Florian Reimair, Peter Teufl, Thomas Zefferer - "WebCrySIL - Web Cryptographic Service Interoperability Layer",
11th International Conference on Web Information Systems and Technologies (WebIST), 2015

WebCrySIL
Web Cryptographic Service Interoperability Layer

Florian Reimair[1], Peter Teufl[1] and Thomas Zefferer[1]

[1]*Institute for Applied Information Processing and Communications, Graz University of Technology, Inffeldgasse 16a, Graz, Austria*

{florian.reimair, peter.teufl, thomas.zefferer}@iaik.tugraz.at

Keywords: cloud security, central cryptographic solutions, advanced cryptographic protocols, heterogeneous applications, mobile devices

Abstract: Today's applications need to work with a heterogeneous collection of platforms. Servers, desktops, mobile devices, and web browsers share data and workload. Many of these applications handle sensitive data or even have security as their core feature. Secure messaging, password storage, encrypted cloud storage applications or alike make use of cryptographic algorithms and protocols. These algorithms and protocols require keys. The keys in turn have to be provisioned, securely stored, and shared between various devices. Unfortunately, handling the keys and the availability of cryptographic APIs evokes non-trivial challenges in current heterogeneous platform environments. Also, the implementation of APIs supporting cryptographic protocols on arbitrary platforms require significant effort, which is a major challenge when new cryptographic protocols become available. Our approach, the Crypto Service Interoperability Layer (CrySIL), enables applications to securely store/use/share key material and supports a wide range of cryptographic protocols and algorithms on heterogeneous platforms. CrySIL complements existing solutions that mitigate the aforementioned problems through central services by allowing for more flexible deployment scenarios. In this work, we explain the motivation of CrySIL, describe its architecture, highlight its deployment in a typical heterogeneous application use case and reflect on achievements and shortcomings.

1 INTRODUCTION

In recent years, a highly heterogeneous platform landscape has become the standard scenario for the development and deployment of applications. Especially mobile device platforms introduced an unprecedented level of architectural heterogeneity. Modern web browsers support rich HTML5-based applications, well known mobile devices are here to stay, new wearable computing systems claim their place in the world and classic desktop operating systems still stand their ground.

Applications present the user with use cases that became available only recently. Developing applications for today's and tomorrow's diverse environments boosts complexity of the development processes significantly. Especially security-related aspects – the deployment of cryptographic algorithms and protocols to offer confidentiality, integrity and authenticity for the processed data – suffer from this complexity in several ways: First, cryptographic algorithm and protocol implementations are not available on every platform. Second, the capabilities of different platforms to handle the required key material in a secure way strongly deviates. Third, the high level of complexity causes implementation mistakes that lead to significant security issues (Egele et al., 2013; Fahl et al., 2012).

While existing and well-established solutions, such as smart cards and cryptographic tokens offer a way to solve the challenge of key storage, distribution, and handling, they are only available on limited platforms and cause additional cost and usability issues when being integrated in heterogeneous applications. Platform-based systems, i. e. TPMs (Trusted Computing Group, 2011), mobile TPMs, or proprietary solutions offer reasonable security, given that the security features are activated and configured accordingly. These system seldom share common configuration and features and therefore have to be configured and used one by one. The challenge is to get it right, every time. The challenge becomes even greater when multiple platforms with different hardware or software-based encryption systems are to protect sensitive data. These examples represent only a small excerpt of the entire problem space and are augmented by the fact that the availability of cryptographic APIs is strongly platform-dependent. E.g. when considering the web browser within an application scenario, one has to keep in mind that hardly any cryptographic API nor a secure key storage component is available.

While it might seem reasonable to hold confidential and secret cryptographic material directly at the user's device and refrain from storing these data on a central server, a severe drawback emerges: If stored locally, cryptographic material is not available everywhere and at any time. For instance, access to e-government services is infeasible in case the solicited smart card-based user authentication is impossible because the end-user device does not provide appropriate card-reading capabilities. In times, in which mobile and wearable devices are of growing importance, this is becoming a significant issue.

The industry created central cryptographic service to meet the anywhere and at-any-time requirements. We reviewed some solutions and found that their limited scope make them only suitable for specific use cases and deployment scenarios. For example, the Austrian Mobile Phone Signature solution is limited to the creation of electronic signatures, but does not feature cipher operations. The CloudHSM solution provided by Amazon is limited in terms of possible deployment scenarios. In particular, this solution is tied to a Public Cloud solution and to one specific Cloud provider. Similar limitations apply to comparable server-based solutions that offer storage of cryptographic material and central provision of cryptographic services. Furthermore, these solutions do not use or include locally available key material and cryptographic functions, which still might play an important role in many use cases.

