In Certificates We Trust – Revisited

Florian Reimair, Peter Teufl, Bernd Prünster
Institute for Applied Information Processing and Communications
Graz University of Technology
Graz, Austria
{florian.reimair, peter.teufl}@iaik.tugraz.at

***Abstract*—Today's state-of-the-art workflow – when encrypting data for one or more recipients – requires for the sender to select the respective encryption keys. Naturally, it is crucial for data security to pick the correct keys with sufficient security levels. Yet, for selecting a key, the sender has to trust a recipient-chosen third party and therefore bears the hardly controllable risk of choosing a bad key. We propose to redistribute the tasks and require for a data sender to create an encryption key for himself and grant the recipient access to the key through authentication. The sender therefore can select the authentication methods, key strength, and key lifetime that suits his needs. In order to do this, we take advantage of the (semi-)centralized key storage solution CrySIL and add advanced policy enforcement options. We show the results of our prototypical implementation and present a discussion on the security of the system.**

***Index Terms*—certification, encryption, key selection, trust, heterogeneous platform security**

I. INTRODUCTION

Nowadays, applications need to support a wide range of heterogeneous platforms including smartphones, tablets, browsers and classic desktop-based systems that are provided by different manufacturers and come with different operating systems and functionality. Developing in this environment is challenging in many aspects – especially when security needs to be considered. For security-aware applications the deployment of cryptographic functions for data encryption, electronic signatures etc. is essential. Having the required high-level or low-level cryptographic functionality available on the respective device, and distributing, storing and handling the key material poses significant issues for applications that are deployed in multiple-platform environments. These issues can be addresses fairly well by providing central cryptographic platforms that store the key material in a secure storage place (e.g. HSM) and provide associated cryptographic functions via a web-based interface that is consumed by standard APIs such as PKCS11, JCA/JCE etc. on the respective devices. Such solutions have already been deployed and offer high-level or low-level cryptographic functionality. Examples therefor are qualified signatures services within the e-government context (e.g., [17]) or the Amazon Cloud HSM[1], respectively.

Due to missing flexibility in these services we have introduced and deployed the Crypto Service Interoperability Layer (CrySIL) [20], which in general provides a central instance that offers cryptographic APIs to arbitrary devices. The key material never leaves the secure storage place (HSM) and is utilised within cryptographic functions consumed via standardised cryptographic APIs by the clients. Access to those functions is regulated via a flexible authentication architecture that can easily be adapted according to the security needs of a specific deployment scenario[2]. CrySIL has a highly flexible architecture, which allowed us to extend the initial use case of a central-instance by mobile use cases, local installations and even distributed scenarios, where key material and/or cryptographic functions are handled by multiple CrySIL nodes. The current CrySIL functionality plays an important role for a wide range of data encryption and digital signature scenarios related to encrypting/signing cloud-storage data, providing email signing/encryption facilities or offering protection for keys utilized within remote access systems (e.g. VPNs).

However, the current CrySIL system as well as related systems lack functionality that is required for use cases where data has to be encrypted for entities that are not associated with the CrySIL system. Examples are entities that either do not have a (trusted) certificate, that do not and will never actively use the CrySIL system, that are not allowed to decrypt the data at the time of encryption but might be allowed to do so at a later time due to changed attributes/access policies, or that are only allowed to temporarily access data (e.g. sharing files with external users).

In essence such problems are typically addressed with identity-based or attribute-based encryption systems. However schemes like [2] and [8] offer this functionality but are not compatible to wide-spread standards used for data encryption/signatures such as CMS, S/MIME, XMLENC, XMLDSIG etc. A primal aspect of the CrySIL architecture is the flexible authentication system that allows us to protect cryptographic operations and the utilization of the stored key material within these operations via arbitrary authentication methods.

Based on this functionality, we propose an extension to CrySIL that is based on the following idea: A CrySIL instance encrypts data for other recipients via keys that are generated within the same CrySIL instance and defines policies – especially related to authentication – that govern who (which entities) and how (e.g. time related limitations) a CrySIL key can be used. Whenever an external entity needs to access such encrypted data, this entity sends a decryption request to the CrySIL instance, then – according to the pre-defined

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

[2] By deploying adequate authentication methods and parameters related to session life time, number of allowed key operations per time frame etc.

978-1-4673-7952-6/15 $31.00 © 2015 IEEE
DOI 10.1109/Trustcom-BigDataSe-ISPA.2015.437

policy – needs to stand the authentication challenge and finally is allowed to use the CrySIL key for decryption. To add additional flexibility, the generated keys and the key-usage policy can be protected with internal CrySIL keys and exported as a container that can be stored together with the encrypted data and the CrySIL instance's URL at an external location (cloud storage, transfer via email etc.).

