Manuel Schallar, BSc

Enabling Digital Signatures on Smartphones

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Dipl.-Ing. Dr.techn. Peter Teufl
Dipl.-Ing. BSc Florian Reimair

Institute for Applied Information Processing and Communications (IAIK)

Graz, May 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Contents

List of Figures

Abstract

Over the last years, smartphones have become essential devices to access online services, whereby those services often require user authentication. In Austria, services of this kind are often provided by public authorities to interact with citizens while digital signatures are used for authentication. By relying on digital signatures and thus eliminating the need to appear in-person, citizen requests can be processed more efficiently. Unfortunately, some smartphones are not equipped with suitable secure storage for cryptographic keys needed for digital signatures.

Due to the heterogeneous device landscape regarding hardware- and software combinations, there is no standard way to securely store cryptographic keys on smartphones and protect those cryptographic keys from unauthorized access. As a solution, cryptographic keys may reside at a secure service, whereby the keys are not extractable onto smartphones. Instead, smartphones will be used to authorize access to the cryptographic keys, e.g., digital signatures keys.

In this theses, we enable smartphones to create digital signatures. To overcome the limitations of smartphones, cryptographic operations are delegated to a remote service, the ServerBKU [33]. The ServerBKU is a secure and flexible server-based eID and e-signature solution developed at IAIK[1]. This remote service securely protects the cryptographic keys used for signature creation. Communication with the ServerBKU is realized using the CrySIL[34] protocol. Latter can be used for accessing cryptographic keys and use them for signature creation in consequence.

For our purpose, the CrySIL protocol does not support all requirements and therefore, needs modifications.

To establish secure communication between two parties, e.g., the ServerBKU and the smartphone, we had to develop a CrySIL compatible end-to-end encryption. Furthermore, the CrySIL protocol does not support the discovery of new nodes (i.e., services, endpoints) within the CrySIL network. We had to introduce a suitable but simple discovery mechanism for the CrySIL protocol. Also, the CrySIL protocol does not support routing within the CrySIL network, as the CrySIL protocol is for academic purposes. This disadvantage had to be overcome to talk with multiple endpoints, e.g., the ServerBKU and the public authority. Finally, we evaluated our system and created a security analysis based on Common Criteria for Information Technology Security Evaluation [5].

[1] http://www.iaik.tugraz.at

Kurzfassung

Im Laufe der letzten Jahre wurden mobile Endgeräte (z.B.: Smartphones) essenzielle Bestandteile der Online Kommunikation. Beansprucht z.B. in Österreich ein Bürger den Online-Dienst einer Behörde, wird statt einer herkömmlichen Benutzername- und Passwort Authentifizierung eine digitale Unterschrift auf einem Dokument verlangt. Durch diese Unterschrift ist es der Behörde möglich, die Identität des Bürgers auf automatisiertem Wege festzustellen und in weiterer Folge das Dokument großteils automatisiert zu verarbeiten. Die digitale Unterschrift bringt dem Bürger und der Behörde den großen Vorteil, dass kein persönliches Erscheinen auf der Behörde mehr von Nöten ist. Daraus resultierend entstehen für beide Beteiligten auch weniger Kosten.

Die für eine digtale Signatur benötigten kryptografischen Schlüssel müssen in einem sicheren Speicher auf den Smartphones abgelegt werden. Nicht jedes Smartphone unterstützt jedoch diese Anforderungen.

Aufgrund der unterschiedlichen Hard- und Softwarekombinationen der Smartphones, gibt es kein einheitliches Konzept für die sichere Speicherung und die Authorisierung der kryptografischen Schlüssel. Diese müssten auf einen sicheren Server ausgelagert werden und dürfen den Server nie verlassen. Daraus resultierend müssen die Smartphones lediglich die Authorisierung zu den Schlüsseln verwalten.

In meiner Arbeit wird ein Konzept vorgestellt, mit dem Smartphones digitale Signaturen erstellen können. Um das oben genannte Problem des sicheren Speichers zu lösen, delegieren wir die Operation der kryptografischen Signatur auf einen Server, die ServerBKU. Diese ist ein sicherer und flexibler Identitäts- und digitaler Signatur-Server und wurde vom IAIK entwickelt. Der Signaturserver schützt die kryptografische Schlüssel für die Erstellung einer digitalen Signatur. Die Kommunikation mit der ServerBKU, die Authorisierung der kryptografischen Schlüssel und die Erstellung der digitalen Signatur erfolgen mit dem CrySIL Protokoll.

Im CrySIL Protokoll gibt es keine Möglichkeit, eine verschlüsselte Ende-zu-Ende Verbindung wie z.B.: zwischen dem Smartphone und der ServerBKU aufzubauen. Aus diesem Grund wurde eine auf dem CrySIL Protokoll aufbauende Ende-zu-Ende Lösung entwickelt.

Weiters ist es mit dem CrySIL Protokoll nicht möglich neue Dienste (z.B.: die ServerBKU, eine Behörde) im CrySIL Netzwerk zu erkennen, oder Daten im Netzwerk zu routen. Für diese Herausforderungen wurde ein einfacher Algorithmus entwickelt.

Zum Schluss haben wir unser System evaluiert und einer Sicherheitsanalyse, ähnlich dem *Common Criteria for Information Technology,* unterzogen.

1 Introduction

Qualified digital signatures provide a legal basis for transactions in online processes and government interactions [29]. Online interactions speed up communication between citizens and public authorities and help reduce costs, as the citizen does not need to appear in-person. Therefore it is reasonable to assume that it should be easy for citizens to create qualified digital signatures, with their computer at home but also while traveling or at the office. In Austria, a digital signature concept referred to as the Austrian Citizen Card was established, leading to several implementations following this concept in consequence. One of those is a mobile phone signature called "Handy Signatur". It uses a server-side secure signature creation device (SSCD). However, to use the Handy Signatur as a service on smartphones, the service has to be integrated into existing mobile applications. The integration is quite difficult might sometimes not be possible.

Furthermore, encrypting arbitrary data enables smartphone applications to keep sensitive data private. Currently, there are some heterogeneous concepts to encrypt data on smartphones. Some concepts encrypt the smartphone [2], and some concepts encrypt data in separate containers [13]. Such concepts require cryptographic encryption keys to encrypt data. As the data is only as secure as the encryption key, it is important that keys are stored in a secure environment.

Due to heterogeneous device landscapes on smartphones, it would be a great approach to store sensitive cryptographic key material remotely, and that those keys are not extractable onto smartphones. The remote location must operate in a secure environment and must be accessible from every device we work. It does not matter what kind of service the remote environment is as long as keys are stored securely, and the environment is accessible whenever needed.

Due to smartphone limitations, cryptographic operations are delegated to a remote service, the ServerBKU [33]. The ServerBKU is a secure and flexible server-based eID and e-signature solution developed at IAIK.

Two main challenges arise when storing keys at a remote environment, and using a remote service for cryptographic operations.

The first challenge is to manage the authorization of cryptographic keys [28]. Well defined cryptographic key policies must be defined and implemented to enforce proper cryptographic key authorization, i.e., to allow the usage of those keys.

The second challenge is the cryptographic key access. Although cryptographic keys are stored centralized, a well-defined API must exist to allow (user-) applications to access those cryptographic keys. Such an API concept might include a delegating authorization application for smartphones. This spares user applications of prompting users for cryptographic key credentials and separates the authorization from the use of cryptographic keys. [20]

The Cryptographic Service Interoperability Layer (CrySIL) [35] tries to solve these two cloud storage challenges mentioned above.

Unfortunately, the CrySIL protocol misses some features necessary for using a remote service for cryptographic operations.

First, CrySIL does not support end-to-end encryption between two parties, important for communicating sensitive information. As part of this thesis, support for end-to-end encryption was deployed.

Second, CrySIL does not support the discovery of new nodes within a CrySIL network. Nodes can be (signature-) services or endpoints. This limitation stops new nodes from being discovered in an already existing CrySIL network. In this thesis, we will introduce a suitable but simple discovery mechanism for the CrySIL protocol.

Third, as the CrySIL protocol does not support routing, communication was impossible within a CrySIL network if the network consisted of more than two CrySIL nodes. To overcome this, we propose a simple routing algorithm in this thesis.

1.1 Contribution

There are several disadvantages to the current status quo of the CrySIL protocol.

The first is that the CrySIL protocol does not support end-to-end encryption. Currently, data is transported in plain, which makes it easy for an eavesdropper to gather credentials and metadata information. We will target this challenge by using an approach similar to IPSec's ESP Tunnel mode [7]. After integration, a similar version of that approach, it will be impossible for an eavesdropper to gather any information from the transported data.

The second disadvantage is that currently there are no signature services available for CrySIL although signatures get more popular everyday[1]. They simplify and speed up the communication between citizens and public authorities in Austria by using the popular website *help.gv.at*[2]. The website provides information on all interactions

[1] http://www.a-trust.at/handystat/
[2] https://www.help.gv.at

between Austrian citizens and Austrian authorities and permits electronic processing of some of these procedures. The website offers a wide range of forms, e.g., applying for citizenship, applying for a visa, a conversion of a foreign driving license. *help.gv.at* records 1.749.386 users on their website doing government interactions in January 2016 [15], which is ninety times higher than in January 1999. It is obvious that signatures are almost a necessity when interacting with the Austrian government, and therefore, an approach is needed to create signatures with CrySIL.

To approach the missing signature service challenge, we will use a signature service called ServerBKU [33], a secure and flexible server-based eID and e-signature solution developed at IAIK. It enables users to create signatures similar to the Austrian mobile phone signature. The ServerBKU is capable of generating XML and CMS signatures based on a digital certificate. Latter enables users to use the ServerBKU to sign documents, which may then be accepted by public authorities.

Another disadvantage of CrySIL is that it is not able to detect (new) services within a CrySIL network. This prevents detection and routing to the services needed. We approached this challenge by developing a detection- and routing algorithm.

1.2 Outline

The thesis is structured as follows. In Chapter 2, we will provide related information on already existing similar solutions and point out their advantages and disadvantages. The Chapter 3 provides background information for the reader regarding authentication schemes, digital signatures, the Austrian citizen card concept, the ServerBKU as well as additional information about CrySIL. After the basics are explained, we will focus on our contribution in Chapter 4. Following that chapter, our security analysis will be explained in Chapter 5. We will finish the thesis with future work in Chapter 6 and a conclusion in Chapter 7.

2 Related Work

This section provides an overview of our results and compares them to already existing approaches. We will discuss how our end-to-end encryption (Chapter 4.6) compares to existing solutions, and how our server-based signing service (Chapter 3.3) compares to existing server-based signing services. Both are a necessity for our contribution in Chapter 4.

2.1 End-to-End Encryption

One goal of our approach is to provide end-to-end encryption. With the status quo of the CrySIL protocol, data exchange happens in unencrypted. Data exchanged in unencrypted are an easy target for eavesdroppers. To prevent eavesdroppers from gathering our exchanged data unencrypted, we will need end-to-end encryption.

Internet Protocol Security (IPSec) [8] offers two protocol suites, namely Authentication Header (AH) [6] and Encapsulating Security Payload (ESP) [7]. The Authentication Header protocol suite does neither provide encryption nor integrity protection and does, therefore not satisfy our need. The Encapsulating Security Payload protocol suite offers confidentiality, integrity, and authenticity of data and is therefore suitable for our requirements. As IPSec operates on the Internet Protocol Layer, any exchanged data consists of an IP header and IP payload. Both, the IP header and the IP payload, may be sensitive data worth to protect.

ESP supports two different modes:

1. Transport mode encrypts and authenticates only the payload of the Internet Protocol (IP) packet. The IP header is left untouched since it is necessary for routing, as seen in Figure 2.1. The transport mode cannot be applied to our approach, as we transport sensitive information within the headers. We need an approach which protects the confidentiality, integrity, and authenticity of header and payload.

2. In tunnel mode, the header and the payload of the IP packet are encrypted and authenticated. Tunnel mode puts the origin IP packet (which needs to be protected) into an entirely newly created IP packet. The new IP packet gets

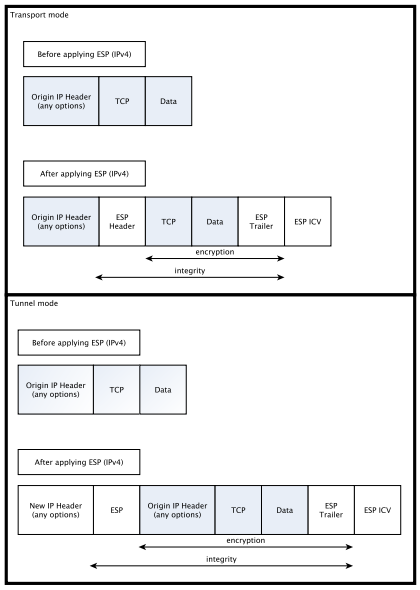

Figure 2.1: IPSec transport and mode with IPv4

a new IP header, with only public data from the origin IP header. This way, the origin IP headers, as well as the payload, are protected, as seen in Figure 2.1. This mode provides confidentiality, integrity, and authenticity to packets.

The fact that IPSec in ESP tunnel mode protects both the header and payload with confidentiality, integrity, and authenticity fits perfectly for our approach. Therefore the idea of the ESP in tunnel mode with encryption and authentication was used for our approach *SecureTransport* (see Chapter 4.6.2).

It is worth mentioning that ESP also can be used in authentication-only or encryption-only configuration, but that is discouraged [18, 31] and therefore does not fit our needs.

2.2 Signing services

In this section, we will take a look at different signing services and remote key storage services ([35]), as another goal of our approach, is to have documents signed by a signature service (see Section 4.5) with a remote signature key.

AWS CloudHSM[1], as an example, offers secure key storage by giving users access to dedicated HSM capabilities in the cloud whereby users retain full control of their

[1]https://aws.amazon.com/cloudhsm/

keys and cryptographic operations in HSMs. Amazon only manages and maintains the hardware without having access to users keys. Amazon offers reliable and durable key storage, secure connectivity, and improved application performance. Furthermore, AWS CloudHSM is integrated into Amazon Redshift, Amazon Relational Database Service (RDS), Oracle and third-party applications such as SafeNet ProtectV volume encryption for EBS, Apache, and Microsoft SQL Server. APIs for PKCS#11, Java, Microsoft CAPI and CNG are available as well. Unfortunately, the commitment binds you to mostly Amazon services. Migration to another HSM provider may not be feasible.