To overcome limitations of existing solutions and to further improve the availability of cryptographic services everywhere and at any time, we propose the Crypto Service Interoperability Layer (CrySIL). CrySIL provides a flexible architecture to use cryptographic protocols and algorithms in a heterogeneous environment and provides secure key storage and key handling capabilities. It features build-in authentication as well as transparent off-device key storage. When used as central service CrySIL can also be seen as a cryptographic platform for the rapid deployment of new cryptographic protocols, which are not yet supported by hardware and software solutions. This feature is especially important for upcoming advanced cryptographic protocols that will play an important role in cloud computing security.

The remainder of this paper is organized as follows. In Section 2, the motivation for CrySIL is outlined by discussing the current deployment of cryptographic algorithms and functions in heterogeneous application environments. Related work in Section 3 is followed by a detailed description of the CrySIL architecture in Section 4. Subsequently, in Section 5 the functionality and security challenges of CrySIL are explained by investigating the prototype of browser application for platform-independent file-encryption. Finally, the work is concluded by giving an outlook on future CrySIL plans and deployment scenarios.

2 BACKGROUND

The following deployment scenarios and application categories highlight the manifold use of cryptographic algorithms and protocols and the associated problems of deploying such technologies in heterogeneous application environments.

In recent years mobile platforms have provoked a rush to applications that allow users to access and process their data on a wide range of devices. Nowadays a heterogeneous application comes in versions for desktop-based systems, different mobile device platforms and web browsers. In addition, specific environments, such as the Chrome Apps platform[1], further specific mobile environments (e.g., tablets, phablets, smartphones) and vendor-specific aspects need to be covered. In recent years the first mass market wearable devices have been released and are expected to gain significant usage numbers in the near future (IDC, 2014). While this heterogeneous device landscape offers significant new opportunities for customers, it comes with a major price tag – complexity. Developing for so many platforms requires different user experience design (UXD) strategies, different platform-specific programming-language and architecture skills and – especially important for the scope of this work – in depth knowledge on how to provide confidentiality, integrity and authenticity for the processed data. Protecting the application data is especially relevant in the context of exchanging this data via cloud infrastructures to provide instant access on heterogeneous application platforms. Noteworthy examples for such applications are note-taking applications, messengers, applications for collaborative document processing, or cloud storage providers. Obviously, the secure handling of this data needs to be considered in the local environment as well as when the data is transferred and stored at the cloud provider. To offer the required level of protection, cryptographic algorithms and protocols are deployed.

[1] https://developer.chrome.com/apps/app_architecture

In a similar way, applications that advertise security features as core functionality require the deployment of cryptographic algorithms and protocols. Examples for this application category are secure messengers, encrypted cloud storage solutions or password managers. On a high-level view those applications face the same challenges as the heterogeneous applications described above. However, the security requirements for those applications are typically more complex due to the higher likelihood of targeted attacks.

Security and especially the deployment of cryptographic functions are core aspects of enterprise-level applications. VPN solutions based on IPSEC or TLS are widely deployed to guard network communications, S/MIME or PGP are required to ensure the confidentiality, authenticity and integrity of emails. While these communication systems are based on well established protocols, a rather new concept – digital signatures to create authentic documents – is not. Legal frameworks (e.g., (Parliament and Council, 2000)) have established the basis for the legally binding use of digital signatures. The applications of advanced digital signatures within private and corporate perspectives are manifold: signing legally binding documents (e.g. contracts), automatically signing documents (e.g. invoices), or providing authenticity and integrity for stored documents in general. The security requirements for enterprise-level applications mostly exceed the requirements for applications for personal use due to much higher security requirements.

