5

Privacy Preservation

This chapter describes the automated privacy preservation framework of the GAMBAS middleware. The framework extends the adaptive data acquisition and distributed data processing frameworks to support the automated sharing of contextual information in a privacy-preserving manner. In the GAMBAS middleware, privacy preservation encompasses mechanisms and protocols to limit the access to contextual information to trustworthy clients, which also allow the user to specify which data items can be used by the system. Furthermore, it includes tools to automatically derive sharing policies by inspecting privacy settings from a configurable and extensible set of web services. Specific care is taken to avoid the use of central points of trust in order to support the policy enforcement at runtime and to maximize the applicability of derived policies to different types of context information. In the following, the chapter first clarifies the focus and contribution of privacy preservation in the GAMBAS middleware. Thereafter, it describes the privacy protocols and mechanisms and discusses the policy generation tools. Finally, the chapter presents details on the integration into the other systems.

5.1 Focus and Contribution

Context privacy is an important and very active research area in the ubiquitous computing domain. Therefore, we briefly review the state of the art in this area, before we discuss the contributions of the privacy preservation framework of the GAMBAS middleware with respect to security and privacy.

5.1.1 Trusted Computing Hardware

Hardware-based privacy approaches try to make use of current security technologies that enable trusted hardware design. This is usually based on the

Intel trusted execution technology (TXT). The TXT uses the trusted platform module (TPM) that is already built-in in many business PCs and laptops. The TXT only allows trusted (and cryptographically validated) software to run on the device. So the software itself cannot be tampered. This implies that the software engineering process is monitored closely and the privacy-preserving quality of the software can be approved [LZD08]. The drawback of such a design beside the costs is the inflexibility in hardware design. For example, simply attaching a new hardware device will tamper the security, so "secure" versions of all hardware components that are used for context processing are necessary. This includes "common" components like a USB-controller or a storage-controller. Thus, in summary, all these approaches are based on special hardware and need the complex creation of trusted software.

5.1.2 Key Exchange and Derivation

Context privacy can be achieved in several ways. One possibility is creating a common symmetric key with users and devices which are allowed to access the produced context [Mis08]. This approach is often used in eHealth scenarios. An alternative is to derive keys based on the context information in the surroundings [HV09], [RB04]. The context information used to create these keys is usually based on physical characteristics like the acoustic "fingerprint" of a room or a similar "fingerprint" based on Wi-Fi radio signals. In general, common symmetric keys cannot be used in dynamic environments, because a new key must be created and redistributed if some device leaves the group of devices that are allowed to access context information. Many of the approaches that create keys from the context information that persists in the current environment either need servers and a central authority [HV09] or can only be used in a very limited physical region [RB04]. Besides using encryption to keep the transferred context information secret, it is possible to create hashes that are distributed instead of the original context information. Because a hash is a one-way function, the original context cannot be reconstructed easily. This is similar to an approach that uses hashes and pseudonyms to hide the context from unauthorized access [EBBS07].

Another centralized approach shifts the context that can be accessed to a central database [HM08]. Similarly, it is possible to rely on several third parties that store (possibly private) context information [MMG11]. Here, the user is supposed to control the access to this context information using permissions (or a user-defined policy). When a user issues requests to a context-based service, k-anonymity [Swe02] can help to make the accessing

user anonymous. However, relying on central databases always means that the user has to put trust in the database providers with regard to their compliance with the user's policy. Additionally, each provider needs to store the data securely; otherwise, a data leak may make a user's private context available.

5.1.3 Obfuscation and Generalization

Obfuscation and generalization of context information can be used to provide context privacy, usually by blurring the context. Often, these techniques are used for the privacy of location [XC09], [ACDCdVS08]. Although k-anonymity is also suggested as a solution to the privacy of location [GG03], [ZH09], [SHL+05], its use is also disputed [STD+10]. Other approaches for location privacy rely on the collaboration of users, either with [SPTH11], [RR98] or without user interaction [BS04]. MobiCrowd [SPTH11] allows users that request information from the location-based service (LBS) to share the information among each other. The information is signed by the LBS, so it can be verified by each user individually. Besides the fact that the LBS cannot gather information about users that share the context among each other, the LBS is queried less regularly, so this also has an effect on load balancing. This idea is roughly based on the use of crowds to anonymized requests [RR98]. Here, a proxy technology is used that (randomly) forwards web requests (e.g. http, ftp, gopher, etc.) to other computers or to the target server on the Internet to make the original user, who created the request, anonymous.

An approach for location privacy which is not based on the interaction between different users uses the so-called "Mix Zones" [BS04]. In Mix Zones, users are changing their pseudonyms secretly to maintain location privacy. Mix Zones require specially marked zones that cannot be used by location-based services. Additionally, using a map, many traces through mixed zones might be guessed successfully due to normal human movement behavior. The IETF working group called "Geopriv" [IET13] is also focused on location privacy. The Geopriv working group uses central servers that apply a user-specific policy and send the data to the location-based service if the policy did approve it. An evaluation of the privacy risk of location-based services [FSH12] using traces from real users concludes that current solutions which use user anonymity are effectively not providing location privacy and this result may encourage "the use of distributed solutions in which users store maps and the related information directly on their mobile devices."

The generalization of context information can also be done with other context information, especially when the information encompasses numerical

values like age or height [PRAB08]. The quality of a service customized on this context information might of course be lower than if the actual context information would have been used; however, the user's privacy is still preserved.

Social networking sites often contain different context information. Additionally, they usually allow a fine-grained access control policy to be defined. Helping the user in creating and maintaining this policy as well as extracting policy information out of the social network will allow an in-depth analysis of privacy settings [FL10]. A similar approach is taken by the privacy policy tool PRiMMA [WCMS10] that allows editing privacy settings in social networks more fine-grained than supported by the network itself. The tool allows co-ownership of shared data (e.g. photos) and allows all owners to edit the privacy settings. Since social networks currently do not support a more complex privacy policy, it is necessary to store the context data on an additional server and use a separate viewer for policy editing. Often, social networks cannot be trusted with private context information, so a decentralized social network that stores the user's profile on the user's devices [NPA10] provides a solution. The necessary access control is directly enforced by the user's devices, according to the user's policy that must be specified beforehand. To have a high availability of the user's profile, the profile's context data is distributed among devices from different, trusted users. Another approach, comparable to our approach in GAMBAS, uses a server-side aggregator [JJFZ11] that crawls through different social networks and collaboration tools to retrieve the user's context that should be shared between users and devices. The user needs to specify a common profile and edit her privacy settings, defining a privacy policy. All approaches which target the privacy policies in social networks usually involve manual user actions that need to be done additionally to defining the privacy settings in the social networks.

5.1.4 Contribution

Privacy-preserved sharing of context information is a very active research area without providing the user with a clear solution. GAMBAS provides concepts and mechanisms that focus on the automated privacy-preserving sharing of context information while still being applicable for devices in the ubiquitous computing scenario, which includes heterogeneous devices, mobility and resource constraints. Hardware-based approaches for privacy-preservation need special hardware and a defined software development

process that allows security audits, which define the "trust" in software. Since GAMBAS is using a dynamic architecture in software and hardware, this approach is not feasible.

Current approaches for context privacy often rely on (external) databases run by third-party providers. In contrast to that, GAMBAS does not rely on central databases, so no infrastructure is necessary to share context information in a privacy-preserving way. Key derivation for context privacy is usually only applicable in special environments and not a general solution in a pervasive scenario where devices exhibit mobility and are not bound to any infrastructure. Additionally, the necessary configuration conflicts with the goal of a distraction-free usage of devices. The context generalization in GAMBAS extends the existing approaches. If a generalization path is available that would make the context information privacy-preserving according to the used privacy policy, GAMBAS tries to use obfuscation or generalization, so customized service access is possible while preserving privacy. This obfuscation can be done in an automatic way, without distracting the user.

Approaches that extend social networks mainly focus on the privacy policies. The policies must be created manually by the users. This requires the user to learn the usually complex privacy policy language. Additionally some approaches require a central server that stores context and/or the defined policy. This requires additional server infrastructure where the user's context is stored. In GAMBAS, we use a decentralized approach where the context is usually stored on the user's device instead of a third-party server. Privacy policies can be retrieved automatically from social networks without user interaction. Also, these approaches must be extended to be applicable to not only one social networking site, but many and to other context providers like physical sensors. To create a privacy-preserved sharing of context information, GAMBAS encompasses extraction tools that gather and generalize privacy policies from a set of web services automatically as well as an associated set of mechanisms and protocols that enforce these policies at runtime.

5.2 Privacy Framework

In GAMBAS, the data acquisition and the interoperable data representation and processing mechanisms are developed to gather and distribute all possible types of data. Furthermore, it is possible to access dynamic as well as static information using one-time and continuous queries. To protect the privacy of users, the privacy framework has to limit the data acquisition and in particular

the data sharing such that it respects the privacy preferences (i.e. policies) of different entities. Enforcing the desired limits is the primary task of the privacy framework.