When comparing this idea with existing solutions such as CAs, webs of trust (PGP) or other methods, the following differences can be observed. While in existing solutions the level of authenticity/trust for a published public key is determined by the recipient, in the proposed system it is defined by the sender. By defining an authentication policy that is associated with the respective key and cryptographic function, the sender determines which trust-level the recipient needs to fulfill. Another key issue is that within such a system the recipient does not need to have a key at the time of the encryption process (comparable to identity/attribute based encryption schemes). The recipient only needs to be able to stand the defined authentication challenge at the time of decryption. Furthermore, encrypting data for the recipient can be based on specific attributes that need to be proven with a specific authentication method.

To gain a better understanding of the consequences of such systems and to analyze an initial CrySIL prototype that will act as test-bed for later implementations, this work describes CrySIL, the proposed extension, demonstrates the system with a simple encryption use case, and discusses the security of the proposed system.

The remainder of the work is structured as follows. section II discusses related work such as certification authority or web-of-trust approaches to the trust challenge. We present our approach to the trust challenge in section III and discuss our prototypical implementation in section IV. We give a security discussion in section V and conclude the work in section VI.

II. Related Work

Public key infrastructure (PKI) systems like PGP or certification authorities (CAs) still are the state-of-the-art solutions in standing the above-mentioned challenges. Their flaws, however, motivated other solutions such as OTR and advanced cryptographic schemes to complement and replace existing solutions.

PKI systems have been around a long time now and thus, evolved and matured over time. They overcame many of their limitations and became the state-of-the-art approach to tackling the trust challenge. Their trust models [14] offer great flexibility and therefore makes them suitable for many use cases. Subsequently, PKI solutions are widely used today [9] as they are well understood and mature.

The main players for PKI implementations are CAs and the pretty-good-privacy (PGP) solution. While both rely on the same cryptographic methods for securing communications, they address the challenges of distributing keys and creating trust relationships differently. The CA approach relies on having a trusted third party that issues a certificate only after verifying the identity of the requester. The resulting certificate can be distributed freely and used for cryptography. The trust level for such a certificate relies solely on the CA and its identity check. However, CAs and their checks are not infallible [25]. In contrast, the PGP solution relies on the community to build a trust network. Members of the community vote for the trustworthiness of a public-key-to-identity-binding and therefore form a web of trust. The trust levels therefore depend solely on the community and the system only works if most of the community members are trustworthy. PGP is free of charge but much more complex to use [21][26] and has some flaws too [5][16].

There are alternative approaches to stand the trust challenge. [10] proposes a "PKI design for the real world" and addresses shortcomings and culprits of current designs, [24] uses postcards or voice messages to establish message authentication, [1] proposes a PKI for peer-to-peer networks, [28] verifies certificates with a trust matrix, [13] propose a PKI-less solution for trust establishment utilizing mobility and physical interaction, [4] and [7] build entire trust networks which calculate a trust rating out of the PGP web of trust, advanced encryption schemes offer novel ways to approach the trust challenge, e.g. identity-based encryption [27], distributed private-key generators [22] and proxy-reencryption schemes [15][29]. Last but not least, [3] proposed the "off-the-record" encryption methodology, an easy to set up encryption scheme for short-lived communications. Yet, most of them rely on a PKI-like scheme with all of its trust issues.

III. Inverting the Trust Relationship

In this work we propose to invert the trust relationship between the recipient and the sender. Within classic approaches, the sender is the entity who has to verify the authenticity of the cryptographic primitive (i.e. the recipients key) he is going to use for encrypted communication. Having to trust a third party (a CA or a web-of-trust or alike) to find the correct key might be against the sender's interest of having the data only readable by a distinct recipient. Therefore, with having (semi-) centralized key management solutions available, we propose to invert this constellation. The sender creates his own key and allows the recipient to use it if and only if the recipient can authenticate herself directly against the sender. Thus, there is no pre-sharing of certificates necessary, the authenticity of the key data is ensured by the sender/key owner, and the sender can authenticate the recipient(s) through authentication challenges that meet his cause.

