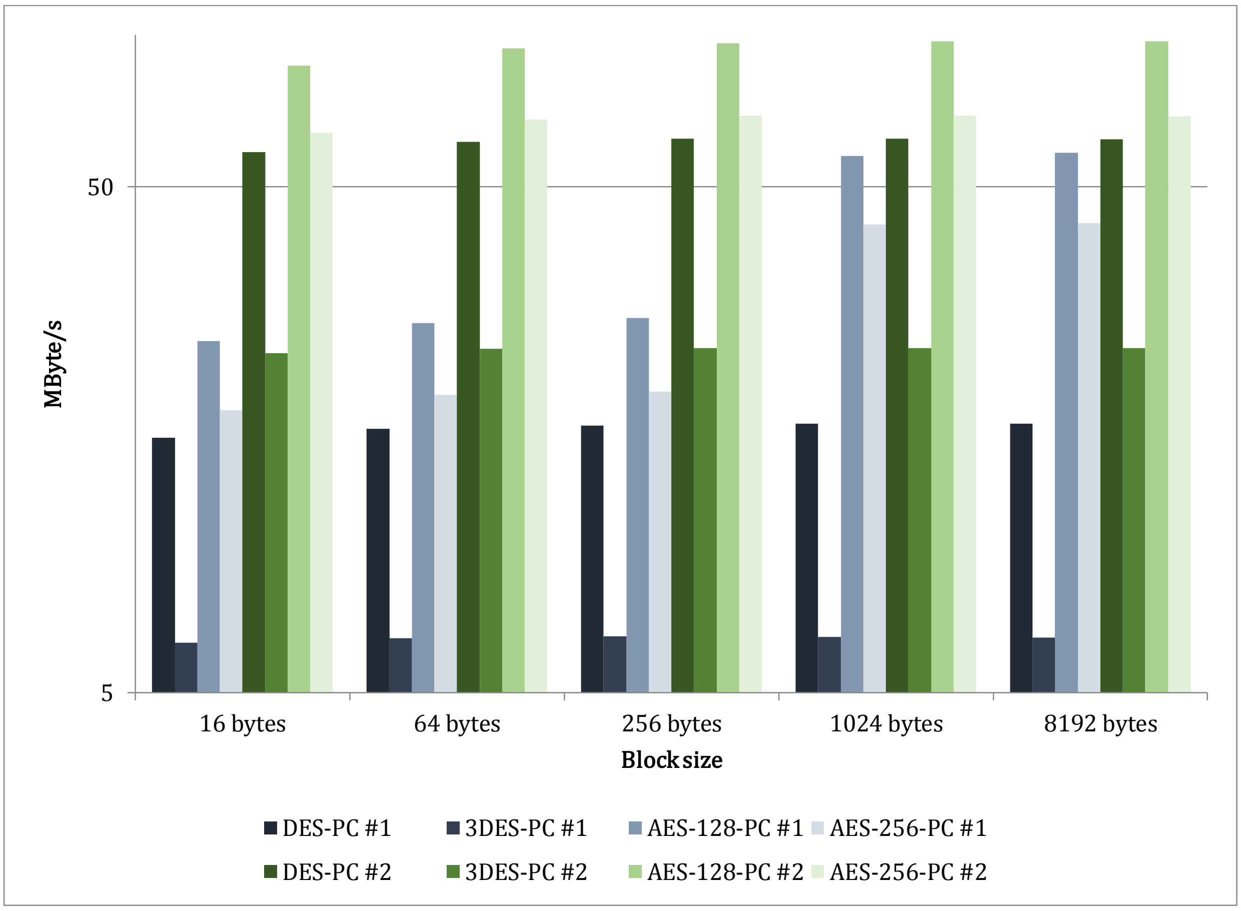


Figure 23: Comparison of OpenSSL encryption performance.