Signinghub[2] offers only the creation of advanced electronic signatures for most common browsers, operating systems and mobile operating systems. The convenient RESTful API for applications is only available to customers with an enterprise service plan.

In Austria, a concept called *BKU (BKU means Bürgerkartenumgebung)* exists. There are different implementations of BKUs. The "BKU Local" implementation forces users to have the software[3] installed on their computer to use the smart card based qualified digital signature. The usability of that use case is low because software must be installed on every computer users want to create such signatures. That is time-consuming initially, and afterward as well due to updates, but also a hustle due to the different system configurations and operating systems.

BKU Online was an alternative to *BKU Local*. *BKU Online* uses, as the name implies, an online version of BKU, which was implemented as a Java applet. The applet was able to access the local smart card plugged in the user's computer for signatures. Unfortunately more and more browser vendors disable Java by default in their product due to vulnerabilities in Java applets. After the support for Java applets is removed from browsers, *BKU Online* will practically not be usable anymore.

Therefore another solution was needed. A solution that does neither rely on certain browser plugins nor special software installed on a computer. The mobile implementation of a BKU, the *Handy Signatur*[4] was created. No special software is needed for using the Handy Signatur; only a standard browser is needed. No special hardware to buy or install is needed anymore. Instead, users mobile phone are used and used as tokens. These tokens have already been rolled out because practically everyone who wants to sign data digitally owns a mobile phone. The "Handy Signatur" service is free of charge, and it uses digital signatures based on qualified certificates, which in Austria, are legally equivalent to handwritten signatures.

[2]https://www.signinghub.com/
[3]https://www.buergerkarte.at/downloads-karte.html
[4]https://www.handy-signatur.at/

3 Background

In this chapter, we will explain the underlying basics used by our approach one by one. First, we will talk about authentication schemes in general and the two-factor authentication concept. Second, we will explain what qualified digital signatures are and how they relate to the Austrian Citizen Card concept. After explaining the Austrian Citizen Card concept, we will describe the ServerBKU, its architecture, components, and usage. We will conclude this chapter by explaining the basic concept of CrySIL.

3.1 Authentication Schemes

Authentication schemes define how an entity (e.g., a user) authenticates against another entity (e.g., a service). We will discuss three authentication schemes, namely knowledge-, possession- and inherent based authentication schemes. If a service uses just one of these three discussed schemes, the type of authentication is called a single factor authentication (SFA). If a service uses two of these three schemes, it is called a two-factor authentication (2FA) or multi-factor authentication (MFA). In this chapter, we are going to explain how each scheme works and what advantages 2FA has over SFA.

3.1.1 Knowledge-Based Authentication Scheme

Knowledge-based authentication is an authentication scheme that requires users to attest their knowledge against a service. This *knowledge* factor could be for example a username and password combination. The following section focuses on a username and password combination as knowledge factor - one of several possibilities, whereby it is byfar the most frequently used option.

Advantages The knowledge-based authentication scheme only needs *knowledge* from an entity and does not depend on any *token* (e.g., smartphone, security token, ..). Because of that, the scheme is platform independent and has zero footprint.

This authentication scheme is widely used on computers (office, home) and web services.

The knowledge based authentication scheme has a superb usability for users as they only have to remember their knowledge factor. This scheme is great for services because it can easily be handled as no special hardware or software is needed to validate the provided knowledge factor. Furthermore because this scheme is widely known and used the user acceptance is excellent.

Disadvantages A disadvantage of the knowledge-based authentication scheme is that the knowledge factor chosen by users must be remembered. Otherwise users will not be able to authenticate themselves against a service. Therefore users tend to choose passwords which are easy to guess [19]. Malicious entities may use this drawback to guess passwords and impersonate users. Accounts have been hacked due to that fact that passwords were easy to guess.

Another disadvantage is that users reuse their passwords for different services. This behavior makes attacker's acquired assets more valuable, as people re-use their passwords on different services regularly [32]. Services use different approaches to store passwords in their databases. If passwords are stored in an unsafe manner, it could be easy for an attacker to retrieve the passwords, once the database gets compromised. If this happens to a user's password, an attacker could try to re-use that same password on other services as well.

Even if users did not re-use passwords, an attacker could still be successful if the attacker abuses the "password recovery" feature. Some services offer such a feature, in case users forget their current password. The feature uses questions and answer tuples to confirm user identity. If correct answer are given, the service enables users to change the password without knowing the current password. Sometimes the questions, which need answers to reset passwords, are easy to answer, e.g., "What was the name of your first pet?", "What is your mother's maiden name?". The disadvantage of a password recovery feature is that users get tempted to choose question/answer tuples, which may also be known to other people, except themselves.

Another disadvantage is the exchanged data when an authentication attempt happens over an insecure channel. If the freshness of the authentication attempt is missing, the attempt is vulnerable against a replay attack.

Conclusion The knowledge-based authentication scheme is well established. To make an authentication attempt the knowledge factor (i.e., username and password combination) has to be remembered. This makes the authentication scheme easy to use from a user perspective and hence provides high usability. However, that simplicity is also the main disadvantage. Easy to guess passwords, password reuse and the possibility of a replay attack create an attack surface for a malicious entity.

3.1.2 Possession Based Authentication Scheme

Possession based authentication schemes are an alternative to knowledge based authentication. These possession based authentication schemes imply that users *have* a token in their possession other users do not have e.g., a personalized smart card or security token. Tokens must be kept safe, as with the passwords in the knowledge based authentication scheme. For the following section, we will focus on smart cards as *token*s. Smartphones are also being used as a token, as smartphones are able to receive TANs via SMS, or use vendor specific, and service account-paired applications.

Advantages Possession based authentication has been used for a long time, e.g., the key to your front door at home. In computer science, this key would be called (security-) token or smart card. These tokens are usually nearby. An example would be a key card in your wallet or a mobile phone in your pocket. As this authentication scheme has been around for a while, it has become more familiar and accepted.

Disadvantages Unfortunately, smart cards have many disadvantages [36]. A disadvantage is that in order to use a smart card, users have to obtain smart cards. This usually happens through the vendor of the smart card and the cards may be expensive. The decision on a certain smart card is especially for companies important, as such smart cards have to be rolled out to all their employees.

Another disadvantage is that card readers are required to communicate with the smart cards. These card readers do cost money as well. In comparison, a smart card fits in your wallet and is nearby. But a smart card reader normally does not.

The next disadvantage is that a smart card reader driver and the associated software have to be installed and configured on the computer. That is not always easy and a hustle for inexperienced users.

Another disadvantage is if smart cards should be deployed to many people i.e., within the same company. The roll out may be complicated, as every employee has to get his personalized smart card.

Similar to the knowledge based authentication, an attacker could try to steal the token and use it for their advantage.

Conclusion The possession-based authentication scheme needs a suitable token to achieve high user acceptance and an inexpensive token roll out. Therefore a token with high usability and an already rolled out token should be considered. Some

services use mobile phones as tokens. Lots of people have mobile phones, most of the time they are nearby. As mobile phones are used on a daily basis the usability of mobile phones is high. In Section 3.1.4 we will discuss how mobile phones are used in combination with another authentication scheme to achieve a 2FA.

3.1.3 Inherent Based Authentication Scheme

Inherent-based authentication happens based on what users *are*. Characteristics of users are, e.g., fingerprints, iris, voice recognition or the user's face itself. These biometric features are unique and make authentication a bit more convenient.

Advantages In comparison to a knowledge-based authentication scheme a biometric feature cannot be forgotten. Furthermore, in comparison to a possession-based authentication scheme, a biometric feature cannot be lost or stolen, and we "carry" it always with us.

Disadvantages Although these biometric features can not be lost, they can be impaired e.g., by accident. In that case, these features can not be used for authentication anymore. For such cases a fall-back method should be configured.

Nowadays smart phone manufacturers implement fingerprint readers in their phones [11] (e.g., Huawei GX8) and some manufacturers implement face recognition (e.g., Microsoft Lumia 950 and Microsoft Lumia 950 XL [10]). Unfortunately the fingerprint readers on smart phone can be fooled [26] as well as face recognition [25].

Conclusion Inherent based authentication schemes are an alternative to knowledge-based and possession-based authentication schemes. They are handy because we *are* them and they can not be forgotten or stolen. More and more device manufacturers implement scanners for those biometric features, which makes using those devices more comfortable.

3.1.4 Two Factor Authentication

A two-factor authentication (2FA) uses two authentication schemes sequentially. For this chapter we will focus on a commonly used combination of authentication schemes i.e., the *knowledge* and *possession* schemes. Some examples of this combination of authentication schemes are e.g., online banking, Dropbox- or Google's two-step verification.

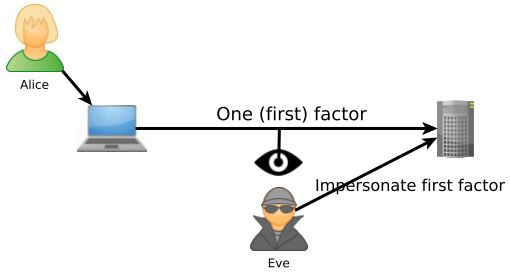

Figure 3.1: One-Factor Authentication

The advantage compared to a single factor authentication (SFA) is that both factors need to be compromised by a malicious entity in order to impersonate users.

For the following two examples we will have a user (i.e., Alice) try to login to a service. We assume that none of her devices (computer, mobile phone) are compromised and that a malicious entity (i.e., Eve) only spies on the network traffic between Alice's computer and the service Alice wants to log onto.

As first example, Alice will use SFA to logon to her desired service. In this case she will use a knowledge based authentication scheme, that means Alice will logon by knowing her username and password combination. With SFA, Eve can spy on the data transmitted between Alice's computer and the service and is therefore able to gather her username and password combination as seen in Figure 3.1. From that moment on, Eve will be able to impersonate Alice on that service (and maybe even on other services as well [32]).

As second example, Alice will use 2FA to logon to her desired service. In this case, she will use a knowledge based authentication scheme together with a possession based authentication scheme. The knowledge-based authentication scheme will again be her username and password combination. The possession based authentication scheme will be a one-time password (OTP) sent to Alice's mobile phone. When Alice starts her login process and she enters her username and password combination (the first authentication factor), the second and in this case last authentication factor will be triggered. The service sends an OTP to Alice's mobile phone, together with some kind of "reference value". Both, the OTP and reference value are important identifiers for the logon process. The reference value identifies her login process uniquely, as every login process, after passing the first authentication factor, creates a new reference value. Therefore Alice is able to match her login process identifier (i.e., reference value) received on her mobile phone, with the identifier shown on the service's login form. After verifying that these two identifiers are the same, Alice then enters the received OTP and completes the second authentication factor as seen in Figure 3.2.

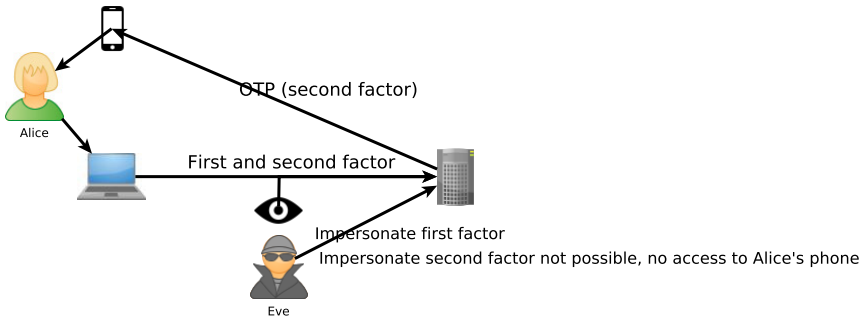

Figure 3.2: Two-Factor Authentication

Figure 3.3: Reference value used to identify the login session

Eve is still able to spy on the network traffic between Alice's computer and Alice's service and therefore gather Alice's username and password combination. Eve is also able to capture the OTP received by Alice, if Alice sends the OTP over the network for her login process. However, as the OTP is, as the name implies, a one-time password, the password is only valid for Alice's login process. If Eve started her own login process by providing the captured username and password combination, Alice would receive an OTP and reference value on her mobile phone. As Alice is either not using the service at all, or the reference value received does not match the reference value shown on the service's login form, Alice will be wise enough not to enter the received OTP for Eve's login process. This way, Eve is not able to finish her login process.

3.1.5 Conclusion

A lot of single-actor authentication schemes are based on the knowledge based authentication scheme. This scheme is quite popular amongst services but unfortunately easy to abuse if the password is badly chosen. Possession based authentication schemes are much more robust against attacks but lack in usability and user acceptance. Inherent based authentication schemes are becoming more popular nowadays but are still not fool proof.

If strong authentication is needed, a two-factor authentication (2FA) should be chosen. A common example is using the knowledge based authentication scheme together with the possession based authentication scheme. Some banks require 2FA for online transactions, and more and more internet services add the second factor as an optional process to their services.

3.2 Austrian Citizen Card Concept

The Austrian Citizen Card is a concept [27] based on a valid identification document together with a certificate, that enables citizens to identify themselves against services and sign documents. The Citizen Card is available for every Austrian citizen for free [14]. To acquire a Citizen Card one has to either visit a registration officer (RO) and provide a valid identification document, receive an RSa letter issued from an authority which already has a valid identification document, or create a Citizen Card by using his FinanzOnline account [1].

After the Austrian Citizen Card has been issued, an integrity-protected, authenticated document with non-repudiation can be created by creating a qualified digital signature over the document. A citizen can either send these signed documents via email, or use available online forms [15] to communicate with public authorities. Public authorities are able to verify the signature of the document, identify the signatory and issue further steps according to the signed document. This way public authorities are able to process a citizen's request quicker which is cost efficient and time-saving.

Because the signature of a citizen on such a document is critical security wise, an important factor for the Austrian Citizen Card concept was to use two factor authentication. It states that an entity has to identify himself by using two factors mentioned in Section 3.1.

