Extension and Integration of an Abstract Interface to Cryptography Providers

Steve Wagner

EI-3nat

Prof. Dr. Axel Sikora
Dipl.-Phys. Andreas Walz

Institute of reliable Embedded Systems and Communication Electronics (ivESK)
Offenburg University of Applied Sciences

January 29, 2016

Table of contents

1 Motivations

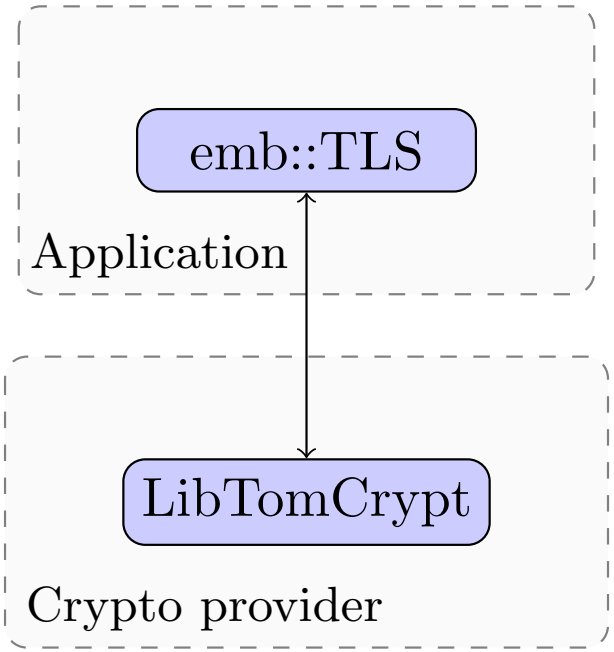
2 Generic Cryptographic Interface

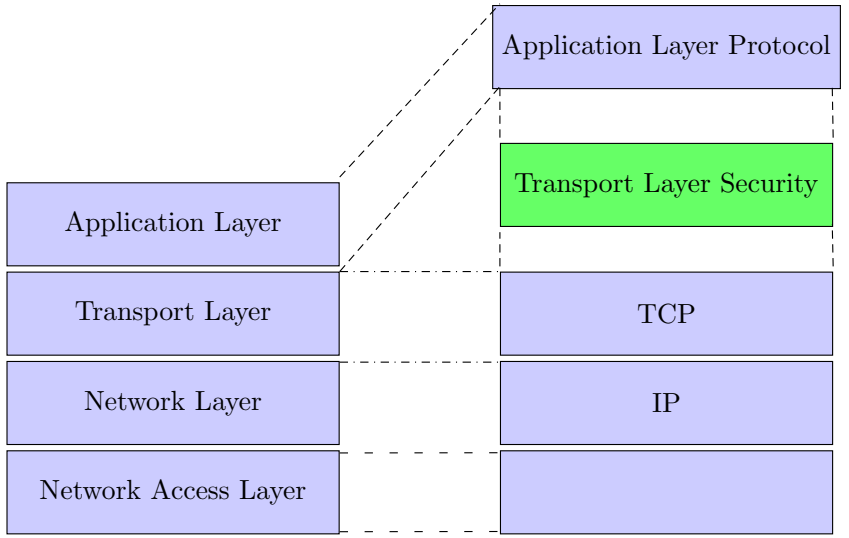
3 Implementation

4 Results

5 Conclusion

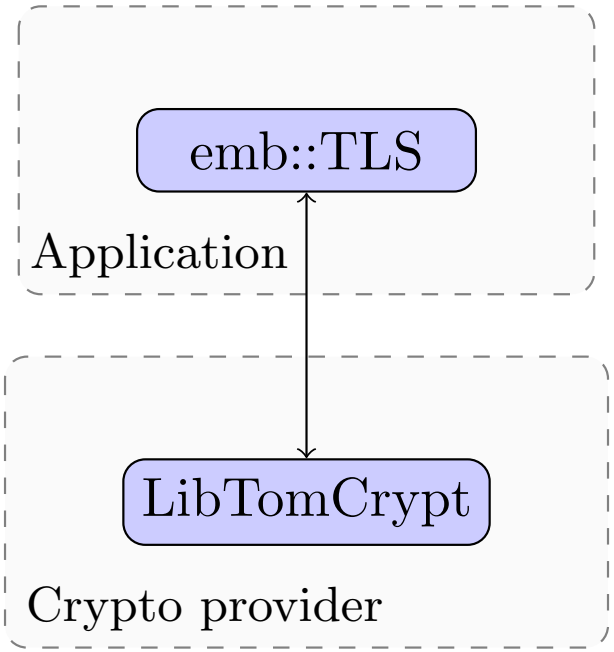
Motivations





- emb::TLS
  - Transport Layer Security (TLS) application for embedded systems
- LibTomCrypt
  - open source cryptographic software library

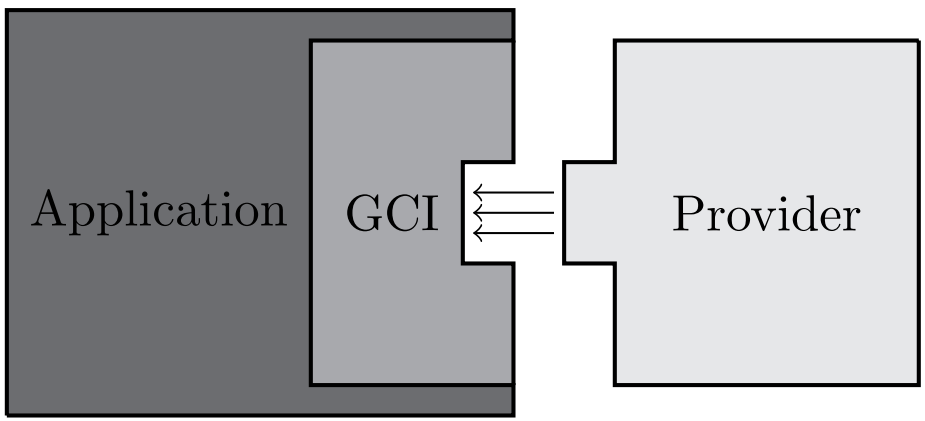
Motivations



Requires significant manual effort:

- Only LibTomCrypt is supported as cryptographic provider
- Many parts of the application have to be modified to use another provider

Motivations



Generic Cryptographic Interface (GCI)

- Provides a base of cryptography
- Facilitates the support of different cryptographic providers
- Facilitates the addition of new cryptographic algorithms

Interface's requirements

1. No hidden states in the interface
   - Behavior of functions only affected by function parameters
2. Interface shall have the possibility to use different cryptographic providers
   - Software libraries
   - Hardware-based cryptographic modules
3. Interface shall enable the use of a provider's internal key managemet services

Cryptographic services

- Key pair generator
- Key management
- Hash
- Signature
- Symmetric cipher
- Asymmetric cipher
- Diffie-Hellman
- Random number generator

Contexts

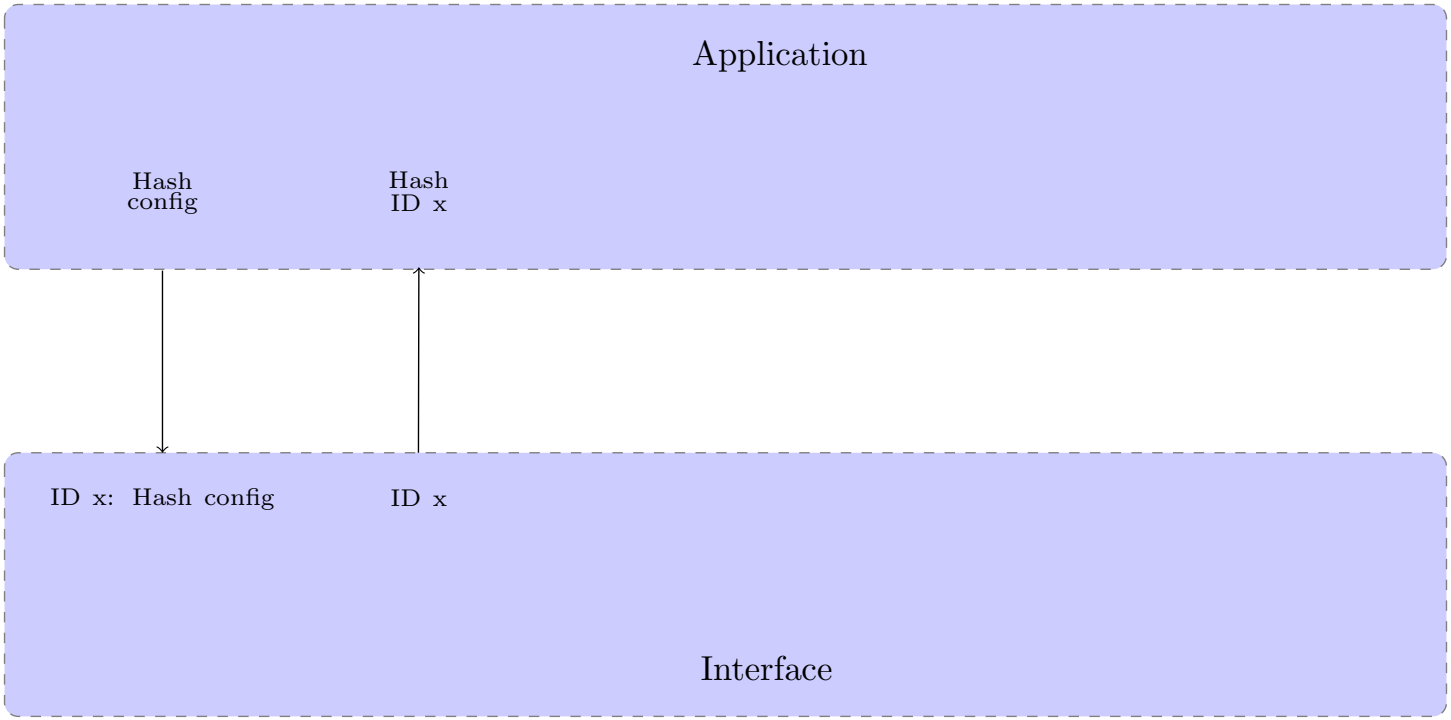
Algorithms which need configurations

- Hash
- Signature
- Symmetric cipher
- Asymmetric cipher
- Diffie-Hellman

Contexts:

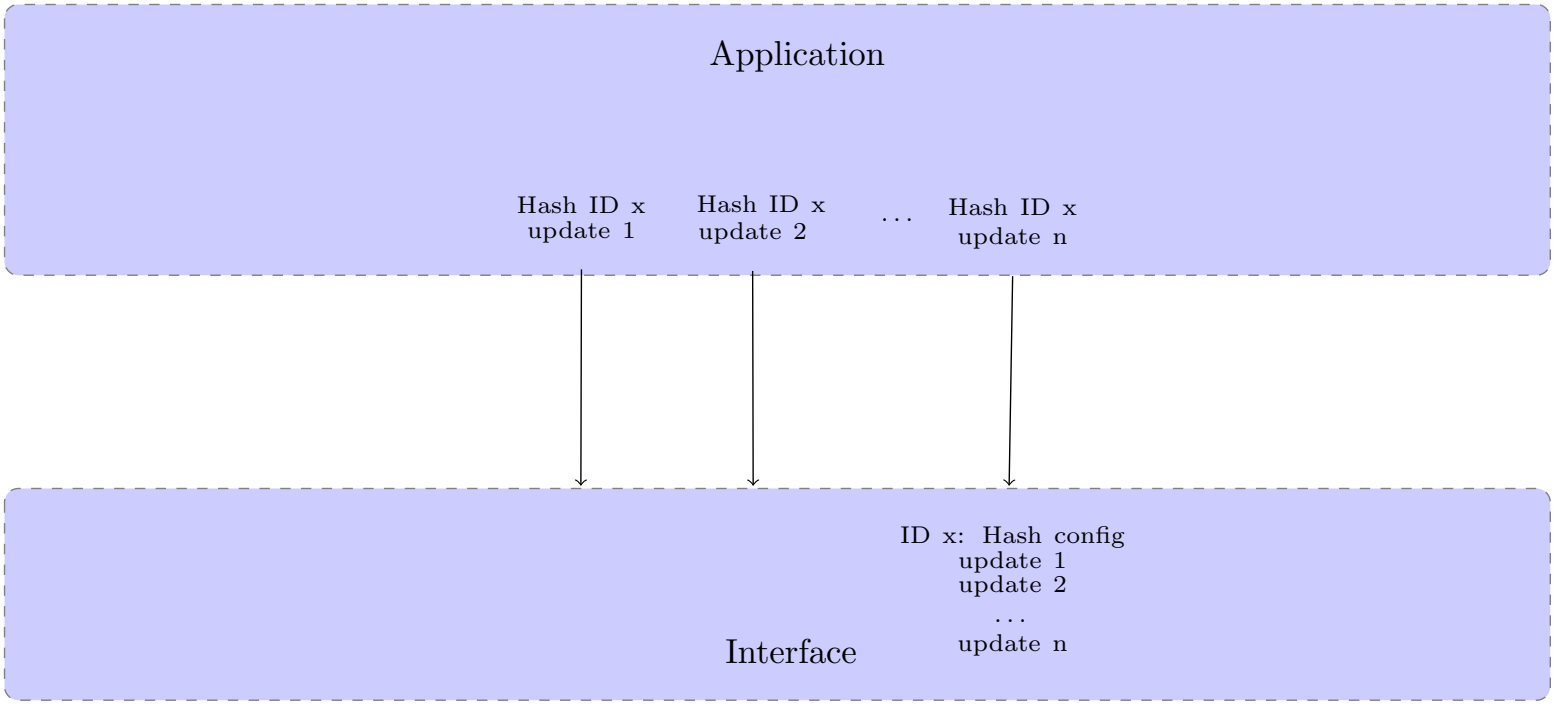
- Represent state of stateful algorithms
- No hidden states in the interface

Context example: Hash



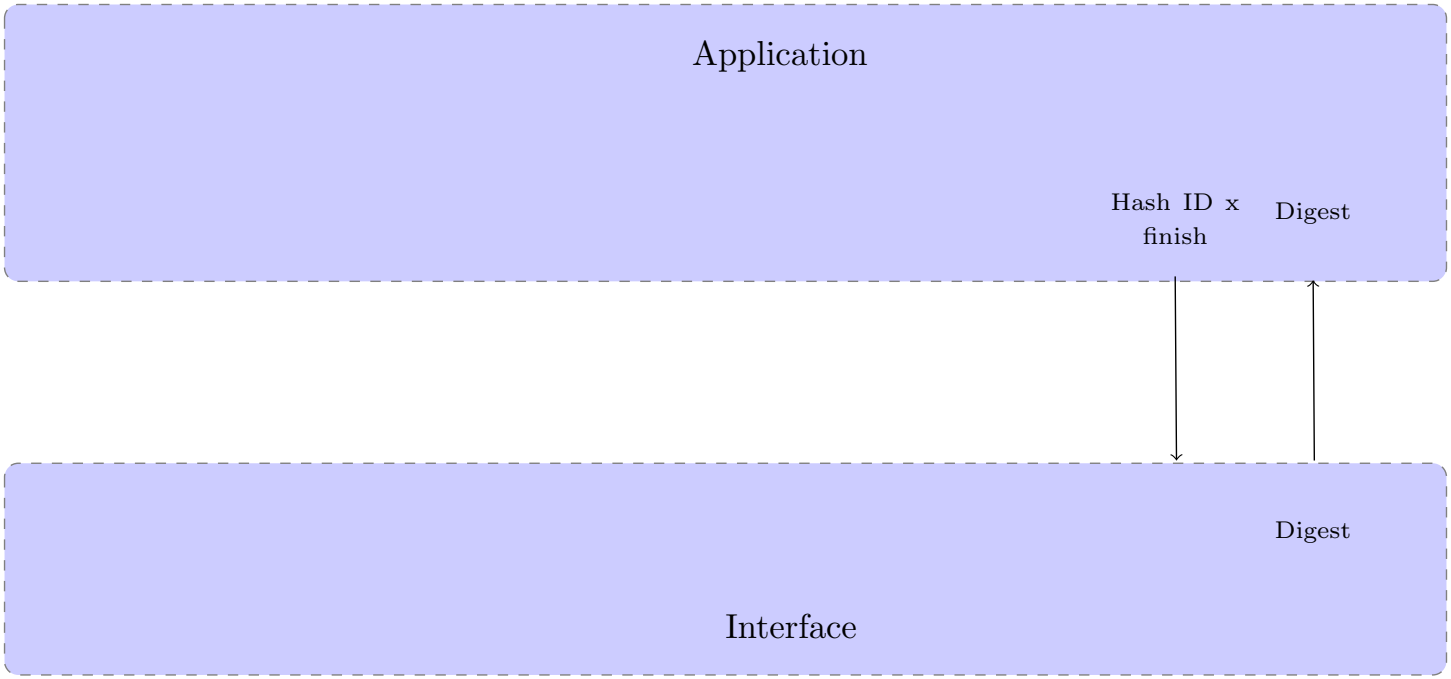
1. Create the hash context with the desired configuration

