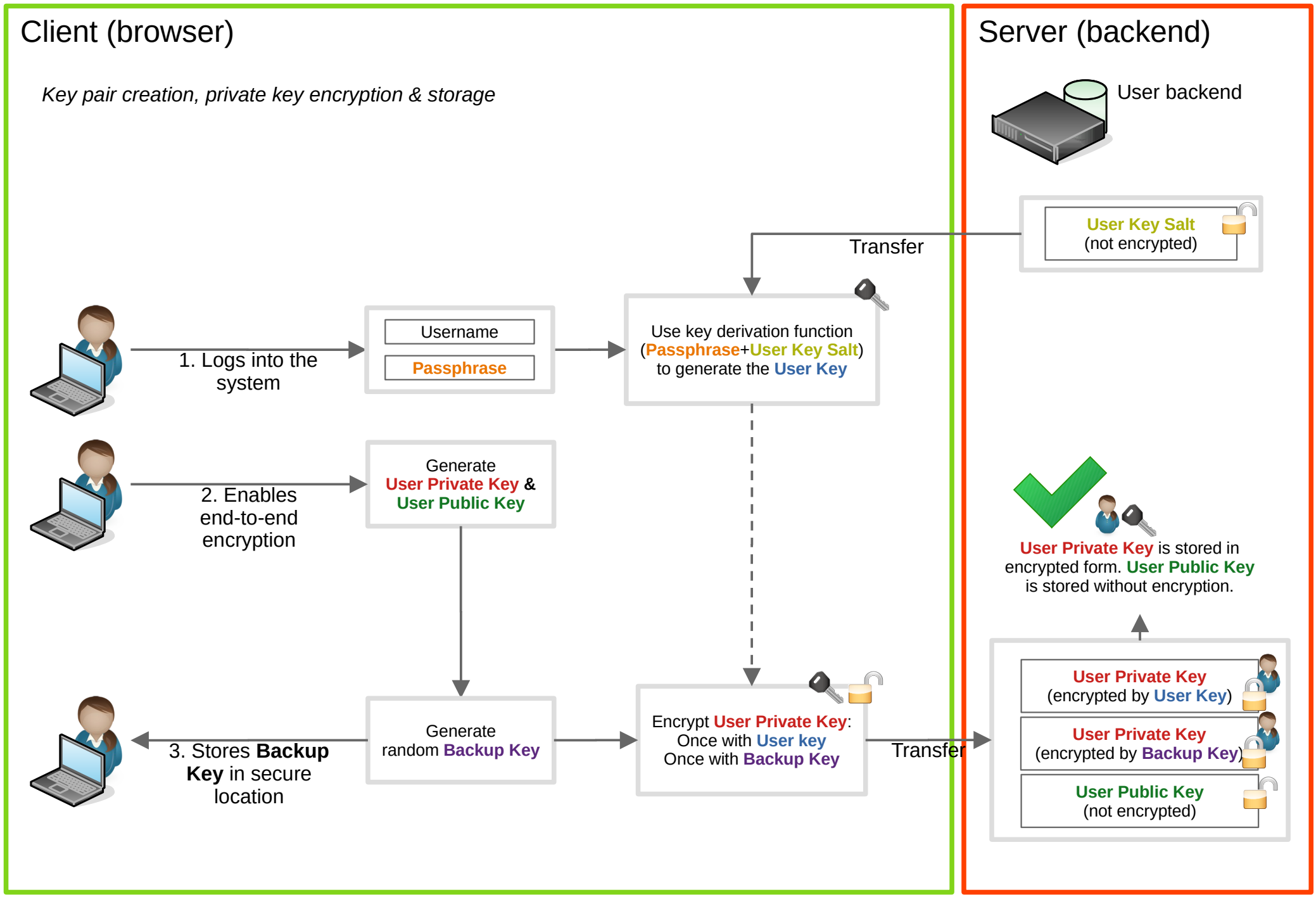