As the Citizen Card concept describes a concept but not a single final solution, there are two implementations available:

1. Citizen Card on a crypto smart card (e.g., e-Card)

2. A mobile phone signature called *Handy Signatur*

Both implementations fulfill the Citizen Card concept but are different from each other. Both serve the same purpose, namely to identify a citizen in the World Wide Web based on their signatures and to enable citizens to create qualified digital signatures which are legally considered equivalent to handwritten signatures. In the following sections we will explain these two implementations briefly.

3.2.1 Citizen Card on a Crypto Smart Card

The first Citizen Card implementation in Austria was the "Citizen Card on a crypto smart card". In Austria, this implementation of the Citizen Card concept is mostly used on so-called "e-Cards", which are health insurance cards issued to Austrian citizens. In order to use the Citizen Card on a crypto smart card, card readers are needed.

Advantages These health insurance cards are issued to every Austrian citizen. Anyone can activate the Citizen Card on his health insurance card for free.

Disadvantages First, smart card readers do cost money especially those with a PIN pad. Second, card reader software has to be installed on a computer, which may be difficult for inexperienced users. These two steps were an obstacle, and therefore the first implementation of the Citizen Card concept was not widely used. Especially the portability of this implementation was suboptimal. If travelling abroad or were using another computer you were not able to use your Citizen Card right away. Either no card reader was installed on the system (and could not be installed due to missing administrator rights) or no smart card reader was available. This limited the usage and application of the Citizen Card.

Conclusion This implementation of the Austrian Citizen Card needed a successor that does not have the drawbacks mentioned.

3.2.2 The Austrian "Handy Signatur"

The second Citizen Card implementation in Austria is a mobile phone signature called Handy Signatur. It enables users to possess a Citizen Card without having the need to possess a card reader or even a crypto token, i.e., smart card. Instead, the *token* a user has to own is a mobile phone. The activation process is quite similar to the Citizen Card on crypto smart cards. Either the user visits a registration officer (RO) or the user uses another service to which he already provided a valid identification document of himself e.g., FinanzOnline, TUGraz online, Online Banking, Post.at.

As an example, we will describe the process of how a user has to identify himself using the Handy Signatur, in order to use the service FinanzOnline[1]:

[1] https://finanzonline.bmf.gv.at/fon/

1. User chooses his favorite browser and navigates to the FinanzOnline website.

2. User chooses "Handy" instead of "Karte" within the "Login with Bürgerkarte" window.

3. User enters his mobile phone number in the field "Mobiltelefonnummer" and his mobile phone signature password in the field "Signatur Passwort", and clicks on "Identifizieren" to start the signature process.

4. The user receives a SMS on his mobile phone. The message contains a "reference value" which binds the SMS received to the document to-be-signed. If the reference value from the SMS does not match the reference value from the login window, the user must cancel the login process. If they match, the user should check the "Signaturdaten". The "Signaturdaten" is the document-to-be-signed. It will only be signed if the user completes the signature process. The service is able, based on the signed data, to determine the user's identity.

5. After verifying the reference value and the "Signaturdaten", the user enters the received TAN from the SMS into the field "Tan".

6. The signature will be generated on the Handy Signatur server and sent to FinanzOnline. FinanzOnline is now able to verify and validate the signature created by the Austrian Handy Signatur and knows which user signed the document. Based on that FinanzOnline knows which user wants to use the FinanzOnline service. After successful validation, the user will be logged into the service.

Advantages As lots of people own mobile phones, it is far more convenient to have the Citizen Card functionality activated for the mobile phone instead of a crypto smart card. Most times the tokens have to be distributed amonst all employees, but in this case the rollout happened some time before the activation, and the usability of mobile phones are great. All in all, it makes the mobile phone a great possession token for the citizen card concept.

Disadvantages Mobile phones (most times smart phones) are sensitive tokens and need to be protected. Users have their emails, social accounts and contacts on their phones. In addition, the mobile phone now serves as a token for possession based authentication. This makes the mobile phone even more valuable and a greater target for theft.

Conclusion As of 14th September 2018, there exist about 1.087.540 Handy Signatur activations and the numbers keep on rising[2]. As this Citizen Card implementation is more popular as the Citizen Card on a crypto smart card we will use an approach, based on the Austrian Handy Signatur, in the following Section 3.3 called ServerBKU.

3.3 ServerBKU

The ServerBKU is a secure and flexible server-based mobile e-ID and e-signature solution [33] developed by the Institute of Applied Information Processing and Communications (IAIK). It is a web-based solution that enables users to digitally sign documents based on digital certificates.

Figure 3.4 shows an example setup in which every ServerBKU module can be deployed on a separate server or virtual server. As the ServerBKU is flexible, fewer servers may be used, or more if redundancy or fallbacks are necessary.

In the following chapter, we will describe the architecture of the ServerBKU. After that we will describe how the two basic modules, namely Activation and Signature work.

3.3.1 Architecture

Figure 3.4 shows the architecture of the ServerBKU. Basically, we can distinguish between the core modules of the ServerBKU and the peripheral services. The four core modules are the Activation- and the Signature module whereby both, the Activation- and Signature module, have a front- and backend. The peripheral services are services from external components. The communication between front- and backend happens via Java Messaging Service (JMS) secured over Transport Layer Security (TLS).

Frontend Both frontends need a shared database for, e.g., user names, emails, and user preferences. Also the Activation frontend needs peripheral services during the Citizen Card activation which are:

- The ID Register, in Austria the "Zentrales Melderegister", a register that holds citizen identification like names, date of birth, address, certain document number.

[2] http://www.a-trust.at/handystat/

- Any CA (Certificate Authority); in this figure, it is explicitly named CAPSO (CA and Pki SOlution, a CA developed by IAIK).
- The IDC (Identity Confirmation) / SIR (Standard Identification Record) Web service is a simple web service able to receive and validates SIRs. SIRs contain data acquired upon user identification.

Backend Both backends need a shared database as well. It contains critical data i.e., the signature key or the person identity link [30].

The Activation backend carries out the signature key generation. Signature keys are generated using a hardware security module (HSM). Furthermore, the Activation backend is responsible for creating a certificate signing request (CSR) sent to the CA in order to get a digital certificate for the signature key.

The Signature backend receives the pre-processed signing request from the Signature frontend and carries out the cryptographic signing operation.

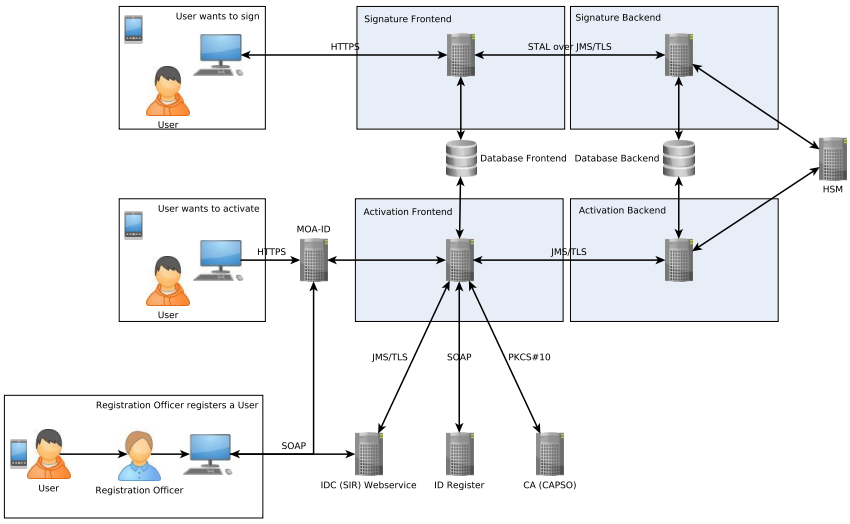

Figure 3.4: ServerBKU Architecture

3.3.1.1 Activation

The Activation module comes into play whenever users need to activate Mobile Citizen Cards (MCC). The activation process is split into two parts. The first part, called *Registration* can only be accomplished with the help of Registration Officers (RO). Users need to identify themselves against an RO. ROs vouch for the

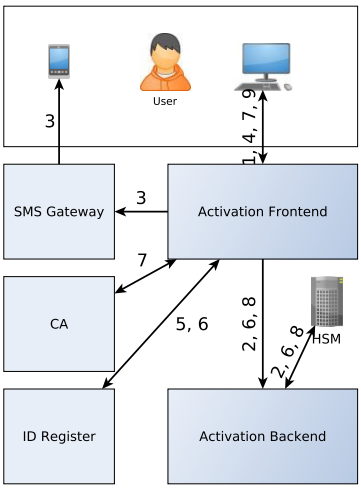

Figure 3.5: ServerBKU Activation Module

identity of users, provided by a valid identification document e.g., passport, drivers license. After users provided enough information for the registration process, a signed record, called Standard Identification Record (SIR), is sent to the ServerBKU. A SIR contains all information necessary to activate users at the ServerBKU. SIRs will be stored in the ServerBKU's database and can be used by users who want to activate an MCC. At the time of storing the SIR in the ServerBKU's database, users receive an activation code on their mobile phone. This activation code is necessary for the second part of the process called *Activation*.

As the activation part is completely separate from the registration part, the activation part of an MCC can be done at home or at the office, it does not need to happen at the RO's office.

The second part activates the Mobile Citizen Card:

1. Users start the activation by entering their mobile phone number and the received activation code. The ServerBKU fetches the Standard Identification Record (SIR) from the database based on the mobile phone number and activation code. In each step data from the SIR is taken and used, and if necessary users are asked to provide additional data. Such data are e.g., a signature password, email address, and a revocation password. It is important to mention that neither the signature password nor the revocation password are asked by the RO and therefore are not stored in the SIR.

2. After the first activation step, a signature keypair is created at the Activation backend inside a Hardware Security Module (HSM). This keypair is wrapped and exported into the backend database. The wrapping algorithm is described in [28].

3. A transaction number (TAN) is sent to the user's mobile phone.

4. The user has to enter the received TAN into the activation form in order to check whether the user truly possesses the mobile phone.

5. An Identity Link (IDL) request containing the user's signature public key and additional user data is generated and sent to the Identity Register.

6. The Identity Register responds with a signed IDL response, which is saved in the Activation backend database.

7. After receiving the IDL response, a certificate signing request (CSR) is generated containing the user's chosen revocation password and the public key of the signature key pair. The CSR is then sent to the Certification Authority (CA).

8. The CA responds with a digital certificate issued for that user based on his CSR. The certificate is stored in the backend database.

9. The user gets notified that the activation process is complete.

After these steps are complete, users are able to use their MCC for signing documents using the Signature module. The Signature module will be explained in Section 3.3.1.2.

Any subsequent MCC activations can be done without the need for an RO. This way users are able to skip the registration phase, once they have a valid MCC.

3.3.1.2 Signature

The Signature module which consists, as the Activation module, of a front- and backend will probably be the most frequently used module from the ServerBKU. The Signature module is responsible for creating XML and CMS signatures. These signatures can be created for documents for public authorities, or to prove the identity of users to external applications.

Communication third Party Application - ServerBKU Third party applications are able to use the ServerBKU as an identification provider. The ServerBKU receives the signature request (e.g., containing a document to sign) from the third party application and requests the user to sign the document. The signed document is returned to the third party application, which is then able to:

1. Verify the signature of the document to check its validity.

2. Identify the signatory based on his signing certificate.

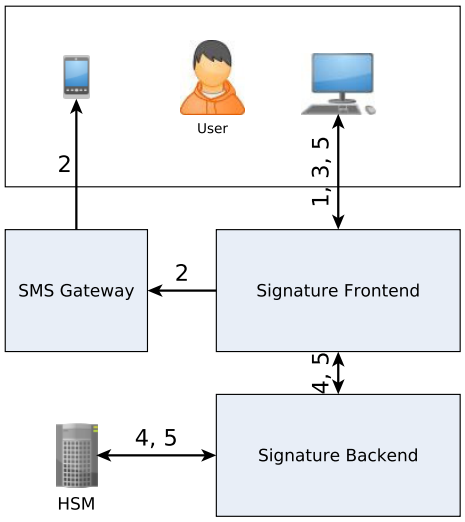

Figure 3.6: ServerBKU Signature Module

Communication ServerBKU - User Upon receiving a signature request, the Signature module interacts with the user. The interaction process between users and the ServerBKU is as follows:

1. The ServerBKU requests the phone number and the signature password from the user.
2. If correct the ServerBKU sends a transaction number (TAN) and a reference value to the user's mobile phone.
3. The user verifies the received reference value, against the reference value seen in the browser. If the reference values match, the user should check the signature data. The signature data is the data which will be signed once the authentication against the ServerBKU is complete. If the user is satisfied with the document which will be signed, the user enters the received TAN and the ServerBKU verifies the TAN.
4. The ServerBKU completes the signature process in the Signature backend.
5. Once the signature is created, the signed document is sent back to the external application. Typically the user gets forwarded to the external application.

The process for XML and CMS signatures is the same. The user is able to view the data which will be signed in step 3. This should always be done before entering the TAN.

3.3.2 Conclusion

In this section, we have taken a look at a secure and flexible server-based mobile e-ID and e-signature solution called ServerBKU. After completing the registration and activation process, users are able to create signatures over documents based on digital signatures. These signatures are considered legally equivalent to handwritten signatures in Austria and can be processed by public authorities.

3.4 CrySIL

The Cryptographic Service Interoperability Layer (CrySIL) [35] concept was created to unify the existing mobile heterogeneous device landscape.

CrySIL offers the possibility to perform cryptographic operations on devices which are not capable of doing these operations themselves. Reasons for that could be that the devices do not have the computing power, a cryptographic chip which may be necessary or the devices do not have the cryptographic keys locally stored to perform the cryptographic operations. Instead, a crypto service provider will carry out the cryptographic operations.

CrySILs concept is to break the approach of conventional cryptographic service providers apart. It does so by separating the cryptographic service provider into several modules. Each module must have only one specific task. An overview of CrySILs architecture can be seen in Figure 3.7.