All in all, the wide deployment of cloud computing in recent years has brought up many issues in relation to security, privacy and data protection laws. E.g. using U.S. cloud computing resources in Europe is linked to significant legal issues (van Hoboken et al., 2012). One technical approach to stand these challenges is to deploy data encryption mechanisms. However, when using current cryptographic schemes, data needs to be encrypted directly at the client before it is stored at the cloud service provider. That, in fact, keeps the data protected while at rest but makes one of the main advantages of cloud computing – online data processing – much harder. Current research, however, focuses on new cryptographic protocols and algorithms capable of processing encrypted data in the cloud. The required functionality is brought – among others – by new cryptographic protocols and algorithms in the areas of homomorphic encryption (Naehrig et al., 2011), searchable encryption (Bellare et al., 2007), verifiable encryption (Camenisch and Shoup, 2003), proxy re-encryption schemes (Ateniese et al., 2006) or redactable signature schemes (e.g, (Hanser and Slamanig, 2013)). There is a high variety in the applicability of these new protocols in production environments. While in certain cases the schemes are not yet suited for practical applications (e.g. homomorphic encryption), other schemes such as proxy-re-encryption have already reached a prototypical stage (e.g., the NICS CRYPT Library[2]). However, even protocols from the latter category cannot yet be efficiently deployed in existing applications due to the lack of compatibility with current high level standards and the lack of hardware and software support on various platforms.

By considering the current challenges related to platform support and secure key storage facilities, the following conclusions on using cryptography in heterogeneous applications can be drawn: Due to the presence of these challenges, heterogeneous applications either do not support cryptographic functions, face usability issues, or suffer from implementation weaknesses due to the wrong use of cryptographic primitives and methods. Furthermore, certain platforms currently do not have any reasonable support for cryptographic algorithms and key storage facilities at all. And that is where novel solutions are needed.

3 RELATED WORK

A number of server-based cryptographic services have been implemented by the industry. This section gives an overview and functional evaluation of selected cryptographic services that can be integrated into cloud-based environments.

SigningHub[3] offers the creation of advanced digital signatures with unique cryptographic keys for different users. As a cloud-based service, it provides centrally-stored keys as well as signature creation using keys stored on smart cards or soft tokens. Signed documents are stored and managed on servers provided by *SigningHub*. The service can be easily integrated into applications and web services using a simple REST-based interface.

Dictao[4] and *Cryptomathic*[5] support digital signatures for transaction security and user authentication. Both services facilitate key access by authenticating clients using simple credentials, such as username/password schemes, or credentials that feature higher strength (eID cards, OTPs, mobile devices etc). While *SigningHub* and *Cryptomathic* are deployed

[2] https://www.nics.uma.es/dnunez/nics-crypto

[3] http://www.signinghub.com

[4] https://www.dictao.com

[5] http://www.cryptomathic.com

as cloud services, *Dictao* requires integration into an enterprise IT infrastructure. Basically, the provided functionality is limited to user authentication and signature creation.

The Austrian citizen card (Leitold et al., 2002), which represents the official eID in Austria, is a technology-neutral concept for unique citizen identification and secure qualified signature creation. Aside from smart card-related implementations, a cloud-based service is operational. The so-called *Austrian Mobile Phone Signature*[6] is operated in a private cloud and uses a hardware security module (HSM) to store the private signature keys of all Austrian citizens. Access to these keys is protected by a strong two-factor authentication mechanism, involving a password as well as an OTP being sent to the citizen's mobile phone. Applications can access the Austrian Mobile Phone Signature and its signature creation functionality through a well-defined XML-based interface. Although the Austrian Mobile Phone Signature meets the demands of the law, the currently deployed implementation fails to support use cases other than signature creation and user identification.

With *AWS CloudHSM*[7], Amazon feeds the demand of integrating secure cryptographic operations into deployed applications without requiring an HSM available on premise. To meet regulatory requirements for data security, customers are able to acquire sole access to appliances on a dedicated HSM and therefore retain full control of the keys and the cryptographic operations of the HSM. The offered functionality of the HSM can be integrated into applications that are deployed within the Amazon Virtual Private Cloud (Amazon VPC) via the provided Java or C programming API. As *AWS CloudHSM* can only be used in conjunction with Amazon VPC, customers are bound to Amazon and a migration to other cloud providers is infeasible.

The content delivery network *CloudFlare*[8] elaborated a solution that enables website visitors to establish a TLS connection with a *Cloudflare* server while retaining the private key on the server of a customer. The technique, entitled *Keyless SSL*, is transparent to a website visitor and takes place between a *Cloudflare* server, acting as a reverse proxy, and the key server where the private key resides. The *Cloudflare* server is fully capable of negotiating the TLS connection but shifts the generation of the signatures for the client random, server random, and public key certificate to the customer side. While this concept ascertains that *Cloudflare* servers have no access to the private key, they still know the negotiated session keys and are thus able to read and write any data flowing between the website visitor and a target web server. As a consequence, end-to-end security is undermined and the overall benefit derogates to having out-sourced all operations involving the private key.