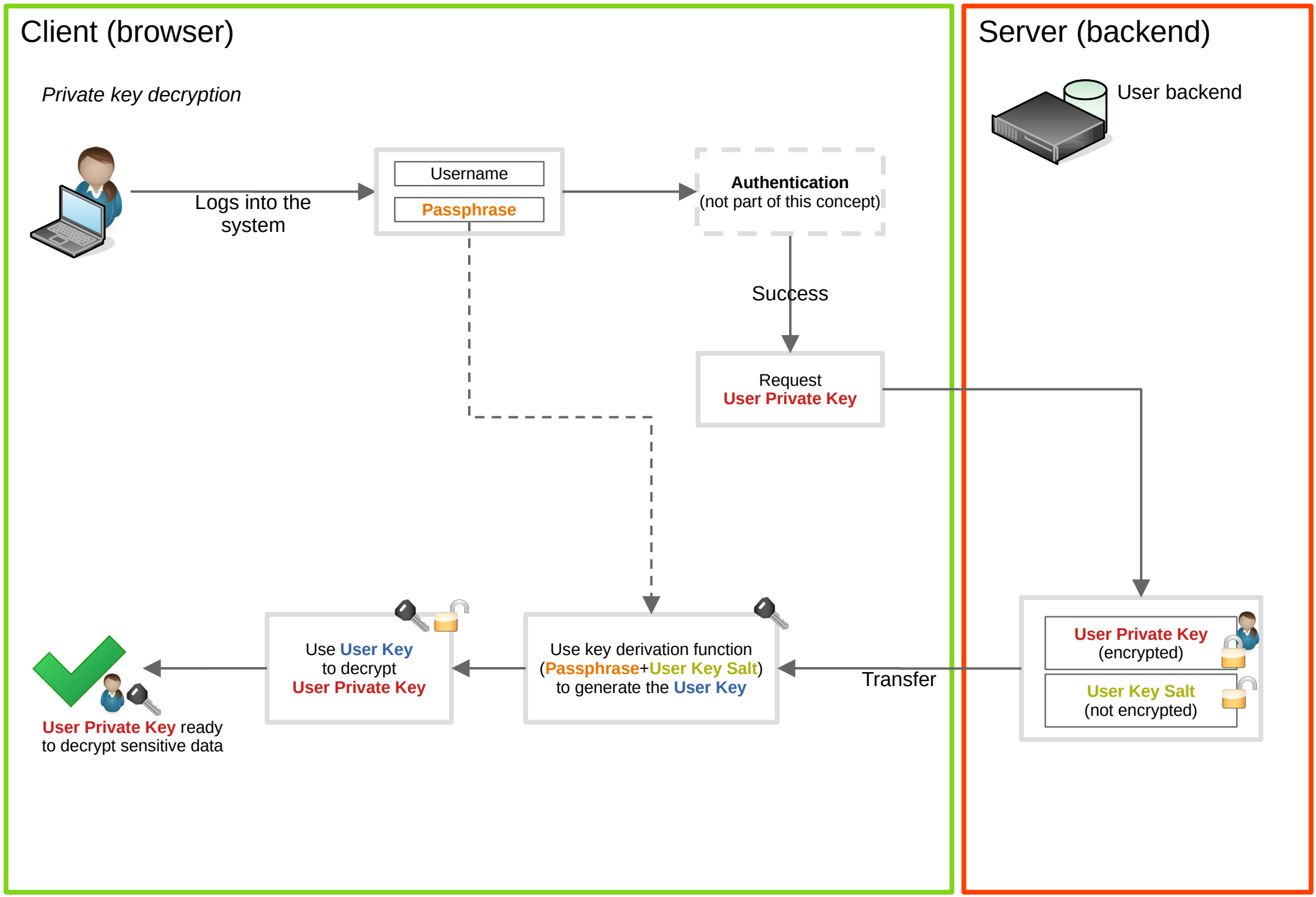
Implementation end-to-end encryption