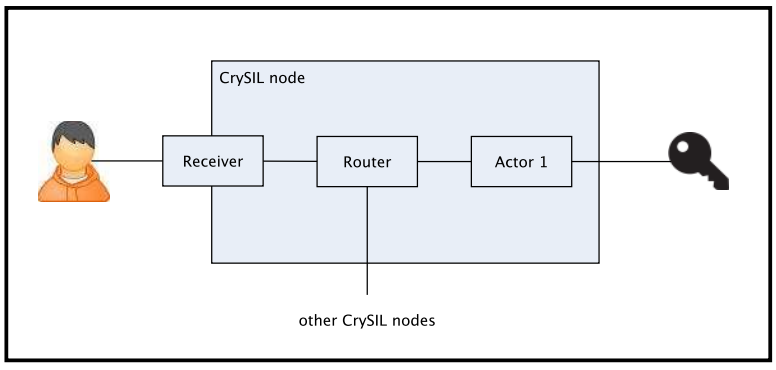

Figure 3.7: CrySIL overview

3.4.1 Architecture, Components and Open Challenges

CrySIL consists of several components, namely *actor*, *receiver*, *router*, which will be explained in this section. These components may form a so-called CrySIL node. Nodes may be connected to each other over a network which may not be secure.

Receivers *Receivers* receive commands from applications and recode (if necessary) these commands into CrySIL commands and send them to the connected router. *Receivers* do not perform any cryptographic operation. After a command was executed the result usually is send back over the receiver to the origin. *Receivers* may be implemented on any platform, device, and technology. An example *receiver* may be an HTTPS receiver, plain Java API receiver or a JSON receiver.

Actors *Actors* expose contents of key providers to the interoperability layer. Key providers may host key material, and key providers may perform cryptographic operations. An actor may have an authentication module configured to enforce authentication.

Routers *Routers* are responsible for forwarding the received command to the appropriate *actor*, or to the next CrySIL node. Currently, if *routers* forward commands to a different CrySIL node the communication happens over an unencrypted channel. The protocol itself does not support any transport encryption. Based on the CrySIL model we've build on, routers do know the network topology beforehand.

3.4.2 Open Challenges

CrySIL is still under development and therefore several challenges exist, therefore.

The first challenge is to preserve privacy between CrySIL nodes. No information should leak to anyone else except the communication partner when communicating over multiple CrySIL nodes. Therefore one big disadvantage is the missing transport encryption between CrySIL nodes. Commands sent over an insecure network must be encrypted. A good approach would be to introduce end-to-end encryption. This way a sender can be sure that only the receiver is able to decrypt the command sent but no party in between.

The second challenge is that CrySIL nodes attached to a CrySIL network must be discovered if needed. Any new service attached to an existing infrastructure must be discoverable and usable. Otherwise, it would make no sense to attach it in the first place.

The third challenge is routing to different CrySIL nodes. Currently CrySIL only supports routing through one CrySIL node. Although the concept is open for attaching CrySIL nodes to each other no routing strategy has been developed yet.

The fourth challenge is the fact that digital signatures are not available for CrySIL. It would be nice to have an adaption for the Austrian Handy Signatur in order to use the Handy Signatur on any application CrySIL is running on. To achieve this CrySIL needs to master the previous three challenges mentioned. Otherwise, any attempt to communicate with the Handy Signatur would either

- leak sensitive user credentials or
- the Handy Signatur service would not be reachable without appropriate routing.

3.4.3 Conclusion

CrySIL enables the heterogeneous device landscape to perform cryptographic operations. As not every device is able to do cryptographic operations itself, CrySIL offers a crypto service provider for these devices. The CrySIL architecture is separated into several modules which have different responsibilities in performing tasks. Although CrySIL is a step forward it still lacks in certain aspects. With our contributions to CrySIL as seen in chapter 4, we will try to fill the missing aspects of CrySIL.

4 Contribution to CrySIL

The goal of our work is to provide users the possibility to create digital signatures securely, on every CrySIL supported device. To demonstrate how this works we will create a use case. In that use case, a user wants to sign a document digitally.

In the following sections, we are going to explain step by step what needs to be done to create a digitally signed document. First, we will describe the use case. Second, we will explain, in the following chapters, which steps users need to perform before they can digitally sign a document. In the end, we will conclude the use case.

4.1 Use case: Signing a document with CrySIL

The user (Bob) wants to file a tax report to the Austrian federal ministry of finance. Bob must digitally sign the tax report. To allow public authorities to verify Bob's signature, the signature must be based on a qualified certificate. As CrySIL is not able to generate signatures based on a qualified certificate, we assume that for our use case that, i.e., the Austrian federal ministry of finance only requires a non-qualified digital signature. Furthermore, as a restriction, the signature creation process must be done with Bob's smartphone as shown in Figure 4.1. As we do not trust smartphones to protect key material adequately, the ServerBKU will store the key material. Bob's smartphone has to communicate with the ServerBKU to sign a document.

We extended the ServerBKU, which only supported SecurityLayer [16] protocol, to support the CrySIL protocol as well. This way Bob's smartphone can communicate with the ServerBKU over the CrySIL network.

The restriction, i.e., that the signature creation process must happen on a smartphone is not uncommon. Not long ago, desktop computers and laptops were needed to do most of the digital work. As smartphones became popular, widely available and more powerful, some of the work shifted from desktop computers to smartphones.

In the following sections, we will take a closer look at our use case. Figure 4.2 showing our use case, consists of players, objectives, and components, which we will explain in the following section. After explaining the players, objective, and components we will take one step towards Bob's goal to create a digital signature over a document, which is described within the next chapter.

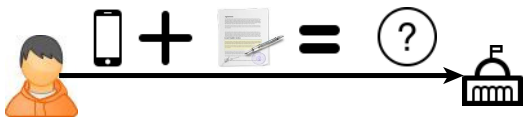

Figure 4.1: The use case: Bob wants to create a digital signature over a document with his smart phone.

4.1.1 Players

Our use case has several players who interact or influence the use case.

The first player is the User, i.e., Bob. Bob is a typical user of CrySIL. Bob wants to send a document to a public authority. To do so, Bob needs to generate a digital signature (QDS) over the document. Otherwise, public authorities will not accept the document. As Bob owns an account of the signature service named ServerBKU, he can use this service to create a QDS over a document. For Bob to use his key material from the ServerBKU, Bob needs to authenticate himself against the signature service with his credentials.

The second player is a malicious party, i.e., Eve. Eve is an attacker who wants to gather information about the document and Bob's credentials, as they are routed through Eve's CrySIL node. Eve could also try to convince Bob to sign a forged document, by manipulating the data exchanged between the signature service and Bob.

The third player is the signature service, i.e., ServerBKU. The signature service is considered trustworthy. With Bob's ServerBKU account, he can create digital signatures.

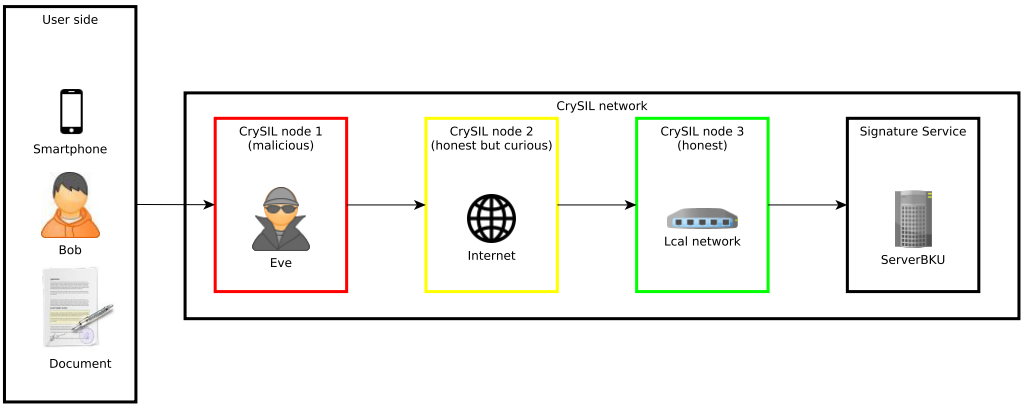

Figure 4.2: The use case: players, objectives and components

4.1.2 Objectives

The players in our use case have several objectives. We will explain the objective for each player.

Bob's objective is to get digitally signed documents to a public authority. Bob wants to use the ServerBKU for creating the digital signature.

Eve's objective is to get a maliciously crafted, but digitally signed by Bob, to a public authority. She wants either be able to have Bob sign a crafted document himself, or even be able to steal Bob's credentials to sign her crafted document in Bob's name.

The ServerBKU's objective is to create a digitally sign a document. The ServerBKU has two-factor authentication in place for authenticating users who want to sign documents digitally.

4.1.3 Components

Several components come into play in our use case.

The first component is the smartphone. In our use case, Bob owns the smartphone. His smartphone supports CrySIL by having a CrySIL App installed and properly configured.

The second component is the document to be signed. Bob possesses the document, and the signature service acting as a third player will sign the document

The third component is a CrySIL node. The node is an element of the CrySIL network. There may be multiple CrySIL nodes within the network. Some of them are honest (e.g., local routers), some may be honest but curious (e.g., nodes in the world wild web) and some may be malicious (i.e., operated by Eve).

The fourth component is the signature service. It creates a signature based on the user's certificate over the user's data (i.e., the document). A two-factor authentication (knowledge- and possession based authentication) is necessary to authorize the usage of a signing key and thus be able to create a signature.

4.2 Status quo and my Contribution

In this chapter, we are going to explain the current state of CrySIL and explain why my contribution was necessary to achieve our use case (see Section 4.1).

Status quo

CrySIL's protocol has a set of commands. We will take a look at a subset of the available protocol commands, needed for our use case. These commands are:

1. *Discover Keys Request* - enables us to fetch signature keys from a signature server

2. *Auth Challenge Request* - enables us to authorize against the signature server for accessing the signature key

In addition to these commands, we will evaluate a question: which service is going to sign our document for our use case.

Discover Key Request The *Discover Key Request* is used to retrieve available keys from a server (i.e., a CrySIL node connecting a server to the CrySIL network). The returned keys will either be handles (i.e., unique reference for a key) or the certificates associated with the keys.

However, the *Discover Keys* Request requires that the server knows the CrySIL node before sending the request. Currently, there is no mechanism to discover servers within a CrySIL network. Furthermore, there is even no mechanism to route requests within a CrySIL network from one node to another. So far, most educational examples consisted of just one client and one server, directly connected. For these examples no routing was necessary, and the only server was always the receiver of a request. A feature is missing to enable a client to discover servers within a CrySIL network.

Auth Challenge Request To access various protected resources, i.e., key material, an *Auth Challenge Request* may be required by the server beforehand. A simple authentication challenge may be a username and password challenge. The username and password tuple will be included within the request and send to the server. The server then verifies the authentication challenge and grants access to the protected resource if eligible.

However, an *Auth Challenge Request* typically contains authentication data for accessing protected resources. These authentication data belongs to the authenticator and must be protected against, e.g., eavesdropper. Currently, the CrySIL protocol does not any support transport encryption. The data (i.e., requests and responses) are always transmitted in plaintext between client and server. This issue would make it easy for an eavesdropper to steal the authentication credentials.

Signature Server Currently, there is no server-side signature solution available for signing data, which is a blocker for our use case as we need to sign a document over the CrySIL network digitally.

My Contribution

My contribution effort was focused on missing features to achieve the use case. Within the next paragraphs I'm going to explain my contributions to the missing aspects we have identified.

Discover Key Request The server's CrySIL node must be known to send the *Discover Key Request.* Also, to detect existing and new nodes within the CrySIL network, we need proper routing to reach the desired destination. We created a new command called *Discover Service Request.* This command enables us to detect existing and new nodes within the CrySIL network. Furthermore, a routing was implemented to route between the sender and receiver CrySIL node properly.

Auth Challenge Request To protect the sensitive data in the exchanged messages, we need transport encryption. The transport encryption was implemented by using TLS under the hood. For a detailed explanation, please see Section 4.6.

Signature Server A server-side signature solution was needed to sign documents. I have extended the ServerBKU (see Section 3.3) to handle the CrySIL protocol. With this extension, it is possible to communicate with the ServerBKU using only CrySIL commands and enables us to use the ServerBKU as a signature server.

4.3 Discover Service

As Bob does not know if and where the signature service is located within the CrySIL network, as seen in Figure 4.3, he must gather information about the CrySIL network. At this point, the *DiscoverService* command comes into play. Bob will send a *DiscoverService* command to his CrySIL network attached receiver, as illustrated in Figure 4.3. This command propagates throughout the CrySIL network. The responses contain information:

1. Which services are available and online
2. Which services support which commands

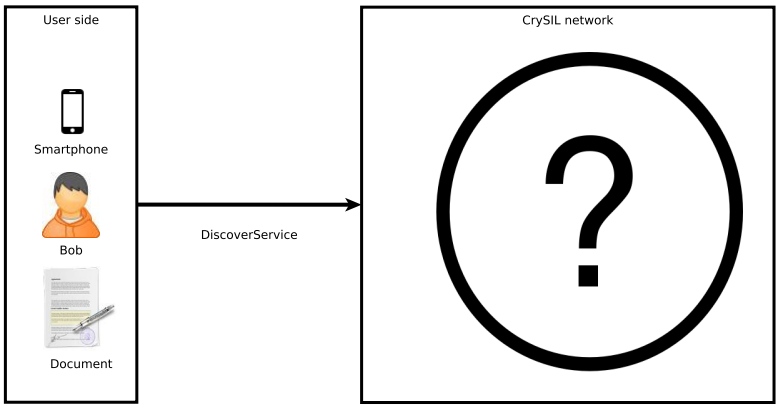

Figure 4.3: DiscoverService: How does the CrySIL network look like and which services are available?