Context example: Hash



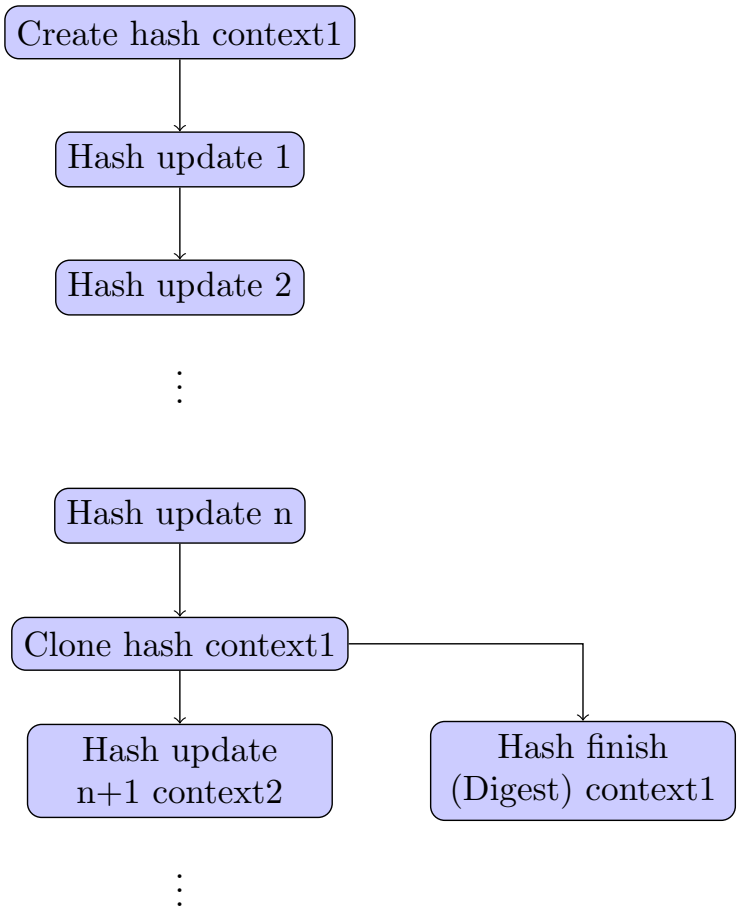
1. Create the hash context with the desired configuration
2. Update the hash context by adding messages

Context example: Hash



1. Create the hash context with the desired configuration
2. Update the hash context by adding messages
3. Get the digest

Context Hash/Signature: Clone



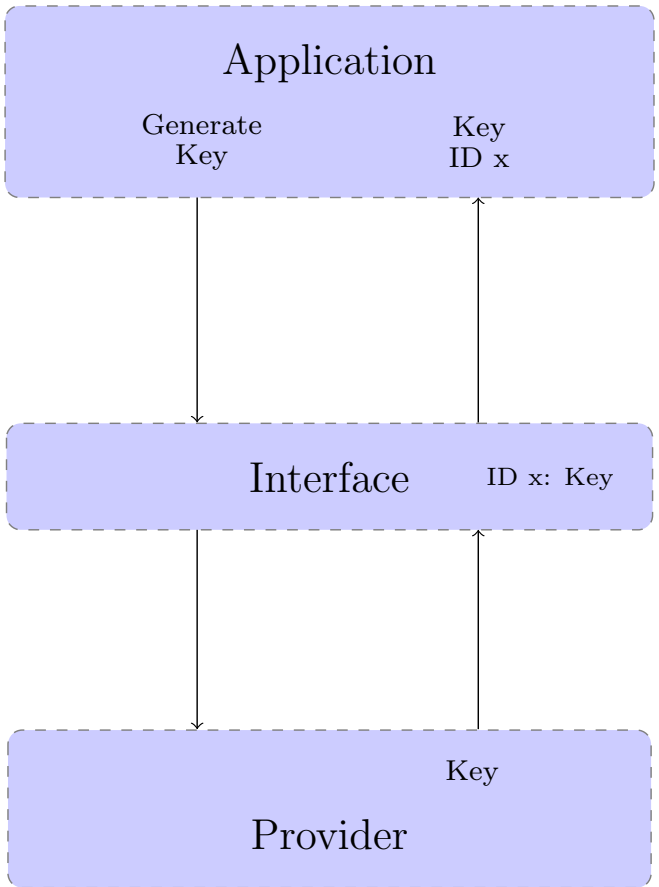
Problems:

- Once the digest has been computed no more updates could be done
- Only the release of the context is possible

Solution: Clone of the context

- Copy of the configuration and the updates of the actual context in another context
- Digest can be calculated for one context
- Other messages can be added to the other context