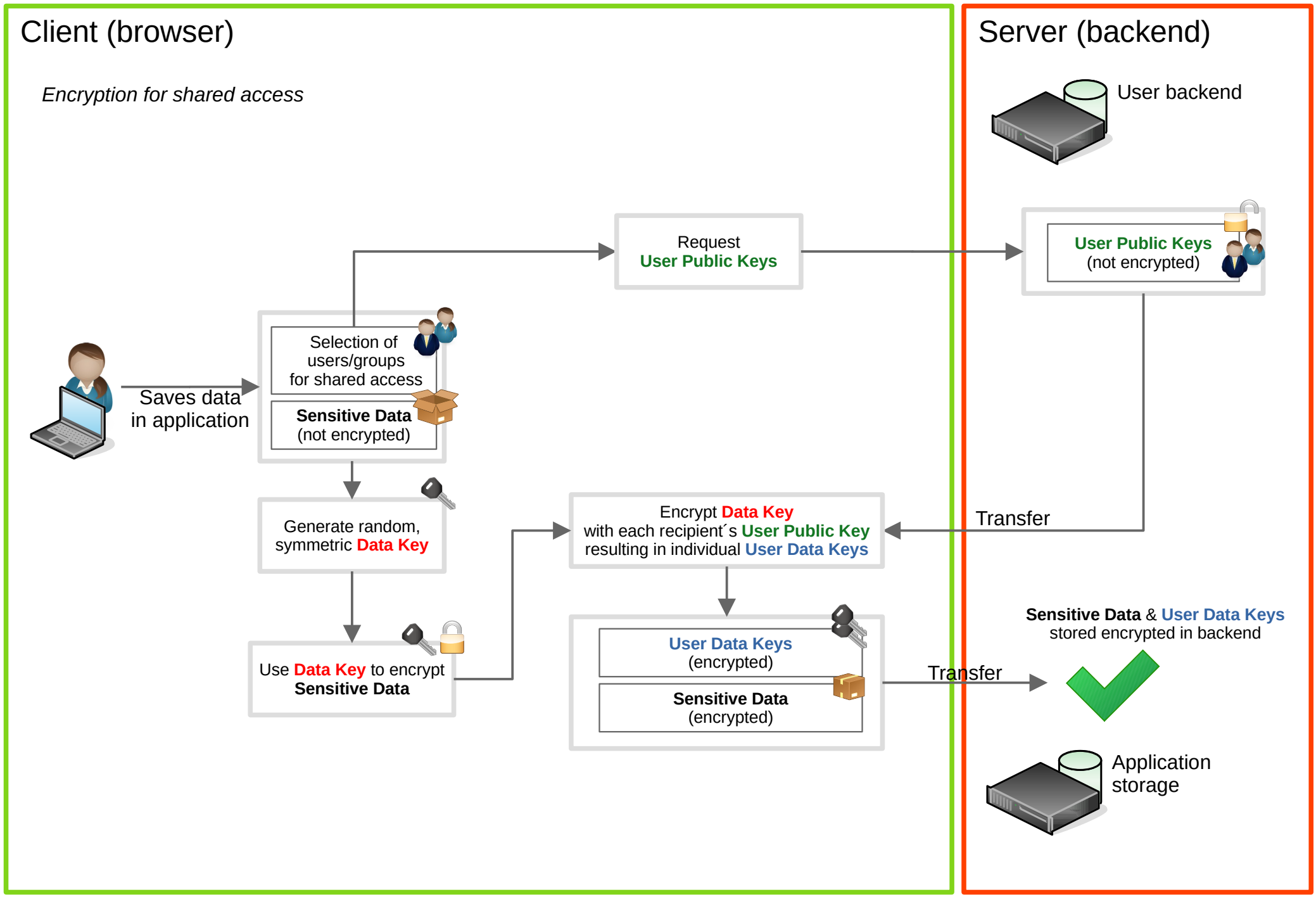
REI3 application platform

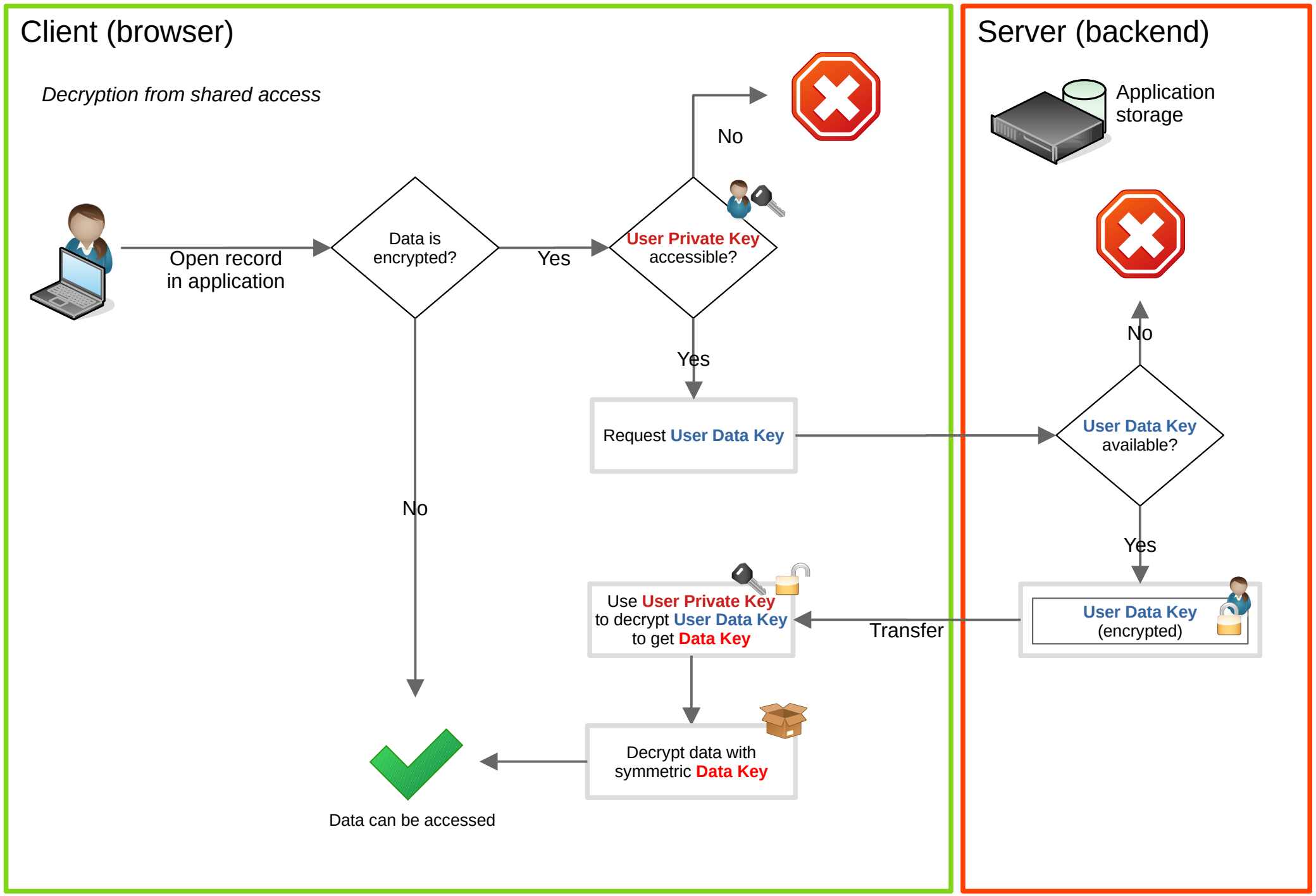
Contributors
Bao Trung Le Nguyen (BaoLe@btln.de): Concept
Fabian Borgwardt (fabian@rei3.de): User feedback/testing
Gabriel Victor Herbert (gabriel@rei3.de): Concept, implementation