The process flow in a nut shell (Figure 1 illustrates the involved entities and their relationships): In order to *encrypt* the *data*, the *sender* needs to create a *key* which he subsequently *owns*. The *key* is protected by a central key storage service throughout its lifetime and hence, only the central key storage can use the *key* for cryptography. The sender then uses the *key* to *blind* the *data* and sends the *data* to the receiver. Additionally, the *sender* attaches some *policy* to the *key* that *protects* it and allows the receiver to use it after standing

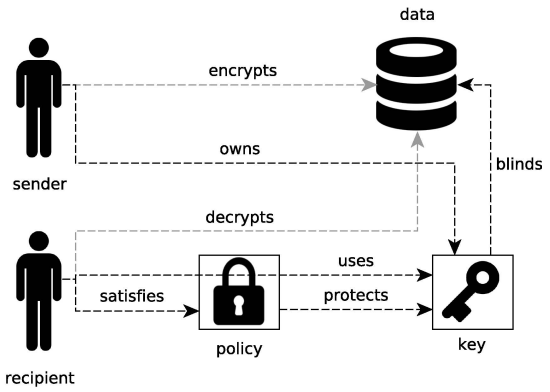

Fig. 1. Entities and relationships

an authentication challenge. In order to *decrypt* the *data*, the *recipient* needs to *satisfy* the *policy* and thereafter can *use* the *key*.

Sending large amounts of data across the internet to be received by a (semi-) central key management service is not feasible for large quantities of data. Therefore, we suggest to use hybrid encryption schemes such as key wrapping to enable bulk encryption of data while the data encryption keys remain at the client and the sender's key remains at the key management service. Using asymmetric keys as key encryption keys enables compatibility with widely deployed methods of encryption such as CMS [11] or S/MIME [23].

All in all, moving the challenge from securely distributing and storing the key data to restricting access to web-accessible keys brings a number of advantages. First and foremost, the recipient does not have to have a suitable (public) key available for the world to use. Thus, a sender does not have to accept trust issues as they occur for example when a CA is compromised or a PGP key is wrongly tagged as trusted/verified or the recipient simply missed to provide her certificate to the public. Next, the sender can easily manage data recipients without any additional cryptographic effort. In order to add a recipient, the sender simply has to add access permission to the key and does not need to re-encrypt and redistribute the data. Revoking data access, as with any other encryption solution, is not possible since the recipient can decrypt and memorise the plain data. He thereafter does not need crypto to read the data. Next, the sender can choose whether unidirectional or full-fledged bidirectional communication is available to the recipient. Allowing the recipient to use the key only for decryption initiates a uni-directional communication. Granting the recipient encryption capabilities too allow the sender to participate in a bidirectional conversation with an authenticated recipient. Last but not least, the sender remains anonymous in terms of certificates and key data. If the sender uses an anonymity framework like [6] against being traced, the sender might even participate in these communications without revealing his own identity at all.

The major drawback is the number of keys a busy sender might have to maintain. To approach this challenge, we propose to wrap (i.e. encrypt) the encryption key and send it to the recipient along with the data. The sender therefore has to maintain only the wrapping key(s). Another drawback is that the trust relationship towards the (semi-) central key management service is to be solved by the sender. For the sender, that might be enough, because he can host such a service all by himself.

IV. Implementation

In order to evaluate our proposal we did a prototypical implementation based on the CrySIL architecture [20]. CrySIL offers (semi-) centralised key storage that can be deployed to be publicly accessible and has built-in authentication features available. The easy to extend architecture made integration efforts low. The option of deploying the CrySIL key storage on a smart phone [19] fits the anonymity use case well. We complemented the CrySIL infrastructure by a key store implementation that features a lifetime key management, protection, and key usage policies.

In this section, we give a brief summary of CrySIL, discuss the custom key store as well as give a detailed work flow description.

A. Building Blocks

1) Crypto Service Interoperability Layer: The Crypto Service Interoperability Layer (CrySIL) concept has been created to meet today's heterogeneous device landscape and allows the user to use her keys within cryptographic operations regardless where the key resides and where it is needed. In a nutshell, 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

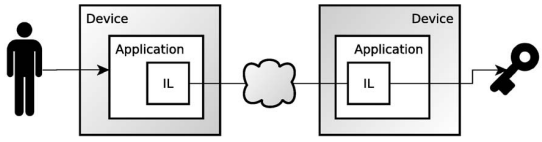

Fig. 2. CrySIL's basic architecture

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 2.

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 parts – modules – have different jobs and work together to form the actual cryptographic service provider. *Receivers, actors*, a *router*, and other modules handle internode communication, protocol mappings, crypto, advanced crypto and authentication. A *receiver* for example acts as a protocol bridge to the interoperability layer with pre-build

implementations for Java's Cryptographic Extension (JCE), the Windows CNG, or the W3C Web Cryptography API as well as PKCS#11 and a SOAP-based web interface. An *actor* makes the services of a crypto provider, e.g. a smart card or an HSM, available to the layer and collects authentication information as they are required. The *router* connects the modules together. Anyhow, all modules are considered as building blocks and are not restricted to any technology, platform, or programming language. Every configuration of a router and other modules is referred to as a *CrySIL node* and forms the heart of the concept. The modular node design enables the flexibility and extensibility of the concept while keeping the overall architecture simple. An illustration of modules and their interconnections is given in Figure 3.

