Source:	**Nokia**
Title:	**Using shared key TLS with GAA NAFs**
Document for:	**Discussion and Decision**
Agenda Item:	**7.9 GAA and Support for subscriber certificates, 7.18 Presence**

1. INTRODUCTION

In the last SA3 ad hoc meeting in September it was agreed that the authentication proxy (AP) is a special NAF in generic authentication architecture (GAA). In this contribution the "authentication proxy NAF" is referred to as AP-NAF. This contribution describes how *shared key TLS* [SharedKey] can be used in GAA. It further describes a use case where AP-NAF utilizes generic bootstrapping architecture (GBA) and shared key TLS.

2. SHARED KEY TLS IN GAA

2.1 TLS session resumption

TLS has a capability to resume previously established TLS sessions. TLS implementations typically store session information (i.e., Session ID, and master secret) to a local TLS session cache. When the client and server decide to resume a previous session or duplicate an existing session (instead of negotiating new security parameters) the message flow is as follows:

The client sends a ClientHello using the session ID of the session to be resumed. The server then checks its TLS session cache for a match. If a match is found, and the server is willing to re-establish the connection under the specified session state, it will send a ServerHello with the same session ID value. At this point, both client and server must send change cipher spec messages and proceed directly to finished messages. Once the re-establishment is complete, the client and server may begin to exchange application layer data. (See flow chart below.) If a session ID match is not found, the server generates a new session ID and the TLS client and server perform a full handshake. [RFC 2246]

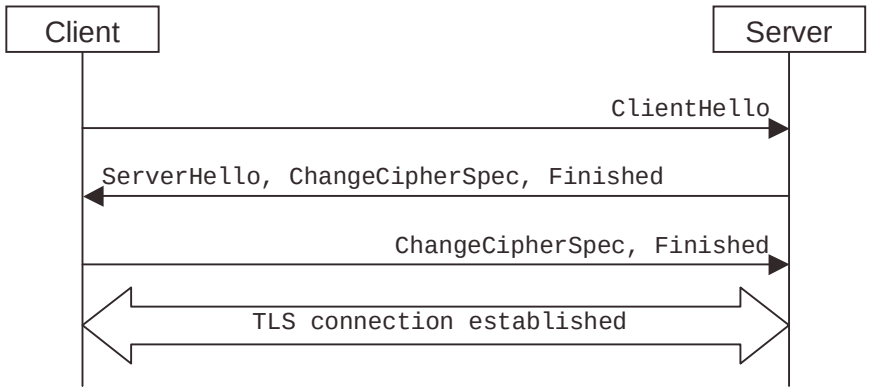

Figure 1. TLS handshake with existing session ID.

2.2 TLS with shared secret

Currently IETF TLS Working Group is defining a specification [SharedKey] to use previously agreed shared secret to establish a TLS connection between a client and a server. The draft uses the session resumption capability TLS (see 2.1).

According to [SharedKey] the functionality is as follows. Before the TLS handshake takes place, the client and server TLS session caches are seeded with a session ID identifying the user/session (variable length value of up to 32 bytes), and a master secret (48-byte value) derived from the shared secret key or password/phrase. Then the client and server resume TLS session with the session ID as described in section 2.1.

2.3 Shared key TLS usage in GAA

Figure 2 depicts the usage of the shared key TLS in GAA.

