Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 10 (2012) 696 – 704

Procedia Computer Science

The 9th International Conference on Mobile Web Information Systems (MobiWIS)

Secure Web Service Clients on Mobile Devices

Jens Bertram, Carsten Kleiner

University of Applied Sciences and Arts Hannover
Ricklinger Stadtweg 120
30459 Hannover
Germany

E-Mail: ckleiner@acm.org

Abstract

Enterprise apps on mobile devices typically need to communicate with other system components by consuming web services. Since most of the current mobile device platforms (such as Android) do not provide built-in features for consuming SOAP services, extensions have to be designed. Additionally in order to accommodate the typical enhanced security requirements of enterprise apps, it is important to be able to deal with SOAP web service security extensions on client side. In this article we show that neither the built-in SOAP capabilities for Android web service clients are sufficient for enterprise apps nor are the necessary security features supported by the platform as is. After discussing different existing extensions making Android devices SOAP capable we explain why none of them is really satisfactory in an enterprise context. Then we present our own solution which accommodates not only SOAP but also the WS-Security features on top of SOAP. Our solution heavily relies on code generation in order to keep the flexibility benefits of SOAP on one hand while still keeping the development effort manageable for software development. Our approach provides a good foundation for the implementation of other SOAP extensions apart from security on the Android platform as well. In addition our solution based on the gSOAP framework may be used for other mobile platforms in a similar manner.

Keywords: Web service; SOAP; WS-Security; Android; gSOAP; kSOAP; enterprise apps

1. Introduction

Most apps currently used on mobile devices are primarily for consumer purposes. In the near future an increasing number of enterprise apps can be expected. With this new domain for mobile applications new requirements regarding the communication with other software components emerge, particularly concerning security features. Enterprise environments are usually organized based on business processes which themselves integrate many different system components. Such loosely coupled software systems implement parts of the processes and offer them to other components as services. Generally these are SOAP based web services as SOAP is a standardized and platform independent protocol enabling flexible and feature-rich service interfaces to be offered and consumed in processes. Furthermore the clients consuming the services can (semi-)automatically be generated. This significantly reduces the implementation effort required for clients during initial implementation and maintenance. Also code generation is required for implementing the vision of autonomous orchestration of services by software agents.

1877-0509 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.procs.2012.06.089

The above discussion already shows the need for SOAP web services as opposed to REST based services in an enterprise context. Adding security aspects the transport-layer security provided by HTTPS (and essentially the only one available for REST services) might be sufficient for consumer apps but does not fulfill the more advanced security requirements for enterprise applications. These requirements are achieved with additional well-known web service securitsy specifications built on SOAP (e.g. [1, 2]).

In summary mobile enterprise apps should be SOAP capable to enable their integration in existing business processes appropriately. In addition (parts of) these apps should be automatically generated to be more flexible on changes of the complex service interfaces and to speed up the applications development process in general. For security aspects the respective specifications have to be supported. This is also true for other well-known web service specifications for similar non-functional requirements (e.g. quality of service).

Unfortunately none of these issues is addressed by the currently existing mobile platforms like iOS and Android. Most of them do not support SOAP services natively at all. In addition advanced specifications like WS-Security are currently also not supported ([3]). A potential reason for this fact could be that the recent development of mobile devices has been focusing primarily on consumer needs. Nevertheless it is a very important question for many companies on how to add these SOAP related features to the mobile platforms to utilize mobile devices in their existing business processes.

In this article we will show how current mobile device platforms can be extended to address these shortcomings. We focus on the Android platform as it is widely used and has the highest growth ratios in the market currently. We will compare the native capabilities of Android and external libraries and tools on how to add support for SOAP and extending specifications. After that we will present our solution that provides the automated generation of SOAP clients for Android with support of the WS-Security specifications in detail. This solution is based on the gSOAP toolkit, which is available for many platforms. Thus we expect that our results may be transferred easily to other mobile device platforms as well.

