3GPP TR 33.823 V2.0.0 (2012-11)

Technical Report

3rd Generation Partnership Project; Technical Specification Group TSG SA Security; Security for Usage of GBA with a UE browser; (Release 12)

MCC selects keywords from stock list.

Keywords
<keyword[, keyword]>

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

Introduction

The most used authentication method in the Internet today is HTML FORM based authentication. It is used with web browsers where a login page is downloaded over HTTPS and which contains an HTML FORM with at least 'username' and 'password' fields. Sometime, this takes place over plain HTTP, which poses a security risk. The current mechanism how GBA could be used from web browser is to use GBA with HTTP Digest as specified in clause 5.3 of 3GPP TS 33.222 [3].

In current implementations, once a web browser has started to use HTTP Digest with a particular web server, it continues to use it until the browser instance is terminated. This is common behavior in web browsers today. This means that there is no way of doing a logout as the browser keeps on sending the HTTP Authorization headers back to the web server.

Another drawback is that using HTTP Digest in parallel to HTML FORM based authentication is not straight forward as the authentication happens in different layers of protocols. Also, the usage of HTTP Digest and HTML FORM based Authentication from the same browser is investigated.

In order to simplify the usage of GBA in web browsers this document describes how to enable access to GBA in HTML layer, namely using Javascript. The usage of Javascript together with GBA raises also some security concerns with regard to protection of GBA credentials, hence the best common practices for this kind of interworking are outlined in this document.

NOTE: Security in a Javascript scenario is implementation-dependent.

1 Scope

This work in this Technical Report has the following scope:

- Study the potential threats for different GBA credentials use scenarios via a web browser. These new use scenarios (e.g. using HTML forms, using Javascript, using widgets) are not covered by current specifications.

The scope of this Technical Report will cover the following:

- Study, identify and specify any protection mechanism that maybe additionally required for the GBA credentials

- Study, identify and potentially specify usage control for GBA credentials

- Study, identify and potentially specify access control mechanism for GBA module

- Study, identify and potentially specify the usage of web based GBA as an extension on the current protocol mechanisms used on Ua reference point (e.g. new Ua protocol identifier)

- Identify and outline how GBA can be used with HTML Forms and Javascript securely (e.g. describing GBA – web specific common practices and examples)

This Technical Report will collect the potential specification improvements, which are then at a later stage of work transferred to the appropriate Technical Specifications. The potential improvements for access control to GBA credentials and potential Ua protocol impacts will then be documented in TS 33.220 [2]. The threat analysis, common security implementation practices and examples may build a new chapter 5 in TS 33.222 [3].

Relation to GBA variants defined in other documents: Web based GBA aims at defining web enhancements for the use of HTML forms with GBA. It is a new variant for the Ua interface and does not affect the Ub interface, as opposed to the GBA variants defined in TS 33.220. Web based GBA is orthogonal to these other GBA variants and can be used with any of them.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 33.220: "Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture".

[3] 3GPP TS 33.222: "Generic Authentication Architecture (GAA); Access to network application functions using Hypertext Transfer Protocol over Transport Layer Security (HTTPS)".

[4] IETF RFC 5705 (2010): "Keying Material Exporters for Transport Layer Security (TLS)".

[5] W3C Candidate Recommendation (Dec 8, 2011): "Web Storage", work in progress, http://www.w3.org/TR/webstorage/

[6] W3C Working Draft (Oct 20, 2011): "File API", work in progress, http://www.w3.org/TR/FileAPI/

[7] IETF RFC 5929 (2010): "Channel Bindings for TLS".

[8] W3C Working Draft (Apr 20, 2012): "HTML5 – A vocabulary and associated APIs for HTML and XHTML", work in progress, http://dev.w3.org/html5/spec/

[9] 3GPP TS 33.203: "3G security; Access security for IP-based services".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in TR 21.905 [1].

HTML FORM: A HTML form is a section of a HTML document containing normal content, markup, special element called controls (checkboxes, radio buttons, text fields, password fields, etc.) and labels on those controls. End users generally "complete" a form on a web page by modifying its controls (entering text, selecting radio buttons, etc.), before submitting the form to an agent for processing (e.g., to a web server).

HTML5: HTML5 is a W3C specification [8] that defines the fifth major revision of the Hypertext Markup Language (HTML), the standard language for describing the contents and appearance of Web pages.

JavaScript: JavaScript is a prototype-based scripting language that was formalized in the ECMAScript language standard. JavaScript is primarily used in the form of client-side JavaScript, implemented as part of a Web browser in order to provide enhanced user interfaces and dynamic websites.

Same origin policy: Some origin policy is a security mechanism in a client browser that permits webpage scripts to access their associated website's data and methods but restricts its access to scripts and data stored by other websites.

GBA web session: A GBA web session is the duration where the NAF can identify that the messages relate to the same individual GBA enabled terminal and a particular browser instance running in that terminal and consist out of a sequence of related HTTP request/response transactions together with some associated server-side state. The lifetime of the session is the lifetime of the Ks_js_NAF which is equal or shorter than the Ks_NAF lifetime and it is also equal or shorter than the lifetime of the TLS session, which was used to derive the Ks_js_NAF.