Conceptually, the privacy preservation framework interacts with the semantic data storage (SDS) as well as the data acquisition framework (DQF) that are deployed on each personal device. In addition, the privacy framework may also be used to limit the access to information that is provided by a particular service. For this, it is also integrated into devices that are offering the services. Using a privacy policy, the privacy framework takes care of exporting sensitive data in a way that it can only be accessed by legitimate entities. The necessary privacy policy can be generated automatically by means of plug-ins that access proprietary data sources. Furthermore, depending on the user preferences, the framework can apply obfuscation in order to limit the data precision and it can also anonymize the data in order to unlink the data from a particular user. Since GAMBAS aims at supporting the use of personal mobile devices as primary sources of data, the privacy framework supports not only traditional computer systems, but also constrained computer systems as its execution platform.

5.2.1 Overview

The architecture presented in Chapter 2 describes different views on data. Regarding privacy, there are two relevant views. One is the data acquisition view in Section 2.2.1, and the other one is the processing view in Section 2.2.2. Here, we first concentrate on the data acquisition view, before we have a look at the processing view from a privacy-preserving perspective.

The data acquisition view envisions two different scenarios. The first scenario is targeting the personal acquisition of data that is used to capture the user's behavior on behalf of the user. The second scenario is targeting the collaborative acquisition of data from a large number of users that is used to improve or provide a particular service upon request of a service provider. In both scenarios, private data may be processed. Therefore, both scenarios are relevant regarding privacy.

For the first scenario, the identity of the user is important to ensure that the resulting profile can be associated with the right user. Consequently, the acquired data may be highly sensitive from a privacy perspective. For the second scenario, the user's identity is not that important, since often an aggregated view of the data will be used. Additionally, for both scenarios, it is necessary for the user to give an explicit consent to the data acquisition

at least once in order to ensure that only the desired data types are acquired. To do this, the user can interact with the privacy framework by means of the intent-aware user interface to define the associated preferences. In the following, we show and describe the architectural figures from Section 2.2.1 that were extended to highlight the relevant parts for the automated privacy preservation framework.

As can be seen in Figure 5.1, the privacy preservation framework is relevant for every step of this scenario. The first two steps include the retrieval of the policy-related data from a third party (e.g. a social network or a business collaboration tool) and the generation of a personalized privacy policy from this data. A Policy Generator can create this policy using the policy language described later on in this chapter. Similarly, the third step concentrates on the policy. Here, the integration and visualization of the policy in the user interface is the focus of this step. It enables the user to manually modify the automatically generated policy to suit his or her needs. The last two steps concentrate on the data acquisition and the storing of collected data. For privacy reasons, the user may limit the data acquisition directly at the data acquisition framework, actively avoiding the gathering of certain data. A second filter step includes the short-time or long-time storage of data in the device-based registry. The user may limit or modify (e.g. obfuscate or blur) the stored data according to his or her policy. Since this scenario is focused

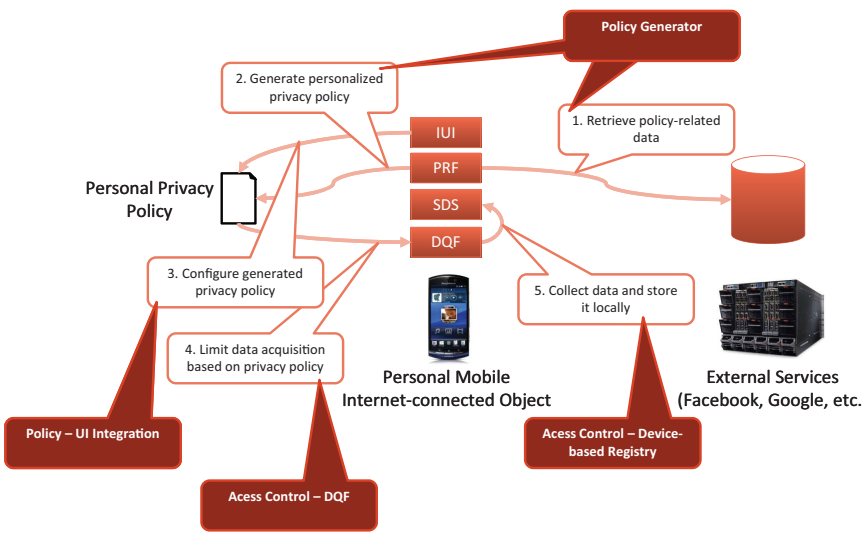


Figure 5.1 Privacy Components for Personal Data Acquisition.

on personal data acquisition, this may affect predictions that are based on the data's history, but it does not affect other devices.

The second scenario is focused on collaborative data acquisition. This includes the sharing of data with third parties, like an external SDS. The scenario is depicted in Figure 5.2. While the first four steps are identical to the ones presented for the first scenario on personal data acquisition, the last step differs. In the last step, data is not stored locally on the device, but transferred to a remote SDS where the data is stored or further processed. At this point, the privacy preservation framework needs to secure the connection to the remote service. This is done by means of mechanisms for device/service authentication and by encryption. The encryption prevents eavesdroppers from overhearing the data transmission and is necessary since the data might be transferred over insecure networks like the Internet. The authentication enables an access control component to identify the remote service and to apply the necessary limitations with regard to the acquired data. Since the data is shared with a remote service, it is often necessary to enforce a stricter policy. The access control component of the privacy preservation framework must therefore limit, anonymize, obfuscate or blur data, if requested by the policy.

In addition to data acquisition, the second relevant view is the processing view. Similar to acquisition, the processing view envisions two scenarios

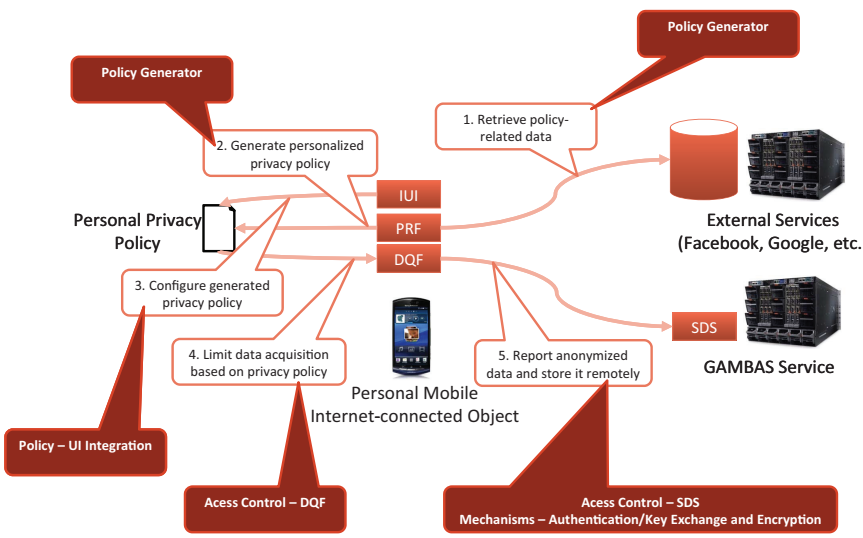


Figure 5.2 Privacy Components for Collaborative Data Acquisition.

that are relevant with respect to privacy. The first scenario describes the processing of shared data using a one-time query to the data discovery registry (DDR), and then accessing the shared data source. In the second scenario, a continuous query is executed at the continuous query processor (CQP) that retrieves and sends data on behalf of the user continuously.

The one-time processing of shared data is depicted in Figure 5.3, which shows how the privacy preservation framework integrates into the GAMBAS architecture for the processing of shared data. As a first step, if the data source's owner decides to share data through GAMBAS, the data source will be exported to the DDR. If now, as a second step, a device (i.e. the query issuer) initiates a query regarding the data source(s), it will look up the data sources at the DDR. After that, the query issuer will remotely access the data, if the user gave his consent to accessing and processing remote data. The consent is provided by means of the privacy policy. The remote data access makes use of the authentication and key exchanging mechanisms that are provided by the privacy preservation framework (Step 5). On access, the shared data sources check the status of the query issuer (i.e. check, if their policy allows data to be shared with this entity) and create a personalized view for this query issuer. In the last step, the query issuer uses the key that was exchanged in Step 5 to access and retrieve the data from the shared data sources. In this step, the communication channel is encrypted to prevent unauthorized devices from overhearing the data in transit.

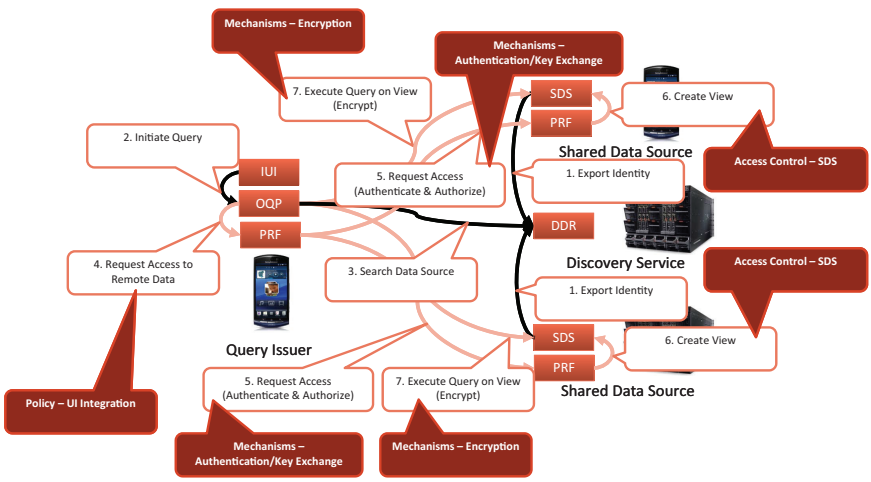


Figure 5.3 Privacy Components for One-time Processing of Shared Data.