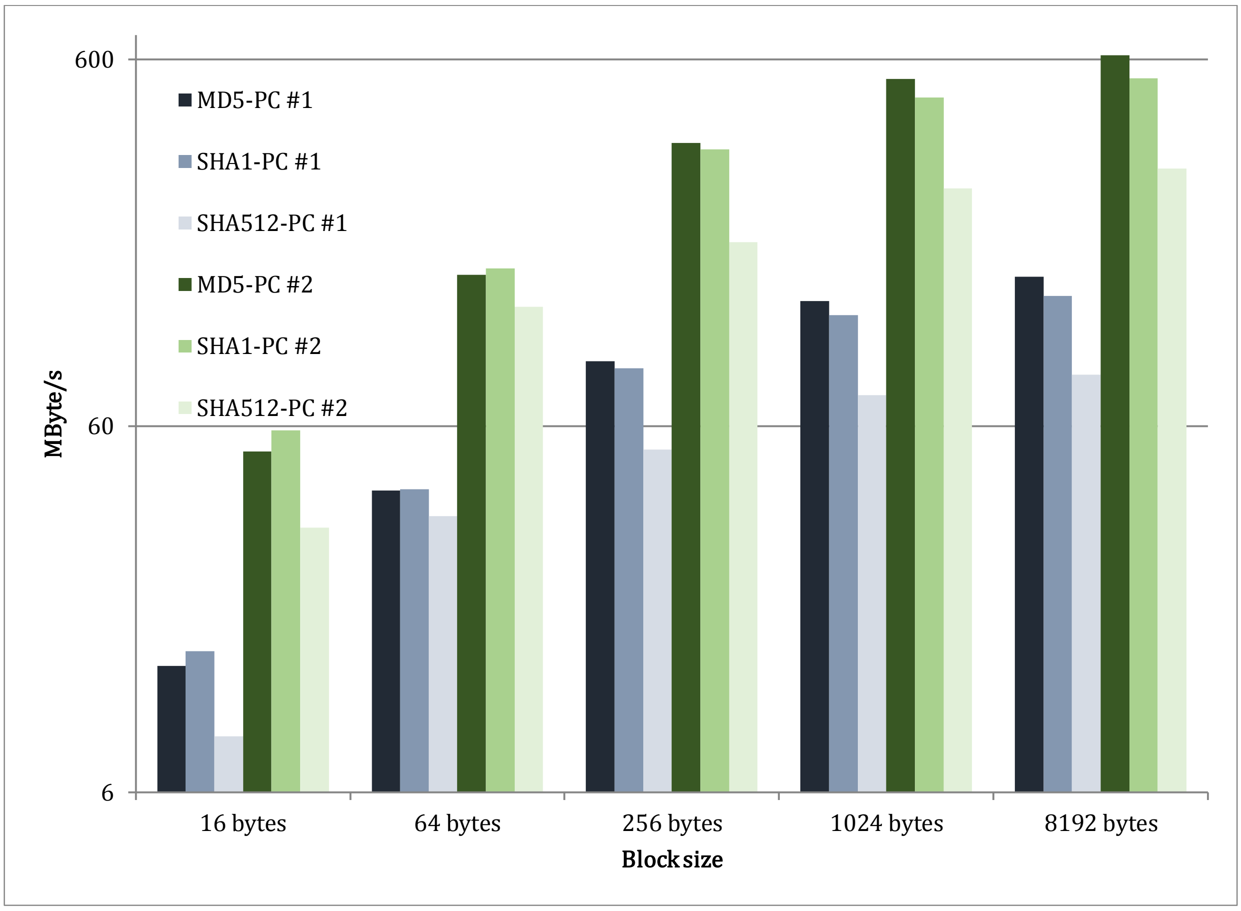


Figure 24: Comparison of OpenSSL digest algorithms performance.