NOTE: The NAF and the UE may have to recalculate the key, when the TLS session is re-established.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in TR 21.905 [1] and TS 33.220 [2].

API	Application Programming Interface
BSF	Bootstrapping Server Function
B-TID	Bootstrapping Transaction Identifier
CA	Certification Authority
DNS	Domain Name System
FQDN	Fully Qualified Domain Name
GBA	Generic Bootstrapping Architecture
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Over TLS
MBMS	Multimedia Broadcast Multicast Service
ME	Mobile Equipment

NAF	Network Application Function
NAF_ID	NAF identifier
TLS	Transport Layer Security
UE	User Equipment
URL	Uniform Resource Locator

4 Objectives for the Architecture using GBA from a UE web browser

4.1 Introduction

The most used authentication method in the Internet today is HTML FORM based authentication. It is commonly used with web browser where a login page is downloaded over HTTPS and which contains an HTML FORM with at least 'username' and 'password' fields.

The current mechanism how GBA could be used from web browser is to use GBA with HTTP Digest as specified in clause 5.3 of 3GPP TS 33.222 [3]. In this case, the GBA enabled web server can detect whether the web browser is able to perform GBA with HTTP Digest by examining the "User-Agent" header. If "3gpp-gba" product token is present in this header, then the web server (NAF) is able to perform GBA with HTTP Digest with the web browser (UE). However, HTTP Digest has one general drawback. In current implementations, once web browser has started to use HTTP Digest with a particular web server, it continues to use it until the browser instance is terminated. This is common behavior in web browsers today.

This means that there is no way of doing a logout as browser keeps on sending the HTTP Digest headers back to the web server. Another drawback is that using HTTP Digest in parallel to HTML FORM based authentication is not straight forward as the authentication happens in different layers of protocols and with different input windows (as web browsers typically implement a dialog window to handle the query HTTP Digest authentication credentials from the end user compared to HTML FORM having query for the credentials implemented as part of the web page itself).

In order to simplify the usage of GBA in web browser this TR outlines the access to GBA in HTML layer, namely using Javascript.

4.2 Objectives

The document has the following objectives for the usage of GBA in web browsers:

- There will be cryptographic separation between different applications using GBA (e.g. MBMS, Presence, browser banking application, browser e-mail application, etc). For non-browser based applications, this is already in use in generic GBA architecture with the usage of NAF specific keys Ks_(ext/int)_NAF with the usage of NAF_Ids and protocol identifiers.
- The authentication token for the use of GBA in web browsers will be protected from man-in-the-middle attacks.
- The GBA based authentication token will be bound to the existing GBA web session between the browser and the web server in such a way that the authentication tokens cannot be reused in another session or reused by another entity.
- The access to NAF specific keys and authentication tokens by JavaScript will be restricted in such a way that a web page executing a Javascript in a web browser will have access to the NAF specific authentication tokens that it is authorized to have access to. For instance, same origin policy could be used so that a Javascript will have access to only that NAF specific authentication tokens that belongs to same origin (e.g. a web page loaded from http://www.3gpp.org/ will have access to only the NAF specific authentication tokens of www.3gpp.org and not be able to request keys or authentication tokens for another origin).

5 Usage Scenarios and accompanying Threats for using GBA from a UE web browser

5.1 Usage Scenarios

5.1.1 Usage scenario 1

End user wants to use some service provider's services (e.g., an operator), and the service provider wants to use GBA to authenticate the user.

1) End user opens web browser application in the ME, and instructs it to go the service provider's web page. The web page redirects the web browser to a login page if end user has not yet authenticated.

2) Service provider's login page has logic to discover whether Javascript access to GBA is enabled in the browser or not (can be done with Javascript). If GBA is not supported, the web page reverts to other means of authentication, e.g., legacy username/password. If GBA is supported, proceed to step 3.

3) The web page has code implemented in Javascript that obtains a NAF specific token and the B-TID from the GBA function in the UE. In simplest case, the browser uses these variables as username and password in an HTML FORM, and instructs the web browser to send this information back to the web server.

4) The web server extracts the NAF specific key and the B-TID, and uses B-TID to fetch the NAF specific key from the BSF over Zn interface. The NAF generates then the NAF specific authentication token and compares it with the received NAF specific token from the BSF with the one received from the UE. If they are equal, end user is authenticated, and the requested service is provided to the ME and the end user.

NOTE: The direct usage of the NAF specific keys Ks_(ext)_NAF introduces an even greater risk. Therefore the usage of a NAF specific authentication token is considered below in 5.2.

5.2 Threats

The usage scenarios described in clause 5.1 are susceptible to three serious threats:

Threat 1: ME downloads a web page from an attacker that has Javascript which requests all NAF specific keys that is interested in.