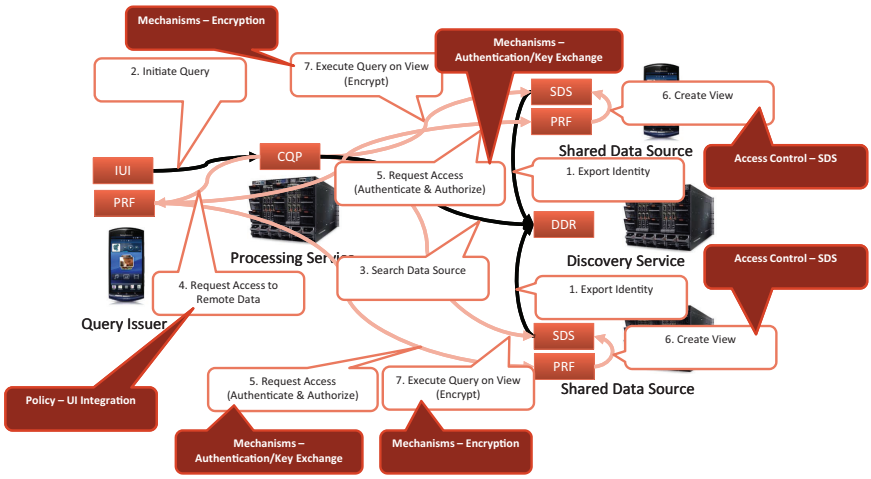


Figure 5.4 Privacy Components for Continuous Processing of Shared Data.

The second scenario is shown in Figure 5.4. In this scenario, a continuous query is executed. In contrast to one-time processing, the query is not executed by the query issuer itself. Instead, an intermediate middleware service, the continuous query processor (CQP), is used. This changes the message flow in comparison to Figure 5.3, since the CQP executes the query (i.e. performs Step 7) and not the query issuer. Therefore, the query issuer needs to trust the processing service that is running the CQP to perform queries and aggregate data reliably. As the CQP is executing the query on behalf of the query issuer, the query issuer is still requesting the access to the shared data sources. The retrieved access token is then handed over to the CQP, which uses it to execute the query. Although the message flow is more complicated, due to the addition of the CQP, the processing load decreases for the query issuer. From a privacy point of view, the CQP executes and analyzes the query, i.e. processes potentially private data. Any query issuer that is using a CQP should therefore either only request and process public data or must exhibit trust in the CQP it is using.

5.2.2 Mechanisms

The privacy preservation framework uses several mechanisms to keep data private, e.g. prevent eavesdroppers from overhearing private data, establish encrypted communication channels and authenticate users and servers. Additionally, the framework uses a privacy policy to describe which data should

be shared with whom. The mechanisms then make sure that the primitives that are defined by the policy (i.e. users, groups, data and access rights) are enforced properly at any point in time. To enforce the policy with regard to users or groups, authentication is necessary. For the security of data, devices and servers need to communicate securely (i.e. using encryption communication channels). Access to the shared data is controlled by combining authentication and secure communication. Additionally, access control must be enforced depending on the different views and scenarios that are targeted by the GAMBAS middleware.

To support remote communication, the GAMBAS middleware relies on the BASE communication middlware depicted in Figure 5.5. Originally, this middleware has been developed by researchers at the Universität Stuttgart [BSGR03] and it has been refined over several years [HWS+10]. For example, in the European research project PECES (PErvasive Computing in Embedded Systems) [PEC12], BASE has been used to enable the secure networking of embedded devices in smart spaces over the Internet [AHM12].

As hinted in Figure 5.5, the BASE middleware provides a rather traditional object-oriented interface for the application programmer, which relies on explicitly defined service interfaces and generated proxies and skeletons. Underneath, it enables spontaneous and secure device interaction and discovery. To do this, BASE relies on an extensible plug-in model that can be used to support different communication technologies and protocols. The extensibility of BASE includes hooks for the integration of authentication and key-exchange mechanisms as well as encryption protocols. However, instead of describing BASE, in the following, we focus on the contributions of GAMBAS that are required to implement the overall system architecture. From a conceptual point of view, these contributions are independent of the

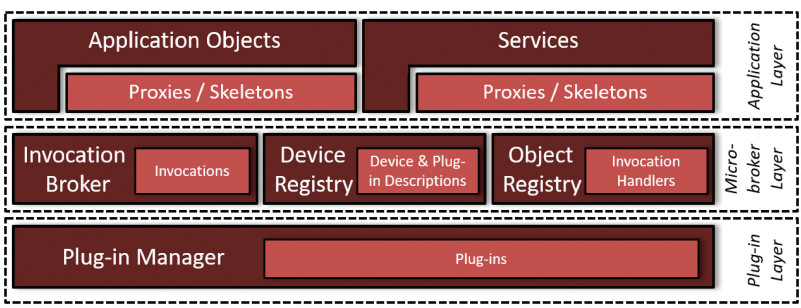


Figure 5.5 BASE Middleware.

concrete implementation and could have been implemented on top of other communication middleware systems as well. However, due to its flexible communication plug-in support, we found that implementing them with BASE was efficient.

5.2.2.1 Authentication and Key Exchange

In order to enable trustworthy and secure interaction between devices, it is necessary to authenticate interacting devices and the data exchanged between them. In particular, it is necessary to authenticate individual devices/services, e.g. during the establishment of a connection or during the access of shared data.

In GAMBAS, authentication relies on both asymmetric and symmetric cryptography, which requires the availability of keys. In the case of symmetric approaches, the keys are available only to a particular set of devices, which may use this key to ensure authenticity with respect to the devices that share the key. In the case of asymmetric approaches, the key consists of a public part (the so-called public key) and a private part (the so-called private key). The keys may then be used to authenticate individual devices.

Both symmetric and asymmetric approaches can be used to distribute further keys on the basis of existing keys. However, there needs to be at least one key available to bootstrap the overall process. Usually, this key needs to be distributed by means of a secure channel. Typically, this is done offline, e.g. as part of the device configuration. In GAMBAS, while still supporting this type of key distribution, we also offer a more convenient key exchange for user-to-user authentication, which is as secure as the underlying service.

5.2.2.1.1 *Server Authentication*

Server authentication enables the authentication of a server or a server-based service to another device. The other device can either be another server (for server-to-server communication) or a user device (e.g. a smartphone). In GAMBAS, servers are used to host services like a traffic information service or GAMBAS-related services like the CQP. The authenticity of these servers and services is important since GAMBAS applications rely on the data retrieved from them.

It is noteworthy that the server infrastructure envisioned by GAMBAS is similar to the server infrastructure in other networks, like the Internet. Here, pre-deployed certificates enable the verification of the authenticity of sites for purposes like Internet banking or e-mail retrieval. Since these mechanisms are in daily usage and have been proven effective for years, GAMBAS also relies

on them. Each server in GAMBAS is therefore equipped with a certificate that is issued by the certificate authority or some trusted third party (e.g. a particular company).

Since certificates rely on asymmetric cryptography, this results in a key pair (a public and a private key) being deployed on every server. While only the server knows its private key, the public key (as part of the certificate) is shown to devices for authentication. Using a common certificate infrastructure, the public key is signed by the authority's key pair, which might then again be signed by the domain authority's key pair, leading to a certificate tree. An example certificate hierarchy tree is depicted in Figure 5.6.

As can be seen in Figure 5.6, it is not necessary for a GAMBAS application to trust a whole company. It is sufficient to trust only the parts of the company that are providing GAMBAS-related servers and services. Accessing a GAMBAS-related server will then trigger a certificate verification. It is possible to verify whether a particular certificate belongs to the GAMBAS-related sub-tree by recursively validating the certificate chain from bottom to top. To do this, the signatures must be verified one at a time. If the chain is valid and if it contains a pre-deployed GAMBAS certificate, the validated certificate belongs to the spanned part of the tree, i.e. it belongs to a valid GAMBAS server.

Similar to other infrastructures, the GAMBAS middleware makes use of the X.509 certificate standard. Among other things, this standard defines a common format for certificates, which enables the use of existing tools to generate keys and certificates offline. Specifically, it is possible to use the implementations provided by the OpenSSL library. This avoids the need for implementing key generation mechanisms and thus, it eliminates the need for providing tools that exist already.

For device authentication, GAMBAS uses an authentication based on the standard ISO authentication framework [CCI89], which can be used with the Diffie–Hellman (DH) key exchange in its original version (using

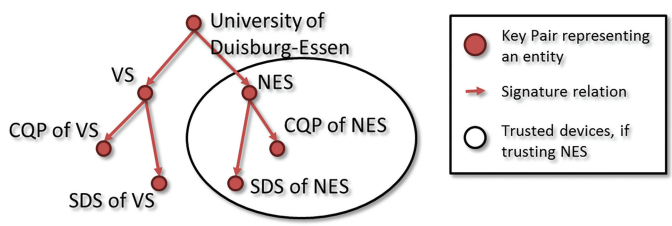


Figure 5.6 Certificate Hierarchy Example.

