NSIS Working Group
Internet-Draft
Expires: April 20, 2006

H. Tschofenig
Siemens
P. Eronen
Nokia
October 17, 2005

Analysis of Options for Securing the Generic Internet Signaling Transport (GIST)
draft-tschofenig-nsis-gist-security-00.txt

Status of this Memo

Copyright Notice

Abstract

This document investigates the different options for securing the Generic Internet Signaling Transport (GIST) protocol with the goal of using existing credentials, user and policy databases and other security infrastructure. We examine, among other options, Transport Layer Security (TLS) with X.509 PKI, Extensible Authentication Protocol (EAP), and 3GPP Generic Bootstrapping Architecture (GBA).

Table of Contents

1. Introduction and Problem Statement

1.1. Background

When designing security mechanisms for NSIS, two important design decisions that have to be made are how exactly the entities involved are authenticated, and how authorization decisions (e.g., whether or not to process a request received from an authenticated peer) are made. In many cases, these two aspects are intertwined: usually the identity (identifier) of the requestor is considered when making an authorization decision, and often the information about how an entity can be authenticated and what resources that entity is allowed to access is stored at the same place.

How exactly the authorization decision is made depends a lot on the NSLP in question; the details are mostly beyond the scope of this document. This document adopts a model where the GIST layer makes only very simple decisions: either it drops a request, or delivers it to the NSLP for processing together with whatever authenticated information about the requestor can be found (e.g., identities authenticated by security mechanisms at the GIST layer or below).

While NSLP-specific policy issues are mostly beyond the scope of this document, it is worthwhile to note that there is an important dependency. If the NSLP's policy expects as input authenticated information GIST cannot provide (and the NSLP cannot obtain itself), the policy cannot be effectively enforced. For instance, if the NSLP is faced with the decision whether to allow opening a firewall pinhole with certain addresses and ports, knowing the request came from a peer authenticated as "joe@example.com" may not be that helpful -- unless the NSLP can somehow look up this identity from some access control list to find out what kind of rules Joe is allowed to create. Thus, authenticating a particular identity (identifier) may not be always needed if the identifier is not actually used in making the authorization decision (but that authentication may still be required for some other purpose, such as

auditing).

1.2. Problem Statement

In many NSIS usage scenarios it is not realistic to assume that new authentication credentials and user/policy databases would be deployed just for NSIS. This is especially the case when the entities are end-user terminals rather than routers or servers in the network.

In this document, we examine how existing security infrastructure can be used to secure NSIS communication. As "existing security

infrastructure" heavily depends on the environment, a single solution is unlikely to be appropriate for all usage scenarios. Thus we examine several options.

We have currently selected three quite different options for consideration.

Section 2 describes how Transport Layer Security (TLS) authenticated with certificates can be used to secure GIST. In this case, entities of the existing infrastructure (PKI) are not actively involved during NSIS session (except possibly for certificate revocation checks). In many other cases, infrastructure elements participate in the exchange when authentication/authorization is needed.

One example is given by Extensible Authentication Protocol (EAP) [RFC3748], as typically embedded in IETF RADIUS/Diameter based AAA infrastructure [RFC3579] [RFC4072]. This is described in Section 3. As EAP alone does not provide mechanisms for authenticating or encrypting the actual NSIS messages, it has to be used together with another protocol. Several alternatives of what this protocol could be and how it fits together with EAP, GIST and the NSLP in question are discussed in this section.

Section 4 presents another example where infrastructure entities participate in the exchange. The 3GPP Generic Bootstrapping Architecture (GBA) [TS33.220] includes an on-line key distribution center, called Bootstrapping Server Function (BSF), that is responsible for distributing keys and authorization-related information to the parties involved in the actual protocol (NSIS

peers).

These three cases are not meant to be exhaustive; Section 5 lists several other possibilities that are not yet considered in this document.

1.3. Disclaimer and Limitations

This document is very much work-in-progress, and its goal is to enable discussion rather than present finished solutions.

Many of the options described may not be suitable for GIST due to various technical and non-technical reasons we do not yet fully know. Nevertheless, the authors believe analyzing options that in the end are rejected is valuable to gain better understanding of why they are not suitable.

In this document, we limit our consideration to cryptographic mechanisms that are visible and explicitly controlled by NSIS. In

some environments, NSIS may be able to rely on the security of the routing infrastructure and link-layer security mechanisms instead.

Consider, for instance, an ISP offering dial-up Internet access using PPP. An NSIS-capable NAT/firewall [NATFW] placed "near" the NAS (Network Access Server) might be able to assume that any packet it receives from its NAS-side interface comes from a valid customer of the ISP, and that the client who sent the packet is the current "owner" of the IP address found in the source address field. This assumption depends on specific characteristics of the environment: the NAS provides access only to authenticated customers and prevents IP spoofing by other clients; possibly additional link-layer security mechanisms are used to secure the link between the client and the NAS; and the network between the NAS and the NAT/firewall is "sufficiently secured" using various various physical and administrative methods (and possibly cryptographic link-layer and/or network layer mechanisms as well).

2. Transport Layer Security (TLS) with X.509 PKI

2.1. Introduction

TLS [RFC2246] is one of the most popular security mechanisms for protecting application-layer protocols. TLS supports authenticating the parties using public-key certificates [RFC2246], pre-shared keys [TLS-PSK], and Kerberos [RFC2712], and also includes an "anonymous" Diffie-Hellman key exchange that does not itself authenticate the parties. TLS is usually run over TCP or SCTP, but datagram TLS [DTLS] also allows the use of unreliable transports such as UDP.

2.2. Proposal Discussion

In this section, we consider how TLS authenticated using certificates

can be used to protect GIST messaging over TCP. The use of datagram TLS and other authentication mechanisms is left for further specification.

This approach is suitable when an existing authentication infrastructure based on X.509 certificates is present, or a new one can be deployed for NSIS purposes. This clearly does not cover all possible scenarios for NSIS: for instance, assuming that all users have X.509 certificates has proven to be unrealistic in many environments.

However, we do allow flexibility in what kind of semantics the certificates have. X.509 certificates bind together one or more identifiers and a public key, but sometimes they have additional authorization semantics: the certificate either implicitly or explicitly grants some kind of permission to do something (that is not simply an identifier uniquely identifying the subject). For instance, we do not rule out a certificate essentially saying the "subject is a valid NSIS QoS NSLP node in Autonomous System (AS) 64512", although we not anticipate seeing such certificates in the near future.

2.3. GIST API Viewpoint

To support this flexibility, we make certificates explicitly visible in the abstract GIST API. This allows different NSLPs to have different certificates and semantics for interpreting them.

- An NSLP can specify the certificate (and private key) to be used for this NSLP's traffic. Most likely this operation is done only once when the node is started.

- When sending a message, the NSLP can verify the peer's certificate before the message is actually sent. This is done using the MessageStatus primitive; certificates and other security details are contained in the Transfer-Attributes parameter.

- When a message is received, the certificates are given to the NSLP for verification, contained in the Transfer-Attributes parameter of the RecvMessage primitive.

- When sending a message, the NSLP can also specify the peer's certificate explicitly: this is useful when the NSLP wants to send a message to the same peer a message was received.