Threat 2: ME uses a public access point that is controlled by attacker, i.e., classic man-in-the-middle attack. When the ME requests the login page from the service provider, the attacker sends back a rogue login web page as it controls the DNS. This rogue login page has Javascript that is able to extract any NAF specific authentication token of the service provider, and send it back to the attacker.

Threat 3: It is possible for any third party on the internet connection to eavesdrop on the B-TID and the NAF specific authentication token, and impersonate the user as long as the B-TID has not expired.

Threat 4: If an attacker gets hold of the authentication token Ks_js_NAF, then he can utilize it to attack the communication between web browser and the NAF.

Threat 5: ME downloads a web page from an attacker that has JavaScript which repeatedly triggers GBA re-bootstrapping to be performed. This causes SQN numbers in the USIM to be consumed which shortens the lifetime of the USIM (i.e. a type of denial-of-service attack). A secondary effect is that the malicious web page can coordinate a distributed DoS attack against the BSF/HSS.

6 Control of GBA Credentials and GBA Module in the UE

6.1 General

The threats identified in clause 5 are counterd using a set of control mechanisms as defined in this clasue. Using only a subset of the control mechanism leaves some threats open. Therefore all control mechanisms need to be applied to mitigate the outlined threats.

6.2 Control Mechanism 1– Same Origin Authentication Tokens

To mitigate threat 1 in clause 5.2, the web browser should limit a web page to access only to those NAF specific authentication tokens that belong to origin web server. This way Javascript has access only to one NAF's authentication tokens, which is the NAF identified by the origin of the web page. All web browsers currently implement a single-origin policy where the Javascript is able to send HTTP requests only to the server from where the original web page came from.

6.3 Control Mechanism 2 – Server Authenticated TLS

To mitigate threat 2 and threat 3, HTTPS, i.e., server authenticated TLS, should be used with integrity and confidentiality protection. This way attacking DNS does not help the attacker as the origin of the web page is authenticated using TLS, and the web page content, and B-TID and Ks_js_NAF are confidentially protected against eavesdropping and the Ks_(ext)_NAF are not used directly here.

6.4 Control Mechanism 3 - Channel Binding

The usage of server authenticated TLS as described in clause 6.3 is not sufficient on its own if one were to consider the threat of a compromised TLS server certificate a likely event.. Given that in commonly used browsers there are 100+ root certificates from certification authorities (CAs) who have different levels of security protection when issuing and managing certificates, it can in principle be questioned, how secure TLS with server authentication really is. If one CA is compromised the attacker can use a compromised certificate to lure the user into believing that the attacker's server is the genuine NAF the user wants to communicate with. The attacker can exploit this to realize the following two threats:

- Threat A: If the Javascript would use the Ks_NAF directly and an attacker obtains the Ks_NAF from the user, then it could use this Ks_NAF to impersonate the user towards the genuine NAF, obtain the services and let the user foot the bill.

- Threat B: The attacker makes the user reveal information valuable for the attacker that the user would want to reveal only to the genuine NAF.

Even though TLS with server certificates can generally be trusted, the TLS channel should for the GBA browser case be bound to the authentication token derivation process of GBA. This shall however not be taken as a general clue that TLS with server side certificate authentication is insecure. As the key derivation of Ks_(ext)_NAF is already defined with a fixed set of input parameters, and backward compatibility by not changing this key derivation should be ensured, a new Javascript specific authentication token (Ks_js_NAF) should be derived from Ks_(ext)_NAF using a channel binding mechanism. This channel binding mechanism shall be based on either RFC 5705 (Keying Material Exporters for TLS) [4] or RFC 5929 (Channel Bindings for TLS) [7].

This mechanism does not help against threat B. The mitigation of threat B is further discussed in clause 8.1.4.

6.5 Control Mechanism 4 – Key Usage

In Threat 4 in clause 5.2, the attacker may get hold of the Ks_js_NAF by one of the following means:

- One of the endpoints can be considered as compromised and the Ks_js_NAF is compromised i.e. NAF or web browser are compromised.

- Ks_(ext)_NAF and authentication token derivation parameters are compromised.

The compromise of an endpoint might be made more difficult by usage of additional hardware functionalities, but those would require that all communication for usage of such keys would be routed over the secure hardware. This would still leave the challenge, how to ensure that no fake traffic is routed over the secure hardware. The handle used to authorize the usage of the Ks_js_NAF authentication token inside the secure module need to be secured to avoid unauthorized usage, but that would require a trustworthy browser, which then negates the effect of using a handle for authentication tokens. The usage of the Ks_js_NAF should be done in the TLS tunnel that was used to create the token. This makes usage in another TLS tunnel impossible, as long as the end points check that the TLS tunnel used to receive the information is the same as was used to derive the token.

If the compromised token has been derived, by usage of the compromised Ks_(ext)_NAF key and corresponding parameters, then usage of additional secure hardware would not gain any significant security improvement for the usage from the token of the originating terminal, since the source of the Ks_js_NAF token is compromised.

7 Potential Extension of Protocol Mechanism used on Ua Reference Point

7.1 Key derivation

In order to ensure the key separation in the HTML FORM based authentication in Ua reference point, a Ua security protocol identifier for the NAF_ID needs to be specified.