3. Which services support secure connections

The *DiscoverService(-Request)* command is typically performed first to get to know the CrySIL network. Therefore, the response to a *DiscoverService* command is always a *DiscoverService(-Response*). That is different from any other CrySIL command. To understand why this is handy, consider the following example:

Let's take another CrySIL command, e.g., a *DiscoverKeysRequest* command. If this command is sent as the first command, the expected result would be *DiscoverKeyResponse* as seen in Figure 4.4. But due to the heterogeneous service configuration and CrySIL architecture, the responses could be:

- Service A responds with an expected *DiscoverKeyResponse*.
- Service B responds with an authentication challenge. That's because the *DiscoverKeysRequest* needs authentication from the sender to respond.
- Service C does not respond at all because the service is not available (i.e., offline).
- Service D responds with a *SecureTransportResponse* (which may contain an error due to the incorrect initial request which should have been a *SecureTransportRequest*). This service only communicates over a secure channel and therefore expected a *SecureTransportRequest* message to be sent, not a *DiscoverKeyRequest*.

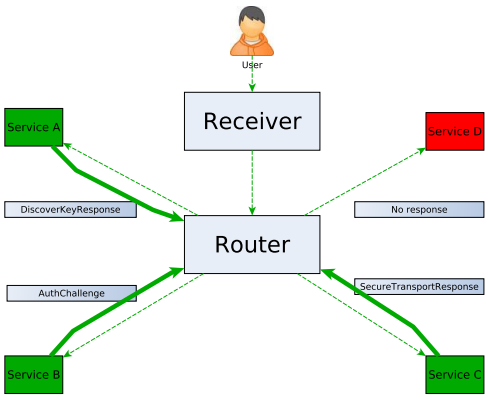

Figure 4.4: DiscoverKeyRequest response possibilities

As the CrySIL protocol is still under development, further responses may be possible. The more responses are possible, the more effort the sender has to process these different responses. An approach was needed to get rid of those different responses. Therefore the *DiscoverService* command was created.

It should be sent as the first command, and the results are always *DiscoverService* responses. As a contract, every service has to answer a *DiscoverService* request with a *DiscoverService* response.

4.3.1 Extending the CrySIL Protocol

The protocol extension for *DiscoverService* consists of two commands, namely *DiscoverServiceRequest* and *DiscoverServiceResponse*.

The user sends a *DiscoverServiceRequest* to the router, which then forwards all requests to all connected CrySIL nodes and actors. These CrySIL nodes forward the command to their connected CrySIL nodes, and actors and so on. This way, the request gets propagated throughout the whole CrySIL network.

As an example consider Figure 4.5. In this example, neither the ACTOR 2 from CrySIL NODE 1, nor ACTOR 6 from CrySIL NODE 2 will respond. Also, CrySIL NODE 4 is not going to respond to such a request either.

The *DiscoverService* request sent contains one parameter, names challenge. The challenge is a nonce and is used to determine the freshness of the responses. Every received *DiscoverServiceResponse's challenge* must be checked against the *challenge* from the *DiscoverServiceRequest* sent.

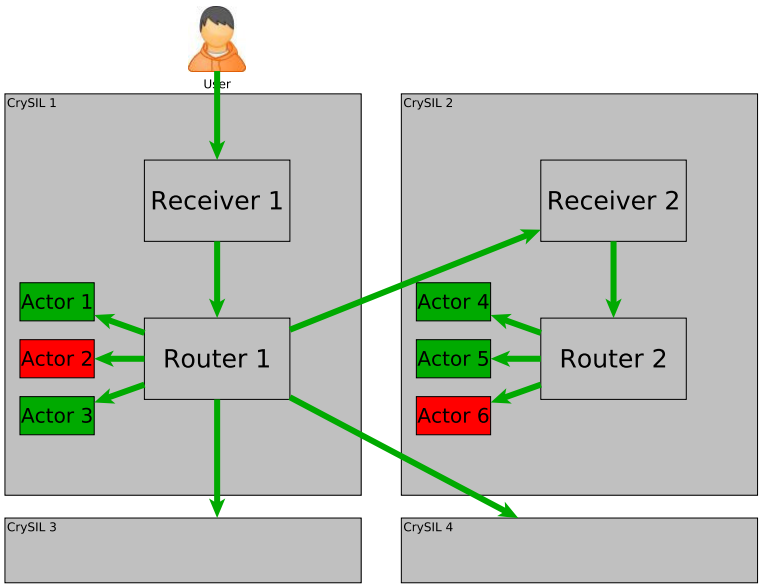

Figure 4.5: DiscoverService request propagated throughout the CrySIL networks

A *DiscoverServiceResponse* is a response which every service endpoint creates in the CrySIL network after they receive a *DiscoverServiceRequest*.

The Listing 4.1 shows an example of a *DiscoverServiceResponse* whereby the CrySIL header is omitted.

```
{
  "timestamp":"2015-08-29T18:37:05.493Z",
  "response":"someresponse",
  "cipherSuites":[
    "TLS_DHE_RSA_WITH_AES_128_CBC_SHA",
    "SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA",
    "TLS_RSA_WITH_AES_128_CBC_SHA",
    "SSL_RSA_WITH_3DES_EDE_CBC_SHA"
  ],
  "name":"the name of this actor e.g. signing actor",
  "commands":[ "discoverServiceRequest", "sign" ],
  "jsonSignature" {
    "value": "base64 encoded signature of 'content' (or payload without 'signature' property)",
    "certificate": "base64 encoded signing certificate, can be validated or is in trust store"
}
```

Listing 4.1: DiscoverServiceResponse

A *DiscoverServiceResponse* has several parameters:

1. *timestamp*: This timestamp denotes the time at which the service, which received the *DiscoverServiceRequest*, created the *DiscoverServiceResponse*

2. *response*: This is the challenge which was initially sent in the *DiscoverServiceRequest*. The challenge must be returned to the sender to let him verify the freshness of the *DiscoverService* response.

3. *path*: This is the path from the sender to the current Actor. It includes every Actor's name in between sender and the current Actor.

4. *cipherSuites*: A list of cipher suites which are supported by the service. If this service does not support TLS, this list must be empty.

5. *name*: The name of the actor creating this *DiscoverServiceResponse*.

6. *commands*: A list of commands which are supported by the responding service, e.g., sign, encrypt.

7. *jsonSignature*: This is the base64 encoded JSON signature [9]. The JWS payload must consist of all parameter mentioned above. Parameters are ordered alphabetically ascending, separated by a semicolon.

```
1 {
2   "timestamp":"2016-08-29T18:37:05.493Z",
3   "name":"name of the actor which did not respond"
4 }
```

Listing 4.2: NotRespondingServiceInformation

In the example from above, Actor 2 from CrySIL node 1 and Actor 6 from CrySIL node 2 do not respond to the *DiscoverService* request, that is denoted with dotted lines. Router 2 will create the missing response from CrySIL node 6. Together with the responses from Actor 4 and Actor 5, the response from CrySIL node 6 to Router 1. The same happens with the missing response from CrySIL node 2. The missing response will be created by Router 1 and sent back to the sender. The response, which is named *NotRespondingServiceInformation* and is created by Router 1, has following parameters:

1. *timestamp*: This timestamp denotes the time the router, which has just detected a missing *DiscoverServiceResponse*, created the response.

2. *name*: The name of the service which did not respond to the *DiscoverServiceRequest*.

All responses, *NotRespondingServiceInformation* and *DiscoverServiceResponse*, are collected into a *CollectionResult* by the router and sent back to the previous CrySIL node and so forth until the sender has received all responses, e.g., see Figure 4.6.

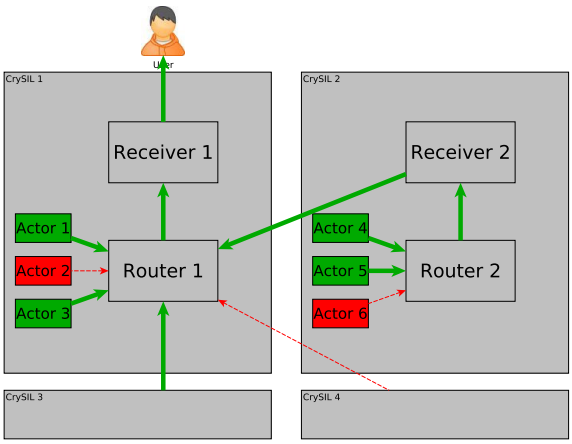

Figure 4.6: DiscoverService response collection

Once the sender received responses to his request the sender has to check:

1. Check the JSON Web Signature (JWS) if the signature is valid. If the signature is invalid, the response was forged and must be discarded.

2. If the JWS is valid, the sender must check the received *challenge* response. If the challenge does not match the challenge sent in the *DiscoverServiceRequest,* the response is not fresh.

3. After the sender has information which services responded to the request and which did not, he may now choose a service.

It is up to the user (or the client's strategy) how often such a *DiscoverService* command should be sent throughout the CrySIL network. If the command is sent too often, the CrySIL network will be polluted with *DiscoverService requests/responses.* If the command is sent too rarely, the client status information may be out of date.

4.3.2 Routing Information

For routing requests within the CrySIL network, source routing is used. In source routing, the sender knows the full path from himself to the receiver, because the network topology (source routing information) is known in advance. See Section 4.4.1 for a detailed explanation of routing in CrySIL. The open problem so far is how to gather this source routing information. To approach this challenge, we are using the DiscoverService to obtain routing information. DiscoverService requests are broadcasted throughout the CrySIL network, as seen in Figure 4.5 and can help to construct the required routing information.

Every actor which processes such a *DiscoverServiceRequest* must attach his unique name into the path element of CrySIL header. After every actor did that and the request reaches the destination, the path element includes the shortest path from source to destination. Figure 4.7 shows an example path element including the shortest path from Alice to Bob, whereas the request originated from Alice.

```
"Actor 1, Actor 2"
```

To encounter manipulation of routing information, the actor processing the received request will sign the routing information using JWS.

4.3.3 Conclusion

The *DiscoverService* command helped Bob to find the signature service he needs to sign his document. Furthermore, the command provided useful information for CrySIL such as routing information and whether end-to-end encryption is available or not.

4.4 Routing

As discussed in Section 4.3 Bob now knows where the signature service is located in the CrySIL network. To exchange messages between Bob, and the signature service, a correct, efficient and reliable routing is needed.

Efficient means that messages must not visit additional CrySIL nodes between sender and destination. Unnecessary hops in-between could mean that messages travel a longer distance, and therefore, it takes more time to reach the desired destination. That reduces the responsiveness of applications, which leads to reduced usability and lower user acceptance. Furthermore, message hops increase the likelihood of message loss as well.

Reliable means that messages must reach the correct destination. If messages are routed to an incorrect destination, the destination will (most likely) not be able to perform the desired action. These messages will most likely be dropped, and the sender will not receive any feedback, i.e., that the messages were dropped, which could lead to erroneous behavior of the application or the sender.

4.4.1 Routing in CrySIL

Routing messages in CrySIL means to deliver messages (in our case requests and responses) from sender to receiver and vice versa. The sender (i.e., Alice), e.g., sends

a *DiscoverKeys* request to a server. The request is forwarded from one CrySIL node to the next CrySIL node until the request reaches the desired destination.

In CrySIL, we use source routing. Source routing requires knowledge of the network topology at the time a message is sent. That means the sender defines a route throughout the CrySIL network. The routing information is included in the CrySIL header and is updated at every CrySIL node. After Bob receives the message, he processes the request and sends the response to back Alice.

We refer to Figure 4.7 for an example. Bob wants to send a message to Alice. In CrySIL, Bob could be a user who wants to access a service, called Alice. The message, i.e., the request must travel the correct path. In this example, the correct path is as following: *Actor1, Actor*2. A different would mean that the routing is incorrect, and the request would reach an incorrect destination.

We have three requirements for the CrySIL network regarding routing to work correctly:

1. (R1): An Actor must only be linked to one receiver. An Actor must not be able to forward data to multiple receivers. In this case, receivers may be services as well.

2. (R2): A router takes requests only from one receiver. A router must not be able to receive data from multiple receivers.

3. (R3): An Actors name must be unique in a CrySIL node. An Actor must not share his name with any other Actors in his CrySIL node.

We demonstrate these requirements in Figure 4.8, Figure 4.9 and Figure 4.10.

The example in Figure 4.7 shows the path the message needs to be forwarded: Actor 1, Actor 2. Path elements are in order, from left to right, meaning that the request must be forwarded to the next path element which is highlighted by the pointer. If an element was traversed the pointer moves to the next path element until the destination is reached.

After the request was processed by Bob, it must be sent back to Alice. Therefore, the routing information must be traversed in reverse order, as seen in Figure 4.11.

As source routing requires routing information before sending a message, it is vital that routing information is correct and up to date. For more information about the routing information, see Section 4.3.2.

Routing messages from the sender to the receiver means that messages must be routed according to the path information stored in the CrySIL header. Path information consists of the following parameters:

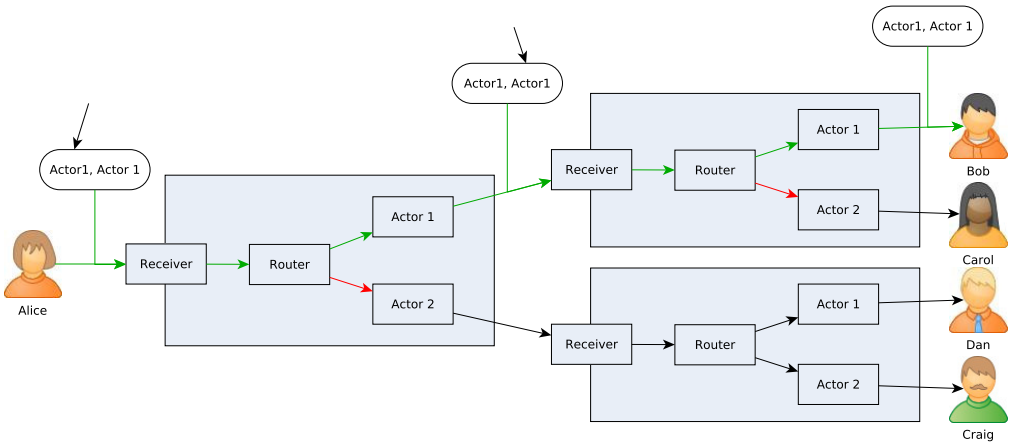

Figure 4.7: An example routing from Alice to Bob

- *Path*: A list of CrySIL node entries which must be traversed to reach the desired destination. The first entry denotes the sender, the last entry the destination. When returning a response to a message, the list must be traversed in reverse order.
- *Current*: Denotes at which CrySIL node the message is currently is.
- *Next*: Denotes the next actor the message must be forwarded to.