Key management

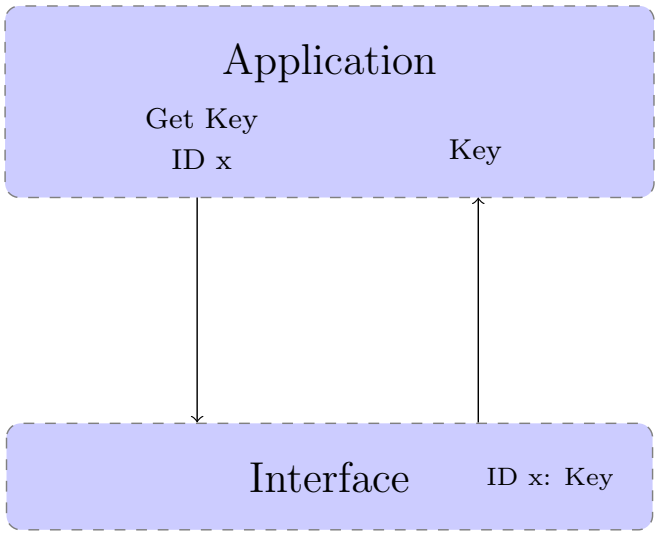


- Interface shall enable the use of a provider's internal key managemet services

Key management:

- Generate the key and store it in the interface
- Return an ID which identify the key

Key management

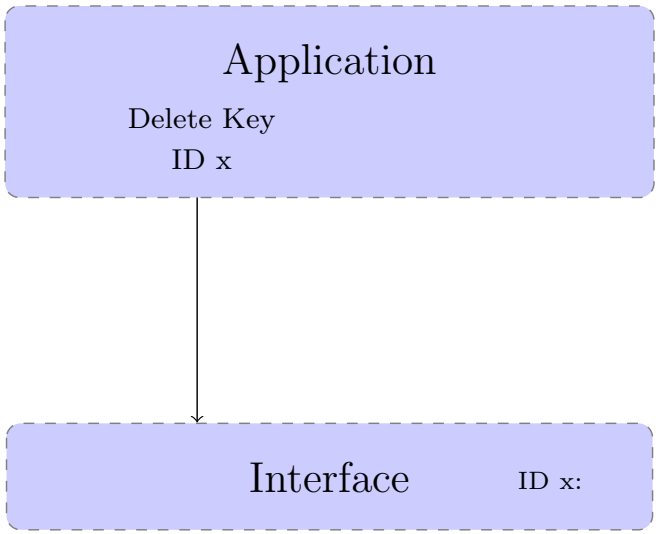


- Interface shall enable the use of a provider's internal key managemet services

Key management:

- Get the key by passing the ID

Key management

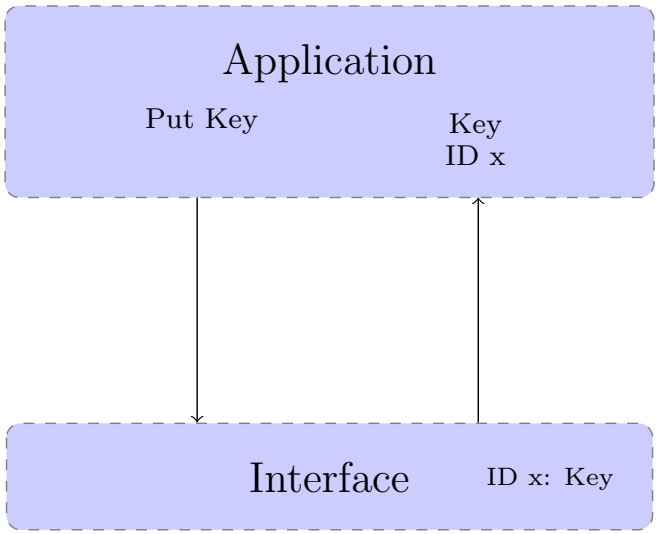


- Interface shall enable the use of a provider's internal key managemet services

Key management:

- Delete the key by passing the ID

Key management

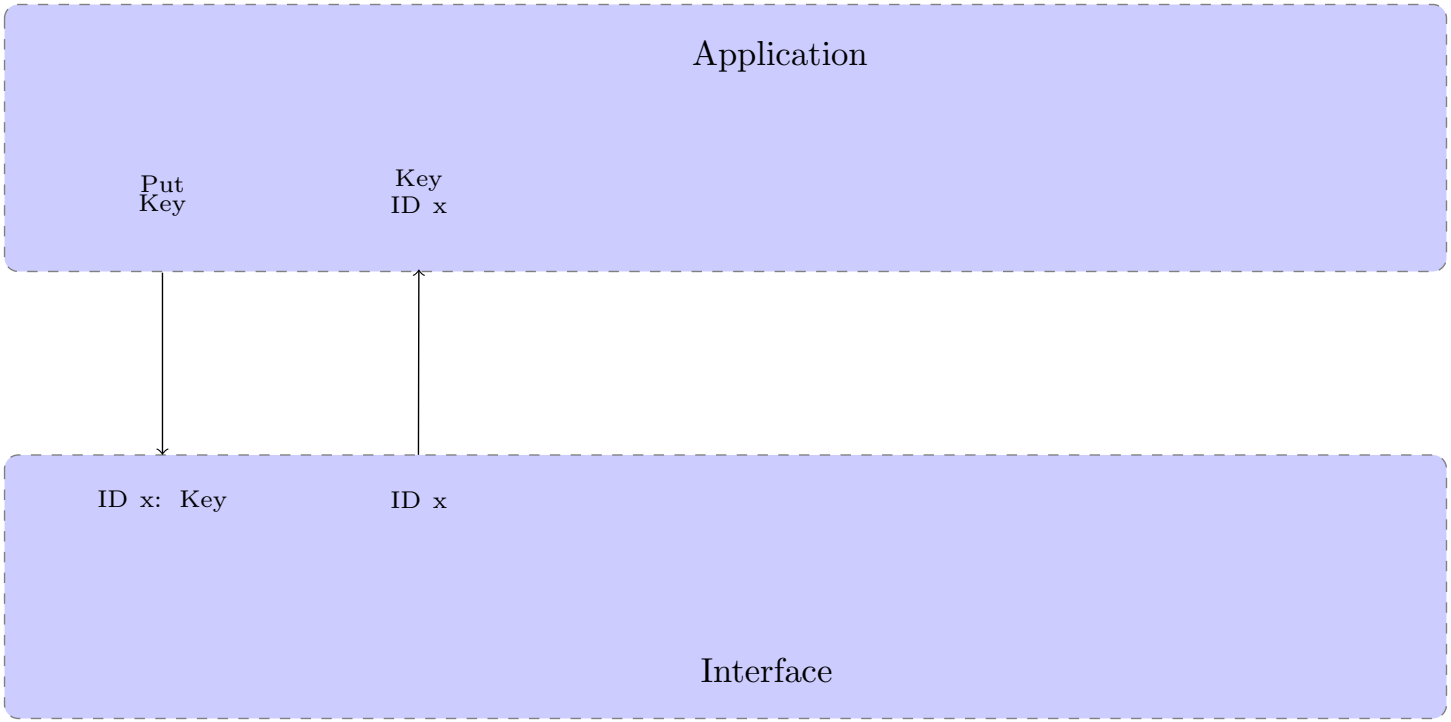


- Interface shall enable the use of a provider's internal key managemet services

Key management:

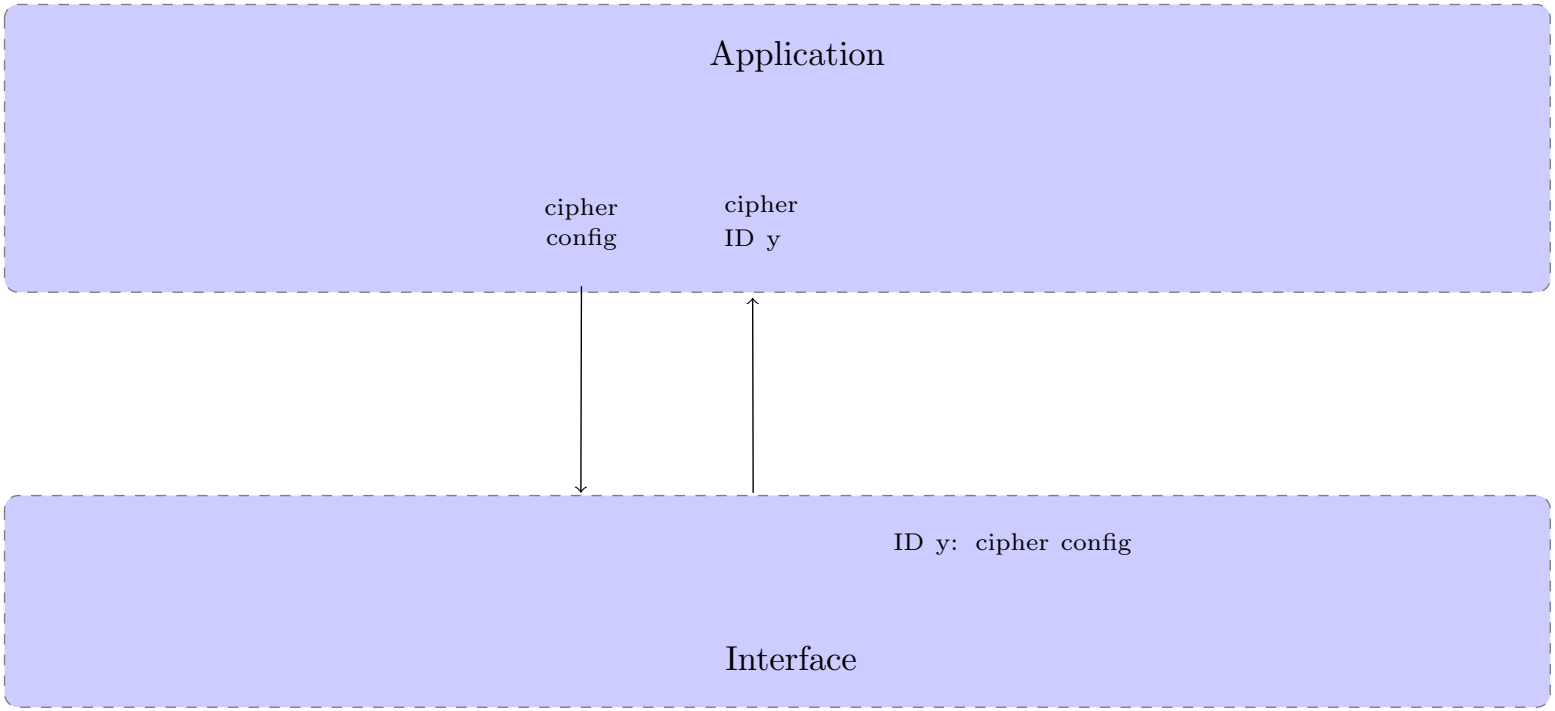
- Store a key coming from outside (not generated by the provider) in the interface
- Return an ID which identify the key