2. Related Work

Important references that our work builds on are several well-known web service specifications. These are referenced directly in the following sections wherever they are introduced. For most desktop platforms several web service frameworks do exist that support all of the required features to implement secure SOAP clients. To the best of our knowledge none of these frameworks has been extended to mobile device platforms with the exception of the .NET framework on Windows Mobile 6/Phone 7. The support on that platform is restricted to a specific security configuration though and the current market share is rather limited. An overview of the options on different platforms has been presented in [3].

Not many work has been published on using SOAP services on mobile devices at all. The papers and online articles that have been, typically describe the implementation of a SOAP client for a specific service. These applications typically do not use client code generation but rather implement the client manually. In our opinion this is of limited use as the main reason for using SOAP is its flexibility. Thus changes in services should not require a complete reimplementation of the clients. For this reason our work focuses on providing solutions that (to the largest degree possible) make use of automated code generation for the client stubs. Consequently we do not cite references that do not make use of code generation here.

Some related work has focused on assessing and improving the performance of SOAP web services to cater for the small bandwidth that has been claimed for mobile networks. In [4] an alternative to WS-Security with reduced overhead is proposed. A detailed performance comparison between REST and SOAP on an iPhone is reported in [5]. We do not consider this to be of great significance nowadays and in the future as XML messages are small when compared to video streaming which is already available on smartphones.

Our work has originally been motivated by a research project where we faced the challenge to implement SOAP clients on mobile devices with specific security features. The goal of the project has been to implement a privacy preserving software system for pay-as-you-drive car insurance where the insurance premium is calculated based on the driving behavior of the policy holder. The driving behavior is assessed by means of GPS positioning on the mobile device. The device is integrated into the insurance software by means of SOAP services, the security features in these services are part of the privacy preservation model for the policy holder. Details on this system have been described in [6, 7].

3. Implementation of SOAP Clients on Android

3.1. Androids' onboard capabilities

The Android OS is based on a Linux-Kernel, which encapsulates hardware specific issues and provides basic concepts like processes, user and rights management, I/O, etc. Besides these core functions the *Dalvik VM (DVM)* – a Java VM especially customized to the hardware properties of mobile devices – is responsible for the application execution. Thus, Android applications are written and implemented in Java, compiled to Java Bytecode and then compiled to Bytecode for the DVM resulting in smaller and less memory consuming code. Due to this fact the Android API consists of some platform specific APIs and a subset of the JavaSE API. Regarding web services the latter includes XML parsers (DOM, SAX, XMLPull) and typical network connections and protocols like TCP/IP and HTTP. Furthermore a JSON [8] API enables the use of a more lightweight alternative format to XML. Regarding security, RESTful services are commonly secured with TLS/SSL [9] and HTTPS [10]. This approach for a point-to-point security context is also available on the Android platform. Stronger mechanisms providing end-to-end security are not standardized for RESTful services and have to be implemented individually, e.g. by utilizing the cryptography API BouncyCastle [11] which is also included in the platform. In summary the Android platform provides a good support for RESTful Web Services which are commonly based on HTTP connections and XML or JSON as message format.

For SOAP based web service support is far worse. Disregarding the network connection and XML APIs, there are no options provided by the platform to consume SOAP based web services. From our point of view, manually implementing the parser code for XML based SOAP messages is not a suitable solution and therefore external libraries and tools must be considered to simplify the consumption of SOAP web services, e.g. by generating a client stub. Unfortunately the well-established SOAP engines like Axis2 and the output of tools like *wsdl2java* are not compatible with Android.

3.2. External SOAP libraries

In fact, with kSOAP2 [12] there is only one project simplifying the use of SOAP on Android. This library encapsulates the details of the underlying transport layer, provides different message (de-)serialization mechanisms and eases the SOAP fault handling. Nevertheless the datatypes defined in the service interface description (WSDL [13]) must be implemented manually with some help from the serialization mechanisms offered. These mechanisms vary in their complexity, some are more suited for simple types, others are a better choice for complex types and require a higher implementation effort. In summary kSOAP2 is a sufficient solution for simple services with limited complexity in their interface. For more complex services with semantically rich and huge messages this library is not an adequate approach. Regarding security features the library also supports point-to-point security with HTTPS. Additional security mechanisms on the application layer as defined in the WS-Security specification [1] are not supported.