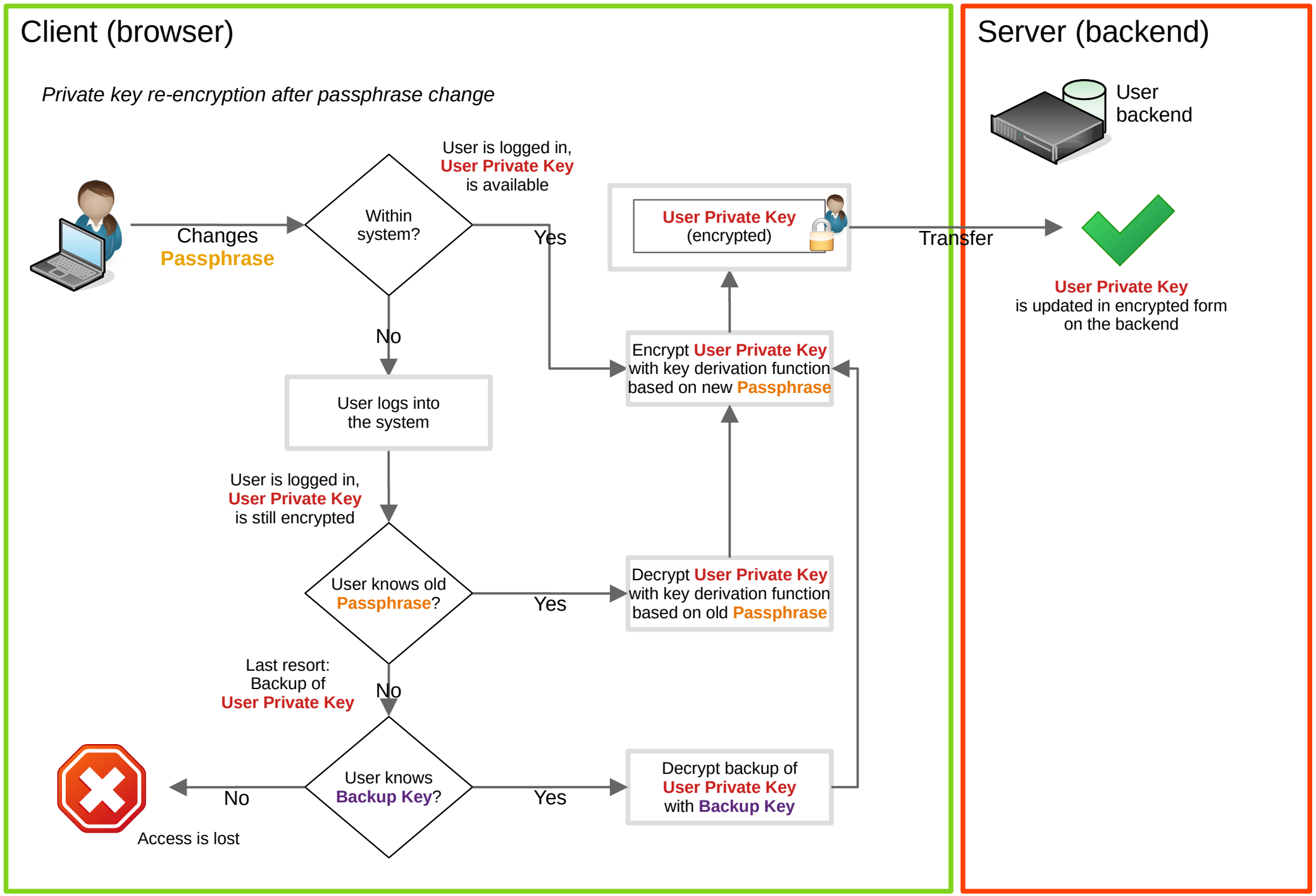
Written by: G. Herbert (2022-03-21)


Client (browser)
Key pair creation, private key encryption & storage
Server (backend)
User backend
User Key Salt
(not encrypted)
Transfer
1. Logs into the system
Username
Passphrase
Use key derivation function
(Passphrase+User Key Salt)
to generate the User Key
2. Enables end-to-end encryption
Generate
User Private Key &
User Public Key
User Private Key is stored in encrypted form. User Public Key is stored without encryption.
3. Stores Backup Key in secure location
Generate
random Backup Key
Encrypt User Private Key:
Once with User key
Once with Backup Key
Transfer
User Private Key
(encrypted by User Key)
User Private Key
(encrypted by Backup Key)
User Public Key
(not encrypted)