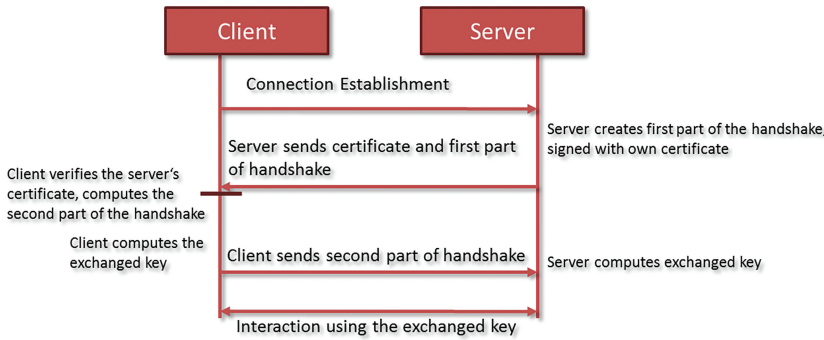


Figure 5.7 Certificate-based Key Exchange.

RSA certificates). The interaction is depicted in Figure 5.7. Additionally, the GAMBAS middleware supports a modified version of DH that relies on elliptic curve cryptography (ECC) certificates, which is more lightweight and therefore better suited for the use on smartphones or other devices with constraint resources. The exchanged keys can then be used with a key-derivation function like PBKDF2 [Kal00] to create a common shared key among any set of devices.

For pre-deployed username/password combinations, we use a hash-based authentication mechanism that does not reveal any user or password information to eavesdroppers which is important, since the past successful attacks on protocols such as MS-CHAPv2 show the necessity of a higher security standard. However, due to the focus on smartphone applications, finding the right balance between security and user convenience is a challenge.

Both the certificate-based and the username/password-based key exchanges result in the computation of a common shared key that cannot be computed by any attacker that might have overheard the communication. This key may be stored by a key store component of the middleware and used for further communication attempts, which speeds up the communication start by skipping the authorization part. This does not result in lower security, because the possession of a common shared key shows that each interaction partner was authorized properly before. Nevertheless, such a key should never become persistent. It should time-out or be renewed after a certain amount of time.

5.2.2.1.2 *User-to-User Authentication*

User-to-user authentication explicitly authenticates one user's device to another user's device, e.g. to share data between two smartphones. This can

be used to share data between users that trust each other, e.g. friends or co-workers. The authentication between users is different from the server authentication described previously, because the devices are not necessarily part of a certificate infrastructure. Only few users set up a certificate infrastructure for their private devices, so we cannot reasonably rely on user certificates.

Clearly, user-to-user authentication is not necessary in all scenarios, e.g. if a user requests information about the next bus from A to B from a service provided by the bus company. It is necessary, however, in scenarios that include the collaboration of users. This includes the sharing of data (like the current location) or a behavior profile that describes a possible future movement pattern of the user. Clearly, such private information should not be shared with anybody, but instead, it should be properly secured. As the first step to the solution, GAMBAS introduces an innovative user-to-user authentication mechanism that makes use of collaboration tools such as Google Calendar or social networks such as Facebook and that can be used as an alternative to the common infrastructure-based certificate architecture.

Many users are using social networks or similar services on a regular basis. They define trusted users in these networks by adding them to their personal network (e.g. friend relationships on Facebook). This information can be used to exchange a shared key, piggybacked on the service. To do this, GAMBAS introduces the so-called PIggybacked Key-Exchange (PIKE) [AHIM13].

PIKE can be used on any service that enables the secure restricted sharing of resources. This means that the service authenticates its users, models relationships between different users with respect to resource usage and enables the specification and enforcement of access rights. From the perspective of the users, the service performs its access control to resources properly. This means that a) it protects the resources from being accessed by illegitimate users and b) it allows access from legitimate users. Yet, beyond proper service operation, we do not assume that the service is necessarily trustworthy. Examples for these services are Facebook or Google Calendar. To use PIKE, the device of the user must be able to access the service regularly through the network. For this, the service provides some API or it uses a mobile application that synchronizes the changes to the resource.

Every time the friend relationship changes, PIKE starts to analyze the friends in order to detect new friends. In case a new friend is found, it will trigger a key exchange between the two friends, using the secure resource-sharing capabilities of the service. To do this, PIKE performs either a local

modification on the triggering resource or, if this is not possible due to a limitation of the mobile application, it uses the API of the service. Once the changes have been made, PIKE simply waits for the next resource synchronization at which point the new friend will have received the key through the secure resource.

Once the personal interaction takes place, these keys can be used for authentication among the devices of the friends. To do this, PIKE simply extracts the keys from the secure resource and provides them during the interaction to the GAMBAS middleware.

To formalize this interaction, Figure 5.8 depicts the resulting logical protocol flow. Conceptually, PIKE involves three entities, namely the two devices of the interaction partners (i.e. "Friend A" and "Friend B") creating a new friend relationship and the service. To establish keys, these three entities interact with each other using three steps.

- After the change in the relationship was triggered (either through an active notification or through a regular service synchronization interval), the two friends contact the service to check if there needs to be a key established between them.
- If so, the two friends compute two keys (K_A and K_B) independent from each other and post them to a secure resource.
- In the next synchronization interval, they recognize and retrieve the key posted by the other friend. Then, they compute the combined key K_{AB} and store it on their device(s).

After the completion of these steps, the interaction partners possess the exchanged key. Once a personal interaction through GAMBAS takes place, the key (or a derived key) can be used to enable group communication as well as private communication and user-level authentication between the two friends.

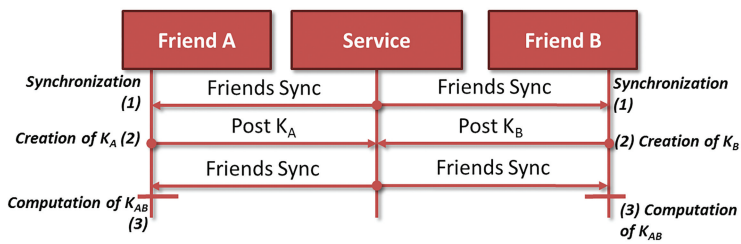


Figure 5.8 PIKE-based Key Exchange.

Figure 5.9 User-level Key Posted on Facebook.

To execute PIKE on top of the Facebook service, GAMBAS uses the Facebook Graph API to access and modify data from the social network. Each user of Facebook has a place for discussions, the so-called *wall*. This wall is used to post the keys K_A and K_B that is then automatically picked up by the friends' devices. Since friends cannot change the visibility of posts on another friend's wall, the keys are posted to the own wall. On this wall, posts can be created with a privacy setting that constraints the access to the other friend (see Figure 5.9 for an example). The friends will then retrieve their keys by going through their walls.

The combination of K_A and K_B to K_AB can use different mechanisms. While simple mechanisms like an XOR of the two values and the use of a key-derivation function to create K_AB will result in the same security as the underlying service (i.e. Facebook, which does not leak the posts, i.e. complies with its security and privacy settings), a more complex mechanism like a Diffie–Hellman key exchange can also provide security against data loss.

The key K_AB that will be exchanged after performing PIKE enables the users to authenticate each other with an exchanged key, even when their devices are not connected with the Internet, but in physical vicinity. A key for every friend relationship ensures that the authenticity is on a user-to-user basis and even malicious users cannot tamper the authentication to another user. Similar to other exchanged keys in GAMBAS, this key may be stored by the key store component and used for further communication attempts, which speeds up the communication start by skipping the authorization part. Also this key should not become persistent, but PIKE should be re-performed from time to time such that the key is renewed.

5.2.2.2 Secure Communication

Secure communication is generally used to avoid eavesdroppers from overhearing private data. In GAMBAS, the communication between different

services, servers and mobile devices may contain private data. Imagine a user searching for the next bus to the mall. If this search (usually a request to a travel planner service) can be overheard, not only the next location of a user (i.e. the mall), but also the planned activity (i.e. shopping) is revealed. Similar problems occur, if personal data like audio recordings, GPS coordinates or movement patterns are shared between users. Any eavesdropper might receive this data if he is in the vicinity and can then later analyze this data, creating user profiles. To avoid this, the GAMBAS middleware relies exclusively on secure communication channels.

To establish a communication channel between two devices, the BASE middleware uses plug-ins that abstract from the used communication technology. Due to BASE's architecture, it is possible to extend this plug-in stack easily. For secure communication, we add an encryption plug-in to the set of existing plug-ins. The plug-in searches the key stored in the device local SDS for a key of the communication partner and uses this key to perform authenticated and encrypted (i.e. secure) communication. An example communication stack using the encryption plug-in is shown (for multi-hop communication) in Figure 5.10.

Although not all applications in GAMBAS require secure communication, recent publications [AHM12] have shown that the overhead by means of communication latency is small. Therefore, secure communication is activated by default and should only be deactivated for public announcements. The encryption technology used in GAMBAS is AES, a symmetric encryption mechanism, which is both fast and secure and available for all devices in the GAMBAS scenarios. AES relies on a shared key between the communication partners that must be exchanged beforehand. The authentication/key exchange in Section 5.2.2.1 shows how such a key can be established in

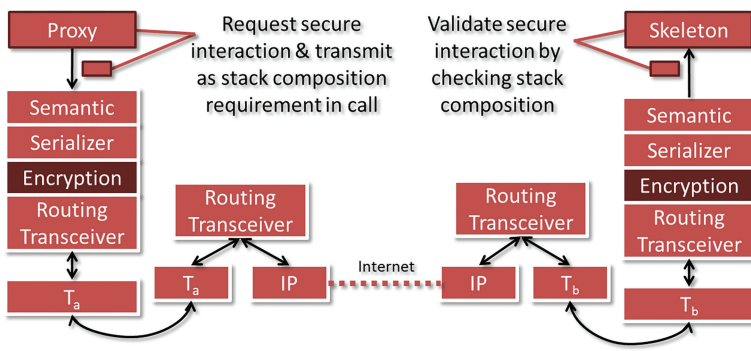


Figure 5.10 Secure Multi-hop Communication Example.