In our work we examined an alternative solution to consume SOAP web services on Android devices accompanied by the convenience of a stub generation process. This approach relies substantially on the C/C++ web service toolkit gSOAP [14] and the Android NDK [15], which enables the reuse of existing C/C++ code and libraries or the implementation of performance critical components like extensive computations. Based on this ability to utilize C/C++ code we implemented and extended the generation process by generating a C based gSOAP stub and encapsulating this stub in Java classes. This process will be described in more detail in the following sections. Concerning security aspects, a gSOAP security plugin provides both, point-to-point (HTTPS) and end-to-end security (WS-Security). These issues will be described in section 4 considering the characteristics of the Android platform.

3.3. The Web Service Toolkit gSOAP

At first we will give a short introduction to the C/C++ Web Service Toolkit gSOAP [14], which is licensed under an open source license and available at [16]. For commercial purposes there is also a proprietary licensed version with varying support levels available. This provides a good base for the approach presented in this paper to be used in business applications and commercial products as well.

The toolkit is especially designed for embedded systems with limited hardware resources like mobile devices. As gSOAP is entirely written in C/C++ and only has a few dependencies (mainly libc) the toolkit is portable across many platforms including traditional desktop and server systems like Windows, Linux, Unix, Mac OS and some more or less obsolete mobile platforms like Symbian and Windows Mobile. The two most important tools are the *WSDL importer* (wsdl2h) and the *gSOAP compiler* (soapcpp2), which provide a top-down approach for the generation of a client stub or a server skeleton. As we concentrate on the client side implementation we will not cover the server side related issues in this paper.

Figure 1 shows an overview of the main steps to generate a C/C++ based web service stub with the gSOAP tools. At first the WSDL importer wsdl2h translates a given WSDL into a header file, which maps the datatypes and operations defined in the service description to C/C++ structs/classes and functions. Based on the header file the gSOAP stub compiler generates the specific stub code with the XML/SOAP (de-)serializers required. In our case we rely on a plain C based stub.

Fig. 1. Stub generation with gSOAP tools

3.4. Integration of a native gSOAP Stub Library

As Android does not support C code on application level we have to look at the Android NDK. As previously described gSOAP has few dependencies, basically this is the standard C library *libc*. Fortunately this library is available on the Android platform and within the NDK, so we can proceed and compile the previously generated stub code into a native library. This library can then be loaded within a Java application and called through JNI. Listing 1 shows a simple and reduced HelloWorld example of a service call through a generated gSOAP stub encapsulated in a JNI method. When the method is called, the request parameter is initialized and the name property is set. In case of a successful service call the response message is printed on the log output.

Listing 1. Web Service call through JNI

```
JNIEXPORT jint JNICALL Java_mypkg_ServiceClass_soap_call_hello(JNIEnv *env, jclass clazz) {
  struct soap *soap = soap_new();

  struct _secclient1__hello hello;
  hello.name = "Homer Simpson";
  struct _secclient1__helloResponse helloResponse;

  if(soap_call___secclient1__hello(soap, soap_endpoint, soap_action, &hello, &helloResponse) ==
      SOAP_OK)
  {
    struct secclient2__HelloWorld * return_ = helloResponse.return_;
    LOGD("Hello Service Response: %s", return_->hello);
  }

  return 1;
}
```

If the compilation is successful, we get an additional library file, for example named *libgsoapJniHelloWorld.so*. This library file must then be loaded on application startup. Thus in case of an Android application the following code (listing 2) should be placed in the main activity file as explained in the NDK documentation.

Listing 2. Loading the native library in the Android main activity

```
static {
  System.loadLibrary("gsoapJniHelloWorld");
}
```

Access to the native library and its methods is encapsulated in a separate class file as shown in listing 3.

