Per-Dossier Encryption with Hybrid Approach

Inhalt

Per-Dossier Encryption with Hybrid Approach 1
2. Executive Summary 3
3. General Approach 5
3.1. Principles 5
3.2. Definition "Entity"/"Profiles" 5
3.3. Definition "Cryptographic Profile" 5
3.4. Public Key Infrastructure (PKI) 5
3.4.1. Handling of private keys 6
3.4.2. Requirements for PKI 6
3.5. Hardware Security Module (HSM) 7
3.5.1. Functional requirements 7
3.5.2. Performance / Scale requirements 8
3.5.3. HSM Interactions 9
4. Design 11
4.1. Creation of a Profile 11
4.2. Creation of a Dossier 11
4.2.1. Step 1: Generation of key $dossier_1$ 11
4.2.2. Steps 2a & 2b: Encryption and storage of dossier documents 12
4.2.3. Steps 3a & 3b: Encryption and storage of the $dossier_1$ key 12
4.3. Sharing Access to Dossier 1 with Entity B (OPE Variant) 12
4.3.1. Steps 4a & 4b: Encryption and storage of the $dossier_1$ key for entity B 12
4.4. Entity B accesses Dossier 1 12
4.4.1. Steps 5a & 5b: Derive profile-protecting key 12
4.4.2. Steps 6a & 6b: Decrypt entity B's private key 12
4.4.3. Steps 7a & 7b: Decrypt $dossier_1$ key 12
4.4.4. Steps 8a & 8b: Decrypt documents of $dossier_1$ 12
4.4.5. Step 9: Entity B can access the documents of $dossier_1$ 13

4.5. Sharing Access to Dossier 1 with Entity B (E2EE Variant) 13
5. Usage Scenarios 13
5.1. Scenario 1: Communication from a Lawyer's Office to Authority using OPE 13
5.2. Scenario 2: Communication between Authorities using E2EE 14
5.3. Scenario 3: Communication from a Lawyer's Office to Authority using Hybrid Encryption 14
6. Technical Scenarios 15
6.1. Membership changes for authority profiles 15
6.2. Change of a profile's key protecting token 15
6.3. Change of a profile's public/private key pair 15
6.4. Change of a dossier key 15
7. References 16
Appendix 16
Cryptographic Algorithms 16
1.1. Recommendation 16
1.2. Quantum-resilient cryptography 17
1.2.1. Impact of quantum computers on standard cryptography 17
1.2.2. Quantum-resilient cryptography not in MVP 17
2. Algorithms for Consideration 18
2.1. Symmetric Cryptography 18
2.2. Asymmetric Cryptography 19
2.3. Hash Functions 19
2.4. Message Authentication Code (MAC) Algorithms 20
2.5. Key Derivation Functions 20

2. Executive Summary

Description

By default, the encryption and decryption of data for the MVP-/Pilot-Phase (see *References*) takes place on the plattform Justitia.Swiss (OPE, on-plattform encryption).

As the platform's functionalities are developed, the option to use end-to-end encryption (E2EE) between judicial authorities is being prepared for easy activation, should the requirement be permitted from a compliance perspective and requested by the authorities.

The conclusions of the present concept are:

- Communication between lawyers/lawyer offices and administrative bodies uses on-platform encryption by default.
- The design of the platform needs to be *flexible and future-proof*. Thus, E2E encryption should be supported in specific use cases.
- Communication between trusted entities can use E2E encryption. The use of E2E encryption requires additional effort on the clients (such as local key management) and makes checking for malware on the platform impossible.

Motivation

The suggested approach is *flexible and future-proof*. The design can be adapted to 1) add extra security requirements on a case-by-case basis and 2) adapt to the (changing) requirements given by the law (BEKJ, see *References*).

Malware checks by the platform are considered to be essential (and Art. 27, Abs. 2 of the "Botschaft zum BEKJ", see *References*).

Assumptions about the IT infrastructure of entities interacting with the platform is not possible. Regulations regarding the IT infrastructure cannot be enforced.

Implementation of the hybrid approach can be conducted in multiple phases. This allows building the foundations, thoroughly testing the fundamental parts and at later point extend the functionality.