FQDN

When the web browser in the ME downloads a web page using HTTPS, the web browser verifies that the FQDN in the URL matches the FQDN used in the TLS certificate used by the server (NAF). It is common practice for web browsers to perform this check today. Any web browser that does not perform this check is not secure enough to be used for security sensitive applications with or withouth GBA, and therefore should not be considered for the purpose of this report.

Once the web browser has verified that the FQDN in the URL matches the FQDN in the server (NAF) certificate, the browser makes this verified FQDN available to the GBA API.

The GBA API uses the verified FQDN to derive the authentication token Ks_js_NAF.

NOTE1: Security associated with the use of the FQDN in Javascript in the manner described above is dependent upon the implementation of the web browser, which is out-of-scope for 3GPP.

Ua security protocol identifier

Since HTML FORM is tunneled through TLS, one possibility is to use the Ua security protocol identifier for Ua security protocols that are based on TLS (HTTP Digest with HTTPS and Pre-shared key TLS) that is already specified in Annex H of 3GPP TS 33.220 [2]: (0x01,0x00,0x01,yy,zz), where yy and zz are the protection mechanism CipherSuite as specified in relevant TLS specifications by IETF. However, the HTML FORM based authentication within TLS is significantly different from this Ua security protocol identifier where the NAF specific key is used as a password in the (TLS tunneled) HTTP Digest case compared to HTML FORM case where the NAF specific key is transferred in plain text inside the TLS tunnel. Therefore it is recommended to specify a new Ua security protocol identifier for Ua protocols that transfer the NAF specific key in plain text inside a TLS tunnel, e.g., (0x01,0x00,0x02,yy,zz), where the third octet (0x02) would distinguish this case from other protocols tunneled inside TLS. The last two octets (yy,zz) would specify the TLS ciphersuite used.

NOTE2: Whenever a new Ua protocol is specified where the client authentication is performed inside a server authenticated TLS tunnel, and the client authentication is based on a protocol (inside a TLS tunnel) not covered by the existing Ua security protocol identifiers, then a new identifier should be specified. In general, this kind of Ua security protocol identifier could be in the form where the used TLS ciphersuite is indicated the same way as above (last two octets of the identifier), and the used client authentication protocol by (subset) of the remaining octets (second and/or third octet).

7.2 Channel binding

7.2.1 Background

To mitigate the threat introduced in clause 6.4, a second level of key derivation is introduced. When Javascript code that is downloaded from the web server via the server authenticated TLS tunnel requests for a GBA based key, the request is first handled by the web browser and more specifically the GBA API module in the web browser. The GBA API module will request the Ks_(ext)_NAF key from the GBA Function in the ME using the Javascript specific NAF_ID as specified in clause 7.1. After receiving the Ks_(ext)_NAF key from the GBA Function, the GBA API will derive a Javascript specific authentication token Ks_js_NAF that is bound to the server authenticated TLS tunnel.

The channel binding can be performed using either RFC 5705 or RFC 5929 [7], as is described below. It is possible for the JavaScript code to select which option to use when it requests the Ks_js_NAF token from the GBA API. An example sequence flow is in clause 8.3.1.

NOTE: Both RFC 5705 and RFC 5929 do not utilize TLS nonces, but only refer to RFC 5246 (TLS 1.2).

7.2.2 Option 1: Channel binding using RFC 5705

After receiving the Ks_(ext)_NAF key from the GBA Function the GBA API obtains the TLS_MK_Extr, which is extracted from the TLS master key using the exporter function as specified in RFC 5705 [4]. The label for the exporter function shall be "EXPORTER_3GPP_GBA_WEB". The Ks_js_NAF shall be derived from Ks_(ext)_NAF as follows:

Ks_js_NAF = KDF (Ks_(ext)_NAF, TLS_MK_Extr)

Editor's note: The label "EXPORTER_3GPP_GBA_WEB" for the exporter function needs to be registered with IANA.

An example sequence flow is in clause 8.3.1.

7.2.3 Option 2: Channel binding using RFC 5929

After receiving the Ks_(ext)_NAF key from the GBA Function, the GBA API obtains either the tls-server-endpoint or tls-unique binding type as specified in RFC 5929 [7]. The Ks_js_NAF token shall be derived from Ks_(ext)_NAF as follows:

Ks_js_NAF = KDF (Ks_(ext)_NAF, tls-server-endpoint or tls-unique value)

The tls-server-endpoint binding type (the fingerprint of the server's certificate) has the advantage that it may be used with existing web servers and server-side proxies without modifications to the web servers or proxies. At the same time it also provides protection in the case where one of the browser root CAs gets compromised. However, if the derived key gets stolen through code injection (e.g., cross-site-scripting or inclusion of malicious third-party-code) then the tls-server-endpoint binding type is not sufficient. To prevent reuse of the authentication token even in this scenario, one has to use the tls-unique binding type (the client's Finished message in the TLS handshake) which binds the credential to the particular TLS connection. The downside of this binding type, however, is the lack of support in web servers and server-side proxies.