GAMBAS. To establish a secure communication, authentication is a crucial step that must not be skipped. Without authenticity, the identity of the communication partner remains unclear. If a secure communication channel does not establish identities, any data that is traveling to a (possibly) unauthorized communication partner must be regarded as public data.

After the authentication, the exchanged shared key is stored in a key store together with the device or user id. To enable secure communication, the device or user id is then used to retrieve the key from the key store. To improve the performance, the shared key can be cached after the communication for the next interaction, but should be changed regularly (e.g., by performing a re-authentication) to avoid impersonation attacks using lost keys for interactions.

5.2.2.3 Access Control

To ensure privacy, GAMBAS relies on user-specific privacy policies. Access control enforces these privacy policies. Using access control, data that is captured by the data acquisition framework (DQF) is protected from unauthorized access. In GAMBAS, access control must take into account the following three points:

- **Authentication:** A user or device must be authenticated, before it may access any resource in GAMBAS that is using access control. Therefore, it must use one of the mechanisms described in Section 5.2.2.1.
- **Encryption:** A user or device must use encryption while accessing data that is using access control. The encryption, described in Section 5.2.2.2, enforces the secrecy of the data while it is being transferred.
- **Policy Compliance:** Before any data is transferred, the access control must check the policy (see Section 5.3) for the data to be sent. The policy contains the users or devices that may access the data (if any) and the access control must follow the policy.

If these points are evaluated properly by the access control mechanisms, the policy is enforced securely. The general process of access control in GAMBAS can be seen as the execution of these six steps:

1. Device A wants to access private data on Device B. Since the data is private, Device B is using access control to protect it from unauthorized access.
2. Device A opens a communication channel to Device B. It sends the plug-in configuration for authentication/key exchange and encryption to signal the need for secure communication.

3. Using the plug-ins, the two devices authenticate to each other. Device A sees that Device B is owned by "Bob", while Device B authorizes the user "Alice" from Device A.
4. Device B now checks, if Device A is using encryption on the communication channel. If this is not the case, the interaction is terminated otherwise the interaction continues.
5. If successful, Device B checks the policy for the data that is to be retrieved by Device A. It searches for the appropriate data type and the access rights of Alice.
6. If the data type can be found and the access rights of this type allow Alice to access the data, Device B grants access and Device A can retrieve the requested data.

In general, it might not be necessary to authenticate Device B in Step 3. Nevertheless, many of the authorization schemes presented in this document are using symmetric authentication, i.e. both communication partners are authenticated at the same time. Additionally, the general process is modified depending on the communication partners in GAMBAS.

In GAMBAS, access control is used to access any private data. Since the scenarios in GAMBAS are manifold, the general access control process needs to be adapted to these scenarios. In the following, the three different access control mechanisms in GAMBAS are presented. At first, we show how access control is used in the data acquisition framework. Then, we concentrate on any device-based registry and at last, we describe how data access and access control with remote data storages is realized.

5.2.2.3.1 *Data Acquisition Framework (DQF)*

The data acquisition framework (DQF) is running directly on the user's device. It is implemented as a module of the GAMBAS middleware, which is realized as a combination of different modules. In GAMBAS, all modules are running in the same process on the device. Since processes in operating systems are isolated against each other, only other GAMBAS modules (running in the same process) can call the internal API. The PRF provides methods that allow the DQF to check whether a certain data type is allowed to be detected. The DQF must call this method before any attempt is taken, to create a recognition stack for detecting any kind of data or context. The method then returns a value that states whether the data or context is allowed to be detected or not. The DQF then changes the recognition stack accordingly to only detect the kind of data or context that is allowed to be detected by the privacy preservation policy.

This access control mechanism does not need any authentication or encryption since it limits the data acquisition directly on the device. Only GAMBAS modules can therefore retrieve and access the policy and the acquired data. On every startup of a GAMBAS application that needs to acquire data using the DQF, the DQF will check the privacy policy for any data type that needs to be detected by this application. If the policy does not allow the gathering of this data type, the application might not be started successfully, but the privacy of the user is preserved. This type of access control enhances the privacy of the user by not capturing data. Data that is not captured cannot get lost or overheard by anybody, even if the user's device gets stolen, the data cannot be revealed since it was not gathered at all. Not acquiring data is therefore a valid privacy goal that can be fulfilled in GAMBAS using the privacy preservation policy. It puts the user in the direct position of defining the data types that are allowed to be used for context or activity recognition.

5.2.2.3.2 *Device-based Registry*

Any device-based registry like the semantic data storage (SDS) stores data that was gathered by the DQF. The data is stored directly on the device itself, not involving remote interaction. Similar to the limitation of data gathering that was described in the previous subsection, this allows the access control to be performed without the need of encryption and authentication.

In GAMBAS, the data stored in a device-based registry is used to predict possible user behavior in the future. To protect his privacy, a user can choose not to store specific data on the device at all, such that no history on the device is created. Additionally, the privacy preservation framework makes it possible to mark stored data as not exportable. This can be modeled using a policy entry for this specific data type, which does not give any access rights to another user. The data is then only processed on the device itself and does not leave the device. Of course, this might result in a limited prediction since the device only has limited processing power. To mitigate this, the preservation policy that is used to limit the access is personalized to each user and may be tweaked, if it is perceived as too restrictive or too liberal.

The PRF contains a method that must be called through the API by any GAMBAS application, if data acquired by the DQF is stored on the device (e.g. using a device-based SDS). This method is similar to the one described in the previous section, but returns whether the data may be stored on the device or not. The application can then see if it is allowed to build a data history for this type of data. It must then comply with the result of this call.

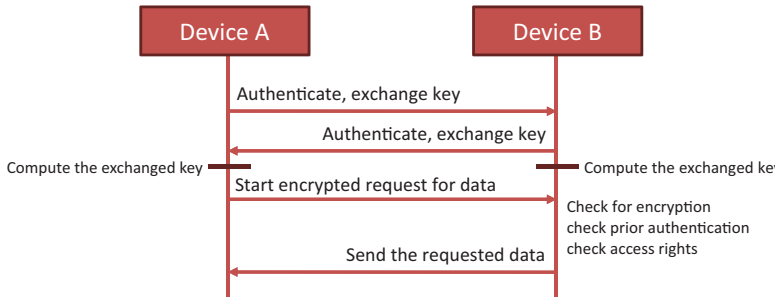


Figure 5.11 Data Request using Access Control.

In contrast to the DQF, a device-based registry like an SDS also contains a remote interface that may be called by other devices. If no data is shared, this remote interface must be inaccessible for other devices. If data is shared, the remote interface uses the device's PRF to perform access control as described in the general process of access control above. A check for authenticity and access rights, as well as the use of encrypted communication is necessary, before any private data may be shared. A simplified message flow for a successful data request can be seen in Figure 5.11.

5.2.2.3.3 *Remote Data Storage and Continuous Queries*

In some scenarios envisioned by GAMBAS, data might be stored outside the user's device. It could be stored in a remote SDS that provides additional computing capacities to give a better prediction on the future values of the data. Storing data remotely requires full user consent and could possibly breach privacy, since private data is transferred and stored on a remote device or server that is usually not owned by the user. This scenario is depicted in Figure 5.2. Here, the data is stored on a remote SDS, for example, to be aggregated for statistical purposes.

Private data is only stored remotely if the privacy policy has a valid entry for the specific private data type and it allows the sharing of the data type at this remote location. Similar to the mechanisms described before, the PRF provides a method that shows, for given values of data type and remote service or server, whether the data is allowed to be transferred there. In addition to the enforcement of the policy (which already includes the authentication of the remote service or server), the remote transfer needs to be encrypted, such that the data cannot be overheard. The access control mechanism is implemented similar to the ones described previously. Since storing data in a remote location is inconvenient for many users, the GAMBAS applications

try to minimize the need for this. One important exception is the remote storing of information that users are obligated to by contract, for example, a bus company that gives out chip cards, which are validated by touching chip card readers at the bus entry, may use the travel information in an anonymized fashion, if it informs its customers accurately.

In addition to the simple one-time query that usually only needs one request–response message flow, GAMBAS supports continuous queries that may be used to notify users if the response data changes. Continuous queries do need a permanent Internet connection and may need more resources than a simple smartphone can provide (in terms of CPU power and RAM). Therefore, the continual query processer (CQP) is realized as a remote GAMBAS service.