For the first procedure, we define a new primitive in the abstract API:

ConfigureSecurity(Mechanism, Local-Certificates, Local-Key, [TLS-Options])

Mechanism: 'tls-certificates'

Local-Certificates: One or more X.509 certificates that will be presented to the peer during the TLS handshake.

Local-Key: A private key (or a "handle" to a private key) that will be used for authentication during the TLS handshake. Depending on the certificates, the key type could be RSA, DSA, Diffie-Hellman, Elliptic Curve DSA, or Elliptic Curve Diffie-Hellman [TLS-ECC].

TLS-Options: Optional preferences about TLS details, such as encryption algorithms and their strengths, or perfect forward secrecy (PFS).

It is an open question whether the ConfigureSecurity primitive should be considered as part of the GIST API or not. One option would be to limit the GIST API to those primitives that map directly to individual messages or flows, and consider ConfigureSecurity to be a part of some other as-yet-undefined configuration/management interface between the NSLPs and GIST.

For SendMessage, MessageStatus, and RecvMessage, we define a number of security-related Transfer-Attributes:

Security: 'integrity' and optionally 'confidentiality'

Mechanism: 'tls-certificates'

Peer-Certificates: In MessageStatus and RecvMessage primitives, the list of X.509 certificates presented during the TLS handshake. The first certificate contains the public key actually authenticated by TLS. In SendMessage primitive, this parameter can be omitted; if included, it includes the end entity certificate that must be authenticated during the TLS handshake.

TLS-Options: Information about TLS details, such as exactly which ciphersuite was chosen and what properties (algorithm, strength, perfect forward secrecy) it has.

2.4. GIST Protocol Viewpoint

From GIST point of view, TLS is used to secure reliable messaging over TCP. Thus, it is used together with the Forwards-TCP protocol defined in [GIST]; the protection is not applied to datagram mode messages.

A new MA-Protocol-ID value, "TLS-Certificates", needs to be assigned to allow the negotiation of TLS using the Stack-Proposal object in the GIST-Query/Response phase. This use of TLS does not require any additional information in the Stack-Configuration-Data object (in addition to the port number for Forwards-TCP).

2.5. Example

When a NSLP is started, it must at least configure the key/certificate for establishing and accepting messaging associations protected with TLS.

```
Host A NSLP --> GIST:
ConfigureSecurity(Mechanism=tls-certificates,
Local-Certificates=CERT_A, Local-Key=KEY_A)

          Host B: GIST <-- NSLP
          ConfigureSecurity(Mechanism=tls-certificates,
          Local-Certificates=CERT_B, Local-Key=KEY_B)
```

Later, host A's NSLP sends a message

```
Host A: NSLP --> GIST:
SendMessage(NSLP-Data=DATA1, NSLP-Message-Handle=H1,
MRI=MRI1, Transfer-Attributes: Security=integrity+confidentiality,
Mechanism=tls-certificates, ...)
```

GIST notices that a new messaging association is needed, and initiates Query/Response

```
   UDP(GIST-Query(
   Common-Header,
   Message-Routing-Information=MRI1,
   Session-Identification,
   Network-Layer-Information,
   Query-Cookie,
   Stack-Proposal=Forwards-TCP+TLS-Certificates,
   Stack-Configuration-Data=NONE)) -->
```

Eventually this reaches host B and it replies

```
     <-- UDP(GIST-Response(
         Common-Header,
         Message-Routing-Information=MRI2,
         Session-Identification,
         Network-Layer-Information=IP_B,
         Query-Cookie,
         Responder-Cookie,
         Stack-Proposal=Forwards-TCP+TLS-Certificates,
         Stack-Configuration-Data=PORT_B))
```

Next, host A establishes a TCP connection to IP_B:PORT_B, and starts the TLS handshake

```
   TLS(ClientHello) -->

     <-- TLS(ServerHello, Certificate=CERT_B, [ServerKeyExchange],
             CertificateRequest, ServerHelloDone)

   TLS(Certificate=CERT_A, ClientKeyExchange, Finished) -->

     <-- TLS(Finished)
```

On host A, GIST now asks the NSLP to verify the certificates before actually sending the NSLP data:

```
   Host A: NSLP <-- GIST
   MessageStatus(NSLP-Message-Handle=H1, Transfer-Attributes:
   Security=integrity+confidentiality, Mechanism=tls-certificates,
   Peer-Certificates=CERT_B, TLS-Options)

   Host A: NSLP --> GIST:
   OK
```

Next GIST sends the Confirm message containing the actual NSLP data:

```
   TLS(GIST-Confirm(Common-Header,
   Message-Routing-Information=MRI1,
   Session-Identification,
   Network-Layer-Information,
   Responder-Cookie,
   Stack-Proposal=Forwards-TCP+TLS-Certificates,
   NSLP-Data=DATA1)) -->
```

GIST on host B delivers the message to the NSLP, together with the peer certificate and other security-related attributes:

```
         Host B: GIST --> NSLP
         RecvMessage(NSLP-Data=DATA1, MRI=MRI1,
         Transfer-Attributes: Security=integrity+confidentiality,
         Mechanism=tls-certificates, Peer-Certificate=CERT_A,
         TLS-Options)
```

The message is now processed by the NSLP, which may eventually decide to reply...

```
         Host B: GIST <-- NSLP:
         SendMessage(NSLP-Data=DATA2, MRI=MRI2,
         Transfer-Attributes: Security=integrity+confidentiality,
         Mechanism=tls-certificates, Peer-Certificate=CERT_B, ...)

      <-- TLS(GIST-Data(Common-Header,
          Message-Routing-Information=MRI2,
          Session-Identification,
          NSLP-Data=DATA2))
```

GIST on Host A delivers the message to NSLP:

```
   Host A: NSLP <-- GIST
   RecvMessage(NSLP-Data=DATA2, MRI=MRI2,
   Transfer-Attributes: Security=integrity+confidentiality,
   Mechanism=tls-certificates, Peer-Certificate=CERT_B, ...)
```

NSLP processes the message.

3. Extensible Authentication Protocol (EAP)

3.1. Introduction

In order to support different deployment scenarios, NSIS signaling should have flexible authentication and authorization capabilities. This can be achieved by interacting with the AAA infrastructure for computing the authorization decision of various NSLP requests. To support existing credentials and the large number of different usage scenarios, the Extensible Authentication Protocol (EAP) might be reused. EAP provides the following capabilities:

- Flexible support for authentication and key exchange protocols.
- Ability to reuse existing long-term credentials and already deployed authentication and key exchange protocols (for example, the UMTS-AKA)
- Integration into the existing AAA infrastructure, namely RADIUS [RFC2869] and Diameter [RFC4072].
- Ability to execute the authorization decision at the user's home network based on the capabilities of protocols like RADIUS and Diameter.

EAP is not the only security framework that could be used. Others, such as the GSS-API (see [RFC2078] and [RFC2743]) or SASL [RFC2222], could theoretically also be used but are from a functional point of view very similar and therefore not explored here. The lack of integration of the GSS-API and SASL into the AAA infrastructure (except for very simple password and challenge-based authentication) makes EAP the preferred choice.

As EAP alone does not provide mechanisms for data origin authentication or encrypting the actual NSIS messages, it has to be used together with another protocol.