An example sequence flow is in clause 8.3.1.

8 Common Practices and Examples

8.1 Security Considerations

8.1.1 General Scripting Security Considerations

JavaScript has been designed as an open scripting language, and it has its own security model. This model has not been designed to protect the server administrator or the data that is passed between the browser and the external application

server. The scripting language security model is designed to protect the user from malicious servers, and as result, capabilities of Javascript have been restricted. For example, currently deployed Javascript implementations cannot read or write files on users' computers, or interact between different web pages that are open at the same time in the browser. W3C has been extending Javascript APIs to include new functionality, including File API [6] enabling reading and writing files, and HTML5 Web Messaging enabling communication between the web pages in the browser.

8.1.2 GBA key control

When the Javascript specific authentication token (Ks_js_NAF) is requested by a web page, its creation is controlled by the web browser as specified in clause 7. The Ks_js_NAf is bound to the web server, to the javascript context, and to the type of TLS tunnel used by using NAF_ID as described in clause 7.1. The Ks_js_NAF should not be used outside of the designed web page context.

8.1.3 User grants

When Javascript in a web page is trying to access the Javascript specific authentication token via the Javascript GBA API, the browser executing the Javascript may prompt the end user with a permission dialog asking the end user to grant access to the token. The end user can then decide whether to allow access or deny it, and also additionally have the browser remember the decision. This mimics the functionality of the browsers today that support geolocation Javascript API. There Javascript notifies the end user, that the current page is requesting location information. The end user has then the possibility to either grant access or deny it. Additionally, the end user may have the browser remember that decision, so that the next time the same page is requesting access to the location information, the answer from the previous query from the end users is used without disturbing the end user.

8.1.4 Root CAs in Browser

Clause 6.3 describes the threats related to a compromised CA where either the CA certificate itself or the certificate of some root CA above the compromised CA is present in browsers' root CA list. With the threat B it is possible to issue certificates containing any DNS name, and therefore pretend to be any server. If the attacker can spoof https://www.facebook.com or https://accounts.google.com for instance, he can easily trick users into entering their username and password to attacker's webpages by just mimicking the look-n-feel of the attacked webpages. Additionally, with the introduction of HTML5 there are additional things to consider as HTML5 introduces new features like WebStorage API [5], where a web site can use "localStorage" function to store name-value pairs to the browser, which can be later accessed only by those web pages that have been downloaded from same server identified by protocol/site/port tuple. With this threat, the attacker can fully read from and write to the localStorage of the attacked site.

There is no way to mitigate this threat if a compromised CA is listed in browsers' root CA list except strongly recommend that the browser vendors should carefully consider which CAs they include to their browser offering as trust roots by default, and that the browser implementation should show proper warnings to the end user, if the user (or some service on behalf of the user) tries to add a new CA as trust root. In addition, root CA stores managed online by some external instance, e.g., browser vendors updating root CA stores of their browsers online, should also be kept up-to-date.

8.2 Javascript GBA API description

8.2.1 GBA API Description

Below is an example how Javascript based GBA API could be specified:

```
[NoInterfaceObject]
interface DocumentGBA {
    readonly attribute GBA gba;
};

Document implements DocumentGBA;

[NoInterfaceObject]
interface GBA {
   void getGBAToken(in GBACallback successCallback,
```

```
                  in optional GBAErrorCallback errorCallback,
                  in optional GBAOptions options);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface GBACallback {
   void handleEvent(in GBATokenInfo keyinfo);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface GBAErrorCallback {
   void handleEvent(in GBAError error);
};

[Callback, NoInterfaceObject]
interface GBAOptions {
   attribute boolean forceBootstrap;          // force bootstrapping; default false
   attribute DOMString bindingType;           // TLS channel binding; the options are

                                              // "tls-key-extractor" for option 1, OR
                                              // "tls-server-endpoint" (default), and
                                              // "tls-unique" for option 2

};

// The NAF_ID is determined by the web browser. The FQDN is taken from the origin URL
// of the web page that has the javascript.  The Ua security protocol identifier is
// (0x01,0x00,0x02,yy,zz) where the yy,zz is CipherSuite in the used TLS tunnel (HTTPS).
// If TLS tunnel was not used, (0xFF, 0xFF, 0xFF, 0xFF, 0xFF) is used as Ua security
// protocol identifier. The latter case is not specified in 3GPP and it should only be
// used for testing purposes.

interface GBATokenInfo {
   readonly attribute DOMString key;          // base64 encoded GBA key: Ks_(ext)_NAF
   readonly attribute DOMString btid;         // B-TID
   readonly attribute long bootstrapTime;     // Bootstrap time; millisecs since 1.1.1970
   readonly attribute long expiryTime;        // Token expiry: millisecs since 1.1.1970
   readonly attribute DOMString fqdn;         // used FQDN
   readonly attribute DOMString uaSecProtId;  // base64 encoded Ua security prot. id;
};

interface GBAError {
   readonly attribute unsigned short code;    // error code (to be specified)
   readonly attribute DOMString message;      // textual description of the error
};
```