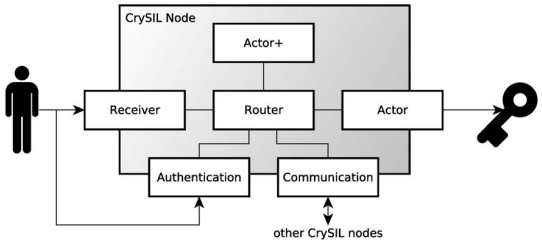

Fig. 3. CrySIL node architecture overview

Cloud-scenarios with multi-device users require 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. 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 Web Messaging, Web-sockets, or IPSec are suitable.

Putting the pieces together, CrySIL renders off-device crypto completely transparent to the user, the developer, and

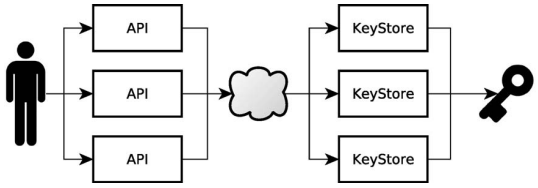

Fig. 4. Interoperability architecture view

to the application while maintaining a simple architecture. The resulting architecture is depicted in Figure 4. A user benefits from her ability to use a variety of keys within different cryptographic operations provided by different crypto providers from a variety of applications on different devices.

The off-device crypto feature of CrySIL requires for key usage constraints as the keys are potentially available to the public. 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 a multitude of authentication methods, such as simple PIN challenges, external identity providers (OAuth, OpenID Connect) or multi-factor authentication methods. The CrySIL infrastructure gathers authentication data from the user as well.

2) A Policy-enabled Key Store Actor for CrySIL: As CrySIL does not come with an actor that meets our requirements, we implemented a prototypical policy-enabled key store actor on our own.

The cryptographic basis of the actor is a simple RSA encryption scheme engine. First and foremost, its versatility and flexibility meets our research scenario well and lets us experiment with different ideas. Second, having RSA encryption scheme implementations freely available for Java did keep our integration efforts low. The engine is designed to use hardware security modules (HSMs) as back-end and therefore the key data is protected from disclosure throughout its lifetime. An illustration of a key's lifetime is given in Figure 5.

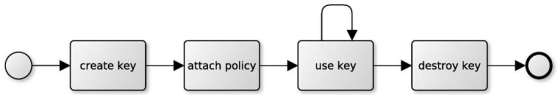

Fig. 5. Key lifetime

The key management is done via a graphical user interface. Basic features like creating and revoking keys are provided as well as attaching policies to a key. Policies specify which authentication challenges are being raised and which authentication outcome is expected to satisfy the policy. We used a simple database-backed scheme for our prototypical implementation but recommend mature policy modelling languages like XACML [18] or alike for productive services.

As for authentication options, we included methods of different strength to test the flexibility of the system. First, we included a simple password authentication option with one expected value. Everyone who knows the correct password can stand the challenge. Further, we modified the password option to incorporate a username too and let the authorization process utilize a Microsoft Active Directory (AD) service via LDAP protocol for credentials validation. With this option, we succeeded in introducing the system into an existing corporate network infrastructure with minimal efforts. Furthermore, we enabled oAuth/OpenID Connect authentication option to work with Twitter/Facebook/Google as an authentication source. And last but not least, we incorporated strong multi-factor authentication in the form of the national mobile eID implementation [12][17] which features strong identity binding as well.

To approach the issue of a busy sender who might need to manage a lot of keys, we added a feature for wrapped export of keys. Wrapped export means that the data encryption key (i.e. the key that is used in a hybrid encryption scheme like CMS or S/MIME to encrypt the symmetric content encryption

key) and the respective set of (aforementioned) policies is put into a data structure. This data structure gets encrypted (wrapped) by another key of the sender. This *key encryption key* resides on the key management service. The wrapped key structure is sent to the recipient alongside the blinded payload data with the URL of the key management service where the wrapped key can be used. In order to decrypt the payload data, the recipient has to provide the data encryption key and the wrapped key structure to the service to get the content encryption key and to subsequently decrypt the data. The advantage of this feature is that the key management service does not get cluttered with lots of keys, yet, the data protection workflow holds. Drawbacks are that once the key is wrapped and sent, it is not possible to change the policy, i.e. adding additional recipients without re-encrypting and resending the key. The second drawback is that there is no "revocation" of single keys possible as it is when the key resides at the key management service and gets deleted.