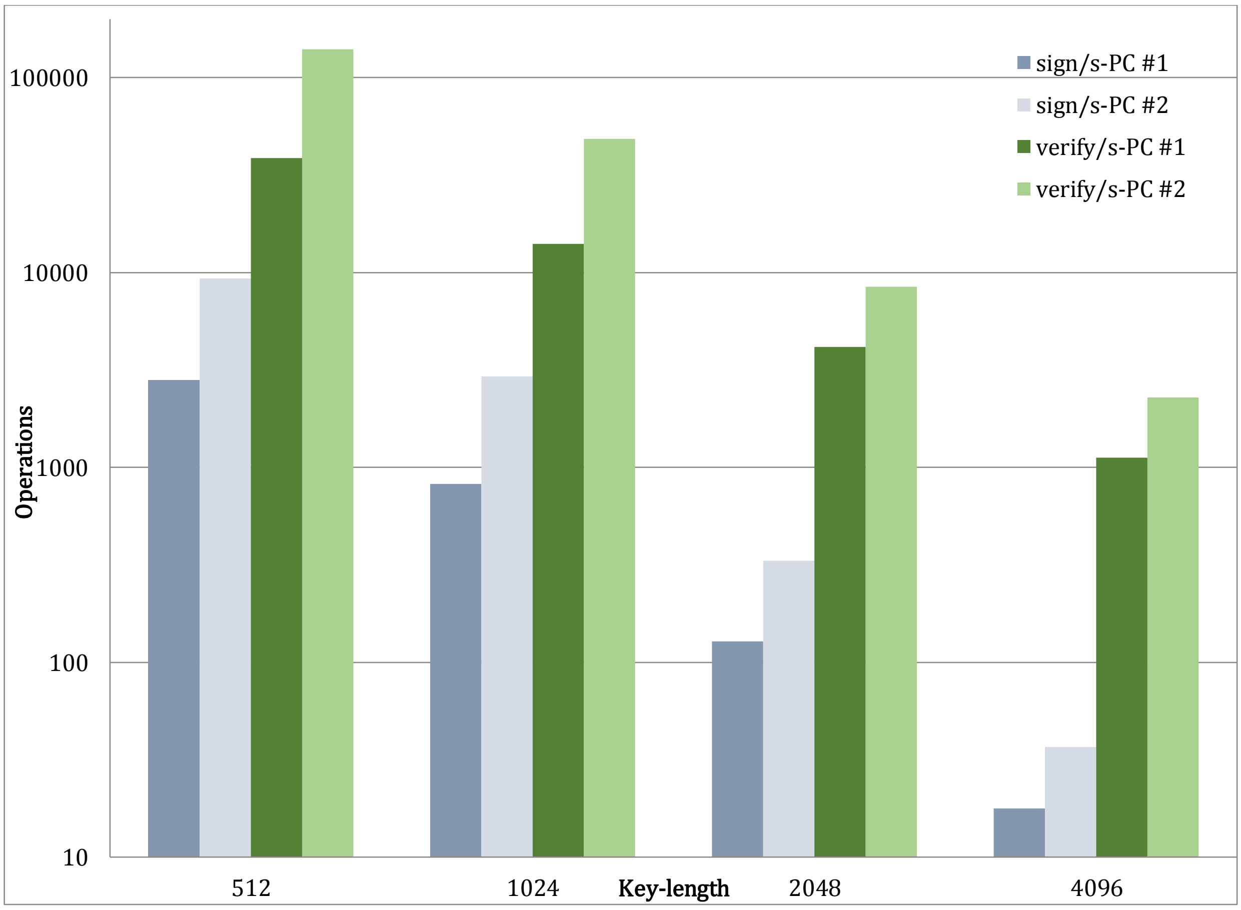


Figure 25: Comparison of OpenSSL RSA operations performance.