A query involving a remote CQP changes the authorization flow, since the CQP is querying other data sources on behalf of the user. As can be seen in Figure 5.12, the device now first authenticates the remote CQP service, which will then issue a request to access a certain data source. The device must now check with the privacy policy whether the CQP service is allowed to access the requested data on behalf of the user. If the policy evaluates to true, the data source is queried to hand out an access token that enables a remote device to act on behalf of Device A. The data source will again check, if the user of Device A is allowed to retrieve the queried data. If that is the case, the data will be processed by the remote CQP service. The PRF of the data source will consult its policy to evaluate these two questions. If they evaluate to true, the data source will transfer an access token to Device A. This access token

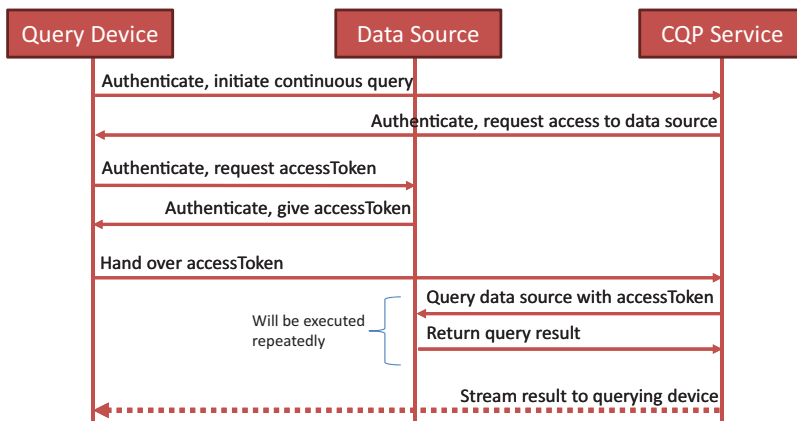


Figure 5.12 Continuous Query Processing using Access Control.

can then be used by the remote CQP service to execute the query, retrieve the data and stream the query result continuously to Device A. The CQP might execute the query repeatedly and update the continuous query result that is streamed to the device accordingly. If used with more than one data source, the CQP service needs an access token for every source and is also used to aggregate data remotely. This aggregation removes burden from the device and makes it possible to execute even complicated continual queries with resource-constrained devices.

To preserve privacy whenever possible, the remote CQP service needs to be properly authenticated and trusted by the device using it. An ideal CQP would be a home server that is in possession by the user itself. In that case, the query result will not depend on the relationship between the data source and the CQP service (since it is identical to the relationship between the source and the user's device). If an external CQP service is used, the data source could change its view on the data, since the policy on the source could have different constraints for the user and the external CQP service.

In the case of a remote CQP, all interactions between devices, data sources and the CQP must be encrypted, because they might contain private data. As shown in Figure 5.12, also all devices must be authenticated, such that the privacy preservation framework can perform the access control properly. Although the access control process is more complicated since more parties are involved, the benefits of using a remote CQP (i.e. a resource-saving query execution) outweigh the drawbacks in many scenarios.

5.3 Privacy Policy

The privacy preservation policy is used to describe the access rights of data types. Additionally, it describes how data types relate to each other. The policy is customizable for the user and can be serialized in a policy language that is based on RDF. When used in the PRF, the policy can specify which data types should be shared with which users (or companies). Therefore, the policy contains the data types and the sharing permissions, individual to each user.

The policy representation shown in Figure 5.13 displays policy permissions (i.e. using triples), which model the access rights on data types. Each

```
PermissionA affects Location
PermissionA grantedTo Bob
PermissionA obfuscation City
```

Figure 5.13 Privacy Policy Permission Example.

of these RDF triples contains a unique name as first argument, and then one of the relations "affects", "grantedTo" or "obfuscation", which denote different aspects of the policy permissions. The third argument of the triple, i.e. "affects", is the data type that should be affected by this permission. The relation "grantedTo" denotes the user that is granted this permission. Since a permission could grant the same access rights to many users, the triple using the relation "grantedTo" can occur more than once (with different users) in one permission. The user name (which could include a unique identifier) links to the profiles that this user is using on social networks or other collaboration tools. The last relation "obfuscation", optionally defines the obfuscation level for this permission. Depending on the data type, different obfuscation levels are possible. For the current location, this could be the actual GPS coordinates (i.e. no obfuscation), the current city or the current country the user is located in. Since the policy is created individually for every user, the user itself is implicitly part of every policy triple and is left out in the policy language. This means instead of creating statements like *Charlie's data type location is "grantedTo" Bob using the "obfuscation" level city*, we simplify the policy triples in Charlie's policy to the ones depicted in Figure 5.13.

The data types are specified in the data model described in Chapter 4 and used by the data acquisition framework discussed in Chapter 3. Since each different data type might provide different obfuscation levels, the levels are also defined as part of the data model. While some data types like "location" can provide more than one obfuscation level, other data types might not provide any. Thus, the use of obfuscation is optional and depends heavily on the underlying data type. In summary, both the data type and the obfuscation level are based on the data model of the GAMBAS middleware.

Another type of policy definition are triples that describe relations between different data types. These relations define a simple hierarchical relationship between data types and can be used to infer access rights for similar data or data that is used as a building block for a more complex data type. Imagine that Charlie shares his current location with Alice. If now Alice asks for the name of the street where Charlie is located, the PRF will search for the data type "street", might fail to find a policy entry for it and will deny access to it. Therefore, the policy language includes the *consistsOf*-relation. Using the privacy policy *Location consistsOf street,city,country*, the data type "street" can be found and if there are no *Permission* policy relations for "street", the "location" data type is checked. In this example, Charlie shares his current location with Alice, so the location data type grants access rights also for the "street" data type.

To support the different level of access control, e.g. storing data on the device itself, device-based registry and the sharing of data with remote devices, the policy introduces another relation that describes the level of sharing data. The *sharingLevel*-relation shows on which level data may be shared or stored. Using this relation, the privacy policy can be easily used to enforce the sharing level on data. GAMBAS relies on three predefined keywords that describe where data may be stored. The tree keywords are "Remote", which defines that data may be stored by remote devices, "Device", which denotes that the data should only be stored at the device itself and must not be shared with others, and "DetectOnly", which does not allow the data to be stored anywhere. When "Device" or "DetectOnly" are chosen, the *Permission*-relations are ignored, i.e. data may not be shared with anybody, when using this keyword.

An example for the *sharingLevel*-relation is presented in Figure 5.14. Here, location data is shared with remote devices; for the access rights, the *Permission*-relations that are linked with the location data type must be considered. Data about the current travel path may be stored on the device and used for prediction that is executed on the device. This policy triple does not allow sending the current travel path data to remote devices. In this example, audio data might only be used for detection using the DQF, but not be stored anywhere.

In summary, the privacy policy consists of three relation types. All of them can be described using privacy triples:

- The *Permission*-relations that define access rights and obfuscation levels of data types.
- The *consistsOf*-relation that defines hierarchical relationships between data types.
- The *sharingLevel*-relation that defines the sharing level of the data type.

Next, we describe the policy generator, which enables the automatic generation of the policy from social networks or other collaboration tools. Thereafter, we describe the integration of the privacy policy with the user interface to enable the user to modify the policy manually.

```
Location sharingLevel Remote
CurrentTravelPath sharingLevel Device
Audio sharingLevel DetectOnly
```

Figure 5.14 Privacy Policy Sharing Level Example.

5.3.1 Automatic Generation

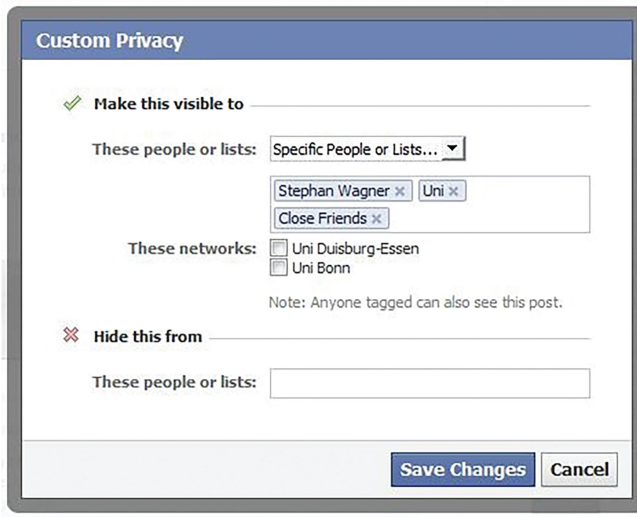
The privacy preservation policy that is used by the PRF to constrain the access to data gathered by the DQF can be created automatically, by the policy generator. This enables the user to use GAMBAS applications without an extensive (manual) configuration phase, while still having a privacy policy that protects private data. The policy generator is therefore one of the key concepts to enable automation in the privacy preservation framework.

Many users use social networks or collaboration tools like Google Calendar as part of their everyday routine. They post messages to colleagues and friends, share photos and create shared appointments. Often, it is possible to constrain access to messages to a pre-defined user group. This could be a list of friends on Facebook or individual users for a shared event in Google Calendar. Similar, the access to other data in the social networks or collaboration tools can be constrained by the user. Figure 5.15(a) depicts an example. Using the APIs that are provided by the social networks or collaboration tools, these privacy settings can be retrieved automatically. An example using Facebook's graph API is shown in Figure 5.15(b).

A user that is using such a social network or collaboration tool is therefore already creating one or more privacy policies (depending on the number of tools that are used). The privacy policy generator can query these policies using the tools' APIs. Since the policy generator operates on the user's own device(s) and uses the user's accounts to access the tools, a policy which is individual for each user can be generated. This generated policy is then also tailored to the needs of the user, because it is only an import of a user-defined policy into the privacy framework. To support as many social networks or collaboration tools as possible, the policy generator has a modular structure. This structure makes the policy generator extensible with regard to other collaboration tools.