Three main choices

- TLS Inner Application [TLS-IA]
- TLS (unauthenticated Diffie-Hellman) below GIST, with channel bindings to EAP run in NSLP.
- Intermediate "TLS/EAP-shim" layer above TLS (unauthenticated Diffie-Hellman) but below GIST. This has the advantage of not requiring modifications to the individual NSLPs but thas the disadvantage of lacking the knowledge about the specific

authorization requirements. Since this approach is very similar to the previous one from an encoding and protocol point of view we will omit the description in this version of the document.

Note that TLS I/A does not preclude the use of certificate based authentication of the server to the client and vice versa. In fact, the usage of certificates for server-based authentication is recommended.

The aspect of enhancing NSIS with an RSVP alike integrity object or the usage of the Cryptographic Message Syntax (CMS) [RFC2630] that use the EAP derived session key for signaling message protection is not considered in this version of the document and is left for further study.

3.2. EAP with TLS Inner Application

3.2.1. Introduction

TLS I/A provides a means to perform EAP-based user authentication between the EAP client and the EAP server integrated within the TLS handshake. TLS I/A can thus be used both for flexible user authentication within a TLS session and as a basis for tunneled authentication within EAP. The TLS I/A approach is to insert an additional message exchange between the TLS handshake and the

subsequent data communications phase. This message exchange is carried in a new record type, which is distinct from the record type that carries upper layer data. Thus, the data portion of the TLS exchange becomes available for NSIS (or another protocol that needs to be secured).

3.2.2. Proposal Discussion

When authentication and key exchange is provided with TLS I/A (and therefore the embedded EAP exchange) then the specific NSLP payloads are not yet processed. Hence, authorizing the specific NSLP operation (such as a request for reserving a certain amount of bandwidth) can only be provided at the QoS NSLP layer. As such, the AAA interaction triggered as part of the TLS I/A (and the EAP method processing) performs only authentication, key establishment and a separate exchange might be required at the NSLP.

The main advantage of this approach is that it does not require modifications for the NSIS protocol suite since the EAP exchange is encapsulated within the TLS Handshake exchange.

3.2.3. GIST API Viewpoint

The GIST API for usage of TLS I/A might need to be make TLS based credentials and the EAP capabilities visible in the abstract GIST API. This allows different NSLPs to use different security mechanisms.

For the first procedure, the previously defined primitive needs to be refined in order to offer an abstract API:

ConfigureSecurity(Mechanism, [EAP-Context], [Local-Certificates, Local-Key], [AAA-Context], [TLS-Options])

Mechanism: 'tls-ia-eap'

EAP-Context: This parameter is present only on the "client" side, and contains information to be used during the EAP authentication (which EAP method, what credentials, etc.).

Local-Certificates: This parameter is present only on the "network" side, and contains certificates that will be presented to the client during the TLS handshake. If authentication is based solely on a mutually authenticating EAP method, certificates can be omitted.

Local-Key: A private key (or a "handle" to a private key) corresponding to Local-Certificates.

AAA-Context: This parameter is present only on the "client" side, and contains information needed to operate as EAP "pass-through" authenticator; i.e., information about the AAA protocol and its details.

TLS-Options: As described in Section 2.3.

For SendMessage, MessageStatus, and RecvMessage, we define a number of security-related Transfer-Attributes. Please note that public key based server authentication inside the TLS handshake might be support and alternatively an unauthenticated TLS handshake is used.

Security: 'integrity' and optionally 'confidentiality'

Mechanism: 'tls-ia-eap'

Peer-Certificates: This parameter is present only on the "client" side. In MessageStatus and RecvMessage primitives, the list of X.509 certificates presented during the TLS handshake (if any). The first certificate contains the public key actually

authenticated by TLS. In SendMessage primitive, this parameter can be omitted; if included, it includes the end entity certificate that must be authenticated during the TLS handshake.

EAP-Information: On the "client" side, information that has been provided by the EAP method (or in SendMessage primitive, has to be provided). The exact details depend on the EAP method in question, but typically includes the EAP method used and identifiers authenticated during the EAP exchange. Since there is no standarized EAP API it is difficult to provide a full set of

details. Proprietary APIs (such as [EAP-API]) might give some ideas about the needed parameters.

AAA-Information: On the "network" side, information that has been provided by the AAA infrastructure (or in SendMessage primitive, has to be provided). The exact details depend on the AAA infrastructure and its configuration, but the AAA server could, for instance, provide information about what identities were authenticated and what types of NSIS requests the client is authorized to make.

TLS-Options: As described in Section 2.3.

3.2.4. GIST Protocol Viewpoint

From a GIST point of view, the TLS Record Layer is used to secure reliable messaging over TCP. Thus, it is used together with the Forwards-TCP protocol defined in [GIST]; the protection is not applied to datagram mode messages.

A new MA-Protocol-ID value, "TLS-I/A", may need to be assigned to allow the negotiation of TLS I/A using the Stack-Proposal object in the GIST-Query/Response phase. Alternatively, a Stack-Proposal for "TLS" is indicated and the support for TLS I/A is indicated as part of the ciphersuite negotiation.

3.2.5. Example

The following example shows the interaction of GIST / QoS NSLP with TLS I/A. In the following example we assume that server-side authentication using certificates is provided within TLS I/A. Client-side authentication is provided using a specific EAP method (such as EAP-AKA).

When a NSLP is started, the server side of the communication must at least configure the key/certificate for accepting messaging associations protected with TLS.

Host A NSLP --> GIST:
ConfigureSecurity(Mechanism=tls-ia-eap,
EAP-Context={EAP-AKA, local USIM card, ...})

```
            Host B: GIST <-- NSLP
            ConfigureSecurity(Mechanism=tls-ia-eap,
            Local-Certificates=CERT_B, Local-Key=KEY_B,
            AAA-Context={Diameter EAP, ...})
```

Later, host A's NSLP sends a message

```
   Host A: NSLP --> GIST:
   SendMessage(NSLP-Data=DATA1, NSLP-Message-Handle=H1,
   MRI=MRI1, Transfer-Attributes: Security=integrity+confidentiality,
   Mechanism=tls-ia-eap, ...)
```

GIST notices that a new messaging association is needed, and initiates Query/Response

```
   UDP(GIST-Query(
   Common-Header,
   Message-Routing-Information=MRI1,
   Session-Identification,
   Network-Layer-Information,
   Query-Cookie,
   Stack-Proposal=Forwards-TCP+TLS-IA-EAP,
   Stack-Configuration-Data=NONE)) -->
```

Eventually this reaches host B and it replies

```
     <-- UDP(GIST-Response(
         Common-Header,
         Message-Routing-Information=MRI2,
         Session-Identification,
         Network-Layer-Information=IP_B,
         Query-Cookie,
         Responder-Cookie,
         Stack-Proposal=Forwards-TCP+TLS-IA-EAP,
         Stack-Configuration-Data=PORT_B))
```

Next, host A establishes a TCP connection to IP_B:PORT_B, and starts the TLS handshake. The InnerApplicationExtension TLS extension indicates that TLS I/A will be used.

```
TLS(ClientHello, InnerApplicationExtension) -->

  <-- TLS(ServerHello, InnerApplicationExtension,
          Certificate=CERT_B, [ServerKeyExchange],
          ServerHelloDone)

TLS(ClientKeyExchange, Finished) -->

  <-- TLS(Finished)
```