7.5 IKE overhead

As explained in section 6.3, all the IPsec result reported in this thesis do not include the costs of the overhead introduced by the IKE exchange.
In order to have an idea of what is the cost of this exchange, I will briefly report the analysis made by Shue et al. in [22]. They analyzed the time cost of an IKEv1 exchange, using two x86 Dell Optiplex GX Pentium IV machines. The first had a 1,66GHz processor and a 100Mbps network interface card, while the second had a 1,8GHz processor and a 1000Mbps network card. Both the machines had 512MBytes of RAM and were connected to each other through a 100Mbps Ethernet switch. The machines ran Debian Linux with a 2.6.8 kernel. For the IKE daemon they used Openswan, in version 2.3.1dr3 and Kernel Level Internet Protocol Security (KLIPS), the stack provided by Openswan for IPsec support.
Table 21 shows the cryptographic timing measurements for the IKE protocol for the responder when main mode with digital signatures as the authentication method was used for phase one. The reported numbers are averaged over 25 trial runs. As Table 21 shows, the biggest contributor to the cryptographic overheads at the VPN server was the RSA signature generation. Out of the total 373,82 ms recorded for the IKE process for 3DES, including 117,59 ms for the cryptographic operations, this one operation took approximately 21% of the total time required for the IKE process for all key sizes and encryption algorithms tested. However, verification of signatures sent by the client was much more efficient (1,07 ms, 1,09 ms, and 1,06 ms for 3DES, AES128, and AES256 respectively). The Diffie-Hellman computations were the second biggest overhead at the server, consuming a total of 35,50 ms (17,59 ms and 17,91 ms during phases one and two respectively) for 3DES. The overheads associated with symmetric key encryption and decryption operations in both phases are quite low and vary with the size of the data being encrypted. Finally, hashing contributed the least to the cryptographic overheads.

Operation	3DES	AES128	AES256
Signature generation	78,8	79,1	78,96
D-H computation	35,5	35,59	35,5
Signature verification	1,07	1,09	1,06
Encryption	0,14	0,34	0,34
Decryption	0,35	0,11	0,11
Total of cryptographic operations	118	117,93	117,66

Table 21: Details of time required for the most demanding cryptographic operation of IKE, in ms.

From this analysis, we can see that in over the overhead introduced is due to several messages that have to be exchanged, but is mainly made by the RSA sign operation. In [22] this operation take about 79 ms, but as seen in section 6.1, on PC #1 this operation take much less time and even less on PC #2 as reported in Table 22. Thus, the overhead introduced by IKE computation is much smaller on a newer machine.

Key-length	Time required on PC #1	Time required on PC #2
512	0,358	0,108
1024	1,211	0,340
2048	7,812	3,003
4096	56,034	27,275

Table 22: Time required for RSA sign operation, in ms.

7.6 Comments

As explained before, these results are representative of my pseudo-real scenario. Modifying the scenario it is possible that the results change, but because I tried to introduce the minimum number of variable that is not under direct control, the behaviors should not change significantly. The thing that could significantly change the result is the implementation: if in place of my PC is used a hardware SIP router I expect that the result could be different. This could derive from many factors, such as:

- Dedicated hardware: are based on hardware used only for processing the SIP request.
- Optimized software: makes uses of lightweight specialized OS, instead of heavy multipurpose OS.
- Hardware acceleration: can makes use of hardware to speed-up the computation of the cryptographic computations, at the same time leaving the CPU free from this work.

Moreover, the software implementation can be much or less optimized, and this can change the result. For reducing the risks, I used stable and well known software, with years of work behind.

Moving on to review the results, I deeply analyzed three of the well-known and implemented methods for securing the signaling plane of SIP. I think that they could be ordered in some way, but each one has its benefits and their disadvantage:

- No security: is feasible only in a well-protected, not accessible from internet, local domain. In every other case, a security is needed.
- HTTP digest: is the most used, and has some advantages if used in combination with an AAA mechanism. From the security point of view is weak, and its performance is not brilliant.
- IPsec: is the best performing scheme tested and it is secure, but it is more difficult to setup and may not be supported from all the machines on the net. Also it could have some implementation related-bug. The system administrator that decides to use IPsec must consider which configuration to use, because according to the network topology and the desired level of security, it is possible to have various security services and achieve different performances. The encryption is not required for the purpose of authenticating the SIP messages, but if the overhead and the performance degradation are acceptable, it allows to give less information to the opponent, for instance hiding the number of the client.
- TLS: is the most secure and easy to configure mechanism, also for the standard it is supported on all SIP UA. The disadvantage is the impact that it has on the

performance. In addition, it requires a large-scale PKI. As for IPsec, the performance are influenced by the configuration and the services used, so the implementers should decide what are the services desired and should consider the security cost.