Listing 3. Encapsulation of the JNI method

```
public class ServiceClass {
  public static native int soap_call_hello();
}
```

With this example we have a first solution to call a SOAP web service inside an Android application through JNI and a generated gSOAP stub. Unfortunately with this solution it is impossible to change any parameters and compose a request at runtime. This is an essential feature of web services and any kind of communication in distributed systems. In case of the presented HelloWorld example one could extend the JNI method with an additional String parameter to set the requests name property. But this approach is only suitable for this specific service and not adjustable to more complex services and therefore not a general solution. We have to spend more effort on the encapsulation of the C based stub in Java classes with JNI.

3.5. Extended Stub Generation Process

Based on the stub generation with the gSOAP tools shown in figure 1 we extended the process as shown in figure 2. As depicted we developed a Python script that also generates Java classes and an intermediary layer for the data transformation between Java and C, both on the basis of the previously generated gSOAP stub. The Java classes map the C structs and methods, which itself map the services datatypes and operations. The intermediary layer (*JNIAdapter*) relies heavily on JNI and is responsible for the transformation of Java objects into C struct instances and vice versa. The reason for using Python is that for the extended generation process it is necessary to parse the stub header file (*soapStub.h*). As the parsing of C code is an expensive task it is not advised to implement that manually. With *gccxml*[1] there exists a tool that produces an XML based representation of C/C++ source code. This is a big step forward but nevertheless we now have to parse the XML output and process it to generate the classes and JNI code. With *pygccxml*[17] there exists an extension to gccxml that provides a Python API to easily handle the XML output. The API provides direct access to structs/classes, class attributes, methods, method parameters, method return types and so on. Thus we can easily access the information required for the Java code generation. Furthermore this solution is decoupled from the gSOAP toolkit to a maximum degree and therefore independent of its ongoing development. gSOAP is very actively developed and thus many new versions are released within a short time frame .In the future we expect the main principle of our solution to be implemented and integrated into the gSOAP stub compiler.

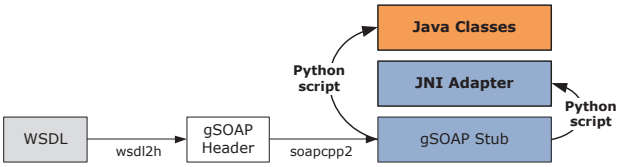

Fig. 2. Extended stub generation process

We previously described the main idea and the tools involved in the extended stub generation. Now we will take a closer look at the different steps of the process.

3.5.1. Generating a gSOAP stub

At first we generate a plain C based gSOAP stub as explained in 3.3. More details can be found in the detailed documentation of the toolkit.

3.5.2. Identifying dynamic arrays

An important issue is the handling and identification of dynamic arrays. The size of these arrays is only known at runtime, not at compile time. An example of such an array definition is shown in listing 4 (in XML Schema).

[1] http://www.gccxml.org/

Listing 4. Definition of a dynamic array in XML schema

```
<xs:complexType name="arrayTest">
  <xs:sequence>
    <xs:element name="values" type="xs:double" nillable="true"
      minOccurs="0" maxOccurs="unbounded"></xs:element>
  </xs:sequence>
</xs:complexType>
```

The resulting C struct definition of the generated gSOAP stub is shown in listing 5. The array is represented by two variables: a size variable contains the arrays' size, a pointer variable provides access to the arrays' data fields (here: double values). This is a common approach to represent dynamic arrays in C. For a human the combination of size variable, pointer and comments ("sequence of elements") is a sufficient hint that the pointer actually hides an array and must be treated differently from a simple pointer. Unfortunately this semantic gap can not be filled by machine based and automated parsing, e.g. with gccxml. The parser does not interpret any comments and therefore does not recognize dynamic arrays, variables are just handled as simple pointers.

Listing 5. Definition of a dynamic array in soapStub.h

```
struct hello1__arrayTest
{
  int __sizevalues; /* sequence of elements <values> */
  double *values; /* optional element of type xsd:double */
};
```

This is a significant issue as we need to map dynamic C arrays to corresponding Java arrays and vice versa. Thus it is necessary to identify dynamic arrays in the generated stub code by some kind of preprocessing. We scan the stub header file for the occurrence of the particular comment strings and filter the variables' names and types with basic string operations. The result is a data structure (see table 1) that facilitates checking each attribute of a struct for dynamic arrays. This will be used in the next step of code generation.

