1

Remote Document Encryption - encrypting data for passport holders

Eric.Verheul@keycontrols.nl
KeyControls, Radboud Universiteit Nijmegen

TNC2019
18 June 2019

(*) Research done for Dutch Vehicle Authority (RDW) based on a question from Gert Maneschijn.

Paper: https://arxiv.org/abs/1704.05647

1

2

Agenda

- RDE introduction and demonstration
- RDE outline
- Example RDE applications
- RDE application in (SURF) FileSender
- Implementation of RDE in (SURF) FileSender
- Conclusion

2

RDE introduction

RDE introduction

- Data Group 1 (file): Name, date of birth, …
- Data Group 2 (file): Facial image
- ….
- Data Group 14 (file): CA (anti-cloning) Public Key **“RDE public key”**
- “EF.SOD” (file) Public Key Certificate

In essence an identity document (passport) is a contactless USB flash drive with some files on it.

RDE introduction: *crux*

- Remote Document Encryption (RDE) is a tweak on identity document protocols.
- It allows any party to encrypt data for the holder of an electronic identity document (passport, identity card, driving license), such that:
- Decryption is only possible with physical possession of the document and takes place *inside* the document, typically by the holder.
- RDE allows for 160 bit security on European identity documents where 128 bit is current good practice, i.e. RDE is $2^{32} \approx 4$ billion times stronger.

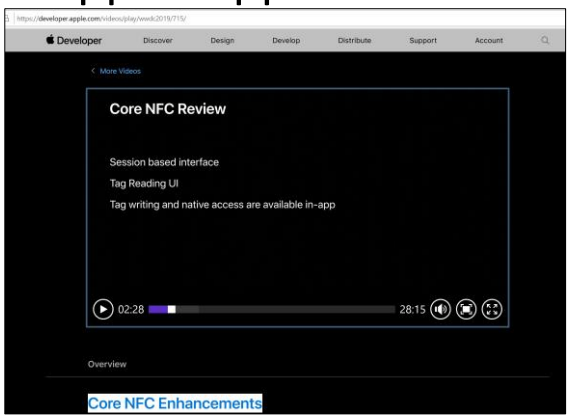
Illustrative application

A hospital wants to send its patients (RDE) encrypted e-mails. The hospital develops an RDE mobile APP allowing:

a. the hospital reading a RDE "public key certificate" from the identity document for e-mail encryption,
b. the patient to perform RDE decryption using a PC/mobile device together with her identity document.

5

Apple support for NFC



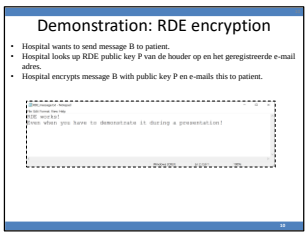
- Until recently, reading electronic identity documents was limited to Android devices supporting NFC or devices with a separate contactless card reader.
- Starting with IOS13 (September 2019) Apple will also "open" NFC allowing mobile APPs reading electronic identity documents, also allowing RDE for iPhones (≥iPhone7).

6

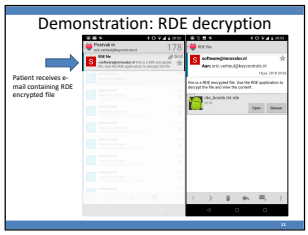
Demonstration ‘RDE secure mail’

The ‘RDE secure email’ use-case consists of three phases.