8 Conclusions

The problem of security in SIP is beginning to be considered important for the economic damage that it may cause and for the privacy problems that could arise. The scope of this thesis it to give an overview, denoting advantages and disadvantages, and to make a performance analysis of the main actual solutions for securing the signaling plane of the SIP. Comparing and discussing them, I want to give a way to choose between them. For what we have seen, taking into account that using SIP without security is really dangerous at the moment there are many proposed solutions but only few that are really usable, HTTP digest authentication, IPsec and TLS. The first, HTTP digest, is not secure if used with weak password and should be implemented in the right way for protecting against replay attack. The last two, IPsec and TLS, both are good mechanisms, but have disadvantages: IPsec must be manually configured for each host and can present some issues related to the implementation, TLS instead is secure and scalable but has a huge performance degradation. The hope is that in the near future unsecure method will not be used anymore and hopefully new security methods that can outperform those revised in this thesis will be adopted.

9 Bibliography

1: Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

2: Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

3: Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

4: Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

5: Kent, S., "IP Authentication Header", RFC 4302, December 2005.

6: Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

7: Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, April 2007.

8: Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

9: Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

10: Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

11: Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

12: Niemi, A., Arkko, J., and V. Torvinen, "Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA)", RFC 3310, September 2002.

13: Wang, Fengjiao; Zhang, Yuqing, "A New Provably Secure Authentication and Key Agreement Mechanism for SIP Using Certificateless Public-Key Cryptography," Computational Intelligence and Security, 2007 International Conference on, vol., no., pp.809, 814, 15-19 Dec. 2007

14: S. Al-Riyami and K. Paterson, "Certificateless Public Key Cryptography," in *Proc. AsiaCrypt*, pp. 452–473, November/December 2003.

15: Acme Packet Net-Net 4500 datasheet, http://www.acmepacket.com/collateral/acm/datasheet/APKT_DS_NetNet4500.pdf

16: Butcher, D.; Xiangyang Li; Jinhua Guo, "Security Challenge and Defense in VoIP Infrastructures," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on , vol.37, no.6, pp.1152,1162, Nov. 2007.

17: Marc Bevand, http://blog.zorinaq.com/?e=42, http://blog.zorinaq.com/?e=43

18: Degabriele, J.P.; Paterson, K.G., "Attacking the IPsec Standards in Encryption-only Configurations," *Security and Privacy, 2007. SP '07. IEEE Symposium on* , vol., no., pp.335,349, 20-23 May 2007.

19: Sebastian Gajek and Mark Manulis and Olivier Pereira and Ahmad-reza Sadeghi and Jörg Schwenk. "Universally Composable Security Analysis of TLS. " p. 313-327.

20: Lawrence C. Paulson. Inductive Analysis of the Internet Protocol TLS. . p. 332--351 1997.

21: Nadhem J. AlFardan and Kenneth G. Paterson "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". Royal Holloway, University of London. Retrieved 4 February 2013.

22: Craig Shue; Youngsang Shin; Gupta, M.; Jong Youl Choi, "Analysis of IPSec overheads for VPN servers," Secure Network Protocols, 2005. (NPSec). 1st IEEE ICNP Workshop on , vol., no., pp.25,30, 6 Nov. 2005

10 Appendix A: Kamailio configuration

Below are the used Kamailio.cfg and tls.cfg.