	Technical View	Functional View	Compliance View
Step 1: **MVP / Pilotbetrieb**	In the hybrid approach, on-platform encryption is used by default; for example for communication between a lawyer's office and an official administrative body. Using OPE allows the platform to check all data being transferred via the platform for malware. The use of a hardware security module (HSM) is part of this phase whereas a public key infrastructure (PKI) is *not* yet required.	The platform is required to be usable by the broad community and not only by IT experts. Users must not be excluded from using the platform due to technical barriers. Hence, requiring users to manage their own cryptographic material is not an option. Nevertheless, the platform should be flexible to cover additional/future (functional) requirements.	The existing draft of the BEKJ law (see *References*) is adhered to. For encryption topics, the relevant requirements are the safeguarding of the information incl. malware-checking on the platform as defined in articles 26 and 27 of the mentioned draft law.
Step 2: **Add-on**	The foundations of the platform can be extended to fully achieve a hybrid solution by supporting E2E encryption. This requires setting up a PKI to enable entities to check the authenticity of public keys. For communication between "trusted" entities (such as judicial authorities), they can opt to use E2E encryption which requires additional effort by the communication parties as they need to locally manage their key material.	To determine whether or not to use end-to-end encryption, judicial authorities can use a risk-based approach using factors such as 1) risk-taking propensity, 2) sensitivity of the documents being exchanged via the platform, 3) convenience of the platform managing cryptographic material, etc.	The approach considers future changes to the law towards optionally using E2EE through requirements from stakeholders likely. Therefore, these potential requirement changes should be considered from the beginning (in the sense of a basic investment for the future).

3. General Approach

3.1. Principles

- The encryption is realized on a per-dossier basis. All files ("Akten" und "Aktenstücke", aka "Sendung") contained in a dossier are encrypted using the dossier-specific key.
- The encryption is conducted using OPE by default and E2EE in specific use cases (see Section 5). The platform also guarantees the integrity of the stored data.
- The platform can perform encryption of selected, sensitive metadata (e.g., subject of the dossier).
- The platform provides a mechanisms for users to check the authenticity of other entities.
- Access sharing/delegation is realized with "key wrapping" (also called "key encapsulation"). This avoids re-encryption of the data stored on the platform.
- In addition to cryptographic mechanisms, the platform performs strict authorization checks for users accessing data stored on the platform (e.g. for sub-documents within a file).
- The platform assists with the key exchange between entities that opt to use E2EE.

3.2. Definition "Entity"/"Profiles"

We refer to "entities" as an abstraction of people/groups that interact with our platform. Each entity has an associated profile on the platform. The profile is created during the on-boarding phase or after the first login.

We distinguish between the following entities:

- Users / User profiles: A person who uses the platform to consult filed documents or to participate in electronic communication in the judicial field.
- Authorities / Organisations / Groups / Organisation Profiles: An authority that uses the platform to participate in electronic communication in the judicial field (e.g., prosecutors office). A user might be affiliated with multiple authorities.
- Technical users: Technical users are a subset of users that interact with the platform **only** through its API.

3.3. Definition "Cryptographic Profile"

A cryptographic profile of a specific entity represents a *container* that contains all cryptographic material (such as encrypted keys) associated to that specific entity.

For example, the cryptographic profile of entity A contains encrypted dossier keys that were encrypted using entity A's public key and entity A's encrypted private key (see Section 4.6).

3.4. Public Key Infrastructure (PKI)

Note that a PKI is only required for all entities that opt to use E2E encryption. For on-platform encryption (OPE) a "key server" that provides the public keys for a specific entity is sufficient (trust into the platform is required). Nevertheless, we assume that *all profiles* registered on the Justitia.Swiss platform obtain a public and private key pair during profile creation.

The platform maintains an internal public key infrastructure with an internal certificate authority (CA). The root CA keys are stored offline in a hardware security module (HSM). An intermediate CA is used to issue certificates.

Each entity that is registered on the platform has a public/private key pair. This key pair is generated on the client during the registration process. The generation is conducted during the profile creation. The process is opaque to the user and requires no interaction.

- The public part of the key pair is registered at the PKI of the platform, which in turn issues a certificate that binds the public key to a unique identifier of the entity.
- The private part of the key pair represents the "master key" for the encrypted data stored on the platform.

3.4.1. Handling of private keys

Case A: An external "key-protecting token" is used

The private part of the key pair is stored in encrypted from in the database of the application.

During the login process of an entity, the encrypted private key is loaded into the HSM and decrypted. The key required for the decryption is derived based on a key derivation function (KDF) which takes a user-provided token as an input.
Thus, the private key can only be decrypted if the user is "logged in". Furthermore, only the private keys of active entities need to be kept inside an HSM. The encrypted private keys for offline entities can reside in memory.

Case B: No "key-protecting token" is used

The private key part of a profiles key pair is encrypted using a "profile-protecting key" and stored in encrypted form in memory.

3.4.2. Requirements for PKI

The platform's PKI needs to satisfy the following (standard) requirements:

1. entities that have a valid profile on the platform should be able to register their public key at PKI.
2. the PKI must then generate a certificate for registered public key that binds the key to the entity's identifier.
3. entities must be able to check the certificates issued by the PKI.