4 CRYPTO SERVICE INTEROPERABILITY LAYER

Motivated by the unsolved challenges and already available solutions, we have created the Crypto Service Interoperability Layer (CrySIL). CrySIL creates an interoperability framework that allows the user access to his key data regardless where the key resides.

In short, the user takes one of her devices and launches an application to perform some cryptographic task. The application interfaces with the interoperability layer, CrySIL, which connects to another device. This other device has access to the actual cryptographic primitive, creates and validates authentication challenges if required and performs the requested operation. The result is returned to the interoperability layer and back to the application running on the device of the user. A graphical illustration of the workflow is given in Figure 1.

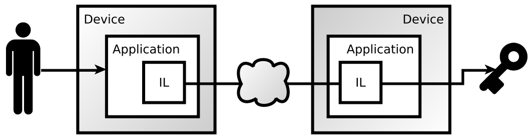

Figure 1: CrySIL's basic architecture

The flexible design of CrySIL allows for numerous deployment scenarios. First and foremost, the device having access to the key material can be the very same device the user interfaces with. Thus, CrySIL fulfils the use case requirements of using local cryptographic services – as provided by smart cards. Further, the device can be a cloud service provider offering the service to a broad range of users. This scenario reflects the central service paradigm as seen in industry solutions. A rather novel deployment scenario possible with CrySIL is to move the central cloud-based service to a user's mobile device. This scenario is motivated by the relative high level of security that can be offered by a mobile device when encryption systems and access protection systems are activated and correctly configured.

[6] `https://www.handy-signatur.at`

[7] `https://aws.amazon.com/cloudhsm/`

[8] `https://www.cloudflare.com`

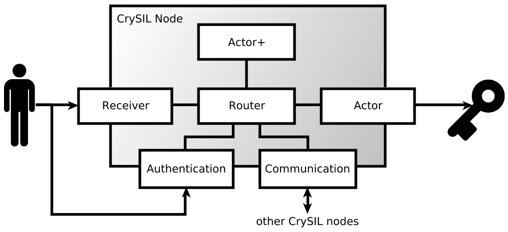

Figure 2: CrySIL node architecture overview

4.1 Interoperability Layer Node

The heart of our CrySIL approach is the interoperability layer node (denoted as IL in Figure 1). The modular node design enables the flexibility and extensibility of our approach while keeping the overall architecture simple.

Most conventional cryptographic service providers receive commands, perform the required actions, and return the result to the caller. To achieve CrySIL's interoperability goal, CrySIL breaks the classic cryptographic provider apart. The resulting modules have different jobs and work together to form the actual cryptographic service provider. The most visible modules are command *receivers* and the modules which act on cryptographic primitives – *actors*. Another crucial module is responsible for connecting *receivers* and *actors*. Other modules handle inter-node communication, protocol mappings, advanced crypto and authentication. All are considered as building blocks and are not restricted to any technology, platform, or programming language. An illustration of modules and their interconnections is given in Figure 2.

A *receiver* offers a set of cryptographic functions to the application developer. Being a design concept, a *receiver* can be implemented to run on any device, any platform, and any technology. A *receiver* does not perform any cryptographic operations but interfaces with the routing module after having the command encoded as interoperability protocol command. A realisation of a receiver can be cryptographic APIs on a programming language level like JCE, CSP, or the W3C Web Cryptography API. Another realisation can serve as a web-service providing a SOAP-based cryptographic interface. Having a realisation that interfaces with clients of the PKCS#11 standard or PC/Smart Card daemons (PCSCd) can bring remote key storage capabilities to existing applications.

An *actor* makes the contents of a specific key provider available to the interoperability layer. A key provider hosts key material and performs the actual cryptographic operation. The *actor* can be implemented to connect arbitrary key providers to CrySIL. Sample key providers are smart cards attached to a PC, USB-Tokens, Software Security Modules (SSMs), as well as Hardware Security Modules (HSMs). An *actor*+ might provide high-level cryptographic methods such as CMS encryption or XMLDSIG signatures while using the cryptographic primitives of other *actors*.

The central routing module – the *router* – receives commands from the *receiver* modules and assigns them to the appropriate *actor* modules. CrySIL supports a many-to-many relationship between *actors* and *receivers* in a completely transparent way.