B. Process Flows

To demonstrate the procedure an example workflow is given. In this workflow, we assume that a person or a company (further denoted as *sending entity*) wants to transfer a *document* securely to another person or company (further denoted as *receiving entity*) which does not provide a public key in some way. However, the *sending entity* knows that the receiving entity has at least one authentication method available that offers adequate security levels. Depending on the required security level, examples could be an authentication method provided by a social network, a combination of multiple methods, or even a strong e-Id method as it is used within e-government processes. For this demonstration workflow we assume that the *receiving entity* is a person who has social networks accounts at Facebook and Twitter. We also assume, that the *receiving entity* has a strong interest to protect these profiles and thus the associated credentials. An overview about the process is depicted in Figure 6.

Preparing CrySIL: The *sending entity* has a local CrySIL installation setup that offers a publicly available web interface via HTTPS. The CrySIL installation has already been set-up with a key pair[3]. This key pair will be further denoted as *static-crysil-key*. The public/private keys will be denoted as *static-public-key* and *static-private-key*, respectively. Depending on the use case/deployment scenario, the *static-private-key* should be stored in a hardware element.

Preparing the document: The *sending entity* utilizes some software capable of encrypting the *document* following the CMS standard.

Determining the authentication policy: Prior to generating the key data for encrypting the *document*, an *authentication policy* needs to be chosen that allows the *receiving entity* to later authenticate against the CrySIL node ((1) Define authentication-policy). In this example a policy is defined that requires the *receiving entity* to authenticate via her Twitter and Facebook accounts.

Generate key pair: The *sending entity* instructs the local CrySIL node to generate a key pair which will be used for the encryption/decryption process ((2) Generate-Wrapped-Key). This key pair will further be denoted as *generated-key* and is comprised of private key (*generated-private-key*) and a public key (*generated-public-key*). The *generated-key* and the predefined *authentication policy* is arranged in a data structure which is then signed with the *static-private-key* and then encrypted with the *static-public-key*. The resulting blob is further denoted as *protected-generated-key-structure*. The *protected-generated-key-structure* and the *generated-public-key* are the handed over to the *sending entity*.

Encrypting the document: The *document* is then encrypted via the CMS standard by generating a random *AES key*, which is used for encrypting the *document* ((3) Encrypt document). The *plain-AES-key* is then encrypted (wrapped) with the *generated-public-key.* The *wrapped-AES-key* and the *encrypted document* are then arranged in a container according to the CMS standard.

Transferring the encrypted document: The *encrypted document*, the *protected-generated-key-structure* and the *CrySIL node URL* are arranged in a data structure (*data-package*). This *data-package* is then transferred via some channel (e.g. via an unprotected email) to the *receiving entity* ((4) Hand over encrypted document).

Preparing the decryption request: The *receiving entity* receives the *data-package* and extracts the *protected-generated-key-structure, CrySIL node URL* and the *encrypted document.* Also, the *wrapped-AES-key* is extracted from the CMS structure of the *encrypted document* ((5) Extract wrapped-AES-key). The *receiving user* then prepares a CrySIL decryption request containing the *wrapped-AES-key* and the *protected-generated-key-structure*. The decryption request is then sent to the *CrySIL node URL* of the *sending* entity ((6) Decrypt request). The authenticity of the *CrySIL node URL* is ensured via the TLS protocol and trusted server (CrySIL) certificates[4].

Processing of the decryption request: The CrySIL node of the *sending entity* receives the decryption request and extracts the *wrapped-AES-key* and the *protected-generated-key-structure*. The *protected-generated-key-structure* is decrypted via the *static-private-key* and the signature of the decrypted data structure is verified via the *static-public-key*. The *authentication policy* is extracted and parsed. Here the *authentication policy* requires the successful Twitter and Facebook authentication of the *receiving entity*. CrySIL generates authentication requests for both methods and transmits one of them to the *receiving user* ((7) Auth challenge). The *receiving entity* stands the challenge of the request and by responding the correct data back to the CrySIL node ((8) Auth response)[5]. The same procedure is carried out with the second authentication

[3]The public key could be represented within an X509 certificate. There are no requirements concerning the trust level of this certificate/public key.

[4]This can be either achieved via standard CAs that are shipped with web browser or custom trust-relationships. Regardless of the method, trusting the CrySIL node is essential for avoiding possible man-in-the-middle attacks.

[5]The details of this procedure depend on the chosen authentication method.