Example: Cipher



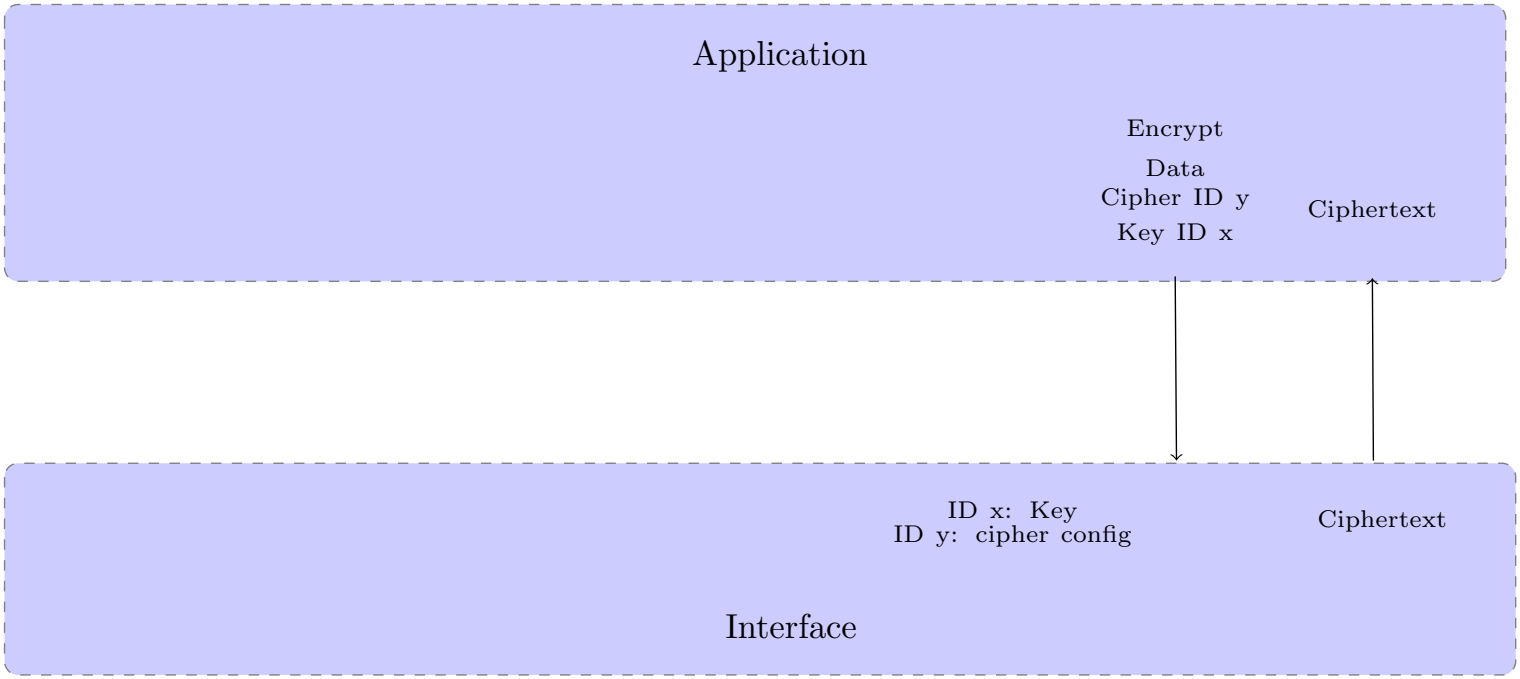
1. Put a key coming from outside to the interface

Example: Cipher



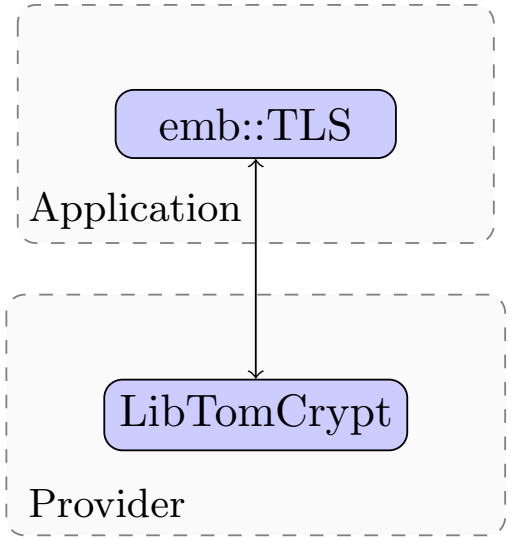
1. Put a key coming from outside to the interface
2. Create the cipher context with the desired configuration

Example: Cipher



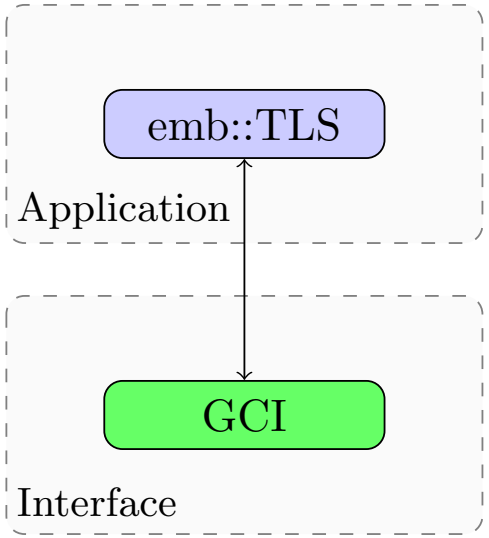
1. Put a key coming from outside to the interface
2. Create the cipher context with the desired configuration
3. Encrypt a data with the key and configuration done previously