3.5. Hardware Security Module (HSM)

The platform contains a hardware security module (HSM) that is used for the following four purposes:

1. generation of public/private key pairs for profiles that use 1) OPE, 2) employ the platform for bootstrapping in E2EE.
    1. entropy source is of higher quality than software-only mechanisms (such as HashiCorp Vault).
    2. lower predictability compared to software-only approach.
2. derivation and temporary storage of "profile-protecting keys" based on an 1) HSM-internal secret key ("HSM seed") and 2) external input ("key-protecting token").
    1. derivation is session-based.
3. maintenance of private keys of profiles while they are active. The goal is that the private key (in unencrypted form) is available during a "session" inside the HSM.
   This includes:
    1. loading encrypted private keys from database.
    2. decrypting encrypted private keys of a profile (symmetric decryption) with key from key derivation step above.
    3. store private keys in HSM while a profile is active.
    4. encrypting private key of a profile (symmetric encryption).
        1. Reverse operation of previous step.
        2. Only needed for profile creation and rotation of public/private key pair.
    5. decrypt encrypted symmetric keys using private key (asymmetric decryption).
4. storage of the root key for the PKI (E2EE case only).

3.5.1. Functional requirements

The HSM must satisfy the following functional requirements for the use-cases described above.

1. interface from the platform using PKCS#11
2. load / store operations from database.
3. symmetric encryption/decryption (e.g., based on AES-GCM).
4. asymmetric encryption/decryption (e.g., based on ECIES).
5. storing of private keys within HSM.
6. key derivation function with HSM-internal secrets and external input.
7. export/import of HSM seed via HW-token and PIN

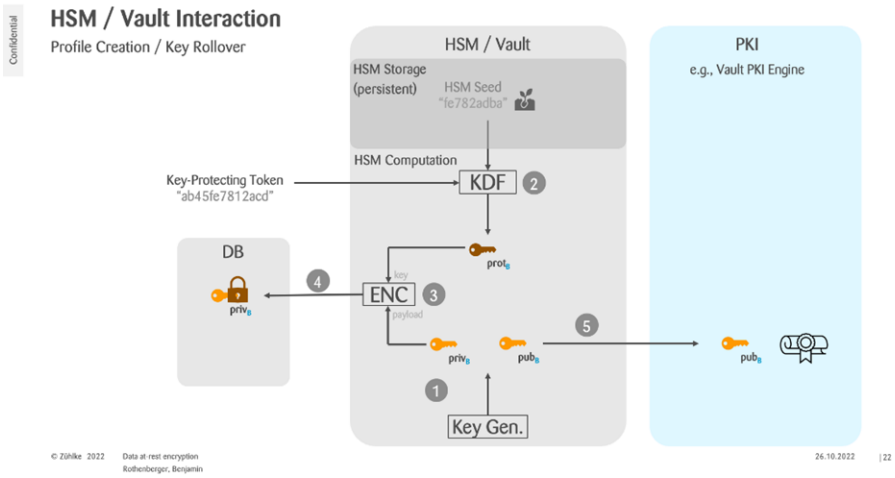
3.5.2. Performance / Scale requirements

	Partial operation **(2024-2027)**	**Full operation** **(2028+)**
Concurrenct users	5'000	15'000 (12'000 lawyers + 20% judicial authority members)
Concurrent symmetric encryption/decryption operations (avg.)	85 operations per second	250 operations per second
Concurrent asymmetric encryption/decryption operations (avg.)	170 operations per second	500 operations per second
Concurrent key derivations (avg.)	35 operations per second	100 operations per second

3.5.3. HSM Interactions

In the following, the main interactions between the HSM and the application are described. Some interaction such as HSM-login are omitted for clarity.

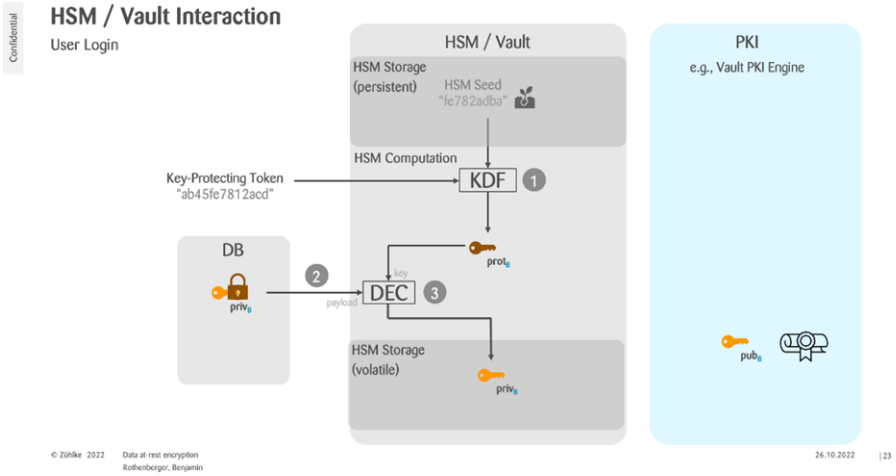
3.5.3.1. Profile Creation / Key Rollover



Creating a profile involves the following steps:

1. Key generation of a public/private key pair for the profile to be created.
2. Key derivation based on HSM seed (persistent secret key in the HSM) and external key-protecting token.
3. Encryption of the private key using the key derived in the previous step.
4. Store the encrypted private key in the DB.
5. The public key is:
    1. published on a key server for OPE.
    2. registered with the platform's PKI in case of E2EE. The PKI generates a certificate for the public key.