Usually, the user is able to define privacy settings in each social network or collaboration tool individually. Because the user may edit this settings freely and independent from each other, the settings might be inconsistent. The policy generator is then not able to create a consistent policy. If this is the case, the generator can detect and display the conflicting settings and suggest possible solutions to the user. After the conflicts are (manually) resolved, the policy generator creates a consistent policy.

Using the *consistsOf*-relation, the policy generator proposes different generalization strategies to apply the policy to a broad set of data types that can be acquired by the DQF. This enables the generalization of the policy,



(a)


```
[{"object_id":105482406163077,"id":105482406163077,
"value":"CUSTOM","description":"Uni","allow":"101909386520379",
"deny":null,"owner_id":100001039541728,"networks":null,
"friends":"SOME_FRIENDS"}]
```


(b)

Figure 5.15 Privacy Settings in Facebook. (a) User Interface and (b) Programming Interface (JSON).

which includes new data types that might be related to data types retrieved from the privacy settings in social networks or collaboration tools.

In summary, the policy that is used to allow access to different data types is generated automatically using the policy generator. The generator is designed to pick up policies or privacy settings that are pre-specified by the user in a social network or collaboration tool and to create a policy that is compatible to the GAMBAS policy format. The generator includes tools to resolve conflicting settings and is able to generalize data types. The generated policy can also be fine-tuned by the user using the user interface presented in the next sub-section. Even without the fine-tuning, the generated privacy policy is consistent and tailored to the user's needs, without putting the user's privacy at risk.

5.3.2 Manual Fine-Tuning

The user interface developed as part of the middleware enables the user to fine-tune the privacy policy. In general, the privacy policy is created automatically using the policy generator. The automated creation takes into account the settings of the user in social networks and other collaboration tools, like the Google Calendar. Although this automatically derived policy is therefore created on an individual basis, a user may want to modify the policy. To do this, the privacy preservation framework encompasses methods that allow retrieving the current policy and methods that can modify the existing policy.

Using the user interface, the user can change the policy triples visually, without having to use the policy language. This allows also non-expert users to edit the policy successfully. The user interface displays the data types and then shows the relevant policy relations graphically. The user can modify the data types by clicking on them and, for example, choose users from a list of users for the *Permission*-relations. Editing the other relations is similar. Any change in the graphical user interface results in a change of the policy, i.e. causes the addition, deletion or modification of policy triples. The user might also use the interface to export or import the privacy policy, which enables expert users to modify the RDF representation of the policy directly.

5.4 Privacy Integration

To clarify the mechanisms and protocols of the privacy framework, we describe how they are integrated into data transfer, data acquisition and data processing defined in the previous chapters.

5.4.1 Data Transfer

To support data exchange and possibly the exchange of context information, data will be transferred between different devices. This data – for example, a user asking the servers of a public transit network operator for the route of a bus trip – can breach privacy. In this case, an eavesdropper could get the current and future location of the user. Therefore, the data should be transferred securely. In GAMBAS, the Privacy Preservation Framework (PRF) is responsible for all security and privacy needs and therefore is also responsible for securing the data transfer. For this, all data that is transferred should be encrypted. The reason for this is twofold. Firstly, the data might

contain private information that should not be shared with unauthorized users or devices. Secondly, the shared data might be transferred over an insecure communication channel (e.g. the Internet or an insecure WiFi network).

To apply the efficient concept of symmetric encryption (AES) to secure communication, a shared key must be exchanged before any encrypted communication can take place. During the exchange of a cryptographic key, the communication endpoints show that they are eligible to access the data that should be transferred by authorizing themselves. After the authorization process, both endpoints possess a shared cryptographic key that allows them to transfer data securely.

In the GAMBAS PRF, authorization can be performed in two different ways. The first way uses asymmetric cryptography and is based on certificates, similar to the implementation of SSL in the Internet. This allows an ad-hoc identification of devices that belong to a certain domain. If the domain root is trusted, the authorization will be successful. Also, the access rights may depend on the trust in this root. For authentication, the device's certificate is transferred together with a challenge that proves that the device is in possession of the certificate's private key. Together this data forms the device's credentials that are checked at the other endpoint. The alternative of using compute intense asymmetric cryptography is symmetric cryptography. Using symmetric cryptography, a key (256 Bit) can be attached to a connection between two endpoints. The first half (128 Bit) of this shared key allows the identification of the other endpoint. The other half (128 Bit) can either directly used for the secure communication or can be used to exchange a new session key securely. For efficiency reasons, both of these checks (i.e. for asymmetric and symmetric cryptography) are performed transparently by the communication system of the GAMBAS middleware.

The secure data transfer is generally foreseen for every transmission of data. The communication endpoints must first authorize each other at the remote privacy preservation framework, before a key for the secure communication is computed. The authorization that is performed by the privacy-preserving framework incurs some overhead during the data transfer. However, without the authorization, the communication partner is unknown to another device and this contradicts the privacy of the transferred data. Therefore, while the authorization is a crucial mechanism, it is possible to use more lightweight security mechanisms, but this would result in a decrease of the security level.

To enable encrypted data transfer, it is necessary for both communication endpoints to use a cryptographic key. In GAMBAS, we support devices

with different capabilities with regard to the available resources (like RAM, CPU and battery power). The encryption is therefore based on a hybrid scheme that allows an efficient and secure encryption. With respect to data processing, we can differentiate between three different cases. The first case is the communication between a user's device (a client) and a server (e.g. the server is asked for a bus route by a user's device). The second case is the communication between devices of two users. An example could be two friends who want to exchange photos with each other. The third case is the communication between two servers. An example could be server of a public transit network operator who communicates with a weather service server to get the current forecast (which might influence the bus planning for the day).

The difference between a user's device and a server in GAMBAS is that the server is able to authenticate itself using a cryptographic certificate. This ensures the identity of the server and allows for server authentication, before the connection is established. In contrast, the user's device does not have a certificate since it is not bound to a certificate domain. Therefore, different methods of authentication must be used, if a user needs to be authenticated and/or identified. Because of these differences, the three cases mainly differ in the authentication phase.

5.4.1.1 Client and Server Communication

When a device (client) contacts a service that is provided by a server, the connection will be established as shown in Figure 5.16. The server will use its certificate to authenticate itself against the client device. The client application validates the certificate of the server against either a pre-deployed service certificate or a pre-deployed certificate root. The root certificate can be used for companies that support different kinds of services and eases the deployment without lowering the provided security. This is a one-sided authentication, i.e. the server authenticates itself against the client, but the client is not authenticated. If it is necessary to authenticate the client, the server may add any authentication scheme after the secure connection is established. A username/password authentication could, for example, use the secure remote password protocol (SRP 6) to authenticate the clients.

Using the (unencrypted) authentication messages, an elliptic curve Diffie–Hellman (ECDH) key exchange is performed. As a result, both sides will be able to compute a secret key that cannot be computed by eavesdroppers. The signature of the message that is created by the server also prevents man-in-the-middle attacks. After the authentication handshake and the key exchange, the interaction is encrypted by using the exchanged secret key.

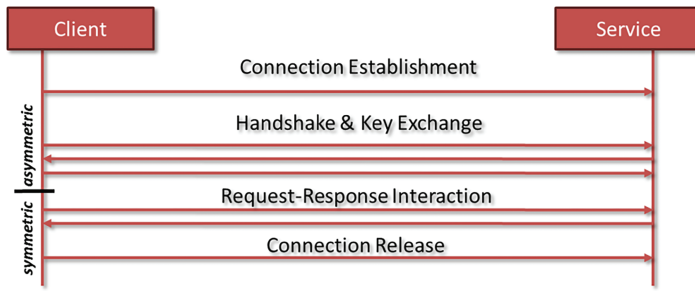


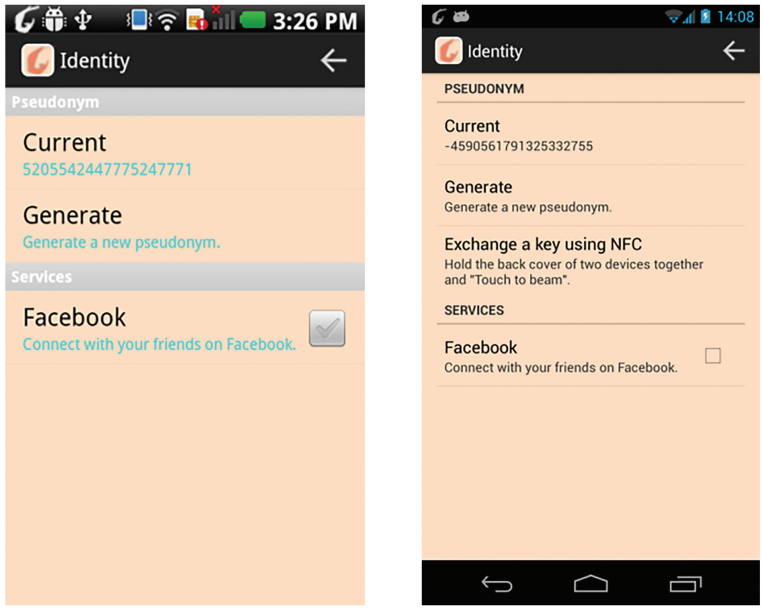
Figure 5.16 Client and Server Communication.