Table 1. Table for identification of dynamic arrays

Struct	Attribute	Type	Size Attribute
hello1__arrayTest	values	double	__sizevalues
hello1__testArrayMessage	values	double	__sizevalues

3.5.3. Code generation

After the stub generation and identification of dynamic arrays the most important step is the generation of the additional source files. These are mainly the *entity classes* representing the datatypes defined by the service, one or more *service classes*, which encapsulate the remote web service calls in local methods and the implementation of the intermediary layer (*JNIAdapter*). As this code generation is pretty straightforward but requires a deep look into details of our script we will skip this part and just take a look at the resulting code. This leads us back to the already introduced HelloWorld example. For reasons of simplicity we also omit the C based implementation of the JNIAdapter. Instead we present the Java side implementation of a service class in listing 6, which is also called JNIAdapter here. In contrast to listing 3 the method now takes an object of a distinct class PHello and returns an object of the class PHelloResponse. In case of a SoapFault, an exception is thrown. This looks like a typical stub method definition except it is declared as **static native**.

Listing 6. Implementation of JNIAdapter.java

```
public class JniAdapter {
  public static native PHelloResponse soapcallhello1hello(String endpoint, String action, PHello
      hello) throws SoapFaultException;
}
```

The class PHello itself is a simple entity class as shown in listing 7. The instantiation of new objects (response, C to Java) and the access to attributes of existing objects (request, Java to C) is entirely implemented on the C side of the JNIAdapter. It is important to notice that any manual changes in the generated Java classes could affect object access and data transformation and thus could result in faulty behavior.

Listing 7. Implementation of PHello.java

```
public class PHello {
  private String name;
  public String getName() { return this.name; }
  public void setName(String name) { this.name = name;  }
}
```

3.5.4. Compilation of the native code

As a last step the generated gSOAP stub code and the output of the extended generation process (JNI-Adapter) are compiled to a native library. Therefore a specific Android makefile is also generated which simplifies the use of the NDK tool ndk−build.

In summary a web service call is passed through different layers as shown in figure 3. Based on the web service description, which is provided by a service producer, all other layers except the Android Activity are automatically generated. Only the activity layer is under responsibility of the developer who wants to consume the SOAP service in his application. The generated Java classes are his entrypoint for the service. Thus in case of changes in the service interface just the generation process has to be executed again and (if necessary) the application activity adjusted. This leverages the flexibility benefits of the SOAP protocol.

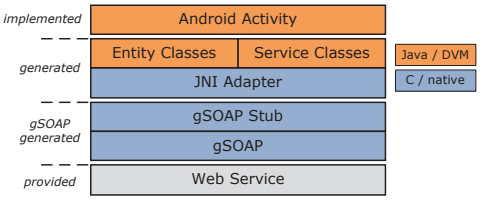

Fig. 3. Different application layers for the integration of a native gSOAP stub

As already explained earlier we also need to address security features of SOAP web services on the Android platform. Thus we will focus on the gSOAP security plugin and its compatibility with Android in the next section.

4. Secure Services on Android with WS-Security

The gSOAP toolkit provides different plugins which implement several specifications and standards, defined on top of SOAP. One of these plugins provides security as defined in the WS-Security specification [1]. This plugin enables security mechanisms like asymmetric/symmetric encryption, signatures and username tokens which satisfy the common security requirements of (message-) confidentiality, integrity and authenticity.

4.1. OpenSSL for Android

The plugin relies on the crypto library OpenSSL [18], as it already implements the cryptographic algorithms and operations needed for encryption/decryption and signing. OpenSSL is also utilized for the establishment of a potentially less secure point-to-point security context with TLS/SSL and HTTPS. On one side the crypto library is included with the Android platform, on the other side OpenSSL is unfortunately not made accessible for the NDK and native code. Therefore it is necessary to compile it separately and provide it as a native library with the application. Fortunately this is simplified by a sample NDK project to build OpenSSL as a library for Android[2]. In summary, OpenSSL is available with the Android platform which is the main basis to integrate the security plugin into our previously generated and extended stub for Android.