This routing strategy implies that the actor names are unique in a CrySIL node, see Figure 4.10 as an invalid example. Duplicate actor names lead to incorrect routing and endanger the CrySIL infrastructure and its usage.

The actual routing of messages happens at the router. The router reads the path

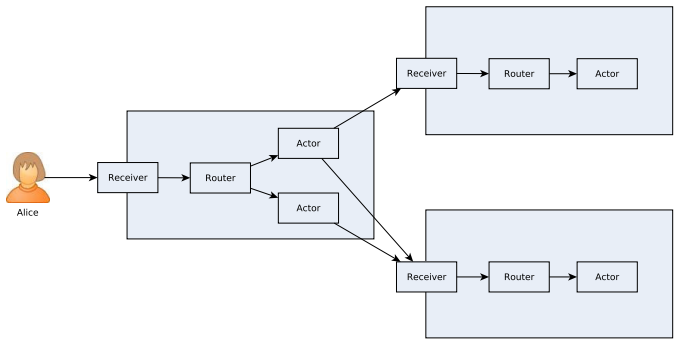

Figure 4.8: Invalid routing example which violates prerequisite R1

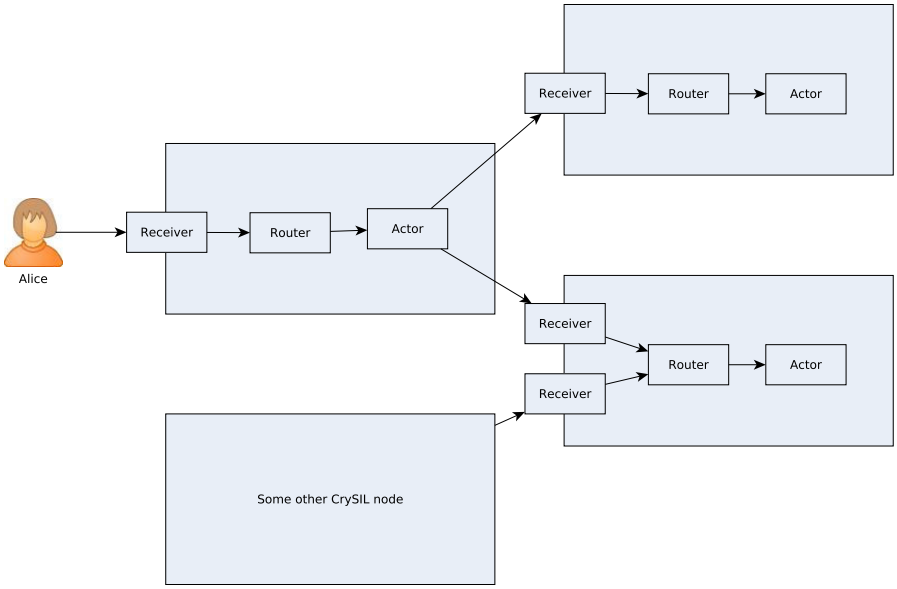

Figure 4.9: Invalid routing example which violates prerequisite R2

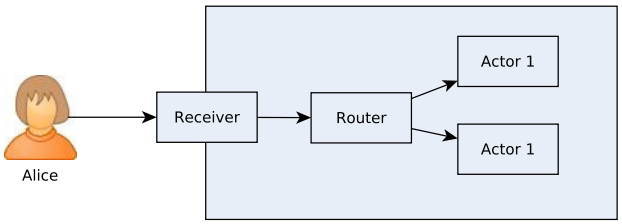

Figure 4.10: Invalid routing example which violates prerequisite R3

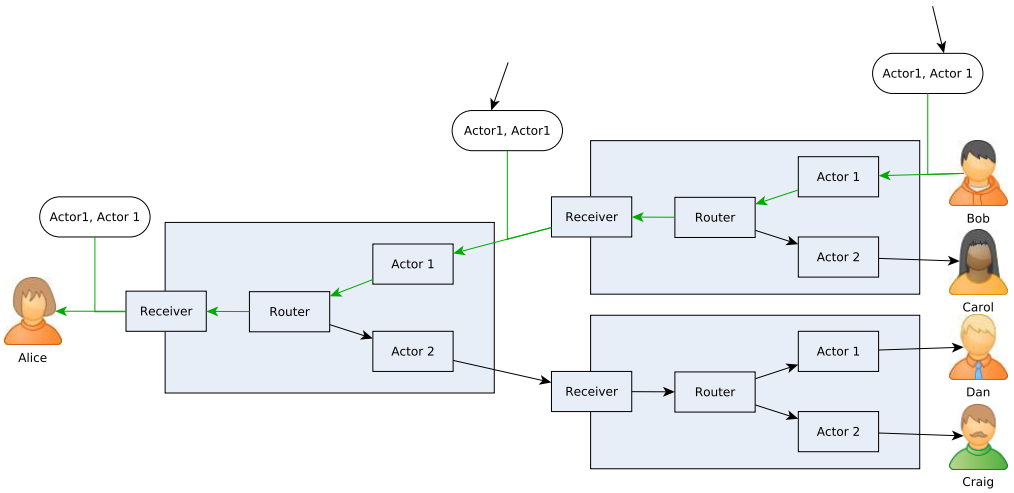

Figure 4.11: Routing back from Bob to Alice

information from the received message and checks if a correct route can be found. A route is found when the next actor name in the path information is registered at the router. The router then forwards the message to the actor, updates the *Current* and *Next* fields within the header, and the actor continues with the processing.

The actor either performs the desired action or forwards the message to the next CrySIL node.

If the actor has finished processing, a message is returned to the sender by setting the path information from the received message (e.g., request) into the path information from the created message (e.g., response) in reverse order.

If the actor does not perform any action, the message is forwarded to the next CrySIL node, and the previously described routing protocol is performed.

4.4.2 Conclusion

By deploying the proposed routing mechanism, Bob's message reaches the desired destination within the CrySIL network. This way, Bob's can interact with the signature service.

4.5 Digital Signatures with ServerBKU

Because of *DiscoverService* and routing, Bob is now able to exchange messages with the signature service. To create a digital signature, Bob has to perform the following steps:

1. Fetch a signing key for the signature process by executing a *DiscoverKey* command. This command offers Bob the ability to choose the key he wants to use to create a signature, as Bob may have multiple keys. For choosing a key, each signature key is identified by an identifier.
2. Create a *SPayloadSignRequest* request. Provide the document (or hash values of the document) to be signed and the chosen signature key identifier to be used, and send the *SPayloadSignRequest* to the ServerBKU.
3. The ServerBKU returns with an authentication challenge to Bob to verify whether Bob is entitled to use the signature key for signing.
4. Bob needs to provide credentials (knowledge-based authentication, i.e., username and password) to access the signing key chosen in step 1, and send the authentication response back to the ServerBKU.
5. Upon successful verification of the first authentication factor, the ServerBKU responds with another authentication challenge. This time the ServerBKU wants to verify the OTP (i.e., possession-based authentication) received on Bob's mobile phone.
6. Bob returns the OTP to the ServerBKU. If the OTP is valid, the ServerBKU signs the document and returns the signature to Bob.

These steps are very similar to the signature process described in Section 3.3.1.2. In this case, the HTTP(S) receiver which normally consumes an *XMLSignatureRequest* will not be used. Instead, a CrySIL JSON receiver will be used to process all CrySIL commands as seen in Figure 4.12.

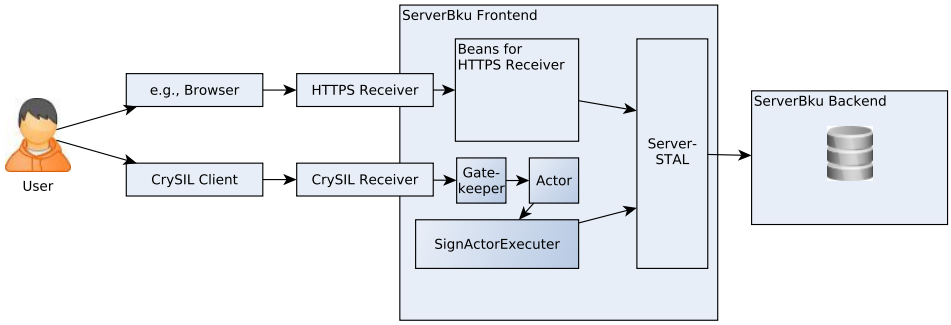

Figure 4.12: Digital Signatures with CrySIL and ServerBKU

Apart from using a different interface to communicate with the ServerBKU, the workflow within the ServerBKU is the same.

4.5.1 Conclusion

Bob successfully signed his document with the ServerBKU. As we have assumed, that the Austrian federal ministry of finance only requires a non-qualified signature, the document will be accepted and processed.

4.6 Endpoint Security

An open challenge so far was the insecure communication channel between two CrySIL nodes, i.e., Bob's smartphone and the signature service. As the exchanged messages may be routed over insecure CrySIL nodes, data protection is important. Some CrySIL nodes may be honest, some honest but curious, and some may even be malicious. These "intermediate" CrySIL nodes are not under Bob's control. Therefore end-to-end encryption, between Bob and the signature service, is necessary to preserve Bob's privacy.

In the following chapter, we will explain how we establish end-to-end encryption between two CrySIL nodes over an insecure network by using TLS on top of the CrySIL protocol.

4.6.1 Using TLS for End-To-End Encryption

To create end-to-end encryption, we have to encrypt all CrySIL request/response messages. A message consists of header and payload. It is important to encrypt both, the header and the payload. The header might include sensitive information, e.g., a session id or authentication data. The payload might include sensitive information as well, e.g., a username-password combination for a privileged operation, and therefore has to be encrypted as well.

By choosing TLS, we rely on suitable and studied technology for ensuring end-to-end encryption.

The CrySIL protocol is transport protocol independent. Therefore we must not solely encrypt messages via TLS and send the encrypted messages via Transport Control Protocol (TCP).

To preserve the CrySIL protocol's transport independence, we must send all TLS encrypted messages via the CrySIL protocol. To achieve this, we have to put the TLS messages into a new CrySIL message. The CrySIL message will then be sent to the receiver. Figure 4.13 shows an example of a message exchange between a sender and a receiver with an already established TLS connection. The message exchange over TLS was named *SecureTransport*.

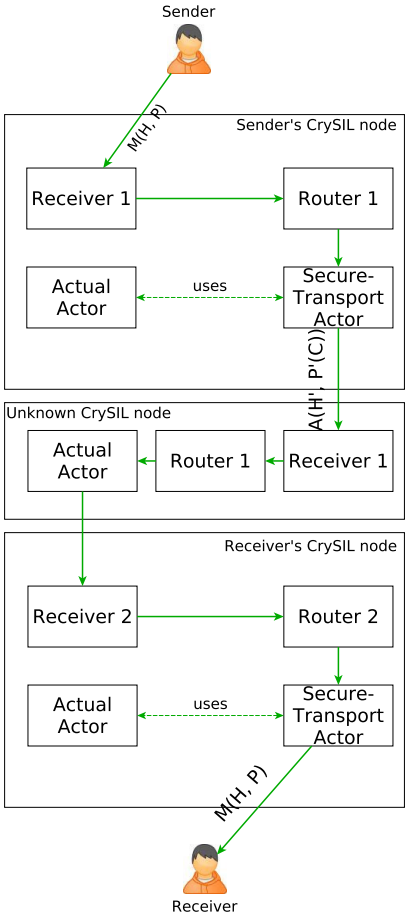

Figure 4.13: Overview of a SecureTransport message exchange

1. The sender puts the message to be sent (M), consisting of a header (H) and a payload (P) into his already initialized TLSEngine. The TLSEngine generates an encrypted message (C).

2. The encrypted message C will be put into a newly generated CrySIL message's (A) consisting of a header (H') and a payload (P').

3. The CrySIL message (A) will be sent over the CrySIL network to the receiver.

4. The receiver reads the encrypted message (C) from the payload (P') and puts it into his TLSEngine.

5. The TLSEngine decrypts the message (C) and thereby obtains the plain text message (M), which can be used for further processing.

It is essential to understand that the CrySIL message (A) is sent in plain text. This needs to be done to guarantee a correct routing through all CrySIL nodes. The message (A) contains routing information, in plaintext, in the header, but not any sensitive data in plaintext. The payload of the message (A) contains an encrypted message from the TLSEngine. Only the receiver can decrypt the message of the payload.

4.6.2 SecureTransport: TLS connection establishment over the CrySIL protocol

The TLS handshake messages have to be put into a newly generated CrySIL message (A), to establish end-to-end encryption between two CrySIL nodes (i.e., the sender and the receiver), as shown in Figure 4.13. Message (A) will be sent to the receiver. The receiver responds with the next handshake message according to the TLS protocol. Sender and receiver exchange as many handshake messages as necessary until the connection is established [12].

Once the TLS handshake is has been performed, messages can be exchanged securely between sender and receiver.

4.6.3 Conclusion

Bob was able to create his digital signature with the signature service. The exchanged messages were transported over a secure channel by establishing end-to-end encryption between Bob and the signature service. This was done by exchanging TLS messages over the CrySIL protocol.

4.7 The Big Picture Concluded

As seen in Figure 4.14, Bob was able to create a digital signature for his document by using our four building blocks proposed.

1. *DiscoverService* gave Bob the location of the signature service.