Implementation



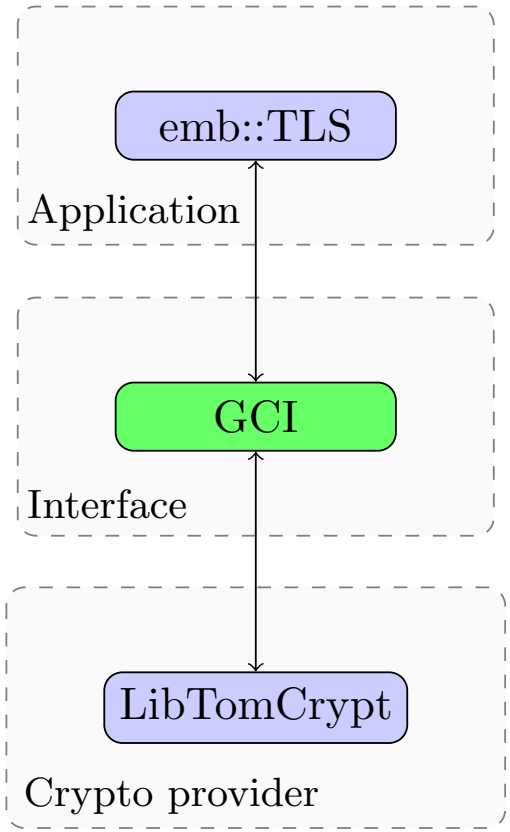
- Previous implementation

Implementation



- Replace the interface instead of the provider

Implementation



- Add a provider (LibTomCrypt) for the interface

Test with cipher suites

```
Protocol Version: tls1_2:

  > -success-: [ passed ] TC0030: Cipher: TLS_RSA_WITH_RC4_128_MD5
  > -success-: [ passed ] TC0031: Cipher: TLS_RSA_WITH_RC4_128_SHA
  > -success-: [ passed ] TC0032: Cipher: TLS_RSA_WITH_3DES_EDE_CBC_SHA
  > -success-: [ passed ] TC0033: Cipher: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
  > -success-: [ passed ] TC0034: Cipher: TLS_RSA_WITH_AES_128_CBC_SHA
  > -success-: [ passed ] TC0035: Cipher: TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  > -success-: [ passed ] TC0036: Cipher: TLS_RSA_WITH_AES_256_CBC_SHA
  > -success-: [ passed ] TC0037: Cipher: TLS_DHE_RSA_WITH_AES_256_CBC_SHA
  > -success-: [ passed ] TC0038: Cipher: TLS_RSA_WITH_AES_256_CBC_SHA
  > -success-: [ passed ] TC0039: Cipher: TLS_RSA_WITH_AES_128_CBC_SHA256
  > -success-: [ passed ] TC00310: Cipher: TLS_RSA_WITH_AES_256_CBC_SHA256
  > -success-: [ passed ] TC00311: Cipher: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
  > -success-: [ passed ] TC00312: Cipher: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
  > -success-: [ passed ] TC00313: Cipher: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
  > -success-: [ passed ] TC00314: Cipher: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  > -success-: [ passed ] TC00315: Cipher: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
  > -failure-: [ failed ] TC00316: Cipher: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
  > -failure-: [ failed ] TC00317: Cipher: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
  > -failure-: [ failed ] TC00318: Cipher: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
```

- Some cipher suites are used

ex: TLS_ *DHE_RSA* *_WITH_* *AES_256_CBC* *_SHA256*

DHE_RSA: key exchange algorithm

AES_256_CBC: bulk encryption algorithm

_SHA256: MAC with Hash

Test with cipher suites

```
Protocol Version: tls1_2:

  > -success-: [ passed ] TC0030: Cipher: TLS_RSA_WITH_RC4_128_MD5
  > -success-: [ passed ] TC0031: Cipher: TLS_RSA_WITH_RC4_128_SHA
  > -success-: [ passed ] TC0032: Cipher: TLS_RSA_WITH_3DES_EDE_CBC_SHA
  > -success-: [ passed ] TC0033: Cipher: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
  > -success-: [ passed ] TC0034: Cipher: TLS_RSA_WITH_AES_128_CBC_SHA
  > -success-: [ passed ] TC0035: Cipher: TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  > -success-: [ passed ] TC0036: Cipher: TLS_RSA_WITH_AES_256_CBC_SHA
  > -success-: [ passed ] TC0037: Cipher: TLS_DHE_RSA_WITH_AES_256_CBC_SHA
  > -success-: [ passed ] TC0038: Cipher: TLS_RSA_WITH_AES_256_CBC_SHA
  > -success-: [ passed ] TC0039: Cipher: TLS_RSA_WITH_AES_128_CBC_SHA256
  > -success-: [ passed ] TC00310: Cipher: TLS_RSA_WITH_AES_256_CBC_SHA256
  > -success-: [ passed ] TC00311: Cipher: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
  > -success-: [ passed ] TC00312: Cipher: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
  > -success-: [ passed ] TC00313: Cipher: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
  > -success-: [ passed ] TC00314: Cipher: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  > -success-: [ passed ] TC00315: Cipher: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
  > -failure-: [ failed ] TC00316: Cipher: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
  > -failure-: [ failed ] TC00317: Cipher: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
  > -failure-: [ failed ] TC00318: Cipher: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
```

- Most cipher suites are working with this new implementation
- ECDSA as key exchange algorithm isn't implemented

Achieved

- Design of the new cryptographic interface
- Implementation of the new interface in an application (emb::TLS)
- Implementation of a provider (LibTomCrypt) in the interface

Future work

- Implementation of ECDSA
- Test with other cipher suites
- Writing of the new interface's documentation

Thanks for your attention

Questions?