5.4.1.2 Device-to-Device Communication

When two devices establish a connection, usually they cannot authenticate each other. The authentication cannot rely on certificates, since the devices do not know each other and normally do not possess a certificate. The users might identify their devices manually, by device id, and may then create a manual key on both of them. But GAMBAS also allows two other easier methods to establish a key for each device.

The first method uses PIKE to exchange a key using an online social network such as Facebook that is used by both users. This key is exchanged before the interaction takes place and allows two or more devices to interact securely with each other. The user identification is extracted from the relationship in the online social network and can then be used at the time later on. This allows for a completely automatic key exchange that does not need any user interaction. The only necessary step a user has to take is to connect GAMBAS with the social network, e.g. through Facebook Connect as shown in Figure 5.17(a).

The second method uses the NFC technology. Nowadays, many smartphones are equipped with an NFC reader system that may also be used for short-range one-way communication. The GAMBAS middleware implementation for Android integrates with NFC to exchange a key. For this, the middleware encompasses a service that is used to redirect the communication back to the device that initiated the one-way NFC communication. This backward channel is necessary to transfer the device id of the device that received the NFC message. Using this service, we can establish a key just by holding two devices together. For this, a user simply needs to press a button in the user interface shown in Figure 5.17(b) and then bring two phones in physical proximity to each other.



(a) (b)

Figure 5.17 Device to Device Authentication. (a) PIKE-based Key Exchange via Facebook and (b) Manual Key Exchange via NFC.

After the successful key exchange, the communication between the two devices can be encrypted. Additionally, the devices can identify themselves using the already established keys.

5.4.1.3 Server-to-Server Communication

The server to server communication in GAMBAS is similar to other communication on the Internet. Each server authenticates to the other server using its own, pre-deployed certificates. Similar to the client–server communication, the certificates can be verified by using the server certificates or by using a common root certificate. Again, the authentication includes an ECDH key exchange, which results in a shared key for this connection. Then, the communication between the two servers will be encrypted using this key.

5.4.2 Data Acquisition

The Adaptive Data Acquisition Framework (DQF) enables the collection of data using various sensors built into the user's mobile device. The collected

data can then be used personally (i.e. by the device, in the case of personal data acquisition) or collaboratively (i.e. by a remote service, in the case of collaborative data acquisition) to optimize services based on the users' behavior. Clearly, the data acquired by means of sensors built into the device of a user may raise privacy concerns. Furthermore, the preferences with respect to privacy may vary drastically from user to user. In order to empower users to exercise control over which data can be collected, the access to the data acquisition framework is guarded by the Privacy Preservation Framework (PRF). Thereby, all accesses made to the data acquisition framework are checked against the user's privacy preferences with respect to data collection. This allows the user to limit the data types that can be collected at all. In extreme cases, a user may limit the collection of all data through the GAMBAS middleware. In less extreme cases, the user may limit the collection of a particular type of context information, such as location-related information or audio information.

The PRF-DQF interface enables the data acquisition framework to check whether the user has given consent to the acquisition of a particular type of contextual information. To do this, the DQF performs calls to the PRF in order to verify that the data types that shall be captured are permissible under the user's current preferences. Furthermore, since the user's preferences may change at any point in time, it is necessary that the PRF provides functionality to signal a change to the DQF whenever the user's preferences with respect to a particular data type change.

The PRF therefore has two different duties. First, it checks the data type that is about to be captured against the preferences of the user and returns a Boolean to indicate whether the user permits the acquisition of the specified data type. If the access is denied, the acquisition is aborted. If access is granted, the acquisition task can be started. Additionally, a user could modify his privacy settings. Therefore, the PRF needs to signal a change to the preferences with respect to a particular data type such that the DQF can check all currently executed data acquisition tasks against the updated set of preferences. If a data acquisition task is no longer permitted by the user, it must be aborted by the DQF.

In order to guarantee that all data acquisition tasks continuously conform to the user's preferences, the GAMBAS middleware implements the continuous and gapless usage of this interface for all calls to the DQF. This means that all tasks that are started within the DQF need to pass through the check method of the PRF with the associated data types. In addition, as long as

the DQF is executing any tasks, it needs to react to changes indicated by the signal method. If a signaled change affects a data type that is currently acquired, the check for the associated (set of) task(s) needs to be reevaluated, possibly aborting any conflicting tasks.

As every export of user's context information is filtered based on the privacy policy of the user, for every request of data by the service provider, the DQF checks it with the privacy framework. If the PRF allows the data to be sent, only then the users' context information is exported. The PRF and DQF communicate this information check through control interfaces provided by the PRF. Specifically, PRF provides different methods that allow the DQF to check if a certain data type is allowed to be detected. The DQF must call these methods before creating a recognition stack for detecting any kind of data or context. Based on the results from these methods, the DQF detects the context data and subsequently sends it to the service providers.

In order to allow acquisition and subsequent export of user's context information, the GAMBAS middleware ensures that the context recognition applications can gather and export only the allowed context features. In order to achieve this, the data DQF checks for permissions with the privacy-preserving framework whenever a new application is started.

When the data acquisition framework starts to acquire data, it analyzes the feature requirements of the application and then checks with the PRF whether the desired features are allowed to be gathered. The PRF will decide, based on the privacy policy that is set by the user and will inform the acquisition framework whether the requested features are allowed to be gathered or not. If the requested features are allowed, then the DQF starts gathering context information.

When the PRF refreshes the privacy policy (either through a user that edits the policy or through an update issued by the Privacy Policy Generator), the GAMBAS core service indicates this change to the DQF, which again checks the permissions with the privacy framework. If an already running application does not adhere to the new privacy policy, then the application is shut down immediately.

The list of privacy features that a user can edit in the privacy policy includes features related to the acquisition of sensing data such as audio sensing, location sensing, motion sensing, ambient sensing and features related to the communication such as enabling of remote gateway communication, enabling of Wi-Fi and Bluetooth as communication technologies, etc.

5.4.3 Data Processing

Dynamic and distributed data processing is an essential part of the GAMBAS middleware. Data processing in GAMBAS is performed by the Query Processors (xQP), which provides GAMBAS applications with the necessary data. Often, the processor executes remote queries. These queries are executed at remote devices and may try to access private data. The GAMBAS Privacy Preservation Framework therefore has to check the access to the requested data types and allow/deny access based on the policy of the remote user.

During the query execution, the query processor identifies the sources needed to answer the query and then sends a request to the registry. The registry resolves the sources and sends back to the processor the list of endpoints (remote storages) that contain needed data. For shared data, however, before the query processor can access the data on the remote source, a privacy control is performed to check if the query initiator has the rights to access the data. A view of the data matching the privacy rules in place is created and shared with the query processor. The query processor forwards the identity and data requirements to the privacy framework, which in turn checks with the privacy framework of the remote device hosting the shared data. A view of the data is created based on the access control. The view can reflect the original data, or it can modify the original data according to the privacy in place. For example, it can aggregate or hide parts of the original data, like changing GPS coordinates to the name of the city or country.

If a one-time query is issued, the access is granted based on the privileges of the user that is trying to access the data. The query can then be directly executed and will be transferred over a secure connection. If a continuous query is issued, a secure access token is generated and sent to the query processor. If a remote endpoint is trying to access the shared data, the secure access token will allow transferring the shared data securely over the chosen communication channel.

The interface between the xQP and the PRF checks whether the query initiator is allowed to access the data. Additionally, if the xQP is executing a remote query, the communication must be properly secured. The user and data access credentials are sent over a secure data connection between the two endpoints. Since the middleware manages the secure communication transparently, the interface does not include a method that enables the exchange of security tokens or start the encryption. Instead, this is done through the authentication and key exchange plug-ins that the PRF integrated into the GAMBAS middleware.

The access to data by the query processor must be checked through an interface at the PRF. The interface consists of one function that checks if the query initiator (i.e. the user requesting the data) is allowed to access the data. The data types that are being requested also need to be specified. The PRF queries the privacy policy of the device using the specified input and decides whether the query is allowed or not. Each request is handled by the privacy framework of each semantic data storage; therefore, this function is performed locally.

The PRF therefore has a local PrivacyManager that implements and interface that can be used by the xQP to check with the PRF if executing a received query is allowed according to the currently active privacy policies. To do so, the query processor hands the PRF (1) a set of classes in the GAMBAS ontology that specify what data types the query will access and (2) the origin of the query, e.g. if it was a local query or a query from a remote user. The PRF then returns whether this query is allowed or not. The PRF needs to be contacted for every query execution, when shared data is involved. The query processor must first interact with the privacy framework, which is responsible for allowing or denying data access, for data encryption/decryption and for device authentication.

Of course, the privacy-preserving framework incurs some overhead in the query processing, specifically an additional method call, device authentication and data encryption. However, the PRF is crucial to maintain the privacy of the users' data. To minimize the performance impact, the PRF uses lightweight privacy rules and lightweight encryption mechanisms (e.g. symmetric encryption using AES) to allow a secure and privacy preserving execution of queries by the xQP. More lightweight encryption mechanisms could be applied, but this would result in a decrease of the privacy and security level without a high speed-up compared to the used security mechanisms, if measured on current smartphones.