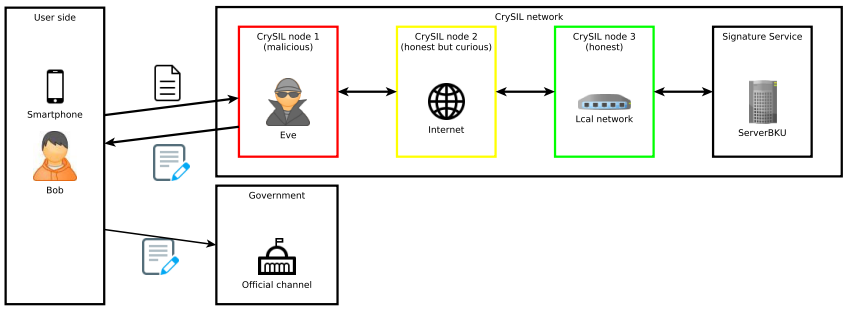

Figure 4.14: After interaction with the ServerBKU, Bob is able to send the signed document to the public authority

2. Routing enabled Bob to exchange messages between his smartphone and the signature service.

3. The signature service (ServerBKU) created the signature for his document

4. The secure channel prevented malicious people from eavesdropping on the communication.

Bob would not have been able to securely create a digital signature for his document if any of these building blocks would be missing:

1. Without *DiscoverService,* Bob would not be able to know the signature service's location within the CrySIL network.

2. Without routing, the sent messages would not reach the signature service.

3. Without the signature service ServerBKU within the CrySIL, Bob would not be able to create a digital signature on his smartphone.

4. Without the secure channel, Bob's credentials would be exposed to malicious eavesdropping parties eavesdropping the communication between Bob and the signature service.

5 Security Evaluation

In this chapter, the security evaluation of the implemented contribution is presented. For the evaluation, we will take a simplified version of the Common Criteria for Information Technology Security Evaluation [5]. Section 5.1 will present the methodology of our security evaluation, followed by Section 5.2, which explains our use case. Section 5.3 will define the Target of Evaluation (TOE). Section 5.4 will proceed by explaining the security assumptions of the TOE. The following Section 5.5 will explain assets our TOE aims to protect. After finding all possible threats in Section 5.6 we will extract the risks in Section 5.7. By using our contribution as countermeasures in Section 5.8, we can conclude and identify residual risks in Section 5.9.

5.1 Methodology

The used methodology for our security evaluation is derived from the Common Criteria for Information Technology Security Evaluation [3, 4].

Our methodology starts by defining the TOE (i.e., CrySIL) the environment and the parties involved in our use case, in Section 5.2.

The security assumptions we assumed are the technologies used, cryptographic aspects, and behavior of involved parties. These assumptions have to be made to define some limits to our security evaluation.

After that, we will identify the assets our TOE aims to protect against malicious parties. Assets may contain sensitive data, e.g., user credentials or a secret key.

How malicious parties attack the TOE are called threats. These threats have to be identified and addressed. A threat to our assets may be a gateway, and therefore, the threat has to be dealt with. An identified threat which may expose one or more of our assets is a risk. We have to process all threats and assets to get a list of risks.

After that, we try to reduce and eliminate the identified risks by applying our countermeasures and security assumptions to risks. The outcome will be a list of residual risks. Residual risks are risks which were neither mitigated by our security assumptions nor by our countermeasures.

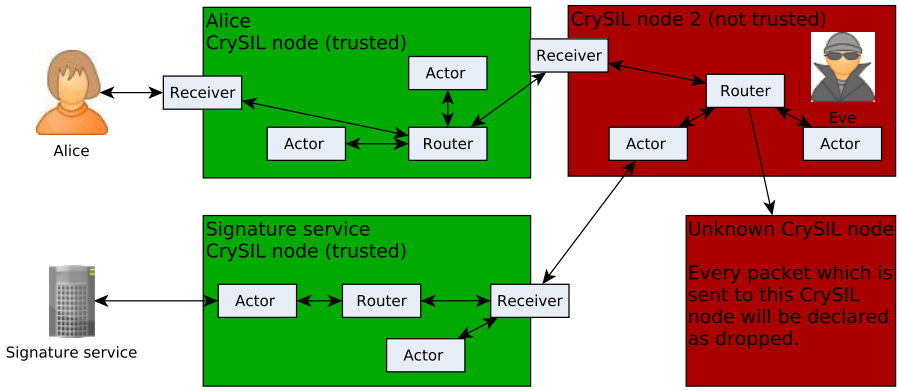

Figure 5.1: Security Evaluation Use Case

5.2 Use case

We will evaluate our approach by creating a use case, as seen in Figure 5.1. The use case consists of Alice, who wants to create digital signatures over CrySIL by using the signature service *ServerBKU*. To communicate with the ServerBKU, Alice has to connect to an unknown CrySIL network. This network will, in total consist of four CrySIL nodes. The first node is under the control of Alice and therefore considered as trusted. The second CrySIL node is Eve's CrySIL node and considered as not trusted. Eve is a malicious party and wants to steal Alice identity to be able to sign documents digitally on her behalf. The third CrySIL node is located at the ServerBKU. The ServerBKU uses its node to be able to receive commands over the CrySIL network. This node is considered as trusted. The fourth and last CrySIL node will drop all received data as this simulates a so-called "black hole".

5.3 Target of Evaluation

The Target of Evaluation (TOE) defines the scope of our security evaluation. The TOE will consist of all components of our use case in Section 5.2, with some security assumptions made in Section 5.4. Within the definition of the TOE are also the communication channels in-between the CrySIL nodes, as sensitive data is exchanged between Alice and ServerBKU.

5.4 Security Assumptions

The following security assumptions define the environment and configuration of the TOE.

1. (AS1) The deployed cryptographic algorithms are considered secure and correct. [21, 24, 22, 23, 17]

2. (AS2) The devices (i.e., Alice's devices: her computer and mobile phone) do not leak any information and are trustworthy, i.e., she has no virus, Trojan or keylogger installed. Alice has her devices configured properly, i.e., her key- and trust stores are set up correctly to trust the ServerBKU but not Eve and the passwords/PINs to access these devices are well chosen.

3. (AS3) The ServerBKU is considered to be secure. The ServerBKU will neither misbehave nor leak any sensitive data to a malicious party.

4. (AS4) The TLS setup was done correctly and only algorithms, considered as secure and correct, were chosen as available Ciphers. Furthermore, the certificates in the trust- and keystore used were checked whether they are trustworthy or not. That means that no secure connection to a malicious party will be possible.

5. (AS5) Alice behaves according to the guidelines of electronic signatures e.g., she uses a different device for the signature process and for receiving the OTP; double check the reference value against the value from the received OTP; have a look at the actual document to be signed within the secure viewer.

5.5 Assets

Assets may be data or other components within the TOE. The TOE should protect its assets using TOE security functions against an attacker.

1. (A1) *User credentials* (e.g., username and password): These credentials belong to Alice and are used to authenticate against the ServerBKU. Being able to gather these credentials is crucial for Eve as it creates the possibility to use these credentials for the *first factor* to authenticate against the ServerBKU.

2. (A2) *The second factor* of the two-factor authentication is, i.e., the OTP. As the ServerBKU uses a two-factor authentication before issuing digital signatures, the attacker must be able to gather the one-time password (OTP) sent to Alice's mobile phone.

3. (A3) The document to be signed (*DTBS*). The document itself may be sensitive, e.g., a business contract. The attacker must neither be able to eavesdrop nor modify the DTBS sent by Alice to the ServerBKU.

4. (A4) The *signature key*. If the signature key gets stolen by the attacker, Eve will be able to forge signatures on behalf of Alice without the need to steal Alice's credentials (A1) and the OTP (A2) or injecting a crafted document to be signed (A3).

5. (A5) Alice's *mobile phone*. Alice's has registered her mobile phone at the ServerBKU. Therefore the ServerBKU can send Alice OTPs.

5.6 Threats

Threats are attack vectors against one of the assets listed in Section 5.5.

1. (T1) *Eavesdropping* on the communication. Eve listens to the communication between Alice's CrySIL node and the ServerBKU's CrySIL node. Eve will try to read and manipulate the data in such a way that she profits the most.

2. (T2) Eve is *routing* packets to different CrySIL nodes. Eve is trying to provoke any misbehavior (from Alice or the CrySIL components) which might let her gather any information regarding an asset.

3. (T3) Eve may be able to trick Alice by using *social engineering,* e.g., a phishing email. If Alice falls for it, she might enter her credentials on Eve's malicious website.

4. (T4) Eve may inspect the *source code* of CrySIL and the ServerBKU. She hopes that to find a vulnerability which can be exploited to gain access to an asset

5. (T5) Eve may *steal* Alice's *mobile phone* to access OTP messages required for the two-factor authentication.

As listed, different threats require different attacks on the TOE. T1 and T2 require access to the network of the CrySIL nodes, which Eve has. T3 requires Eve to know or contact Alice directly via social engineering. T4 requires access to the internal, i.e., the source code of CrySIL and the ServerBKU. T5 requires Eve to be nearby Alice to steal her mobile phone.

We will consider all threats and the assets Eve may be able to gather or access in Section 5.7.

5.7 Risks

In this chapter, we are going to examine the list of threats and the list of assets. The outcome of this examination are risks. A risk is a threat which dangers an asset. We will take a look at all risks and conclude them at the end of this section with a table.

T1 Eavesdropping & A1 User Credentials If Eve can eavesdrop on the communication between Alice's CrySIL node and the ServerBKU's CrySIL node, Eve can gather Alice's credentials. Alice's credentials are the first factor of the two-factor authentication, which is needed by the ServerBKU for generating digital signatures. (R1)

T1 Eavesdropping & A2 Second Factor Eve will not be able to eavesdrop the second factor by listening to the communication between Alice's CrySIL node and the ServerBKU's CrySIL node. That is because the second factor is not sent over the same channel as the first factor. As the second factor, the ServerBKU sends an OTP message to Alice's mobile phone out of band (e.g., mobile network channel). Therefore this is not considered to be a risk.

T1 Eavesdropping & A3 DTBS If Eve can eavesdrop on the communication between Alice's CrySIL node and the ServerBKU's CrySIL node, she can gather the document Alice wants to sign. But if Eve can read the document by listening to the communication, she will be most likely also be able to modify and inject a crafted document to be signed. That enables Eve to get any document she wants to, be signed by Alice. (R2)

T1 Eavesdropping & A4 Signature Key If Eve can gather the signature key, she neither needs the first nor the second factor of the two-factor authentication to sign arbitrary documents on behalf of Alice. (R3)

T2 Routing & A1 User Credentials Eve might reroute messages sent between Alice's CrySIL node and the ServerBKU's CrySIL node to provoke some misbehavior, i.e., a message is sent to Eve's destination instead of Alice's destination. (R4)

T2 Routing & A2 Second Factor Even if Eve reroutes all messages, she will not be able to gather the second factor of the two-factor authentication because the second factor is sent to Alice's mobile phone. The second factor will not be transmitted over the same channel as the first factor. Therefore this is not considered to be a risk.

T2 Routing & A3 DTBS Eve might try to inject or modify the document to be signed by rerouting packets exchanged between Alice's CrySIL node and the ServerBKU's CrySIL node. (R5)

T2 Routing & A4 Signature Key If Eve can gather the signature key by rerouting the packets exchanged between Alice's CrySIL node and the ServerBKU's CrySIL node, she will be able to sign crafted documents on behalf of Alice. (R6)

T3 Phishing & A1 User Credentials Eve might perform a phishing attack on Alice to gather her credentials. The phishing attack may come in the form of an, e.g., email. Such emails are sent quite often, but most of them are filtered by a properly configured email server. But not every phishing email may get filtered, and therefore such a phishing email may reach Alice's mailbox. (R7)

T3 Phishing & A2 Second Factor Eve might perform the same phishing attack, from the previous paragraph, on Alice to gather the second factor. (R8)

T3 Phishing & A3 DTBS Eve might try to make Alice sign a crafted document by sending Alice the crafted document and demanding Alice's signature. (R9)

T3 Phishing & A4 Signature Key Eve might try to make Alice believe that she needs to send her signature key to Eve. (R10)

T3 Phishing & A5 Alice's Mobile Phone Eve might try to make Alice believe that she needs to bring her mobile phone to her (e.g., for a "software update", or any other bogus reason). (R11)

T4 Source Code & A1 User Credentials If the source code of CrySIL and the ServerBKU is open Eve can inspect the source code. She might find a vulnerability either in CrySIL or in ServerBKU. (R12)

T4 Source Code & A2 Second Factor Eve might inspect the source code of the ServerBKU to gather any information about the OTP generation, e.g., whether it is implemented securely or not. (R13)

T4 Source Code & A3 DTBS If an arbitrary document could be injected into the signature process at the ServerBKU, an attacker would be able to get any crafted document signed by Alice. (R14)

T4 Source Code & A4 Signature Key If there exists a vulnerability within the ServerBKU which would expose the signature key used by a user, i.e., Alice, it will compromise the whole signature process. Neither would Eve need Alice's credentials, nor the OTP sent to Alice's phone. Instead, Eve could use Alice's signature key directly and sign any document. (R15)

T5 Steal Mobile Phone & A1 User Credentials If Alice has her credentials stored on her mobile phone and it gets stolen by Eve, she could use those stolen credentials for the signature process. (R16)

T5 Steal Mobile Phone & A2 Second Factor If Eve possesses Alice's mobile phone, she might be able to read the OTPs from the ServerBKU. (R17)

T5 Steal Mobile Phone & A3 DTBS If Alice uses her mobile phone for the signature process and for receiving the OTP, Eve might be able to modify the document to be signed on the mobile phone. Altering the document-to-sign will lead Alice to sign a crafted document. (R18)

T5 Steal Mobile Phone & A4 Signature Key If the signature key would be on Alice's mobile phone and Eve possesses the mobile phone, Eve could use that signature key to sign crafted documents. (R19)

T5 Steal Mobile Phone & A5 Alice's Mobile Phone Stealing the mobile phone would accomplish acquiring the asset. (R20)

Summary As seen in Table 5.1, the threat T3 (phishing) and T5 (steal mobile phone) could have the most impact on security. Unfortunately, these threats cannot be counter-measured in any way. It is up to the user to not fall into such phishing attacks and to keep the mobile phone save.

Threats / Asset	A1	A2	A3	A4	A5
T1	R1		R2	R3	
T2	R4		R5	R6	
T3	R7	R8	R9	R10	R11
T4	R12	R13	R14	R15	
T5	R16	R17	R18	R19	R20

Table 5.1: Threats and Assets

5.8 Countermeasures

After defining the assets and threat, risks evolved. We will try to mitigate the risks by using the countermeasures we have created, and the assumptions we have made.