10.1 Kamailio.cfg

```
#!KAMAILIO
##!define WITH_MYSQL
##!define WITH_AUTH
##!define WITH_TLS

####### Defined Values #########
# *** Value defines - IDs used later in config
#!ifdef WITH_MYSQL
# - database URL - used to connect to database server by modules such
#       as: auth_db, acc, usrloc, a.s.o.
#!ifndef DBURL
#!define DBURL "mysql://kamailio:kamailiorw@localhost/kamailio"
#!endif
#!endif

####### Global Parameters #########
debug=2
log_stderror=no
memdbg=5
memlog=5
log_facility=LOG_LOCAL0

fork=yes
children=4

alias="mysip.com"

listen=192.168.1.6
#listen=172.1.6.1
port=5060

#!ifdef WITH_TLS
enable_tls=yes
#!endif

tcp_connection_lifetime=3605

####### Modules Section ########
mpath="/usr/local/lib64/kamailio/modules_k/:/usr/local/lib64/kamailio/
modules/"
#!ifdef WITH_MYSQL
loadmodule "db_mysql.so"
#!endif

loadmodule "mi_fifo.so"
loadmodule "kex.so"
loadmodule "corex.so"
loadmodule "sl.so"
loadmodule "pv.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "xlog.so"
loadmodule "sanity.so"
loadmodule "ctl.so"
loadmodule "cfg_rpc.so"
loadmodule "mi_rpc.so"

#!ifdef WITH_AUTH
loadmodule "auth.so"
loadmodule "auth_db.so"
#!endif

#!ifdef WITH_TLS
loadmodule "tls.so"
#!endif

# ----------------- setting module-specific parameters ---------------
#
# ----- mi_fifo params -----#
modparam("mi_fifo", "fifo_name", "/tmp/kamailio_fifo")

# ----- registrar params -----#
modparam("registrar", "method_filtering", 1)
/* uncomment the next line to disable parallel forking via location */
# modparam("registrar", "append_branches", 0)
/* uncomment the next line not to allow more than 10 contacts per AOR
*/
#modparam("registrar", "max_contacts", 10)
modparam("registrar", "min_expires", 0)
# max value for expires of registrations
modparam("registrar", "max_expires", 1)
# set it to 1 to enable GRUU
modparam("registrar", "gruu_enabled", 0)

# ----- auth_db params -----#
#!ifdef WITH_AUTH
modparam("auth_db", "db_url", DBURL)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("auth_db", "load_credentials", "")
modparam("auth_db", "use_domain", 0)
#!endif

#!ifdef WITH_TLS
# ----- tls params -----#
modparam("tls", "config", "/usr/local/etc/kamailio/tls.cfg")
#!endif

####### Routing Logic ########
request_route {

      # per request initial checks
      route(REQINIT);

      # authentication
```

```
	route(AUTH);

	# handle registrations
	route(REGISTRAR);
}
# Per SIP request initial checks
route[REQINIT] {
	if (!mf_process_maxfwd_header("10")) {
		sl_send_reply("483","Too Many Hops");
		exit;
	}
	if(!sanity_check("1511", "7"))
	{
		xlog("Malformed SIP message from $si:$sp\n");
		exit;
	}
}
# Handle SIP registrations
route[REGISTRAR] {
	if (is_method("REGISTER"))
	{
		if (!save("location"))
			sl_reply_error();
		exit;
	}
}
# Authentication route
route[AUTH] {
#!ifdef WITH_AUTH
	if (is_method("REGISTER") || from_uri==myself)
	{
		# authenticate requests
		if (!auth_check("$fd", "subscriber", "1")) {
			auth_challenge("$fd", "0");
			exit;
		}
		# user authenticated - remove auth header
		if(!is_method("REGISTER|PUBLISH"))
			consume_credentials();
	}
	# if caller is not local subscriber, then check if it calls
	# a local destination, otherwise deny, not an open relay here
	if (from_uri!=myself && uri!=myself)
	{
		sl_send_reply("403","Not relaying");
		exit;
	}
#!endif
	return;
}
```