Next, the EAP exchange begins. On the "network" side, EAP messages are not processed by the NSIS responder, but rather a separate AAA server.

```
TLS(ApplicationPayload={EAP-Response, Identity,
                        joe@example.com})  -->

  <-- TLS(ApplicationPayload={EAP-Request, AKA-Challenge, ...})

TLS(ApplicationPayload={EAP-Response, AKA-Challenge, ..}) -->

  <-- TLS(FinalPhaseFinished)

TLS(FinalPhaseFinished)  -->
```

On host A, GIST now asks the NSLP to verify the certificates and the result of EAP authentication before ctually sending the NSLP data:

```
Host A: NSLP <-- GIST
MessageStatus(NSLP-Message-Handle=H1, Transfer-Attributes:
Security=integrity+confidentiality, Mechanism=tls-ia-eap,
Peer-Certificates=CERT_B, EAP-Information=..., TLS-Options)

Host A: NSLP --> GIST:
OK
```

Next GIST sends the Confirm message containing the actual NSLP data:

```
TLS(GIST-Confirm(Common-Header,
Message-Routing-Information=MRI1,
Session-Identification,
Network-Layer-Information,
```

Stack-Proposal=Forwards-TCP+TLS-IA-EAP,
NSLP-Data=DATA1)) -->

GIST on host B delivers the message to the NSLP in addition to the information about the client authentication:

Host B: GIST --> NSLP
RecvMessage(NSLP-Data=DATA1, MRI=MRI1,
Transfer-Attributes: Security=integrity+confidentiality,
Mechanism=tls-ia-eap, AAA-Information=..., TLS-Options)

The message is now processed by the NSLP, which may eventually decide to reply...

Host B: GIST <-- NSLP:
SendMessage(NSLP-Data=DATA2, MRI=MRI2,
Transfer-Attributes: Security=integrity+confidentiality,
Mechanism=tls-ia-eap, ...)

<-- TLS(GIST-Data(Common-Header,
Message-Routing-Information=MRI2,
Session-Identification,
NSLP-Data=DATA2))

GIST on Host A delivers the message to NSLP:

Host A: NSLP <-- GIST
RecvMessage(NSLP-Data=DATA2, MRI=MRI2,
Transfer-Attributes: Security=integrity+confidentiality,
Mechanism=tls-ia-eap, Peer-Certificates=CERT_B,
EAP-Information=..., TLS-Options)

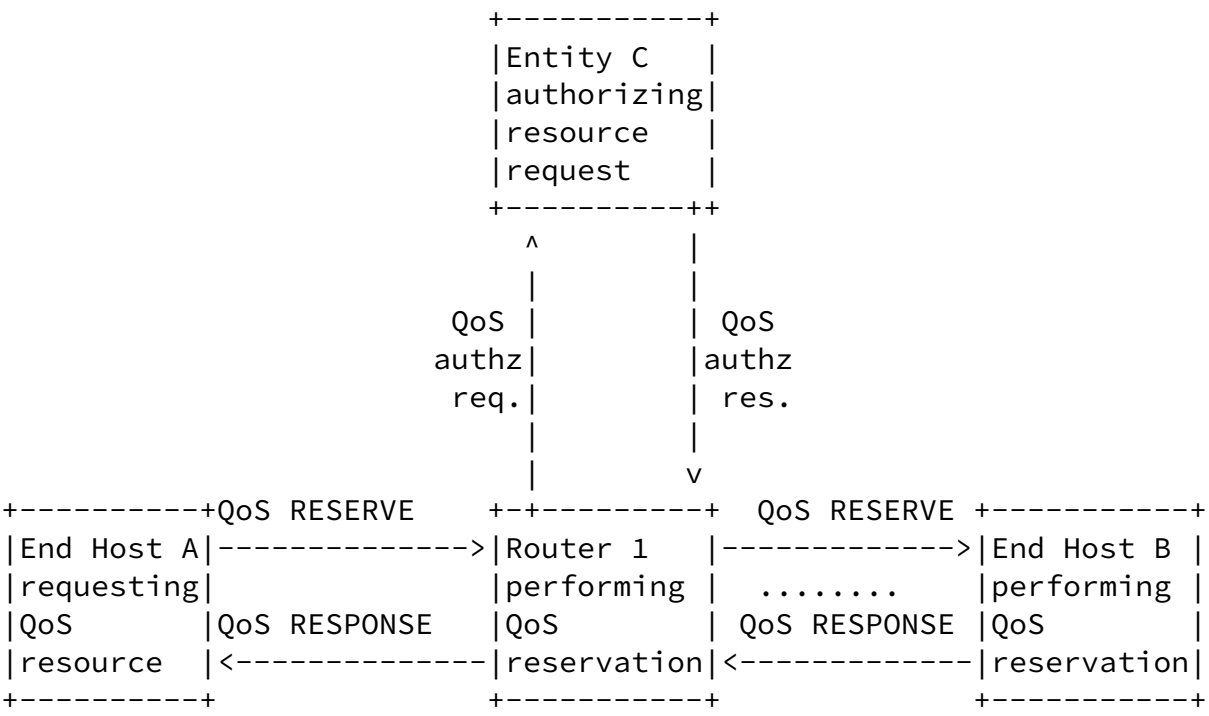
NSLP processes the message.

3.3. EAP in NSLP

3.3.1. Introduction

This section describes the aspects of the EAP integration into the NSLP. It should be noted that integrating EAP encapsulation into QoS

signaling protocol is more than just adding an object into the QoS message exchange. The suggested enhancement of supporting EAP based authentication and authorization for the QoS signaling needs to be investigated for the impact of the integration on the QoS signaling protocol framework. The aspect of authentication and authorization in the QoS environment has raised a number of security concerns in the past.



3.3.2. Proposal Discussion

For integrating EAP into a NSLP application a new NSLP payload object should be defined to carry the EAP packets. The EAP session will be established between a QoS-NSLP initiator (Host A), a QoS-NSLP policy aware node (Router 1) and an Authorizing entity (Entity C). Node A and Router 1 insert the EAP packets into exchanged QoS-NSLP messages between them and are the EAP peer and EAP authenticator in the session. The Router 1 communicates with Entity C (EAP server) via AAA infrastructure and completes the EAP session. Other IETF protocols that decided to use EAP have analyzed some aspects of EAP integration already, such as PANA [PANA]. This experience is reused for identification of the following aspects that necessitate further

investigation:

Transport requirements of EAP and their support by the NSIS framework. EAP itself is a container, and sets requirements on specific transport service features - fragmentation, reliability, in-order delivery and retransmission.

Seamless integration of EAP into the QoS-NSLP message processing.

Number of security considerations related to inter-working with lower layer security mechanism (if present) or trust placed on the QoS NSLP policy aware node or provision of secured transport methods for QoS NSLP messages, utilizing the resulting exchanged EAP method session keys.

In more details, two approaches are investigated in [EAP-QoS] for provision of transport integration between EAP and the NSIS framework. The first one does not introduce any new requirements on the NSIS framework but suggest that the used EAP methods should deal with fragmentation, reliability and re-transmission of the EAP data into the QoS NSLP messages. However, this approach loads QoS NSLP application with functions that are not specific to it (e.g., fragmentation). In the second approach, QoS-NSLP should request reliable transmission for the messages of the signaling session, which involves EAP authentication and authorization. Reliable transport service provided by GIST utilizing TCP and SCTP protocols includes fragmentation and in-order delivery. Considering that a MA establishment (for reliable transport) may be initiated only by the requesting (upstream) peer, additional attention should be paid to the case in which the QoS-NSLP initiator is not initially aware of the EAP usage and does not request reliable transmission from the GIST layer [EAP-QoS].