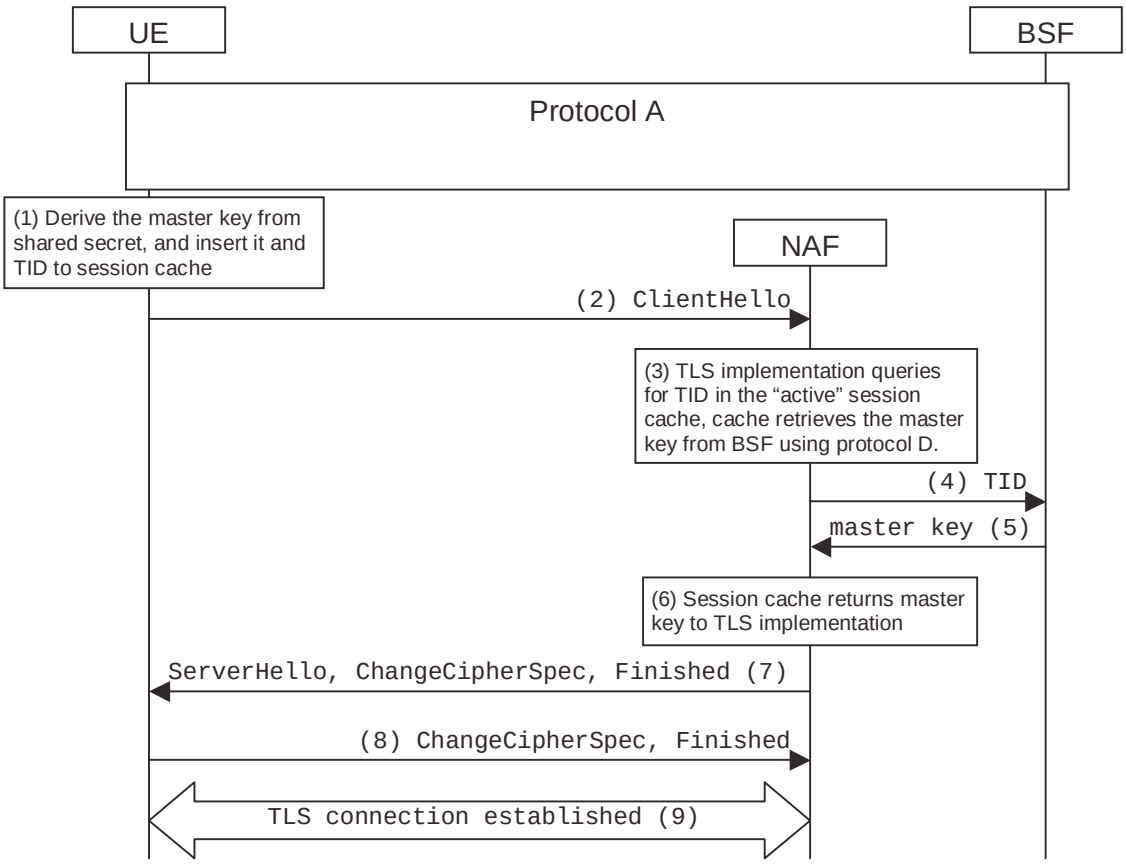

Figure 2. Shared key TLS usage in GAA.

UE wants to mutually authenticate towards an application server (NAF) it first runs the protocol A with BSF resulting to both having established common TID and bootstrapped shared secret keymaterial.

(1) UE derives a master key from the shared bootstrapped keymaterial, and inserts it and TID (as the session ID) to the TLS session cache.

(2) UE sends ClientHello message with TID as the session ID to the NAF.

(3) NAF's TLS implementation queries for the TID from the "active" TLS session cache.

(4) If "active" TLS session cache does not find the TID from the local cache, it retrieves the master key from BSF using protocol D.

(5) Master key is retrieved from BSF to local cache.

(6) "Active" TLS session cache return master key to the NAF's TLS implementation.

(7) ServerHello, ChangeCipherSpec, and Finished messages are sent to UE.

(8) UE sends ChangeCipherSpec and Finished message to NAF to complete the TLS handshake.

(9) Shared key based TLS tunnel is established.

2.4 Analysis

Major advantage provided by the shared key TLS draft [SharedKey] is that it gives wide protection against various MitM attacks (including the Tunneling attack) as the TLS session keys are based on AKA. The authentication is achieved by leveraging the mutual authentication built into AKA, which is done during protocol A. This removes dependency on server certificates in the TLS connection establishment. This also removes a potential external dependency on non-cellular CAs who would issue these server certificates. The needed computation power is also far less required due to the dismissal of the asymmetric key operations of full-fledged TLS (i.e., certificate validation and key agreement functions).

Shared key TLS does not require any changes in the TLS specification itself [RFC 2246]; it uses the existing session resumption functionality. Thus, the TLS protocol itself is not changed. The current shared key TLS draft is just informational explaining how the session ID and master key are derived. Using this approach in GBA may add a new dependency on IETF since the corresponding specification is still in draft stage. However, this specification is likely to proceed fast towards RFC status since it became TLS working group draft (i.e., not a personal draft) during last summer.

Although TLS specification itself is not changed, TLS implementations need to meet the following requirements:

1. Both TLS client and server side need to have a capability to insert a session ID and master secret to the TLS session cache, and

2. TLS server side need to have an IPSec-like “hook” capability to plug in the fetching bootstrapped key material from BSF (hence the name “active” TLS session cache).