Client (browser)
Private key decryption
Logs into the system
Username
Passphrase
Authentication
(not part of this concept)
Success
Request
User Private Key
Server (backend)
User backend
User Private Key
(encrypted)
User Key Salt
(not encrypted)
Transfer
Use key derivation function
(Passphrase+User Key Salt)
to generate the User Key
Use User Key
to decrypt
User Private Key
User Private Key ready
to decrypt sensitive data


Client (browser)
Encryption for shared access
Server (backend)
User backend
Request
User Public Keys
User Public Keys
(not encrypted)
Saves data
in application
Selection of
users/groups
for shared access
Sensitive Data
(not encrypted)
Generate random,
symmetric Data Key
Encrypt Data Key
with each recipient´s User Public Key
resulting in individual User Data Keys
Transfer
Use Data Key to encrypt
Sensitive Data
User Data Keys
(encrypted)
Sensitive Data
(encrypted)
Transfer
Sensitive Data & User Data Keys
stored encrypted in backend
Application
storage


Client (browser)
Server (backend)
Decryption from shared access
Application storage
Open record in application
Data is encrypted?
Yes
No
User Private Key accessible?
Yes
No
Request User Data Key
User Data Key available?
No
Yes
Use User Private Key to decrypt User Data Key to get Data Key
Transfer
User Data Key (encrypted)
Decrypt data with symmetric Data Key
Data can be accessed


Client (browser)
Private key re-encryption after passphrase change
Changes
Passphrase
Within system?
User is logged in,
User Private Key
is available
Yes
No
User Private Key
(encrypted)
Transfer
Server (backend)
User
backend
User Private Key
is updated in encrypted form
on the backend
Encrypt User Private Key
with key derivation function
based on new Passphrase
User logs into
the system
User is logged in,
User Private Key
is still encrypted
User knows old
Passphrase?
Yes
Decrypt User Private Key
with key derivation function
based on old Passphrase
Last resort:
Backup of
User Private Key
No
User knows
Backup Key?
No
Yes
Decrypt backup of
User Private Key
with Backup Key
Access is lost

Questions and answers

Q: Is it a hassle for users to work with end-to-end encryption?

Usually not. The system includes key management functions for creation and storage of encrypted user keys. Users only need to generate personal keys once (a 1-click operation) and store a personal backup-code in a secure location.

Q: Is there a chance, a user looses access to encrypted data permanently?

Yes, because the user´s private key is encrypted with the user´s credentials. Without access to the user credentials the private key is inaccessible. For this case, the system generates a personal backup code, which encrypts a copy of the user´s private key. If a user looses access to both its credentials and backup code, data decryption is not possible.

Another user might still be able to decrypt the same data, if it was also encrypted for this user beforehand.

Q: Can my encrypted data be read, if the database is compromised?

No. Data is encrypted on the client before it is being sent to the database. Decryption keys are stored in the database only in encrypted form.

Q: Is encrypted data protected after the server has been compromised?

No. As a web-based application, the application server sends the code required for encryption to the client. If the server is compromised, this code could be changed, telling the client to send decryption keys or just not encrypting data.

A client trusts the authenticity of a server via its certificate and chain-of-trust. If the system is breached, the server manipulated and trust re-established, there is little any web application can do without requiring additional, client-based software-components (browser-plugins for example).

All web-based applications share this issue, because client-code is distributed by the trusted server. It is therefore important to secure web application servers.

- Key derivation function for user keys
  - Library: **Web Crypto API**
  - Standard: **PBKDF2**
  - Function: **subtle.deriveKey()**
  - KDF salt
    - Library: **math/rand (Golang)**
    - Function: **Intn()**
    - Length: **16-character string, base 62, for each user**
- Symmetric data keys
  - Library: **Web Crypto API**
  - Function: **getRandomValues()**
  - Length: **64-character string, base 82, for each record**
- Symmetric encryption / decryption
  - Library: **Web Crypto API**
  - Function: **subtle.encrypt() / subtle.decrypt()**
  - Standard: **AES-GCM (128 iv, generated by getRandomValues())**
- Asymmetric encryption / decryption
  - Library: **Web Crypto API**
  - Function: **subtle.generateKey(), subtle.decrypt(), subtle.encrypt()**
  - Standard: **RSA-OAEP**
  - Length: **4096 bit**