[2]OpenSSL for Android `https://github.com/android/platform_external_openssl`

4.2. Integration in the generation process

The security plugin must be integrated before the compilation and within the gSOAP stub generation process as shown in figure 4. Before the execution of the stub compiler the gSOAP header file must be changed. Basically this change is the inclusion of a distinct C header file which indicates the stub compiler to integrate the security plugin. After the compilation, the stub code is prepared for secure web service calls (WS-Security compliant). The security context is not automatically configured within the stub code e.g. by interpreting a services policy (WS-Policy [19]). This configuration has to be adjusted by hand with the help of the plugins' API. In our Python script we already integrated two basic security configurations, asymmetric encryption and signatures based on X.509 certificates (WS-Security) and point-to-point security with HTTPS. Regarding these two configurations we can completely generate a secured stub.

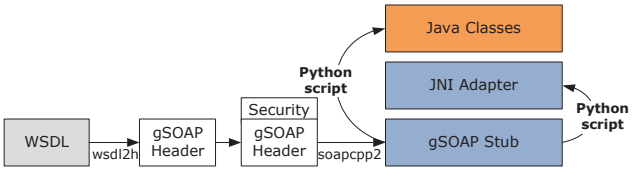

Fig. 4. Stub generation process with security plugin

4.3. Security configuration

As previously described the security plugin provides an API to configure the web services security context. As we extended the C based stub through to the Java side there are different solutions possible on how to implement and provide access to the security configuration. In practice we examined two different solutions: configuration at compile time and configuration at runtime. The first approach is shown in figure 5 where the gSOAP stub is prepared for the security plugin and the JNIAdapter prepares the security configuration by utilizing the plugin API. In this case the configuration is hard coded in the C code and the resulting native library. Nevertheless it is also possible to encapsulate the security plugin API with JNI as well. The security context may then be configured from Java classes at compile time or at runtime. In summary the presence of OpenSSL makes the use of the gSOAP security plugin on the Android platform possible.

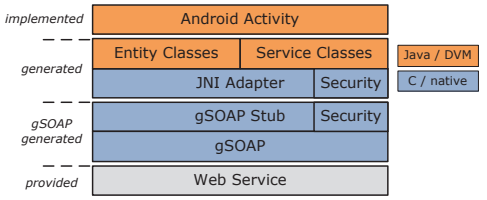

Fig. 5. Application layers with security plugin integration

5. Conclusion and Future Work

In this paper we have presented different options on how to implement SOAP web service clients on mobile devices that require support for advanced web service specifications. In particular we have been using Android on the client side and WS-Security as advanced specification. In order to leverage one of the main benefits of SOAP web services (flexibility) we put special emphasis on automated code generation to the highest degree possible. This also aids the client developer by keeping his effort minimal.

The optimal solution taking everything into account in this situation has been to use the gSOAP library with the plugin for WS-Security in conjunction with the JNI interface of Android. There are several benefits

of this solution: firstly all components dealing with SOAP communication have been generated automatically, even for the security context to a large degree. Thus only the mobile GUI and application logic had to be implemented manually. Similar solutions using gSOAP are transferable to other mobile device platforms that support gSOAP, e. g. iOS and Symbian, as well. Just the code generation part would have to be adjusted to the specifics of the platforms. Finally additional advanced web service specifications such as WS-Addressing ([20]) or WS-ReliableMessaging ([21]) can be used on mobile devices in the same way by using other existing gSOAP plugins.