Each of the two protocols, EAP and QoS-NSLP, has its message processing rules and state machines. Desired integration of both protocols should not require adjustment or even extension of the QoS-NSLP message exchange. [EAP-QoS] shows that a proper encapsulation of the EAP payloads into QoS NSLP Reserve/Query/Response messages allows a seamless inter-working between both protocols.

Until recently, engineers and scientists thought that running one

authentication and exchange protocol on top of another increases the security of the entire exchange. Running these two exchanges completely isolated may allow for Man-In-The-Middle (MITM) attacks, as shown in [EAPBinding]. For example, when running EAP over TLS (as analyzed in [EAPBinding]) the two exchanges need to be cryptographically tied together. This can be accomplished using a number of ways including combining the session keys of both exchanges into additionally generated so called Compound Message Keys according to [EAPBinding]. If the transport layer is secured by TLS and additional authentication and authorization is provided at the QoS signaling layer then the same MITM vulnerabilities need to be addressed. In the NSIS case, these two exchanges are handled at two different NSIS layers. Hence, additional security attributes need to be exchanged through the API between the transport and the QoS signaling layer. Currently, the GIST specification does not provide a strict API specification. Instead, a few clarifying hints to the implementers are given. Specifying additional security attributes, e.g., EAP method derived session keys to be pushed to the GIST layer, are therefore possible. This is, however, not the only obstacle: Exporting keying material from TLS or IKE/IPsec using standard mechanisms in order to combine it with the EAP method derived keying material to compute a Compound Message Key is difficult.

Alternatively, if mutual authentication is provided with TLS to secure GIST signaling then the authenticated identities of both peers can be exchanged via the existing API definition and used to create a binding.

Another common problem for EAP is the validation of the authorization rights of the EAP Authenticator (QoS policy aware NE) that functions in pass-through mode. An adversary might identify itself to the EAP peer and the EAP server with a different identity or a different role. This vulnerability is known as the 'Lying NAS' problem. For example, in the QoS signaling case, an attacker might act as QoS policy aware NE and could misuse an EAP exchange to create the illusion for the EAP server that the context is different (e.g., wireless LAN access instead of wired access). Then a user, who is authenticated and authorized through the usage of EAP, might be charged for services that he has not received.

In the currently discussed framework, EAP should be used to provide authentication and the session key as part of an AAA application,

which is responsible for providing the authorization decision, based on an EAP method that authenticates the user identity. This implies that a QoS policy aware NE should be authenticated and authorized to be one side in an AAA QoS Authorization session. In addition, the authorization decision is based not only on an authenticated identity, but also on the description of requested QoS parameters, which would clearly identify the type of the requested service. It should be noted that the second case is valid only if integrity protection of the exchanged QoS data is available between the End Host and the Authorizing entity. In addition, the problem in general is identified and addressed by some EAP extensions, e.g., [ServiceInfoAuth]. These solutions carry additional authorization related data between the EAP peer and EAP server. However, all above considerations that address the problem do not involve any change to the interworking between EAP and QoS signaling protocol.

3.3.3. GIST API Viewpoint

EAP does not provide a mechanism to secure the NSIS signaling communication. Instead, keying material might be provided by the EAP method that is delivered by the AAA infrastructure to the Authentication and also available to the EAP peer via an API. As such, it is possible to use this keying material for subsequent signaling message protection. Instead of creating a new object that is used to protect the signaling message communication at the NSLP layer the approach described in the subsequent example is based on the combination of TLS. Note that the TLS exchange might not utilize any authentication mechanisms (neither server nor client side authentication). The GIST specific API considerations therefore

appear in the context of configuring lower layer security mechanisms (such as TLS with or without authentication) and the ability to accomplish the channel binding.

As described in [EAPBinding], there are several different ways to bind the channel at GIST layer with the EAP authentication. From layering point of view, it seems that the simplest option would be to pass channel binding information from the GIST layer to the NSLP, and let the NSLP do the binding. In particular, [ChannelBindings] contains a suggestion for channel binding information in the TLS case.

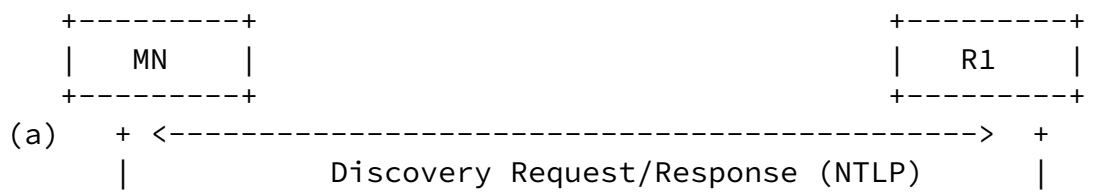
3.3.4. GIST Protocol Viewpoint

From a GIST point of view, the TLS Record Layer might be a good approach to offer data origin authentication, integrity and confidentiality protection of the GIST and the NSLP communication. Thus, it is used together with the Forwards-TCP protocol defined in [GIST]; the protection is not applied to datagram mode messages.

A new MA-Protocol-ID value, "NSLP-EAP", may need to be assigned to allow the negotiation of EAP usage in the NSLP using the Stack-Proposal object in the GIST-Query/Response phase. This would allow an NSIS node to indicate the ability to support this functionality and to thereby avoid relying on a policy for client-side TLS authentication which thereby is provided by EAP at a later stage in the protocol exchange.

3.3.5. Example

An example message flow is shown in Figure 7 which uses the EAP-AKA method [EAP-AKA] for authentication and session key establishment. Please note that the AAA messages triggered by this exchange are not shown for editorial reasons.