If the NAF and BSF are co-located, an optimised BSF implementation can avoid requirement 2 as long as the TLS implementation meets requirement 1.

Protocol A can be based on either version of HTTP Digest AKA, as long as HTTP Digest AKAv2 is not mandated to be run always inside a server-authenticated TLS tunnel. Protection by a TLS tunnel is not mandatory for Protocol A in order to meet its mutual authentication and key agreement requirements.

Lastly, shared key TLS may also be used as “generic protocol B” in the present TS SSC [S3-030488].

3. AUTHENTICATION PROXY

3.1 Existing scenarios

Siemens contribution [S3z030011] to S3#29 ad hoc meeting described three scenarios how Authentication Proxy (AP) could use GBA. These were:

- AP would act as a NAF (see section 6.1 of [S3z030011]) and use draft TS SSC protocols as described in [S3-030488].

- AP and BSF would be co-located (see section 6.2.1 of [S3z030011]) and use draft TS SSC protocols as described in [S3-030488].

- AP and BSF would be co-located (see section 6.2.2 of [S32030011]) and use optimised message flow.

Next chapter describes the fourth option for AP to use GBA using shared key TLS. It is quite close to the first option.

3.2 AP using TLS with shared secrets

This section describes how GAA can use AP using shared key TLS. In this scenario, the BSF and AP-NAF can be either co-located or separate network elements. Figure 3 depicts the latter case.

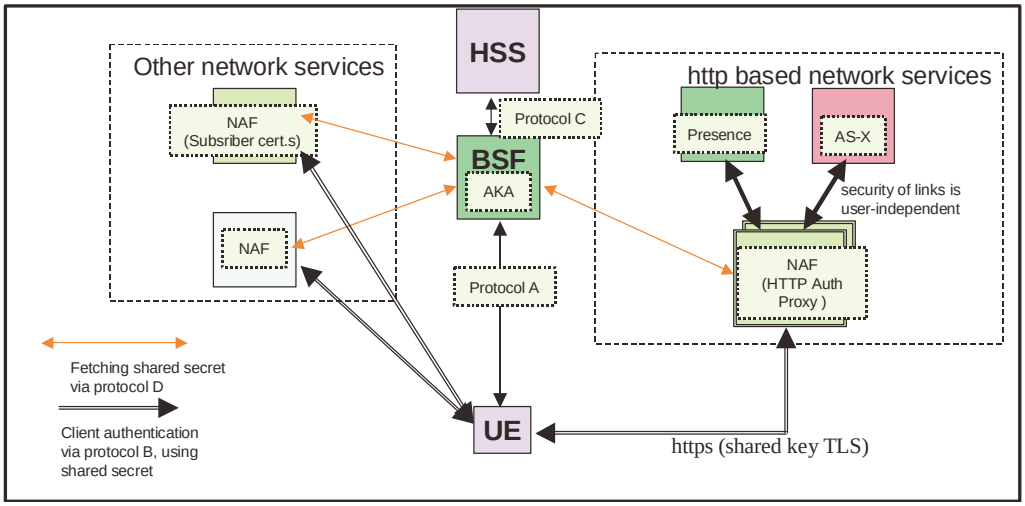

Figure 3. AP-NAF using shared key TLS.

When the UE wants to access one of the application servers, which are attached to the AP-NAF, on the right hand side of the figure, then the sequence of events is as follows:

1) the UE starts HTTP Digest AKA[1] (rfc3310) or HTTP Digest AKAv2[1] (outside a TLS tunnel) with the BSF. The BSF may contact the HSS to fetch authentication vectors (protocol C). After step 1), the UE and the BSF share a secret key and transaction ID (TID), cf. TS SSC, section 4.3.1.

2) The UE sends an http request towards an application server. The http request is intercepted by the AP-NAF. The AP-NAF instructs the UE to upgrade the HTTP connection to TLS/1.0 (see [RFC 2817]).

3) The UE establishes TLS tunnel with the AP-NAF to perform client authentication using the shared key TLS (see section 2.3). In the process, the AP fetches the agreed key from the BSF (protocol D), as described in TS SSC, section 4.3.2.

4) The UE runs the application protocol with the application server through the AP-NAF.