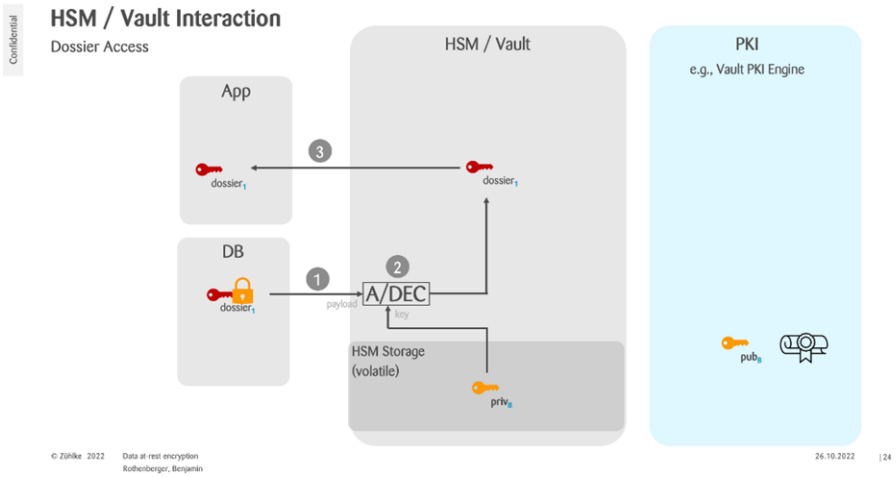
3.5.3.2. User Login



User login requires the following steps from an HSM:

1. Key derivation based on HSM seed (persistent secret key in the HSM) and external key-protecting token.
2. Loading of the encrypted private key from DB.
3. Decryption of the private key using the key derived in the previous step. The private key should be “maintained” in the HSM by storing it.

3.5.3.3. Dossier Access



In case a user wants to access a dossier, the HSM needs to conduct the following steps:

1. Get encrypted dossier key from database
2. Decrypt encrypted dossier key using private key (currently stored in HSM).

3. Pass the dossier key to application. The application should cache the dossier key among multiple requests.

4. Design

In the following, we describe which steps are required to:

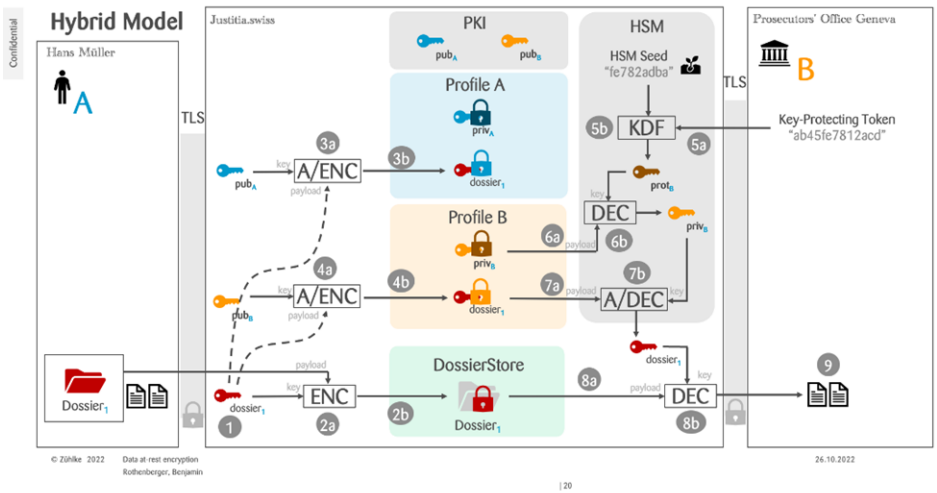
1. create a profile
2. create a dossier
3. share access to a dossier (both OPE and E2EE variants)
4. use shared access to a dossier

4.1. Creation of a Profile

The creation of a profile includes the following steps:

1. generation of public/private key pair in the HSM
2. fetching the key-protecting token
3. deriving the profile-protecting key
4. encrypting the private key using the profile-protecting key
5. publishing the public key at a key server (OPE) / registering the public key with the PKI (E2EE)

4.2. Creation of a Dossier



4.2.1. Step 1: Generation of key dossier$_1$

The platform locally generates a symmetric key dossier$_1$ using a pseudo-random number generator (PRNG).

4.2.2. Steps 2a & 2b: Encryption and storage of dossier documents

The $dossier_1$ key is used to encrypt all the documents of dossier 1 and store them in the dossier store (blob storage).