4.2 Inter-Node Communication

The use cases ask for cryptographic primitives and services to be available anywhere and at any time. CrySIL's answer is transparent off-device cryptography, which mandates inter-node communication. Off-device cryptography allows a device that is not capable of doing a certain cryptographic operation on its own to use the cryptographic engine of a remote service. The device therefore can do the operation at the cost of having to trust the remote service.

The *communication* modules are in charge of inter-node communication. They simply take a request and send it to another off-device communications module. The most basic implementation is HTTP(s). Yet, arbitrary transport protocols, such as HTML5 PostMessage, Web-sockets, or IPSec are suitable.

Putting the pieces together, CrySIL renders off-device crypto completely transparent to the user, the developer, and to the application while maintaining a simple architecture. With inter-node communcation, there can be one or multiple nodes per device. The resulting architecture is depicted in Figure 3. Our approach enables almost complete key flexibility. A user benefits from her ability to use a variety of different keys provided by different key providers from a variety of applications on different devices.

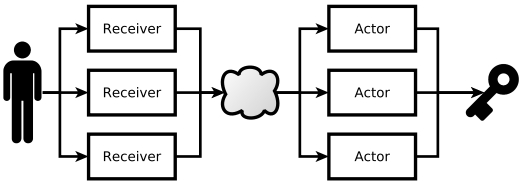

Figure 3: Interoperability architecture view

4.3 Authentication

Whenever cryptographic keys are used by an application, there must be access and usage policies in place. For local deployments, the simplest policy is that everyone can use the key. Similarly, a policy might require a PIN code for a local smart card. Such rather simple policies can be enforced by the device or the operating system. However, the cross-device/inter-node key access feature of CrySIL asks for a policy enforcement system that meets the requirements beyond in-device solutions.

As CrySIL can interface with key providers that may already require for authentication information, CrySIL has to collect and provide the authentication data. Within CrySIL only the *actor* knows about the authentication requirements of its key provider and has to challenge the user accordingly. CrySIL can support simple PIN challenges over external identity providers (OAuth, OpenID Connect) to multi-factor authentication methods. As for standing the above mentioned challenges, the CrySIL infrastructure gathers authentication data from the user as well.

The authentication modules offered by CrySIL are organized in a flexible and extensible manner. Different authentication modules can handle different authentication concepts from a simple PIN query over OpenID Connect to strong two-factor authentication systems. A developer can use any service with any authentication mechanism but does not have to bother with authentication concerns. Applications can therefore offer strong authentication mechanisms with little extra effort. Furthermore, authentication modules are designed to keep sensible credentials away from the *receiver* and therefore from the application. A malicious application therefore might have a harder time to eavesdrop or attack the credentials which results in an overall security boost.

Last but not least, the interoperability layer protocol foresees the use of session information. The session feature allows an *actor* to create one of the established session management systems to allow sessions with lifetime exceeding that of a single request. An application can therefore use an authorized key a number of times before he has to re-authenticate again. That might enable the user to accept stronger authentication as a hurdle during the key unlocking procedure.

5 EVALUATION

In order to demonstrate and evaluate our approach, we have implemented a number of prototypes in the course of our research. The focus lies on evaluating the flexibility and combinability of the CrySIL building blocks in order to solve different use cases.

In this section, we will describe our protected-data-at-rest prototype in terms of features, deployment scenarios, practical applicability, and benefits over other solutions. The prototype addresses the scenario where a user gets some data from a friend and wants to use the data on multiple devices. The objective is that the data is encrypted whenever it is not on a device owned by the user or the friend. The scenario is depicted in Figure 4. The setup is done with state-of-the-art communication and crypto. Trust relationships as well as attack vectors are well known from any hardware security module deployment scenario. Therefore, a thorough security analysis would not yield new information on security and trust issues, and is therefore omitted.

5.1 Deployment Scenario

The users, denoted as user 1 and user 2, have no profound understanding of cryptography and are using web browsers on PCs and mobile devices. It is assumed that for users who are no experts in the field of IT security, it is too much of a hurdle to perform manual key exchange in a secure way.

The storage solution used in the scenario is defined to be some shared storage solution with instant sharing. For example, a public cloud storage service like Dropbox[9]. This solution solicits no manual exchange of the data like it would be when using technologies like instant messaging or electronic mail and therefore keeps the prototype clean and easy to understand.