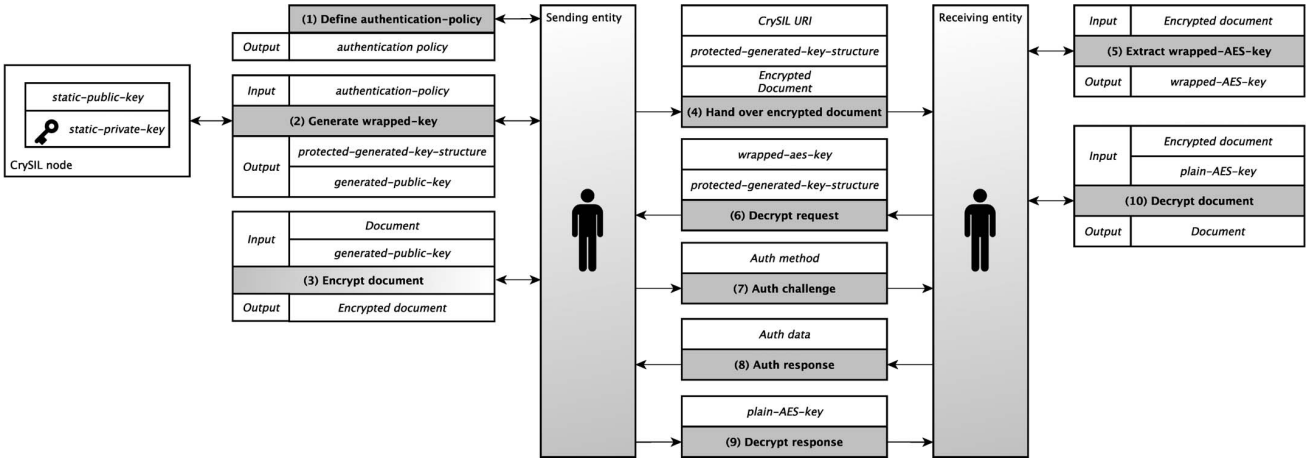

Fig. 6. Processes for sending an encrypted document.

method . If both were successful the CrySIL node extracts the *generated-private-key* of the *protected-generated-key-structure* and decrypts the *wrapped-AES-key*. The *plain-AES-key* is then sent as response to the decryption request to the receiving user ((9) Decrypt response). The CMS library of the receiving user utilizes the *plain-AES-key* to decrypt the *document* ((10) Decrypt document).

V. Discussion

A. Security and Trust

1) Trusting the Receiving Entity: Apart from the encryption algorithm that ensures the confidentiality of the transferred data, the authenticity of the receiving entity is essential for ensuring that the correct entity receives the data. Typically, the authenticity of the receiving entity is bound to a cryptographic (public) key by a (signed) certificate. There are, however, different methods to sign (certify) a certificate. A certificate can be self-signed, i.e. the certificate is signed by the very (public) key that it binds to the entity. Thus, the quality of the binding is merely a claim by the entity itself. Alternatively, a certificate can be signed by a CA after the CA assesses the entity's identity. In this case, the quality depends on the chosen CA. All in all, the quality of the binding, i.e. the trust level and therefore the quality of the authentication of the entity, solely relies on the binding method the receiving entity chose. In contrast, the CrySIL enabled process defines the required trust level by choosing authentication methods for the *receiving entity* that provide adequate security levels for a specific document exchange process. This results in several primal differences to the standard methods: **(1)** By selecting authentication methods, the *sending entity* defines the desired quality of authentication. In classic methods, the *receiving entity* selects the quality of authentication by choosing the binding method. **(2)** With classic methods, the deployed trust-model is typically chosen at some point in time and remains static. I.e. when using certificate authorities (CA) to certify (public) keys, a CA is chosen and a certificate is issued. The chosen CA determines the level of trust that can be achieved. The issued certificate remains static until it is revoked or becomes invalid. This implies that the *sending entity* does not have the option to define the required trust level. It can either trust the existing certification and use the public key for encryption or stop the process. In the proposed method, the decision on the required level of trust can be made every by the *sending entity* every time a document is encrypted for the receiving entity.[6] **(3)** Therefore, the level of trust is not defined via the certification policy (e.g. the requirements of a CA that issues certificates to its users), but by the authentication method. Each authentication method has a different maximum level of trust. This level depends on security of the authentication method and the level of authenticity it is able to provide. E.g. a social network account will have a much lower level of authenticity than e.g., an eGovernment related eId method. By choosing the authentication method or an arbitrary combination of multiple methods, the *sending entity* can decide how the required level of trust can be achieved for the protection of the transmitted data.

2) Trusting the Sending Entity: In this work we focus on the encryption of documents for *receiving entities.* The authenticity of the encrypted document – e.g., via digital signatures – is not considered. However, trusting the *sending entity* is an important aspect when analyzing the security of the communication channel used to transmit decryption and authentication requests/responses. The relevance of the authenticity of the CrySIL node will be discussed in the following section.