Note: Encryption and storage of the document is conducted asynchronously.

4.2.3. Steps 3a & 3b: Encryption and storage of the $dossier_1$ key

The key $dossier_1$ is encrypted using entity A's public key and stored in the database of the platform.

4.3. Sharing Access to Dossier 1 with Entity B (OPE Variant)

4.3.1. Steps 4a & 4b: Encryption and storage of the $dossier_1$ key for entity B

To allow entity B access to dossier 1, encrypts the $dossier_1$ key using the public key of entity B and stores it in the database of the platform.

4.4. Entity B accesses Dossier 1

Entity B logs into the platform and wants to access documents of dossier 1.

4.4.1. Steps 5a & 5b: Derive profile-protecting key

The derivation of the profile-protecting key uses a key derivation function with the HSM seed and the key protecting token as an input. The key protecting token is fetched as an external input.

4.4.2. Steps 6a & 6b: Decrypt entity B's private key

The private key of entity B is decrypted using the profile-protecting key and and stored inside the platform's HSM memory. The private key is maintained in the HSM while the user is active.

4.4.3. Steps 7a & 7b: Decrypt $dossier_1$ key

The HSM fetches the encrypted dossier1 key from the database and decrypts it using entity B's private key.

4.4.4. Steps 8a & 8b: Decrypt documents of $dossier_1$

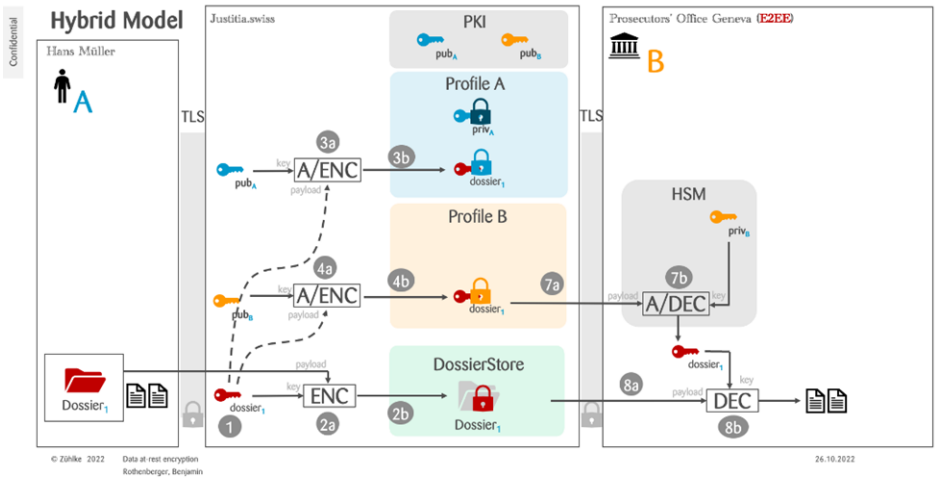
The applications obtains the decrypted $dossier_1$ key from the HSM and can use it to decrypt the documents of $dossier_1$.

4.4.5. Step 9: Entity B can access the documents of dossier$_1$

Finally, entity B has access to the documents of dossier$_1$

4.5. Sharing Access to Dossier 1 with Entity B (E2EE Variant)

In the following case, entity B uses the E2EE variant to access dossier1 and local key management for the private key is used. Accordingly, the steps 5-6 are omitted.



5. Usage Scenarios

5.1. Scenario 1: Communication from a Lawyer's Office to Authority using OPE

This is the default communication scenario.

The first scenario focuses on communication using the J40 platform between a lawyer's office (or simply lawyer) to an official authority. The communication happens under the following circumstances:

- The authority does neither "trust" the lawyer nor the documents submitted by the lawyer (e.g., they could contain malware).
- The authority opts for malware checks on the platform.

Consequently, on-platform encryption (OPE) is being used to encrypt the files submitted by either of the entities. These files could be submitted either via the API or the SPA.
Before encrypting a file the platform checks the documents for malware using the on-platform

malware scanning service. Finally, the platform shares access to the files to the receiving entity using the steps described in section 4.3.

5.2. Scenario 2: Communication between Authorities using E2EE

This scenario describes communication using the J40 platform between two judicial authorities (such as courts, cantonal bureaus, etc.) and considers the following settings:

- Both authorities "trust" each other and the documents submitted via the platform by either of the authorities.
- Both authorities have a local malware scanning solution installed (e.g., in their local IT infrastructure).
- Furthermore, the authorities have a high security demand and don't want to rely on the platform for encryption of the files.

Consequently, end-to-end encryption (E2EE) is employed where the public/private key pair is managed by the authorities.