10.2 Tls.cfg

```
[server:default]
verify_certificate = no
require_certificate = no
method = TLSv1
#cipher_list="DES-CBC3-SHA"
```

```
cipher_list="AES128-SHA"

###1024 bit
#private_key = /usr/local/etc/kamailio/server1024key.pem
#certificate = /usr/local/etc/kamailio/server1024.pem

###2048 bit
private_key = /usr/local/etc/kamailio/server2048key.pem
certificate = /usr/local/etc/kamailio/server2048.pem

ca_list = /usr/local/etc/kamailio/calist.pem

[client:default]
verify_certificate = yes
require_certificate = yes
```

11 Appendix B: SIPp configuration

11.1 Register.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="register_client">
  <send retrans="500" start_rtd="1">
    <![CDATA[

      REGISTER sip:mysip.com SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:[field0]@mysip.com>;tag=[call_number]
      To: sut <sip:[field0]@mysip.com>
      Call-ID: [call_id]
      CSeq: 4 REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      Expires: 60
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]
    ]]>
  </send>

  <recv response="200" crlf="true" rtd="1">
  </recv>
</scenario>
```

11.2 Register-whith-HTTP-digest.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="register_client">
  <send retrans="500" start_rtd="1">
    <![CDATA[

      REGISTER sip:mysip.com SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:[field0]@mysip.com>;tag=[call_number]
      To: sut <sip:[field0]@mysip.com:[local_port]>
      Call-ID: [call_id]
      CSeq: 4 REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      Expires: 60
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]
    ]]>
  </send>

  <recv response="401" auth="true">
  </recv>

  <send retrans="500">
    <![CDATA[

      REGISTER sip:mysip.com SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From sipp <sip:[field0]@mysip.com>;tag=[call_number]
      To: sut <sip:[field0]@mysip.com:[local_port]>
      Call-ID: [call_id]
      [field1]
      CSeq: 4 REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      Expires: 60
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]
    ]]>
  </send>

  <recv response="200" rtd="1">
  </recv>

</scenario>
```

11.3 User.csv

The following are the first rows of the user.csv.

```
SEQUENTIAL
1;[authentication username=1 password=1]
2;[authentication username=2 password=2]
3;[authentication username=3 password=3]
…
```

12 Appendix C: IPsec-tools configuration

12.1 Racoon.conf

The following is the interesting part of the Racoon configuration file. In the “sainfo” part, the encryption_algorithm can be either AES or NULL_ENC according to the configuration. The authentication_algorithm instead can be HMAC_MD5, HMAC_SHA1 or HMAC_SHA512.

```
remote anonymous {
        exchange_mode main,aggressive;
…
        proposal {
                encryption_algorithm aes;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group 2;
        }
…
}

sainfo anonymous {
        pfs_group 2;
        encryption_algorithm aes;
        authentication_algorithm hmac_sha1;
        compression_algorithm deflate;
}
```

12.2 Ipsec-tools.conf

The following is the interesting part of the IPsec-tools configuration file. According to the desired service uncommented the corresponding lines.

```
#!/usr/sbin/setket -f
flush;
spdflush;

#######ESP
spdadd 172.1.5.1 172.1.6.1 any -P out ipsec
     esp/tunnel/192.168.1.5-192.168.1.6/require;
spdadd 172.1.6.1 172.1.5.1 any -P in ipsec
     esp/tunnel/192.168.1.6-192.168.1.5/require;

#spdadd 172.1.5.1 172.1.6.1 any -P out ipsec
     esp/transport//require;
#spdadd 172.1.6.1 172.1.5.1 any -P in ipsec
     esp/transport//require;

#######AH
#spdadd 172.1.5.1 172.1.6.1 any -P out ipsec
     ah/tunnel/192.168.1.5-192.168.1.6/require;
#spdadd 172.1.6.1 172.1.5.1 any -P in ipsec
     ah/tunnel/192.168.1.6-192.168.1.5/require;

#spdadd 172.1.5.1 172.1.6.1 any -P out ipsec
     ah/transport//require;
#spdadd 172.1.6.1 172.1.5.1 any -P in ipsec
     ah/transport//require;
```