3) Confidentiality: When analyzing the processes for encrypting, transferring and decrypting a document, the security of the communication channel needs to be considered with different impacts during the whole process. Thereby, the core processes of sending the document, handling authentication procedures, and decrypting the document need to be con-

[6]Obviously, the *receiving entity* needs to have authentication methods available for the *sending entity* to pick.

sidered.

(1) Sending the document: This process transfers the encrypted document, the wrapped AES key, the protected asymmetric key pair and the CrySIL URL on an arbitrary channel to the *receiving entity* . Since the data (except for the URL) is encrypted, there are no security requirements on the channel used for this transfer. The authenticity of the *sending entity* , however, needs to be ensured in order to foreclose man-in-the-middle attacks. In our demonstration workflow the confidentiality is ensured by using the TLS protocol, and the authenticity of the *sending entity* is guaranteed by using certificates issued by trusted CAs. **(2) Handling authentication challenges/responses**: When the *receiving entity* starts the document decryption process it sends the decryption request to the *sending entity* . This entity then responds with the predefined authentication challenges. The *receiving entity* replies with the specific authentication responses. The requirements in terms of confidentiality and authenticity strongly depend on the used authentication method, but in general, a channel that guarantees the authenticity of the *sending entity* and the confidentiality of the authentication data is required. However, when certain authentication methods are used these requirements might be lower due to the involvement of trusted third parties that carry out the authentication processes. **(3) Unwrapping the AES key** : After authentication, the wrapped AES key is decrypted by the *sending entity* and the plain AES key is then transmitted to the *receiving entity*. The AES key must be kept confidential.

B. Performance and Integration Efforts

A classic performance analysis cannot be applied to our concept as CrySIL is about bringing crypto to devices and not to create yet another high performance implementation. The prototypical implementation, however, gave us a rough understanding on how efficient, fast, and functional our approach is. The key store extension to CrySIL has about 800 lines of prototypical code. None of the lines are optimised for speed and parallel execution. However, the speed is as expected. There is the time required to do the cryptographic operation itself, be it done by an HSM or a software solution, some round-trip times in order to get the commands through the internet and back and of course the latencies induced by the user.

As for the client side integration efforts, whenever an application utilises well-known crypto APIs, CrySIL can be integrated with a minimal effort. As of today, there are receivers for PKCS11, JCE (for desktop and Android), MS CNG, W3C Crypto API, OpenSSL and OpenSC available as prototypical code. 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. The router and sending communications modules of the Java implementation do have 80 loc each with a common protocol definition of 1500 loc. The lines of codes of dependencies are not included in the numbers given.

VI. Conclusions

CrySIL has already been deployed in environments for providing platform-independent cryptographic APIs in heterogeneous application environments. The flexibility of the architecture of CrySIL allowed for the simple integration of additional facilities that allows us to circumvent two main challenges of classic encryption schemes: (1) How can the public key of the receiver be trusted and (2) how to deal with receivers that do not have a public key available (yet). To address these questions the sender of the proposed system encrypts the data with its own and well-protected keys and allows the receiver to use these keys remotely in the decryption process by standing one or multiple authentication challenges defined by the sender.

Shifting the definition of achievable authenticity from the certification process of the receiver (e.g. via a CA) to the sender, who defines the trust level by choosing adequate authentication methods, is interesting from multiple perspectives: The sender is now able to determine the required trust level on a per-document basis and can even encrypt data for entities which do not have the required authentication credentials and attributes yet. This is similar to attribute-based encryption but uses standard methods that are widely used. Furthermore, by using temporary asymmetric keys that are stored in addition to the required key usage policies, the server CrySIL infrastructure can be kept rather simple. The extensions to the CrySIL infrastructure provide an important basis for future use cases that require sharing encrypted documents with external entities that are not members of the CrySIL infrastructures or are chosen by their identity or attributes. We plan to extend the current prototype with a fully fledged policy language (such as XACML), improve the emulation of the attribute/identity-based encryption functionality, and add additional use cases (e.g. emulation of proxy-reencryption).

References

[1] Agapios Avramidis, Panayiotis Kotzanikolaou, Christos Douligeris, and Mike Burmester. Chord-PKI: A distributed trust infrastructure based on P2P networks. *Computer Networks*, 56(1):378–398, 2012.

[2] Dan Boneh and Matthew Franklin. Identity-Based Encryption from the Weil Pairing, 2003.

[3] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-Record Communication, or, Why Not To Use PGP. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society (WPES '04)*, pages 77–84. ACM Press, 2004.