8.2.2 Example API usage

Below is an example how to use javascript based GBA API:

```
// Basic example of requesting GBA token
document.gba.getGBAToken(gbaSuccess,gbaError);

function gbaSuccess(tokeninfo) {
   // gba token was successfully created, and for example use
   // tokeninfo.btid as username and tokeninfo.token as password
}

function gbaError(error) {
   // an error occured during gba token creation
}
```

8.3 Example sequence flows

8.3.1 Example sequence flow with channel binding

In this example message flow with channel binding the following architecture is assumed:

- **GBA Function:** The GBA Function handles establishment of GBA-specific keys. In particular, the establishment of the key Ks can use any of the methods defined by TS 33.220 [2] (e.g., based on AKA or GBA_Digest). The GBA Function is not part of the web browser.

  NOTE: In the case of GBA_Digest, the GBA Function treats SIP Digest credentials as specified in Annex N of TS 33.203 [9].

- **Web Browser:** The web browser is either native or downloaded and contains some functions which support usage of GBA. In particular we have in the architecture:

  - **GBA_API:** Part of the browser that communicates with the GBA Function and receives GBA authentication token material requests from the Javascript code.
  - **Javascript:** Downloaded Javascript code.
  - **Engine:** Sets up communication with the NAF.

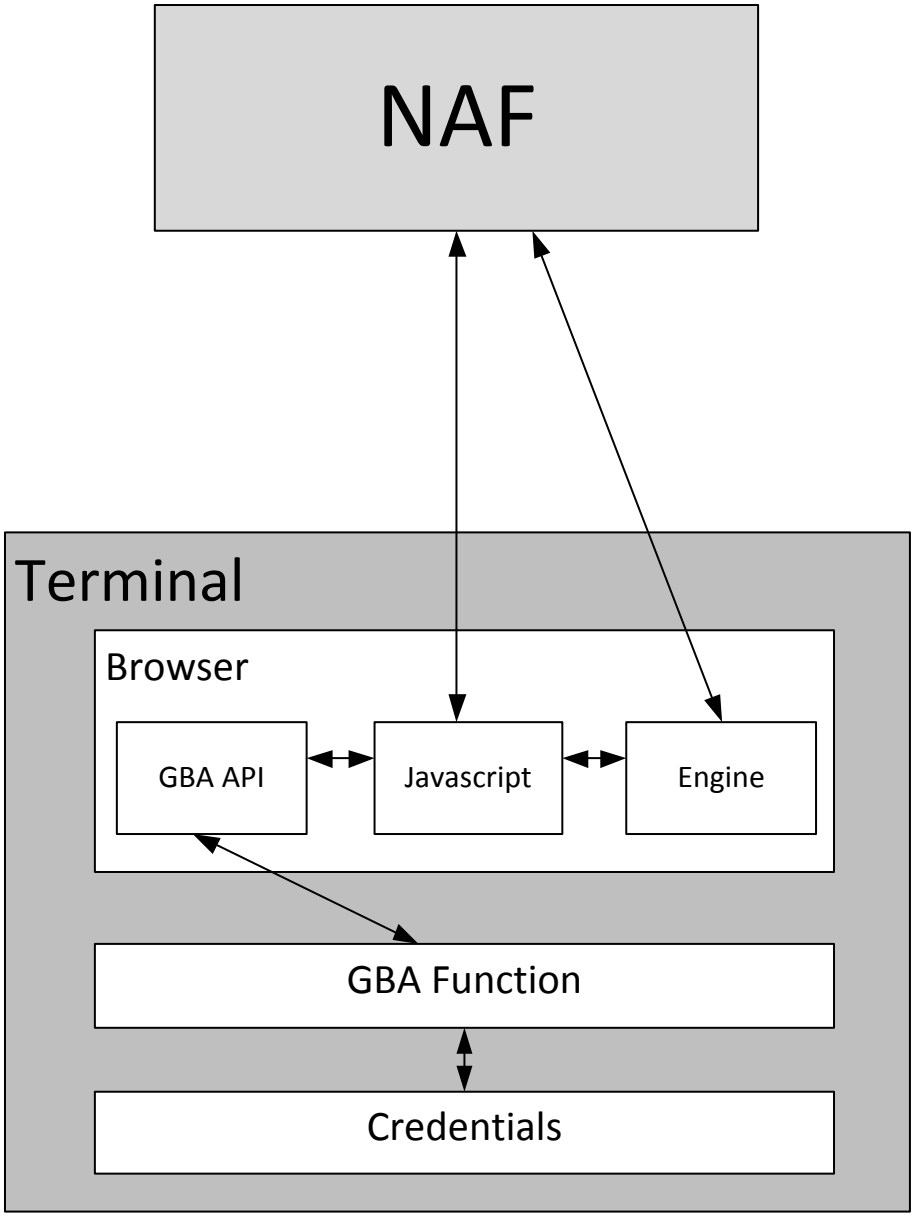

Figure 8.3-1. Example Architecture

Below is an example sequence flow diagram of GBA usage in Web context, i.e., within Javascript.