This scenario includes the following steps:

1. The authorities use the platform to exchange their public keys and check the authenticity of the public keys via the PKI provided by the platform.
2. Both authorities signal the platform that the files exchanged via the platform for this specific case must use E2E encryption.
3. The authorities exchange files either using the API or the SPA.
4. Each file is locally checked for malware upon receival.

5.3. Scenario 3: Communication from a Lawyer's Office to Authority using Hybrid Encryption

lawyer (or a lawyer's office) wants to communicate with an authority.

- The authority does not trust the lawyer.
- The authority opts for malware checks on the platform.
- The authority wants to manage its own keys using their local IT infrastructure.
- OPE for the lawyer and E2EE encryption for the authority is being used.

Note: no E2EE guarantees are given in this scenario.

6. Technical Scenarios

6.1. Membership changes for authority profiles

In case of membership changes with authority profiles, the following options are possible:

1. changing the dossier key and re-encrypting all data in the dossier (full re-encryption)
2. changing the dossier key and re-encrypting all *future* data in the dossier (lazy re-encryption)
3. not changing the dossier key (previous authority members potentially have access to future contents of the dossier)

6.2. Change of a profile's key protecting token

Changing the key protecting token implies that the key used to encrypt the private key part cannot be derived correctly anymore. Thus, the following steps are required:

1. the private key part needs to be loaded into the HSM and decrypted.
2. a new master key needs to be generated based on the new key protecting token.
3. the master key is used to re-encrypt the private key and store it in the entity's cryptographic profile.

6.3. Change of a profile's public/private key pair

Changing a public/private key pair requires re-encrypting all dossier keys of the corresponding profile with the new public key.

1. a new public/private key pair needs to be generated.
2. the new public key needs to be published at the key server (OPE) / registered with the platform's PKI (E2EE).
3. the private key part needs to be loaded into the HSM and decrypted.
4. The dossier keys of the profile need to be decrypted and re-encrypted using the new public key.

6.4. Change of a dossier key

Changing the dossier key requires re-encryption of the dossier data. Given that a dossier might be large re-encryption of the content should be avoided in most cases.
In case a dossier key gets compromised, re-encryption of the dossier data cannot be avoided.
Changing the dossier keys involves the following steps:

1. Generate a new dossier key.
2. Re-encrypt the data with the new dossier key.

3. Encrypt the dossier keys with the public key of the corresponding entity and store them in the respective profiles.

7. References

BEKJ: https://www.bj.admin.ch/bj/de/home/staat/gesetzgebung/e-kommunikation.html

Botschaft zum BEKJ: https://www.fedlex.admin.ch/eli/fga/2023/679/de

Appendix

Cryptographic Algorithms

In the following, we list cryptographic algorithms that could be considered for the implementation of the data at-rest encryption for Justitia.swiss.

The selection is limited to 1) up-to-date and 2) widely-used ciphers. It is important that the selected algorithms can be used in the Springboot backend (using a library) and in the HSM.

1.1. Recommendation

We recommend the following selection of algorithms:

- Symmetric cipher: AES256-GCM
- Asymmetric cipher: ECC
  - ECIES (ECDH + AES256-GCM + HMAC-SHA3-256) for encryption
  - ECDH (secp384r1) for key agreement.
  - ECDSA (secp384r1) for signatures.
- Hash function: SHA3-384
- MAC algorithm: HMAC-SHA3-256
- Key derivation function:
  - Derivation from a password: Argon2
  - Derivation from key material (e.g., symmetric key): HKDF with SHA3-256 as a hash function (to get 32 B keys)

Our recommendations also are highlighted below.

For the selection of key / ciphertext / hash lengths was compared to the current recommendations which are summarized here: https://www.keylength.com/en/compare/

1.2. Quantum-resilient cryptography

This section is added for completeness. But not considered for MVP.

Quantum-resistant cryptography is designed to be secure against attacks from quantum computers. Unlike traditional cryptography, which relies on mathematical problems that are difficult for classical computers to solve but can potentially be solved by quantum computers, quantum-resistant cryptography uses mathematical problems that are believed to be difficult or impossible for both classical and quantum computers to solve. This makes it more secure against potential attacks from quantum computers, which are predicted to be much faster and more powerful than classical computers at certain tasks. Quantum-resistant cryptographic algorithms are currently being standardized and implemented in various applications. While it is still uncertain whether and when quantum computers will become a practical threat to existing cryptographic systems, it is recommended that quantum-resilient cryptography is considered for systems deployed by 2025 and beyond (e.g., NSM 10 from US government).

1.2.1. Impact of quantum computers on standard cryptography

Quantum computers could potentially be used to break some of the algorithms that are commonly used to implement asymmetric cryptography. For example, quantum computers could be used to solve the discrete logarithm problem (Shor's algorithm), which is the basis for many popular asymmetric algorithms such as Diffie-Hellman and DSA. Consequently, if a large-scale quantum computer could be built then, then current public key cryptography systems need to be assumed to be broken. To be specific, a `k`-bit number can be factored in time of order `O(k^3)` using a quantum computer of **5k+1 qubits** (using Shor's algorithm). Accordingly, 256-bit number (e.g. Bitcoin public key) can be factorized using 1281 qubits in 72*256^3 quantum operations (~ 1.2 billion operations).