[4] David Brondsema and Andrew Schamp. Konfidi: Trust networks using PGP and RDF. In *CEUR Workshop Proceedings*, volume 190, 2006.

[5] Kuobin Dai. PGP e-mail protocol security analysis and improvement program. In *Proceedings - 2011 International Conference on Intelligence Science and Information Engineering, ISIE 2011*, pages 45–48, 2011.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium*, 13:21, 2004.

[7] J A Golbeck. *Computing and applying trust in web-based social networks*. PhD thesis, University of Maryland, 2005.

[8] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98. ACM, 2006.

[9] Peter Gutmann. PKI: It's not dead, just resting. *Computer*, 35(8):41–49, 2002.

[10] Peter Gutmann. PKI design for the real world. In *Proceedings of the 2006 workshop on New security paradigms*, NSPW '06, pages 109–116. Acm, 2007.

[11] R. Housley. Cryptographic Message Syntax (CMS). *Rfc 5652*, pages 1–57, 2009.

[12] H. Leitold, a. Hollosi, and R. Posch. Security architecture of the Austrian citizen card concept. *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, 2002.

[13] Yue Hsun Lin, Ahren Studer, Yao Hsin Chen, Hsu Chun Hsiao, Li Hsiang Kuo, Jonathan M. McCune, King Hang Wang, Maxwell Krohn, Adrian Perrig, Bo Yin Yang, Hung Min Sun, Phen Lan Lin, and Jason Lee. SPATE: Small-group PKI-less authenticated trust establishment. *IEEE Transactions on Mobile Computing*, 9(12):1666–1681, 2010.

[14] Hou Liping and Shi Lei. Research on trust model of PKI. In *Proceedings - 4th International Conference on Intelligent Computation Technology and Automation, ICICTA 2011*, volume 1, pages 232–235, 2011.

[15] Song Luo, Qingni Shen, and Zhong Chen. Fully secure unidirectional identity-based proxy re-encryption. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7259 LNCS, pages 109–126, 2012.

[16] Phong Q Nguyen. Advances in Cryptology - EUROCRYPT 2004. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027, pages 555–570, 2004.

[17] Clemens Orthacker, Martin Centner, and Christian Kittl. Qualified Mobile server signature. In *IFIP Advances in Information and Communication Technology*, volume 330, pages 103–111, 2010.

[18] Bill Parducci and Hal Lockhart. eXtensible Access Control Markup Language (XACML) Version 3.0. *OASIS Standard*, (January):1–154, 2013.

[19] Florian Reimair, Peter Teufl, Christian Kollmann, and Christoph Thaller. MoCrySIL - Carry your Cryptographic keys in your pocket. In *Proceedings of the 12th International Conference on Security and Cryptography (seCrypt)*, 2015.

[20] Florian Reimair, Peter Teufl, and Thomas Zefferer. WebCrySIL - Web Cryptographic Interoperability Layer. In *Proceedings of the 11th International Conference on Web Information Systems and Technologies (WebIST)*, 2015.

[21] Steve Sheng, Levi Broderick, Jeremy J Hyland, and Colleen Alison Koranda. Why Johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, pages 3–4, 2006.

[22] Amar Siad. Anonymous identity-based encryption with distributed private-key generator and searchable encryption. In *2012 5th International Conference on New Technologies, Mobility and Security - Proceedings of NTMS 2012 Conference and Workshops*, 2012.

[23] Sean Turner. Secure/multipurpose internet mail extensions. *IEEE Internet Computing*, 14(5):82–86, 2010.

[24] Serge Vaudenay. Secure Communications over Insecure Channels Based on Short Authenticated Strings. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621, pages 309–326, 2005.

[25] Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, Jean-pierre Hubaux, Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, and Jean-pierre Hubaux. The Inconvenient Truth about Web Certificates. In *Proceedings of the 10th Workshop on Economics of Information Security*, pages 11–14, 2011.

[26] Alma Whitten and J.D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. *Proceedings of the 8th USENIX Security Symposium*, pages 169–184, 1999.

[27] Chi Xu and Wenfang Zhang. Improved efficient identity-based encryption scheme. In *Proceedings - 2012 International Conference on Computer Science and Service System, CSSS 2012*, pages 187–190, 2012.

[28] Wen Zhao and San Ping Tang. Certificates verification based on trust matrix in PKI. In *First International Conference on Innovative Computing, Information and Control 2006, ICICIC'06*, pages 52–55, 2006.

[29] Dehua Zhou, Kefei Chen, Shengli Liu, and Dong Zheng. Identity-based conditional proxy re-encryption. *Chinese Journal of Electronics*, 22(1):61–66, 2013.