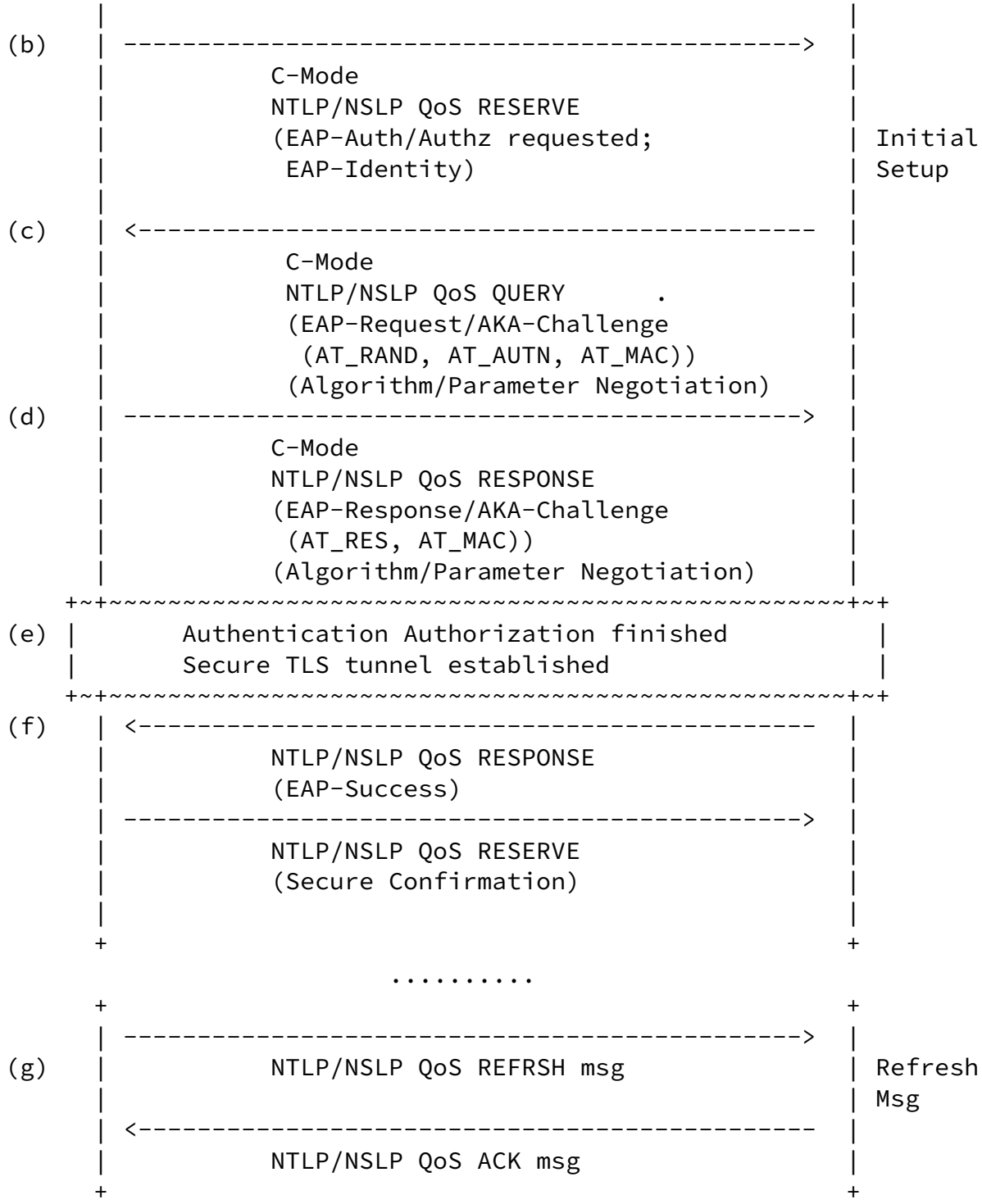


Figure 7: EAP based Auth/Authz exchange using EAP-AKA

The message exchange shown in Figure 7 starts with the optional

discovery of the next QoS NSLP aware node (messages (a)). The first QoS NSLP message with a resource request is sent with the Network Access Identity and a request to perform EAP-based authentication (message (b)). Note that this exchange assumes that the EAP-Request/ Identity and the EAP-Response/Identity exchange is omitted. This exchange is optional in EAP if the identity can be provided by other means. Router 1 contacts the AAA infrastructure, and the EAP server starts the message exchange. The AAA message communication is not shown. Subsequently, two messages (messages (c) and (d)) are exchanged between the EAP peer and the EAP server which contain EAP-AKA specific information. After successful authentication and authorization, session keys are derived and provided to R1 via AAA mechanisms (see [RFC4072] and [RFC3579]). These session keys can then be used to protect subsequent NSLP messages as indicated by (e). The EAP-Success message can already experience such a protection (see message (f)). Furthermore, it is useful to repeat the previously sent objects. Subsequent refresh messages (g) are protected with the previously established session keys and are therefore associated with the previous authentication and authorization protocol execution.

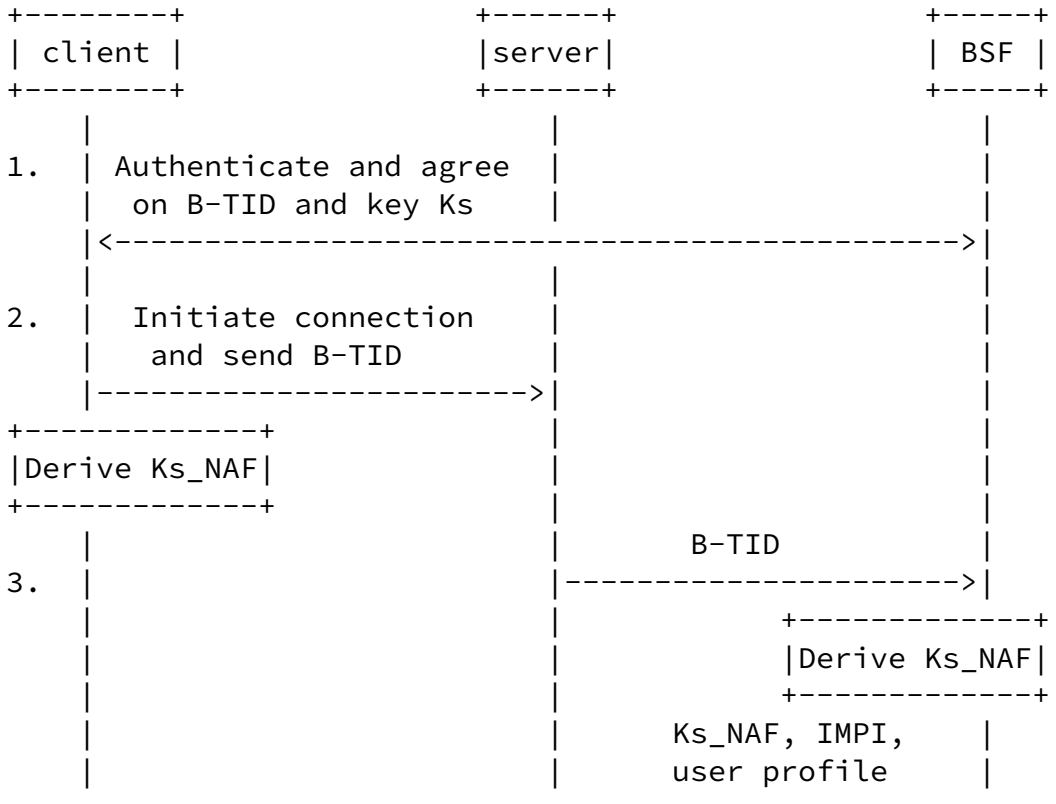
4. 3GPP Generic Bootstrapping Architecture (GBA)

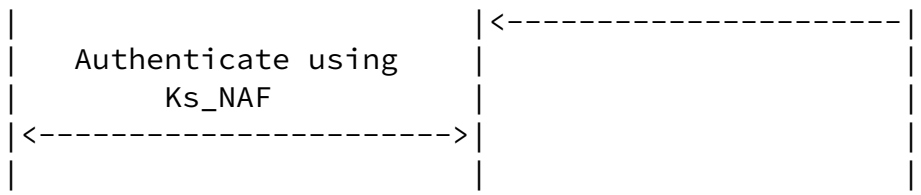
4.1. Introduction

The 3GPP Generic Bootstrapping Architecture (GBA), specified in [TS33.220], is an authentication system with three parties: a trusted third party (called Bootstrapping Server Function or BSF), is involved in authentication and key exchange between two other nodes, a client (called User Equipment or UE) and server (called Network Application Function or NAF).