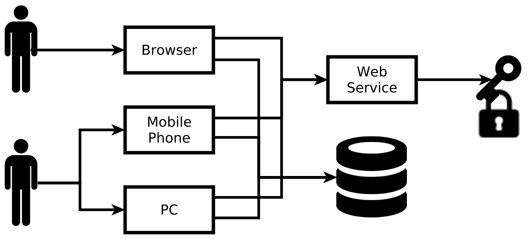

Figure 4: Prototype: protected-data-at-rest deployment scenario

On either device, be it the phone, the tablet, or the PC, a web browser runs an HTML5 and JavaScript browser application. The application encrypts arbitrary data before it submits the data to the shared storage service with the help of the W3C Web Cryptog-

[9] https://dropbox.com

raphy API. In this case, the API is implemented by JavaScript version of a CrySIL node. This node is referred to as the application's CrySIL node later on. Hiding behind the Web Crypto API, the application's CrySIL node offers transparent access to a remote key service over HTTPS.

The key service – acting as trusted third party – is a Java 8/Spring 4 web application hosted on a Tomcat server an referred to as key service CrySIL node later. An HTTPS communication module offers remote access. An *actor*+ capable of handling CMS containers is available as well as an *actor* interfacing with a software key store. Key store access is constraint by the *actor* as follows:

- To retrieve the certificate of the key of user 1, it is sufficient to provide the correct identifier of the user.
- Using the key for decryption purpose, however, requires 2-factor authentication with a username/passphrase tuple in the first place and a mobile TAN[10] as second factor.

To keep the prototype simple, we settled with some less secure implementations. A hardware key store (provided by an HSM) for the key service would bring a major boost to the overall security. However, for the sake of time and simplicity, the current implementations are based of software key stores.

5.2 Use Case

User 2 starts the browser application and add some data that he wants to share with user 1. Before the browser application sends the data to the cloud storage, it interfaces with the CrySIL infrastructure to encrypt the data. This is where the browser application hands over the control flow to the application's CrySIL node. The flow is only returned to the browser application when the encrypted data arrives. An illustration of the control flow is given in Figure 5.

Being asked to encrypt data, the application's CrySIL node asks user 2 to select a certificate that should be used to encrypt the data. User 2 selects the key service and the application's CrySIL node requests a list of certificates from the key service CrySIL node. In any case, the key store actor of the key service CrySIL node answers with an authentication challenge, demanding an identifier. The challenge is picked up by the appropriate authentication module within the the application's CrySIL node. The module interfaces with the human – user 2 – and shows a graphical user interface where it asks for the required identifier of user 1. User 2 provides the identifier. The authentication module sends the information back to the key service CrySIL node which answers with the certificate of user 1. Note, that the authentication information has never been processed by the browser application itself.

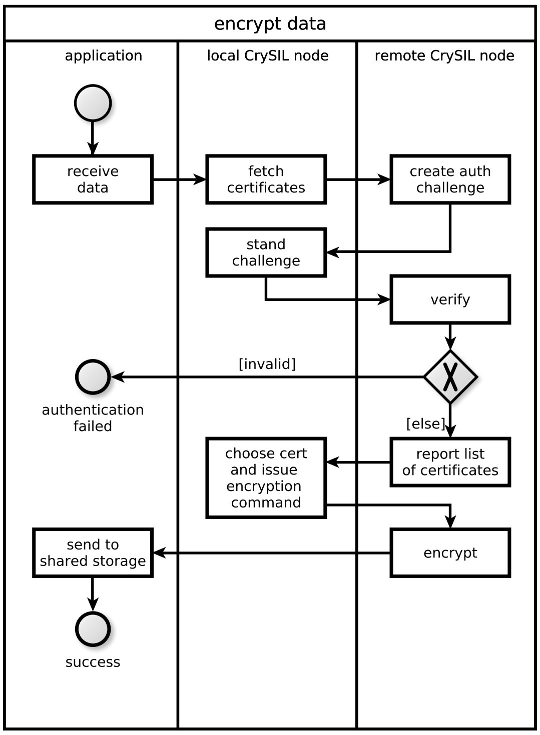

Figure 5: Prototype encrypt command flow

The the application's CrySIL node now sends another request to the CrySIL infrastructure and asks for encrypting the data with the just retrieved certificate using the CMS standard. The *router* of the the application's CrySIL node decides that it has no means of doing CMS locally and therefore forwards the whole request to the key service CrySIL node. The key service node receives the request, its *router* forwards it to the CMS *actor*+ which in turn creates a CMS container from the supplied data and returns it to the *receiver* of the application's CrySIL node within the browser application. Although sending a whole document causes communication overhead, this example highlights that a platform (the web browser) lacking the required cryptographic APIs is still able to cre-