1. User registration (one-time only)



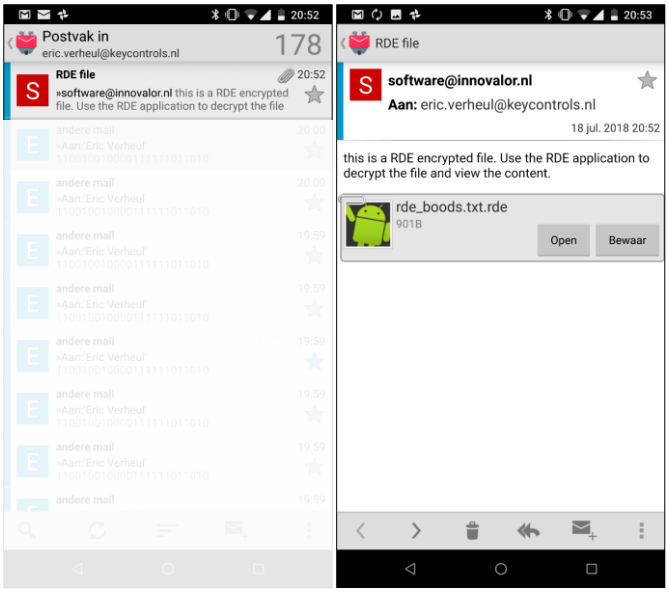
2. Encryption of message by hospital



3. Decryption by patient

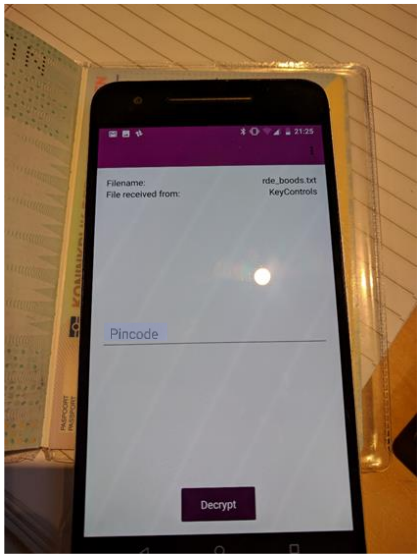
Demonstration: RDE decryption

Patient receives e-mail containing RDE encrypted file



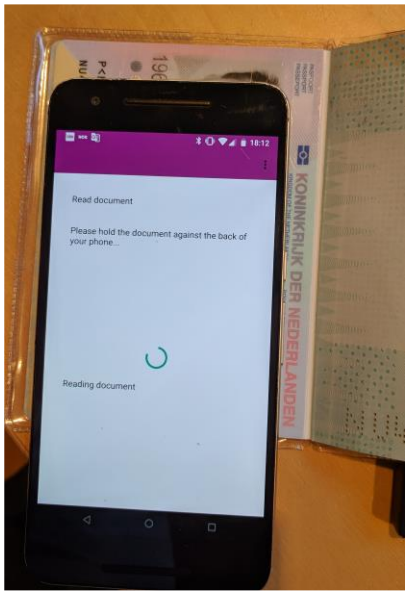
Demonstration: RDE decryption

Automatic Start-up of RDE APP + PIN entry



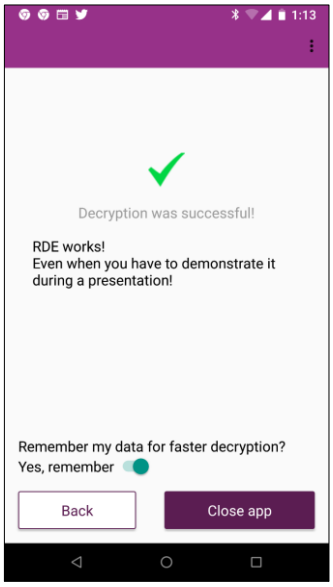
Demonstration: RDE decryption

RDE decryption



Demonstration: RDE decryption

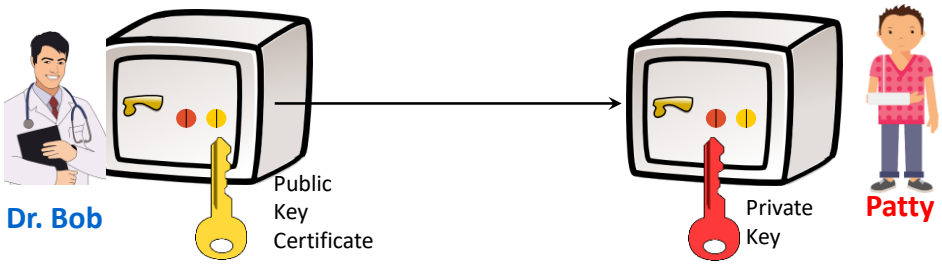
RDE plaintext



11

Outline RDE: public key cryptography

- One key to *close* the safe (*public key*) and another key to open the safe (*private key*).
- Patient **Patty** publishes her public key allowing **dr. Bob** to get hold of it.
- Patient **Patty** keeps her private key secret.
- Dr. **Bob** encrypts data for **Patty** with her public key; only **Patty** can decrypt this with her private key.
- To be sure that the public key really belongs to **Patty**, this key is associated with **Patty**'s identity and signed by a "TTP": a **Public Key certificate**



12

13

Outline RDE: public key cryptography

Private Key

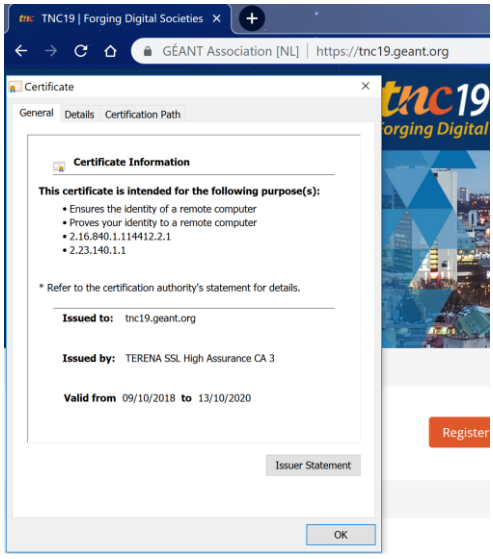
Public key (and certificate)

13

14

Outline RDE: public key cryptography

Example of (TLS) certificate

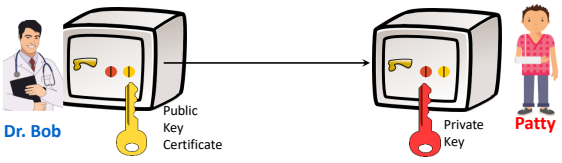


14

Outline RDE: public key cryptography

Outline RDE: public key cryptography

- One key to *close* the safe (*public key*) and another key to open the safe (*private key*).
- Patient **Patty** publishes her public key allowing **dr. Bob** to get hold of it.
- Patient **Patty** keeps her private key secret.
- Dr. **Bob** encrypts data for **Patty** with her public key; only **Patty** can decrypt this with her private key.
- To be sure that the public key really belongs to **Patty**, this key is associated with **Patty**'s identity and signed by a "TTP": a **Public Key certificate**



Hospital Patient

- RDE Public Key certificate is bound to all printed information:
  - First and last name
  - Date of birth
  - Place of birth
  - Facial image (in colour!)