C1: Communication via TLS instead of Plain text Until now, CrySIL communication happened over an unencrypted channel. Messages were sent in plain text from sender to receiver and vice versa. That made an attack of the communication channel in the CrySIL network an easy one. As this problem is critical, we had to introduce a countermeasure. This countermeasure is described in Section 4.6 and called "Secure Transport". This end-to-end encryption may be used between every CrySIL nodes, and therefore no plain text messages will be exchanged anymore. This countermeasure C1 has the following impacts on the risks we have identified.

- R1: Eve is not able to extract Alice's credentials.
- R2: Eve is not able to modify or inject a crafted document which then would have been signed by Alice.
- R3: Eve would not be able to gather the signature key.
- R4: Rerouting or miss-routing will not work because Eve is not able to read the rerouted messages (because they are encrypted), and is not able to inject messages into the communication.
- R5: Eve will not be able to modify or inject a crafted document into the communication.
- R6: Eve will not be able to gather the signature key if Eve reroutes or misroutes packets and tries to provoke misbehavior.

Our countermeasure C1 eliminated five critical risks. We have to note that Eve is still able to gather messages exchanged between Alice's CrySIL node and the ServerBKU's CrySIL node, but Eve cannot read the messages because they are encrypted and Eve does not have the decryption key.

R7, R8, R9, R10, R11: Phishing If Eve wants to listen to the communication between Alice's CrySIL node and the ServerBKU's CrySIL node, Eve has to "bypass" the end-to-end encryption. That means Eve has to trick Alice into establishing a channel to her malicious service instead to the ServerBKU. Eve then plays "man in the middle" (MITM) and creates a channel to the ServerBKU to impersonate Alice.

This phishing attack is avoided by assumption AS2. Alice must have her key- and trust stores set up correctly. That way no encrypted channel can be established to Eve, as her certificates are not trustworthy as they will not be part of Alice's trust store.

Regarding R11. If Alice falls for the phishing attack, Eve might install some Trojan on Alice's mobile phone. This malicious software may forward any OTP received from the ServerBKU to Eve. This way, Eve could gather the OTP, which is the second factor of the ServerBKU. This attack is prevented by AS2. Her devices, i.e., mobile phone, do not leak any information.

R12, R13, R14, R15: Source Code Neither the ServerBKU nor CrySIL must leak any information, as assumed by assumption AS3. Therefore it is assumed that the implementation of CrySIL and the ServerBKU were made correctly.

Regarding R15: The signature key never leaves the hardware security module (HSM), and therefore never leaves the ServerBKU.

R16, R17, R18, R19, R20: Stealing Alice's Mobile Phone If Eve steals Alice's mobile phone, Eve might have access to it if Alice did not activate any protection, e.g., PIN, unlock password. This results in Eve having access to OTPs sent to Alice's mobile phone. Assumption A2 prevents this attack.

5.9 Conclusion

In this chapter, we have created a use case for CrySIL and the ServerBKU. We have made up assumptions as well we have identified assets and threats. By analyzing assumptions, assets, and threats, we have identified risks for the TOE.

Table 5.2 shows in the first column, all identified risks. The following columns denote which assumption (AS), or countermeasure (C) eliminated the risk. As we can see, there are no residual risks.

R / C, AS	C1	AS2	AS3	Residual Risks
R1	•			{}
R2	•			{}
R3	•			{}
R4	•			{}
R5	•			{}
R6	•			{}
R7		•		{}
R8		•		{}
R9		•		{}
R10		•		{}
R11		•		{}
R12			•	{}
R13			•	{}
R14			•	{}
R15			•	{}
R16		•		{}
R17		•		{}
R18		•		{}
R19		•		{}
R20		•		{}

Table 5.2: Risks, Countermeasures, Assumptions

6 Future Work

Our proposed approaches to open problems met the requirements initially set. During the research and development of our approaches, we came across different, initially undiscovered problems. In this chapter, we are going to discuss briefly what those problems are and how we may address them in the future.

Routing Section 4.4.1 states the requirements for routing inside a CrySIL network.

Requirement R1 (an actor must only be linked to one receiver/service) together with requirement R2 (a router takes requests only from one receiver) is quite harsh. It restricts the network topology of CrySIL networks and therefore makes offering services much more complicated. If these restrictions could be lifted, the routing concept must be updated as well.

DiscoverService In Section 4.3.2, we explained how routing information is inserted in the *DiscoverService* request, i.e., that every actor has to attach this unique name into the path information until the desired actor is reached. This way we get end-to-end path information, from a sender (e.g., a user) to an actor (e.g., a service).

Once the desired *actor* has been reached, it will create a *DiscoverService* response and uses JWS to sign the *DiscoverService* response. One attribute of the *DiscoverService* response is the path information collected so far. Due to the JWS, any modification in *DiscoverService* response will be detected, which means that the path element in the *DiscoverService* response is the actual path from the sender to the desired actor.

The open problem is, that the "path element" received from the actors might have been altered from a malicious identity. Once altered, the path from the sender to the receiver will be incorrect, and this will result in an incorrect routing. Because actors do not have any routing information, they cannot detect a malicious modification to the path element. Until now, only routers have path information, namely the name of the actors attached to them. That means that actors will blindly sign any path element and assume it is correct, which may not be the case.

To reduce the risk of not detecting a modified path element from a malicious entity could be that routers have to sign (e.g., JWS) path elements received in DiscoverService requests and attach the signature to the DiscoverService request before sending it further. *Actor*s job change a bit because now they have to copy the JWSs from *DiscoverService* request and attach them to *DiscoverService* responses.

Senders are now able to validate signatures for every router and see if the path elements attached to that particular router signature match the path elements from the previous router signature. Unfortunately, that approach requires lots of calculation power along the path for the routers, which could lead to traffic congestions due to the hold up caused by creating a signature and attaching it to the *DiscoverService* requests.

7 Conclusion

With the continuous rise in the number of mobile phones, more and more applications move from desktop computers to mobile phones. Apart from writing emails, scheduling meetings, and exchanging photos with messengers, an important component is still missing.

The need to sign documents on mobile phones by using an external signature service which offers qualified digital signatures is still a missing feature. It would enable users to sign documents which can be processed by official channels by just using a mobile phone. Existing signature solutions do exist but are either bound to specific companies, are not free of charge or may not be integrated into mobile applications quickly.

In this master thesis, we proposed an approach which makes it possible to use a signature service called ServerBKU [33], which is similar to the Austrian mobile phone signature. This approach enables CrySIL supported mobile phones [34] the possibility to sign documents on mobile phones digitally, which will significantly speed up the communication between users and official channels.

Also, we've proposed end-to-end encryption similar to IPSec's ESP in Tunnel mode [7]. This similar approach makes eavesdropping on data in CrySIL networks in between CrySIL nodes futile. Especially for a signature service, it is essential that user credentials do not get stolen. If an attacker steals user credentials, it would be possible for the attacker to sign documents on behalf of the user.

A *DiscoverService* command was proposed which can be broadcast throughout the CrySIL network to get service and routing information. Service information includes information about each service within the CrySIL network, e.g., if the service supports end-to-end encryption or which commands a service does support and where the service is located.

The routing information gathered can be used to route the commands through a more complex CrySIL network. This routing information ensures that all services attached to a CrySIL network can be reached.

All in all, these approaches enable users to sign documents by just using their mobile phones. As these documents were signed by using a digital certificate, these documents can be processed by official channels. Out approaches speed up the communication with official channels and help reduce costs.

Acknowledgments

(Appearing names not in any order)

This work, which (unfortunately) lasted longer than expected, would not have been possible without the help of lots of people.

Peter Teufl and Florian Reimair for their assistance, my girlfriend and daughters for being patient, my parents for their sponsorship during study and being patient, many people I have become friends with during my studies. Thank you all!

Bibliography

[1] Activate your citizen card, retrieved 4th march 2016, http://www.buergerkarte.at/en/activate-card.html.

[2] Android encryption, retrieved 17th march, https://support.google.com/nexus/answer/2844831?hl=en.

[3] Common criteria for information technology security evaluation, retrieved 31th january 2018, https://www.commoncriteriaportal.org/files/ccfiles/ccpart1v3.1r4.pdf.

[4] Common criteria for information technology security evaluation, retrieved 31th january 2018, https://www.commoncriteriaportal.org/files/ccfiles/ccpart2v3.1r5.pdf.

[5] The common criteria, retrieved 31th january 2018, https://www.commoncriteriaportal.org/.

[6] Ip authentication header, https://tools.ietf.org/html/rfc4302.

[7] Ip encapsulating security payload, https://tools.ietf.org/html/rfc4303.

[8] Ipsec - internet protocol security, https://tools.ietf.org/html/rfc4301.

[9] Json web signature (jws), https://tools.ietf.org/html/rfc7515.

[10] Microsoft lumia 950 - windows hello beta, https://www.microsoft.com/en/mobile/phone/lumia950-xl-dual-sim/specifications/.

[11] Shipments of smartphones with fingerprint sensors worldwide from 2014 to 2018 (in million units), retrieved 12th may 2016, http://www.statista.com/statistics/522055/global-smartphone-fingerprint-shipments/.

[12] The transport layer security (tls) protocol version 1.2, https://tools.ietf.org/html/rfc5246.

[13] Verycrypt - https://veracrypt.codeplex.com/.

[14] www.buergerkarte.at, retrieved 25th march 2016, https://www.buergerkarte.at/en/index.html.

[15] www.help.gv.at, retrieved 4th march 2016, https://www.help.gv.at/portal.node/hlpd/public/content/impressum /seite.731300.html.

[16] Thomas Rössler Martin Centner Arno Hollosi, Gregor Karlinger. Die österreichische bürgerkarte, https://www.buergerkarte.at/konzept/securitylayer/spezifikation/20140114/. online, 01 2014.

[17] European Payments Council. Guidelines on cryptographic algorithms usage and key management, https://www.europeanpaymentscouncil.eu/sites/default/files/kb/files/epc342-08online, 12 2016.

[18] J. Degabriele and K. Paterson. Attacking the ipsec standards in encryption-only configurations. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 335–349, May 2007.

[19] Martin MA Devillers. Analyzing password strength. *Radboud University Nijmegen, Tech. Rep*, 2010.

[20] Johannes Feichtner. Static analysis of cryptography in android applications. Master's thesis, University of Technology, Graz, May 2015.

[21] Bundesamt für Sicherheit in der Informationstechnik. Kryptographische verfahren: Empfehlungen und schlüssellängen: https://www.bsi.bund.de/shareddocs/downloads/de/bsi/publikationen/ technischerichtlinien/tr02102/bsi-tr-02102.pdf. online, 01 2017.

[22] Bundesamt für Sicherheit in der Informationstechnik. Kryptographische verfahren: Empfehlungen und schlüssellängen (ipsec, ikev2), https://www.bsi.bund.de/shareddocs/downloads/de/bsi/publikationen/ technischerichtlinien/tr02102/bsi-tr-02102-3.pdf. online, 01 2017.

[23] Bundesamt für Sicherheit in der Informationstechnik. Kryptographische verfahren: Empfehlungen und schlüssellängen (ssh), https://www.bsi.bund.de/shareddocs/downloads/de/bsi/publikationen/ technischerichtlinien/tr02102/bsi-tr-02102-4.pdf? online, 01 2017.

[24] Bundesamt für Sicherheit in der Informationstechnik. Kryptographische verfahren: Empfehlungen und schlüssellängen (tls), https://www.bsi.bund.de/shareddocs/downloads/de/bsi/publikationen/ technischerichtlinien/tr02102/bsi-tr-02102-2.pdf. online, 01 2017.

[25] et. al. Javier Galbally. From the iriscode to the iris: A new vulnerability of iris recognition systems. 2012.

[26] Young-Hoo Jo, Sung-Yun Jeon, Jong-Hyuk Im, and Mun-Kyu Lee. *Vulnerability Analysis on Smartphone Fingerprint Templates*, pages 71–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[27] H. Leitold, A. Hollosi, and R. Posch. Security architecture of the austrian citizen card concept. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 391–400, 2002.

[28] Clemens Orthacker, Martin Centner, and Christian Kittl. Qualified mobile server signature. In *Security and Privacy–Silver Linings in the Cloud*, pages 103–111. Springer, 2010.

[29] Clemens Orthacker, Martin Centner, and Christian Kittl. *Security and Privacy – Silver Linings in the Cloud: 25th IFIP TC-11 International Information Security Conference, SEC 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings*, chapter Qualified Mobile Server Signature, pages 103–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[30] Bundeskanzleramt Österreich. Xml definition of the person identity link, https://www.buergerkarte.at/konzept/personenbindung/spezifikation/20050214/personenbindung-20050214.en.pdf.

[31] Kenneth G. Paterson and Arnold K. L. Yau. *Advances in Cryptology - EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006. Proceedings*, chapter Cryptography in Theory and Practice: The Case of Encryption in IPsec, pages 12–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[32] Prabaharan Poornachandran, M. Nithun, Soumajit Pal, Aravind Ashok, and Aravind Ajayan. *Password Reuse Behavior: How Massive Online Data Breaches Impacts Personal Data in Web*, pages 199–210. Springer Singapore, Singapore, 2016.

[33] C. Rath, S. Roth, M. Schallar, and T. Zefferer. A secure and flexible server-based mobile eid and e-signature solution. In *The Eighth International Conference on Digital Society*, pages 7–12, 2014.

[34] Florian Reimair, Peter Teufl, Christian Kollmann, and Christoph Thaller. Mocrysil - carry your cryptographic keys in your pocket. In *12th International Conference on Security and Cryptography*, pages 285 – 292, 2015.

[35] Florian Reimair, Peter Teufl, and Thomas Zefferer. Webcrysil - web cryptographic service interoperability layer. In *11th International Conference on Web Information Systems and Technologies*, pages 35 – 44, 2015.

[36] Thomas Zefferer, Vesna Krnjic, Klaus Stranacher, and Bernd Zwattendorfer. *Measuring Usability to Improve the Efficiency of Electronic Signature-based E-Government Solutions book title: Measuring E-government efficiency. The opinions of Public Administrators and other Stakeholders*, pages 45 – 74. 2014.