It is currently not clear yet how much of a threat quantum computers pose to symmetric cryptography (e.g., AES) and cryptographic hash functions (SHA3, blake3). For specific ciphers such as AES, it is known that Grover's algorithm will reduce symmetric cipher security by a square root and thus longer keys are needed (e.g., AES-256). A similar effect is assumed for cryptographic hash functions and thus longer output formats should be considered. Nevertheless, cryptographic hashes (like SHA2, SHA3, BLAKE2) are currently considered to be *quantum-safe*. For more information, see this paper in the attachments.

1.2.2. Quantum-resilient cryptography not in MVP

- The standardization process of quantum-resilient cryptography is still in progress. So far, CRYSTALS-Kyber for encryption and CRYSTALS-Dilithium, FALCON, and SPHINCS+ for digital signatures have been selected by NIST to be standardized.
- The security of quantum-resilient mechanisms have not been extensively analysed yet. Recently an attack on SIKE has been proposed (https://eprint.iacr.org/2022/1026.pdf).
- Quantum-resilient cryptography is currently being tested by companies (e.g., Cloudflare and Google). Practicality problems (e.g., due to the huge signature/ciphertext size) are expected.

Accordingly, in the **design and implementation of the MVP, quantum-resilient cryptography is not considered**.

At a later stage, the use of quantum-resilient cryptography should be re-evaluated.

2. Algorithms for Consideration

Below we list a selection of algorithms/ciphers for consideration to be used in the encryption concept of the Justitia.swiss platform.

2.1. Symmetric Cryptography

Symmetric ciphers in AEAD (Authenticated Encryption with Associated Data) mode are cryptographic algorithms that provide both confidentiality and authenticity for the data they encrypt. The following symmetric ciphers in AEAD mode are extensively used in practice:

1. AES-GCM (Advanced Encryption Standard in Galois/Counter Mode): AES-GCM is a symmetric cipher that uses the AES block cipher algorithm in combination with a cryptographic mode of operation called Galois/Counter Counter Mode (GCM) to provide confidentiality and authenticity for the data it encrypts. It is widely used in a variety of applications, including TLS 1.3, secure storage systems, and encrypted messaging. Furthermore, AES has *hardware acceleration* (AES-NI) on most modern CPUs.
2. ChaCha20-Poly1305: ChaCha20-Poly1305 is another symmetric cipher that uses the ChaCha20 stream cipher algorithm (published in 2008 by DJ. Bernstein) in combination with the Poly1305 message authentication code (MAC) algorithm (published in 2005 by DJ. Bernstein) to provide confidentiality and authenticity. It is often used as an alternative to AES-GCM if no hardware support for AES exists, as it has a lower computational overhead and can be accelerated using vector instructions.
   There also exists the XChaCha20-Poly1305 is a variant that uses an extended nonce to provide even higher security and resistance to attacks.
3. Salsa20-Poly1305: Salsa20-Poly1305 is another symmetric cipher that uses the Salsa20 stream cipher algorithm in combination with the Poly1305 MAC algorithm to provide confidentiality and authenticity. It is similar to ChaCha20-Poly1305, but it has a different structure and design, which makes it more resistant to certain types of attacks. There also exists the XSalsa20-Poly1305 is a variant that uses an extended nonce to provide even higher security and resistance to attacks.

The choice of which cipher to use in a particular application depends on factors such as the level of security required, the computational resources available, and the nature of the data being encrypted. Given that the CPUs used to run the Justitia.swiss platform support AES-NI, an AES-based cipher is used for symmetric encryption.

2.2. Asymmetric Cryptography

Asymmetric cryptographic encryption algorithms are a type of cryptographic algorithm that uses two different keys, a public key and a private key, to encrypt and decrypt data. Widely used asymmetric cryptographic encryption algorithms include:

1. RSA (Rivest-Shamir-Adleman, 1977): RSA is a widely-used asymmetric encryption algorithm that is based on the mathematical problem of factoring large numbers into their prime factors. RSA is widely used in applications such as secure communication, digital signatures, and secure storage. Compared to ECC, RSA requires much longer keys and signatures are required. Furthermore, it is expected that RSA is heading towards the end of its tenure.
2. ECC (Elliptic Curve Cryptography, introduced in 1985 by Victor Miller and Neal Koblitz): ECC is an asymmetric encryption algorithm that is based on the algebraic structure of elliptic curves over finite fields. It is considered to be *more secure and efficient than RSA*, and is often used in applications such as internet protocols, secure storage systems, and encrypted messaging. There exist multiple sub-categories of ECC: ECDH (Elliptic Curve Diffie-Hellman) or ECMQV (Elliptic Curve Menezes-Qu-Vanstone) for key exchange/key agreement, ECDSA (Elliptic Curve Digital Signature Algorithm) for digital signatures (ECDSA was proposed around 2001 by Don Johnson, Alfred Menezes, and Scott Vanstone), and ECIES (Elliptic Curve Integrated Encryption Scheme) to securely exchange encrypted messages.
   Note that ECIES is a hybrid algorithm and thus a ECDH key agreement in combination with a symmetric cipher (such as AES) is used.
   1. Possible curves: https://safecurves.cr.yp.to/
   2. Alternatives to ECDSA: EdDSA (e.g., Ed25519). Ed25519 was proposed in 2011 by Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang.
   3. ECDSA signatures change each time based on the nonce used, whereas EdDSA signatures do not change for the same set of keys and the same message.
   4. ECDSA and EdDSA typically have equivalent performance and security levels.
   5. ECDSA has a random nonce value created within the signature creation, whereas EdDSA does not. In ECDSA, we need to be careful in making sure that we do not reuse this nonce value, and that it is random.

2.3. Hash Functions

Cryptographic hash functions are a type of mathematical function that takes a digital input of any length and produces a fixed-length digital output, known as the hash value. These functions are designed to be one-way functions, meaning that it is computationally infeasible to reverse the function to obtain the original input. This makes them useful for ensuring the integrity of digital data, because if the data has been modified in any way, the hash value will also change, allowing the recipient of the data to detect any tampering.

The following cryptographic hash function families could be considered for use in the Justitia.swiss platform:

1. SHA-2 (Secure Hash Algorithm 2): SHA-2 is a set of cryptographic hash functions are built using the Merkle–Damgård construction, from a one-way compression function itself built using the Davies–Meyer structure from a specialized block cipher. SHA2 was published by the NSA in 2001. The following functions are contained in the familiy: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.
2. SHA-3 (Secure Hash Algorithm 3): SHA-3 is the latest member of the Secure Hash Algorithm family of standards, released by NIST on August 5, 2015. SHA-3 is a subset of the broader cryptographic primitive family *Keccak.* The NIST standard defines the following instances with different output lengths: SHA3-224, SHA3-256, SHA3-384, and SHA3-512. SHA3 is designated to be the successor of SHA2.
3. BLAKE3: BLAKE is a cryptographic hash function based on the ChaCha stream cipher, but a permuted copy of the input block, XORed with round constants, is added before each ChaCha round. BLAKE3 has a binary tree structure, so it supports a practically unlimited degree of parallelism (both SIMD and multithreading) given long enough input. The hash function was designed by Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn and published in 2020. Thus, BLAKE3 is a relatively new hash function.

2.4. Message Authentication Code (MAC) Algorithms

Message authentication codes (MAC) are a short piece of information, known as a tag, that is calculated based on the contents of a message and a secret key. The MAC is used to verify the authenticity and integrity of a message, ensuring that it has not been altered in any way.

1. HMAC (Hash-based Message Authentication Code) uses a cryptographic hash function in combination with a secret key to compute a MAC.
    1. Example: HMAC-SHA512
2. CMAC (Cipher-based Message Authentication Code) uses a symmetric cipher, such as AES, in combination with a secret key to provide message authentication.
    1. Example: AES-CMAC
3. GMAC (Galois/Counter Mode Message Authentication Code) uses the same underlying encryption technique as GCM (Galois/Counter Mode) to provide authenticity for a message.
    1. Example: AES-GMAC

2.5. Key Derivation Functions

A key derivation function (KDF) is a cryptographic algorithm that is used to generate keys from a secret value, such as a password or a passphrase. It is designed to be resistant to attacks such as brute-force guessing, dictionary attacks, and other forms of cryptographic attack. Some examples of secure key derivation functions include:

1. PBKDF2 (Password-Based Key Derivation Function 2): PBKDF2 is a widely-used key derivation function that uses a pseudorandom function, such as HMAC-SHA256, to iteratively generate keys from a password or passphrase.
2. Argon2: Argon2 is a key derivation function that uses a combination of parallelism, memory-hardness, and data-dependent memory access to make it resistant to attacks such as dictionary attacks and GPU-accelerated brute-force attacks.
3. Scrypt: Scrypt is a key derivation function that is designed to be resistant to hardware attacks, such as those using custom hardware or ASICs (Application-Specific Integrated Circuits). It uses a combination of parallelism, memory-hardness, and data-dependent memory access to make it difficult for attackers to efficiently generate keys from a secret value.
4. Bcrypt: Bcrypt is a key derivation function that is based on the Blowfish block cipher algorithm. It uses a variable-round, salted hashing function to generate keys from a secret value, making it resistant to dictionary attacks and other forms of attack.