The goal of the architecture is to isolate knowledge of long-term secrets and credentials to a single trusted node, the BSF. The actual servers (NAFs) do not have access to the clients' long-term credentials, and indeed, do not even have to know exactly what kinds of credentials (such as smart cards) were used between the client and the BSF.

In a simple case, the authentication process is as follows (also depicted in the figure below).





1. First, the client contacts the Bootstrapping Server Function (BSF) and they authenticate each other using long-term credentials. This usually involves contacting back-end authentication databases such as the Home Subscriber System (HSS). As a result of the authentication, the client and BSF share a secret session key (Ks) and a Bootstrapping Transaction Identifier (B-TID) identifying the key and other related state. Currently the authentication between the client and BSF is based UMTS Subscriber Identity Module (USIM) smart cards and HTTP Digest AKA protocol [RFC3310]. However, it is assumed that in the future, other authentication mechanisms may be added between the client and BSF.

2. When the client contacts a server (NAF) it wants to authenticate to, it sends the transaction identifier (B-TID) to the server.

3. The server then sends the B-TID to the BSF, and gets back a server-specific key (Ks_NAF), the user's permanent identity (IP Multimedia Private Identity or IMPI) and application-specific parts of the user profile. This is done using the Diameter-based Zn interface specified in [TS29.109]

4. Finally, the client and server authenticate each other using the server-specific key (Ks_NAF). How exactly this authentication is done depends on the protocol used between the client and the server; this again depends on what exactly the service is. Currently, [TS33.222] specifies that HTTP-based services can use either HTTP Digest authentication [RFC2617] or TLS pre-shared key ciphersuites [TLS-PSK].

4.2. Proposal Discussion

In this section, we consider how 3GPP Generic Bootstrapping Architecture (GBA) can be used to protect GIST messages. Since GBA is by its nature asymmetric, we assume that the querying node is the

client (3G User Equipment) and the responder node is a node in the 3G network (in the role of Network Application Function).

The most promising approach seems to use something like the pre-shared key TLS [TLS-PSK] in a fashion resembling GAA HTTPS services [TS33.222]. However, the HTTPS approach assumes the querying node (3G terminal) knows the FQDN of the responding node beforehand. This would limit the use of GBA to certain message routing methods. Instead, we decided to communicate the FQDN in GIST-Response message, and assume the client can verify whether the FQDN looks correct or not.

4.3. GIST API Viewpoint

The asymmetric nature of GBA is reflected in the GIST API as well: the security-related details contained in the Transfer-Attributes parameter are different on the client and server side.

- When sending a message, the NSLP can verify the server's NAF_ID (or client's IMPI and user profile) before the message is actually sent. This is done using the MessageStatus primitive; the identities and other security details are contained n the Transfer-Attributes parameter.

- When a message is received, the identities and other security details are given to the NSLP for verification, contained in the Transfer-Attributes parameter of the RecvMessage primitive.

- When sending a message, the NSLP can also specify the peer's identity (NAF_ID or IMPI) explicitly; this is useful when the NSLP wants to send a message to the same peer a message received from.

For SendMessage, MessageStatus, and RecvMessage, we define a number of security-related Transfer-Attributes:

Security: 'integrity' and optionally 'confidentiality'

Mechanism: 'tls-gba'

Peer-NAF-ID: This parameter is present only on the "client"

(mobile terminal) side. In MessageStatus and RecvMessage primitives, it contains the NAF identifier (FQDN) that was authenticated using GBA/TLS procedures. In SendMessage primitive, this parameter can be omitted; if included, it specifies the NAF identifier that must be authenticated.

Peer-IMPI: This parameter is present only on the "network" (NAF) side. In MessageStatus and RecvMessage primitives, it contains the user identity (IMPI) that was authenticated using GBA/TLS procedures. In SendMessage primitive, this parameter can be omitted; if included, it specifies the user identity that must be authenticated.

Peer-User-Profile: This parameter is present only on the "network" (NAF) side. In MessageStatus and RecvMessage primitives, it contains the application-specific part of the user profile received from the BSF.

TLS-Options: Information about TLS details, such as exactly which ciphersuite was chosen and what properties (algorithm, strength,

perfect forward secrecy) it has.

4.4. GIST Protocol Viewpoint

From GIST point of view, TLS-GBA looks similar to TLS with certificates. A new MA-Protocol-ID, "TLS-GBA", needs to be assigned.

4.5. Example

Host A's NSLP sends a message

Host A: NSLP --> GIST:
SendMessage(NSLP-Data=DATA1, NSLP-Message-Handle=H1,
MRI=MRI1, Transfer-Attributes: Security=integrity+confidentiality,
Mechanism=tls-gba, ...)

GIST notices that a new messaging association is needed, and initiates Query/Response

UDP(GIST-Query(
Common-Header,

```
   Message-Routing-Information=MRI1,
   Session-Identification,
   Network-Layer-Information,
   Query-Cookie,
   Stack-Proposal=Forwards-TCP+TLS-GBA,
   Stack-Configuration-Data=NONE)) -->
```

Eventually this reaches host B and it replies

```
     <-- UDP(GIST-Response(
         Common-Header,
         Message-Routing-Information=MRI1,
         Session-Identification,
         Network-Layer-Information=IP_B,
         Query-Cookie,
         Responder-Cookie,
         Stack-Proposal=Forwards-TCP+TLS-GBA,
         Stack-Configuration-Data=PORT_B, NAF_ID_B))
```

Next, host A establishes a TCP connection to IP_B:PORT_B, and starts the TLS handshake:

```
   TLS(ClientHello, ServerName(NAF_ID_B)) -->

     <-- TLS(ServerHello, [ServerKeyExchange], ServerHelloDone)

   TLS(ClientKeyExchange(B-TID), Finished) -->
```


Host B fetches the key (Ks_NAF) and the application-specific part of the user profile from the BSF. This is done using the Diameter-based Zn interface specified in [TS29.109].

```
     <-- TLS(Finished)
```

On host A, GIST now asks the NSLP to verify the certificates before actually sending the NSLP data:

```
   Host A: NSLP <-- GIST
   MessageStatus(NSLP-Message-Handle=H1, Transfer-Attributes:
   Security=integrity+confidentiality, Mechanism=tls-gba,
   Peer-NAF-ID=NAF_ID_B, TLS-Options)
```

```
   Host A: NSLP --> GIST:
   OK
```

Next GIST sends the Confirm message containing the actual NSLP data:

```
   TLS(GIST-Confirm(Common-Header,
   Message-Routing-Information=MRI1,
   Session-Identification,
   Network-Layer-Information,
   Responder-Cookie,
   Stack-Proposal=Forwards-TCP+TLS-GBA,
   NSLP-Data=DATA1)) -->
```

GIST on host B delivers this to the NSLP, together with the peer identity (IMPI), application-specific parts of the user profile, and other security-related attributes:

```
         Host B: GIST --> NSLP
         RecvMessage(NSLP-Data=DATA1, MRI=MRI1,
         Transfer-Attributes: Security=integrity+confidentiality,
         Mechanism=tls-gba, Peer-IMPI=IMPI_A, Peer-User-Profile=...,
         TLS-Options)
```

The message is now processed by the NSLP, which may eventually decide to reply...

```
         Host B: GIST <-- NSLP:
         SendMessage(NSLP-Data=DATA2, MRI=MRI2,
         Transfer-Attributes: Security=integrity+confidentiality,
         Mechanism=tls-gba, Peer-IMPI=IMPI_A, ...)

       <-- TLS(GIST-Data(Common-Header,
           Message-Routing-Information=MRI2,
           Session-Identification,
```



```
           NSLP-Data=DATA2))
```

GIST on Host A delivers the message to NSLP:

```
   Host A: NSLP <-- GIST
   RecvMessage(NSLP-Data=DATA2, MRI=MRI2,
   Transfer-Attributes: Security=integrity+confidentiality,
```

```
      Mechanism=tls-gba, Peer-NAF-ID=NAF_ID_B, ...)
```

NSLP processes the message.