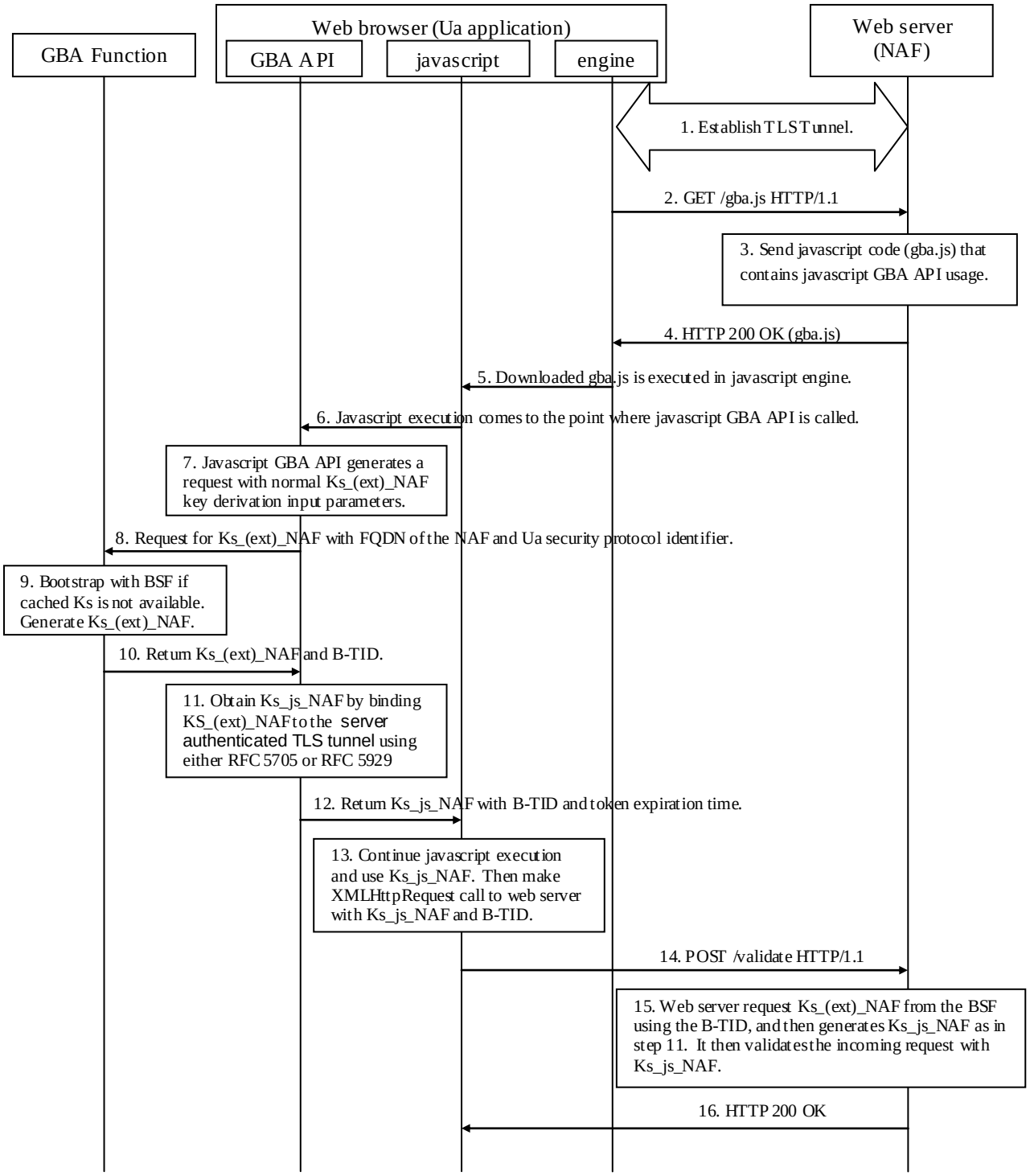

Figure 8.3-2. Example sequence flow.

The web browser is considered to be a trusted application in the sense that the user trusts it to handle security related functions properly, i.e., setting TLS sessions with servers, sandboxing the Javascript code that is downloaded from the web servers, and not leaking sensitive information like a password to third parties. In the sequence flow diagram, the web browser is divided into three functional blocks:

- engine module handles the basic functionalities for the web browser like setting up TLS with web servers, downloading web resources from network, and providing the user interface with the end user.

- GBA API module offers the API towards any Javascript executing in the web browser. As Javascript should not be explicitly trusted, the web browser and the GBA API should not reveal any sensitive information to the Javascript, nor should they accept any sensitive information from the Javascript more than necessary.

- Javascript module executes the downloaded Javascript. Any Javascript executing in web browser should be considered not trusted and should not be granted access to sensitive resources or the access to those resources should be controlled.