[10] Mobile transaction numbers (TANs) are a two-factor authentication process where the user proofs knowledge of a secret and posession of a device, i.e. a mobile phone. The secret is used to identify the user and therefore the mobile phone. Then, a nonce is sent to the phone. The user has to proof knowledge of the nonce. The user can know about the nonce if and only if she has access to the mobile phone in question. Mobile TANs are broadly used for authentication in banking, industry, and cloud services.

ate a CMS document. With the key service being a trusted party and the communications being secured by HTTPS, the encryption process is considered as secure.

The resulting CMS container is returned to the browser application which in turn sends it to the cloud storage after authenticating there.

Now user 1 can receive – i. e. read – the CMS container which was just submitted to the shared storage by user 2. The process is similar to the one described above and illustrated in Figure 5. The user takes one of her devices and downloads the encrypted data. She uses the same browser application to decrypt the CMS file with the help of the CrySIL infrastructure. She has to go through the process of selecting a certificate and standing the challenge and standing a two-factor authentication prior to the decryption process until the result is available in the application's CrySIL node and therefore in the application.

5.3 Discussion

Our approach has a number of advantages over conventional solutions. First and foremost, a centralised key storage location enables a user to access her keys at any time and anywhere. The only dependency is an internet connection, but by having an internet-accessible cloud storage service for data storage renders this dependency fulfilled whenever cloud access is possible. Therefore, this feature closes the gap between classic cryptographic service providers and upcoming requirement to serve multi-device users.

When sensitive key material is not stored on the device itself, there is no need to share key data between devices. The risk of exposing the sensitive key data during transmission is thus foreclosed completely. There is no need to align key storage solutions to be able to translate key material where interfaces and transmission channels are very restricted.

Not having access to the sensitive key material of one key on different devices reduces the attack vectors against the key drastically. Especially, since browsers for example are most vulnerable in terms of protecting sensitive key material. The private key material never leaves the key provider environment.

The level of device and cryptographic expertise required from developers is lower. The developer can focus on creating a feature-rich application, well tested and stable software instead of dealing with the peculiarities of authentication and key exchange and secure key storage on the devices in question. And, nonetheless, create an application that uses cryptography and enhances the privacy and security of the user and her data.

Last but not least, CrySIL offers not only off-device key storage but also off-device cryptographic functions. A cryptographic service provider which offers high level cryptographic methods such as CMS or XMLDSIG for remote use enables a broad range of devices and applications to use cryptography to protect the users' data and privacy. With that, the increasingly popular browser applications are enabled for the use of cryptography in a much more secure manner than with local key storage.

Finally, the CrySIL infrastructure relies solely on well-known building blocks of cryptography. The security aspects of key stores, cryptographic providers, as well as the communication solution are commonly known and well-understood.

All the advantages come with the cost of yet another trusted third party. Establishing certifications and trust relationships are still required and come with all advantages and drawbacks of this concept. Yet, the CrySIL infrastructure also supports the deployment of key stores on the user owned devices (e.g., home servers or mobile devices), or directly supports local crypto devices, such as smart cards.

5.4 Performance

As for performance, CrySIL does not implement any cryptographic service itself. It solely integrates existing solutions and makes them accessible over various APIs even on other devices. Therefore, depending on the used key provider/crypto service, anything is possible between a few up to multi-hundred signatures per second.

The infrastructure adds some overhead in the process of redirecting a command to a crypto service. The overhead is of constant size and in the magnitude of milliseconds. Having to collect authentication information does require some time for fetching the requirements, creating the challenge and reading the response. This time lost is minimal compared to the time a user needs to enter the required information.

Having an off-device scenario, there is of course some delay when sending commands via the Internet. Thus, in addition to performing the actual crypto process and the redirecting process one round-trip-time has to be added per command. For CrySIL, we enabled a batch mode so one can do multiple operations per command as an optimization option.

Anyhow, these performance measurements are made based on our prototypical implementation. The implementation has not received any performance optimizations due to the fact that the main goal is to create availability and not speed.

5.5 Integration Efforts

Whenever an application utilizes well-known crypto APIs, CrySIL can be integrated with an effort next to none. As of today, we work on receivers for PKCS11, JCE (for desktop and Android), MS CNG, W3C Crypto API, OpenSSL and OpenSC.