When UE wants to access the application servers on the left hand side of the figure, which are not attached to the AP-NAF, and if shared key TLS is used between the UE and a NAF, the sequence of events is similar to those above:

1) the UE starts HTTP Digest AKA[1] (rfc3310) or HTTP Digest AKAv2[1] (outside a TLS tunnel) with the BSF. The BSF may contact the HSS to fetch authentication vectors (protocol C). After step 1), the UE and the BSF share a secret key and transaction ID (TID), cf. TS SSC, section 4.3.1.

[1] With HTTP Digest AKA or HTTP Digest AKAv2 need to have the added functionality of protocol A: (1) transport the identity of the user in the first HTTP request and (2) transport the TID in the last HTTP response.

2) The UE sends a request (e.g. an http request) towards the application server (NAF). The NAF instructs the UE to upgrade the HTTP connection TLS/1.0 (see [RFC 2817]).

3) The UE establishes TLS tunnel with the NAF to perform client authentication using the shared key TLS (see section 2.3). In the process, the NAF fetches the agreed key from the BSF (protocol D), as described in TS SSC, section 4.3.2.

4) The UE runs the application protocol with the NAF.

Both sequences follow the current TS SSC [S3-030488] except for the shared key TLS part.

3.3 Ut interface for Presence and other coming SIP applications

The Presence server is a good example of application servers in Figure 3. The right side https connection contains the Ut interface in context of the Presence service, for UE data manipulation such as user groups, subscription authorization policy, and presence lists. When the UE is accessing a Presence server, the procedure is similar to the general procedure described in section 3.2. Connections with other SIP applications over TLS tunnel could be established in same way.

The UE will use the bootstrapped secret and TID to initiate TLS connection with the Authentication Proxy that will forward data manipulation request to Presence server. Once the TID is received, the AP fetches the agreed key from the BSF (protocol D), as described in TS SSC, section 4.3.2. Steps 1) to 3) in section 3.2 are specified in various references; the step 4) needs to be in SA3's Presence specification.

Since mutual authentication is guaranteed by shared key TLS connection, and additional AKA-based authentication in application is not needed any more, which simplifies the procedure significantly. Once the TLS connection is established, the UE can right away send user data in HTTP message. Without shared key TLS, either the client certificate is involved together with server certificate (both in TLS layer) or client authentication based on AKA in application layer together with server certificate in TLS layer.

The AP needs still to check that the private ID would point to the public IDs that are populated inside the HTTP message towards Presence server (see bold part in Figure 4). This is easy to achieve, if the protocol D (Zn interface) will carry user service related data to AP (as a NAF) in which all the allowed IMPUs are contained. If it is needed, the AP can remove the Authorization header to keep the UE identity private to external provider's application server.

```
PUT http://Presence.example.com/services/Presence-lists/users/user1_public1/fr.xml HTTP/1.1
Host: Presence.example.com
Authorization: username="user1_private@example.com"
Content-Type:application/Presence-lists+xml

<?xml version="1.0" encoding="UTF-8"?>
   <Presence-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <list name="friends" uri="sip:friends@example.com" subscribable="true">
     </list>
   </Presence-lists>
```

Figure 4. Example of UE data manipulation message over TLS

There is a companion Nokia contribution for Presence agenda item, which contains a pseudo-CR to the Presencespecification.

4. CONCLUSION

This contribution described a way to use shared key TLS [SharedKey] between UE and NAF. It provides an attractive way to use GBA based shared secret within GAA. It also does not require any changes in TLS protocol itself. However, minor modifications in TLS implementations are needed.

Based on the current knowledge using shared key TLS between UE and NAF seems to be the most promising way forward. This approach uses native AKA mutual authentication, and thus server certificates provided by external infrastructures are not needed. Also this gives wide protection against various MitM attacks. For example, it would not be possible for an attacker to masquerade as the server towards an UE, since it would not be able to find out the TLS session key.

5. REFERENCES

[SharedKey] Gutmann, P., "Use of Shared Keys in the TLS Protocol", Internet-Draft, June 2003. URL: http://www.ietf.org/internet-drafts/draft-ietf-tls-sharedkeys-01.txt

[RFC 2817] Khare, R., and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

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

[S3-030488] Draft 3GPP TS 33.109 v0.3.0 "Bootstrapping of application security using AKA and Support for Subscriber Certificates; System Description (Release 6)".

[S3z030011] S3#29b Adhoc: "Generic Authentication Architecture evaluation", Siemens.