The communication between web browser and web server in the depicted sequence flow diagram is executed inside a server authenticated TLS tunnel. Also, the web browser is in the process of downloading a html page where one of the linked Javascript resources is "gba.js".

1. The web browser and the web server establish a server authenticated TLS session.
2. The web browser engine makes a HTTP GET request to the server to download gba.js resource from the server.
3. The web server sends the gba.js file that contains the Javascript GBA API call on the browser. The gba.js can also contain additional logical elements that make use of the Javascript specific authentication token Ks_js_NAF.

   Example on how a GBA API call could look like:

```
document.gba.getGBAToken(successCallback,
                         errorCallback);
```

4. As a HTTP response to the HTTP request made in step 2, the web server returns the gba.js to the web browser.
5. The engine in the web browser starts to execute the Javascript in gba.js in Javascript sandbox.
6. The Javascript comes to a point where a call to GBA API is made.
7. Browser's Javascript GBA API locates the relevant information about the Javascript, i.e., in what html page it is executing, from what url was the html page downloaded from, and which TLS ciphersuite is used in the TLS tunnel. The FQDN of the NAF can be extracted from the url of the web page, and the Ua security protocol identifier can be derived from the used TLS ciphersuite. FQDN of the NAF and the Ua security protocol identifier form the NAF_ID.
8. Browser's Javascript GBA API makes a call to ME's GBA Function with the NAF_ID derived in step 7.
9. The GBA Function bootstraps with the BSF if there is no valid GBA master key Ks. From the Ks, Ks_(ext)_NAF NAF specific key is derived using the NAF_ID.
10. The GBA Function returns the Ks_(ext)_NAF key to browser's Javascript GBA API with the bootstrapping transaction identifier (B-TID).
11. Upon receiving the Ks_(ext)_NAF key, browser's javascript GBA API will derive the Javascript specific authentication token Ks_js_NAF that is bound to the server authenticated TLS session. The two options are as follows:

    If the value of the bindingType in GBAOptions is "tls-key-extractor " (i.e. RFC 5705 is used with the label "EXPORTER_3GPP_GBA_WEB") then Ks_js_NAF is derived as:

    Ks_js_NAF = KDF (Ks_(ext)_NAF, TLS_MK_Extr)

    If instead the value of bindingType is "tls-server-endpoint" or "tls-unique" (i.e. RFC 5929 [7] is used), then Ks_js_NAF is derived as:

    Ks_js_NAF = KDF (Ks_(ext)_NAF, tls-server-endpoint or tls-unique value)

    The tls-server-endpoint, tls-unique value and TLS_MK_Extr are all related to the TLS connection that established the TLS session in step 1.

    Editor's note: If there are several key-derivation variants then indication of the variant is ffs.

12. Browser's Javascript GBA API returns Javascript specific Ks_js_NAF authentication token, B-TID and authentication token lifetime to the executing javascript.
13. The Javascript continues to execute and it uses the Ks_js_NAF authentication token the way the web server has instructed (via Javascript).

Example on how Javascript can extract parameters from result object in Javascript (continued from step 2).

```
function successCallback(result) {
   var token = result.token;
   var btid = result.btid;
   var lifetime = result.expiryTime;
}
```

14. After executing the client side logic, the Javascript makes a XMLHttpRequest (ajax call, HTTP request) to the web server. This request contains at least Ks_js_NAF or hash of it, and B-TID.

15. The web server fetches the Ks_(ext)_NAF key from the BSF, and it then derives the Ks_js_NAF the same way it was done in step 11. The web server will then compare the received Ks_js_NAF with the locally derived one and validate that the TLS session is the same as was used for the request that established the TLS session in step 1.

16. If the received Ks_js_NAF is valid, the web server will continue to process the request made in step 14 and return the result to the web browser (to the Javascript).

Annex <X>: Change history

It is usual to include an annex (usually the final annex of the document) for reports under TSG change control which details the change history of the report using a table as follows:

Change history							
Date	TSG #	TSG Doc.	CR	Rev	Subject/Comment	Old	New
2011-08-12		S3-110831			Integration of S3-110644, S3-110645, S3-110646		0.0.1
2011-11-08		S3-111214			Integration of S3-111128, S3-111129, S3-111131 and S3-111166 according to the decisions during the meeting	0.0.1	0.0.2
2012-02-10		S3-120227			Integration of S3-120057, S3-120061, S3-120217, S3-120054 and S3-120056 according to the decisions during the meeting	0.0.2	0.1.0
2012-05-25		S3-120530			Integration of S3-120432 and S3-120532 according to the discussion during the meeting	0.1.0	0.2.0
2012-06		SP-120335			SP-120335 presented to SA plenary for information	0.2.0	1.0.0
2012-07-13		S3-120788			Integration of S3-120821, S3-120706, S3-120820, S3-120819, S3-120703, S3-120614, S3-120789, S3-120648, and S3-120790 according to the discussion during the meeting	1.0.0	1.1.0
2012-11-09		S3-121202			Integration of S3-121059 and S3-121061	1.1.0	2.0.0