5. Discussion and Conclusions

We have examined how three different types of existing security infrastructure, namely X.509 PKI, EAP/AAA and 3GPP GBA, can be used to secure GIST messaging. In all three cases, the TLS record layer provides encryption and integrity protection for NSLP messages; the differences are in how the authentication, key exchange, and authorization work. All three cases assume the existence of an infrastructure beyond what is defined in TLS: either a PKI where the infrastructure elements are not actively involved during the GIST messaging, or on-line AAA/bootstrapping servers.

These three cases are not meant to be exhaustive. Some options that have not yet been analyzed include the following:

- IKE/IPsec with X.509 PKI.
- TLS or IKE/IPsec with manually configured shared secrets.
- TLS with Cryptographically Generated Addresses (CGA) [RFC3972].
- IPsec with Host Identity Protocol (HIP) [HIPArch].
- Integration with SAML/Liberty infrastructure [SAMLOverview].
- Usage of Kerberos within the TLS Handshake [RFC2712]. This approach was utilized for enterprise environments with RSVP (see also [MADS01] for information about the authorization information that may be carried in such a ticket).

Moreover, since integration with an existing infrastructure depends largely on what exactly that infrastructure is, it may be more productive to examine NSIS security in some particular deployment scenario, rather than in abstract or generic fashion.

6. Security Considerations

This document discusses different proposals for securing GIST and hence security is addressed throughout the document. As part of future work additional security considerations might need to be added to GIST [GIST].

7. IANA Considerations

A future version of this draft might require IANA actions. Depending on the outcome of the discussions it might be necessary to register

IANA numbers for stack proposals used in GIST.

8. Acknowledgements

The authors would like to Tseno Tsenov, Henning Peters, Mika Kousa, and Robert Hancock for their input to this work. Tseno Tsenov investigated the integration of EAP into the QoS in the past.

This document has been produced in the context of the Ambient Networks Project. The Ambient Networks Project is part of the European Community's Sixth Framework Program for research and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

9. References

9.1. Normative References

[GIST] Schulzrinne, H. and R. Hancock, "GIST: General Internet Signaling Transport", draft-ietf-nsis-ntlp-08 (work in progress), September 2005.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[TLS-IA] Funk, P., Blake-Wilson, S., Smith, N., Tschofenig, H., and T. Hardjono, "TLS Inner Application Extension (TLS/IA)", draft-funk-tls-inner-application-extension-01 (work in progress), February 2005.

[TLS-PSK] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", draft-ietf-tls-psk-09

(work in progress), June 2005.

9.2. Informative References

[ChannelBindings]
Williams, N., "On the Use of Channel Bindings to Secure Channels", draft-ietf-nfsv4-channel-bindings-03 (work in progress), February 2005.

[DTLS] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", draft-rescorla-dtls-05 (work in progress), June 2005.

[EAP-AKA] Arkko, J. and H. Haverinen, "Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)", draft-arkko-pppext-eap-aka-15 (work in progress), December 2004.

[EAP-API] Microsoft Corporation, "Platform SDK: Extensible Authentication Protocol", MSDN Library, http://msdn.microsoft.com/library/en-us/eap/eap/eap_start_page.asp, 2005.

[EAP-QoS] Tschofenig, H. and J. Kross, "Extended QoS Authorization for the QoS NSLP", draft-tschofenig-nsis-qos-ext-authz-00 (work in progress), July 2004.

[EAPBinding]
Puthenkulam, J., Lortz, V., Palekar, A., and D. Simon, "The Compound Authentication Binding Problem", draft-puthenkulam-eap-binding-04 (work in progress), October 2003.

[HIPArch] Moskowitz, R. and P. Nikander, "Host Identity Protocol Architecture", draft-ietf-hip-arch-03 (work in progress), August 2005.

[MADS01] ""Microsoft Authorization Data Specification v. 1.0 for

Microsoft Windows 2000 Operating Systems", April 2000.

[NATFW] Stiemerling, M., Tschofenig, H., and C. Aoun, "NAT/ Firewall NSIS Signaling Layer Protocol (NSLP)", draft-ietf-nsis-nslp-natfw-07 (work in progress), July 2005.

[PANA] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", draft-ietf-pana-pana-10 (work in progress), July 2005.

[RFC2078] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997.

[RFC2222] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[RFC2617] 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.

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[RFC2712] Medvinsky, A. and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC2869] Rigney, C., Willats, W., and P. Calhoun, "RADIUS Extensions", RFC 2869, June 2000.

[RFC3310] Niemi, A., Arkko, J., and V. Torvinen, "Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA)", RFC 3310, September 2002.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication

Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.

[RFC4072] Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.

[RFC4081] Tschofenig, H. and D. Kroeselberg, "Security Threats for Next Steps in Signaling (NSIS)", RFC 4081, June 2005.

[SAMLOverview] Hughes, J. and E. Maler, "Security Assertion Markup Language (SAML) V2.0 Technical Overview", OASIS document sstc-saml-tech-overview-2.0-draft-08, September 2005.

[ServiceInfoAuth] Arkko, J. and P. Eronen, "Authenticated Service Information for the Extensible Authentication Protocol (EAP)", draft-arkko-eap-service-identity-auth-03 (work in progress), July 2005.

[TLS-ECC] Gupta, V., Blake-Wilson, S., Moeller, B., Hawk, C., and N. Bolyard, "ECC Cipher Suites for TLS", draft-ietf-tls-ecc-11 (work in progress), September 2005.

[TS29.109] 3rd Generation Partnership Project (3GPP), "Generic Authentication Architecture (GAA); Zh and Zn Interfaces based on the Diameter protocol; Stage 3 (Release 6)", 3GPP TS 29.109 V6.3.0, June 2005.

[TS33.220] 3rd Generation Partnership Project (3GPP), "Generic Authentication Architecture (GAA); Generic bootstrapping architecture (Release 6)", 3GPP TS 33.220 V6.5.0, June 2005.

[TS33.222]
3rd Generation Partnership Project (3GPP), "Generic Authentication Architecture (GAA); Access to network application functions using Hypertext Transfer Protocol over Transport Layer Security (HTTPS) (Release 6)", 3GPP TS 33.222 V6.4.0, June 2005.

Authors' Addresses

Hannes Tschofenig
Siemens
Otto-Hahn-Ring 6
Munich, Bayern 81739
Germany

Email: Hannes.Tschofenig@siemens.com

Pasi Eronen
Nokia Research Center
P.O. Box 407
FIN-00045 Nokia Group

Finland

Email: pasi.eronen@nokia.com

Intellectual Property Statement

on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.