In case a platform does not have the required modules available, one can easily implement such a module. The Java JCE receiver module for example is implemented using only 1000 lines of prototypical code including some functionalities that are not supported by the JCE framework. Our Java router and sending communications modules do have 80 loc each with a common protocol definition of 1500 loc. The code of imported libraries are not included in the numbers given.

6 FUTURE WORK

Our approach complements classic solutions so that the new requirements of heterogeneous applications and cloud environments can be met. However, there are still gaps neither the related work nor our approach can solve currently.

The first gap is the need to move authentication away from the application. Our approach succeeds in moving the authentication to the library and therefore preventing the sensitive credentials to be directly processed by the application. Since the library runs inside the application and shares its memory, an attacker i. e. a malicious application might still be able to eavesdrop or tamper with the sensitive information. Having the credentials not reaching the application in the first place would foreclose this attack vector completely. For web applications, this could be realized by using a separated iFrame, in case of mobile devices, specific CrySIL apps could be used that are utilized by other apps via IPC calls.

Other future use cases include the emulation of attribute/identity based encryption, or proxy-re-encryption schemes by using flexible and fine grained authentication systems. E.g., proxy-re-encryption schemes could be emulated by handing out authentication tokens to third-parties who – by supplying these tokens – are allowed to re-encrypt data for specific recipients. Similar approaches could be used for the emulation of identity/attribute-based encryption schemes. However, the CrySIL platform could also be used to directly implement such schemes and thereby enable their usage on arbitrary platforms.

These examples represent a small collection of possible future directions.

7 CONCLUSIONS

The deployment of cryptographic functions in heterogeneous applications and storing and handling key material in a secure way faces many challenges in relation to lack of platform support and high complexity for the development teams. One way to approach these problems is the introduction of central services that deploy secure key storage facilities and provide APIs that can be used on arbitrary platforms. Several companies already offer such systems for the deployment of specific cryptographic functions. However, those system lack the flexibility in terms of supported cryptographic algorithms and protocols and have not been intended for generic use cases.

Therefore, this work presents the Crypto Service Interoperability Layer (CrySIL) which has a highly flexible architecture that is capable of combining central and local cryptographic services. The current system has already been successfully used for several prototypical applications and is constantly improved by adding additional support for cryptographic algorithms and APIs for different platforms.

REFERENCES

Ateniese, G., Fu, K., Green, M., and Hohenberger, S. (2006). Improved proxy re-encryption schemes with applications to secure distributed storage.

Bellare, M., Boldyreva, A., and O Neill, A. (2007). Deterministic and Efficiently Searchable Encryption. In *Proceedings of the International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 535–552. Springer.

Camenisch, J. and Shoup, V. (2003). Practical Verifiable Encryption and Decryption of Discrete Logarithms. In Boneh, D., editor, *CRYPTO 2003: Advances in Cryptology*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer Berlin Heidelberg.

Egele, M., Brumley, D., Fratantonio, Y., and Kruegel, C. (2013). An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pages 73–84, New York, New York, USA. ACM Press.

Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., and Freisleben, B. (2012). Why eve and mallory love android. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, page 50, New York, New York, USA. ACM Press.

Hanser, C. and Slamanig, D. (2013). Blank digital signatures. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, page 95, New York, New York, USA. ACM Press.

IDC (2014). Worldwide Wearable Computing Market Gains Momentum with Shipments Reaching 19.2 Million in 2014 and Climbing to Nearly 112 Million in 2018, Says IDC. `https://www.businesswire.com/news/home/20140410005050/en/Worldwide-Wearable-Computing-Market-Gains-Momentum-Shipments`. last visited on March, 25th 2015.

Leitold, H., Hollosi, A., and Posch, R. (2002). Security architecture of the Austrian citizen card concept. *18th Annual Computer Security Applications Conference, 2002. Proceedings.*

Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop - CCSW '11*, pages 113–124. ACM Press.

Parliament, E. U. and Council (2000). Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. *Offcial Journal of the European Communities*, L 013:12–20.

Trusted Computing Group (2011). TCG TPM specification version 1.2 revision 116. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`. last visited on January 29, 2013.

van Hoboken, J. V. J., Arnbak, A., and van Eijk, N. (2012). Cloud Computing in Higher Education and Research Institutions and the USA Patriot Act. *SSRN Electronic Journal*.