In the future we plan to implement and validate the gSOAP solution for other mobile platforms as well. Depending on a positive evaluation of this extension we would like to use the process for the application scenario considered in our work (cf. section 2). Moreover it would be helpful to verify the process for other advanced web service specifications as well as applications domains. Also an extension of the Android API to natively support advanced SOAP services would be helpful to remove the dependency on the third party library gSOAP as well as to simplify the rather complex deployment scenario (cf. figure 5). Finally a detailed evaluation of the communication overhead associated with WS-Security when compared to REST and HTTPS on mobile devices is the main content of another paper. Based on these results the benefits and drawbacks of both approaches can be compared more easily and the correct choice for the particular application can be made.

References

[1] A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker, Web services security: SOAP message security 1.1 (WS-Security 2004), OASIS Standard Specification, wss-v1.1-spec-os-SOAPMessageSecurity, OASIS (2006).

[2] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, H. Granqvist, Ws-securitypolicy 1.3, OASIS Standard Specification, ws-securitypolicy-1.3-spec-os, OASIS (2009).

[3] C. Kleiner, T. Schneider, Securing soap web services for mobile devices on different platforms, in: Proceedings of the 6th conference MMS, Vol. 185 of LNI, Springer, 2011, pp. 25–38.

[4] H. Lufei, W. Shi, V. Chaudhary, Adaptive secure access to remote services in mobile environments, IEEE Transactions on Services Computing 1 (1) (2008) 49 – 61.

[5] K. Hameseder, S. Fowler, A. Peterson, Performance analysis of ubiquitous web systems for smartphones, in: Proceedings of the International Symposium on Performance Evaluation of Computer & Telecommunication Systems, SPECTS 2011, IEEE Computer Society, Washington, DC, USA, 2011, pp. 84 – 89.

[6] J. Bertram, C. Kleiner, D. Zhang, Enhancing customer privacy for commercial continuous location-based services., in: Y. Cai, T. Magedanz, et al. (Eds.), Third International Conference on Mobile Wireless Middleware, Operating Systems, and Applications, Vol. 48 of LNCIST, Springer-Verlag, 2010, pp. 326–337.

[7] J. Bertram, C. Kleiner, D. Zhang, Increasing user privacy in continuous location-based services, in: R. J. Kpper, A. (Ed.), 7.GI/ITG KuVS-Fachgespräch Ortsbezogene Anwendungen und Dienste, Logos Verlag, Berlin, 2011, pp. 175–182.

[8] D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (Informational), `http://www.ietf.org/rfc/rfc4627.txt` (Jul. 2006).

[9] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246 (Proposed Standard) (Aug. 2008).

[10] E. Rescorla, HTTP Over TLS, RFC 2818 (Informational) (May 2000).

[11] The legion of the bouncy castle, `http://www.bouncycastle.org/`.

[12] ksoap2-android, `http://code.google.com/p/ksoap2-android/`.

[13] R. Chinnici, J. Moreau, A. Ryman, S. Weerawarana, Web services description language (wsdl) version 2.0 part 1: Core language, W3C Recommendation, `http://www.w3.org/TR/wsdl20/` (Jun. 2007).

[14] R. A. van Engelen, K. A. Gallivan, The gsoap toolkit for web services and peer-to-peer computing networks, in: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '02, 2002, pp. 128–135.

[15] developer.android.com, What is the ndk?, `http://developer.android.com/sdk/ndk/overview.html`.

[16] gSOAP, gsoap: Soap c++ web services, `http://www.cs.fsu.edu/~engelen/soap.html`.

[17] pygccxml, C++ python language bindings, `http://sourceforge.net/projects/pygccxml/`.

[18] OpenSSL, Openssl - cryptography and ssl/tls toolkit, `http://www.openssl.org/`.

[19] A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, U. Yalcinalp, Web services policy 1.5 - framework, W3C Recommendation, `http://www.w3.org/TR/ws-policy/` (Sep. 2007).

[20] W. S. A. W. Group, Web services addressing 1.0, W3C Recommendation, `http://www.w3.org/2002/ws/addr/` (Sep. 2007).

[21] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, U. Yalcinalp, Web services reliable messaging (ws-reliablemessaging) version 1.2, OASIS Standard Specification, ws-rm-1.2-spec-os, OASIS (2009).