RDE registration

Public Key + "certificate" + email address

PubK

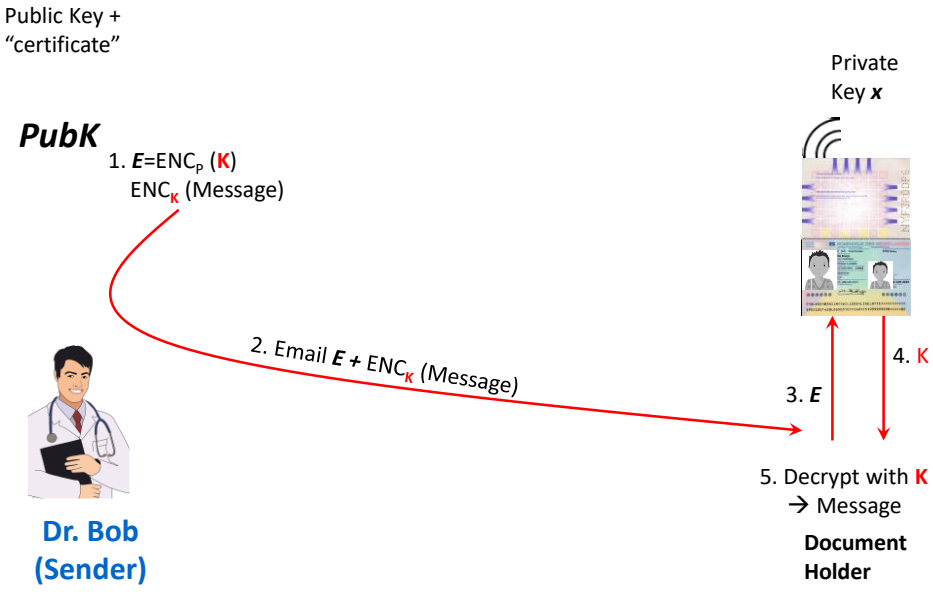
"Registration"

Private Key ***x***

Dr. Bob (sender)

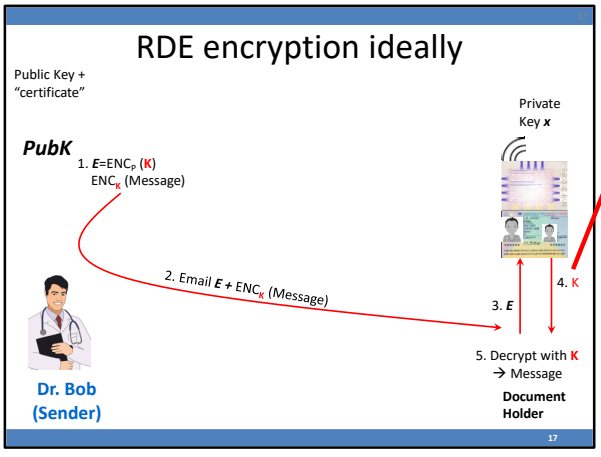
Patty (receiver)

RDE encryption ideally



17

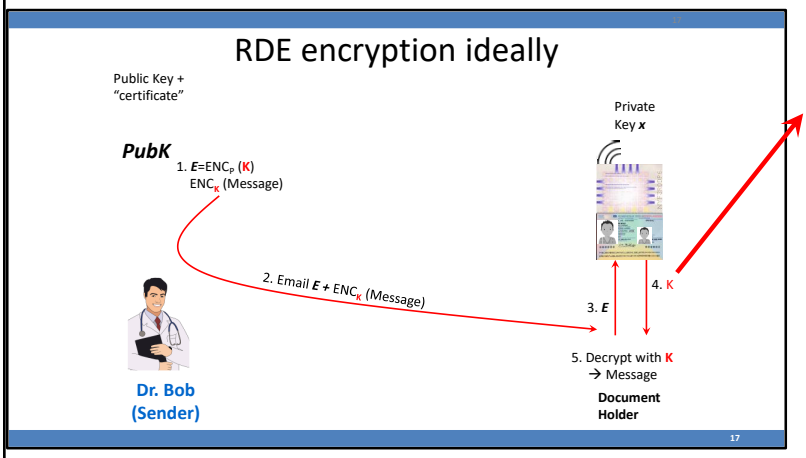
RDE encryption



Unfortunately: passport delivers *different* K than the original
But: sending party can predict what the holder passport will deliver, namely ***K'***.
RDE Crux: let sending party and holder use *that* key, i.e. ***K'***, as encryption key.

18

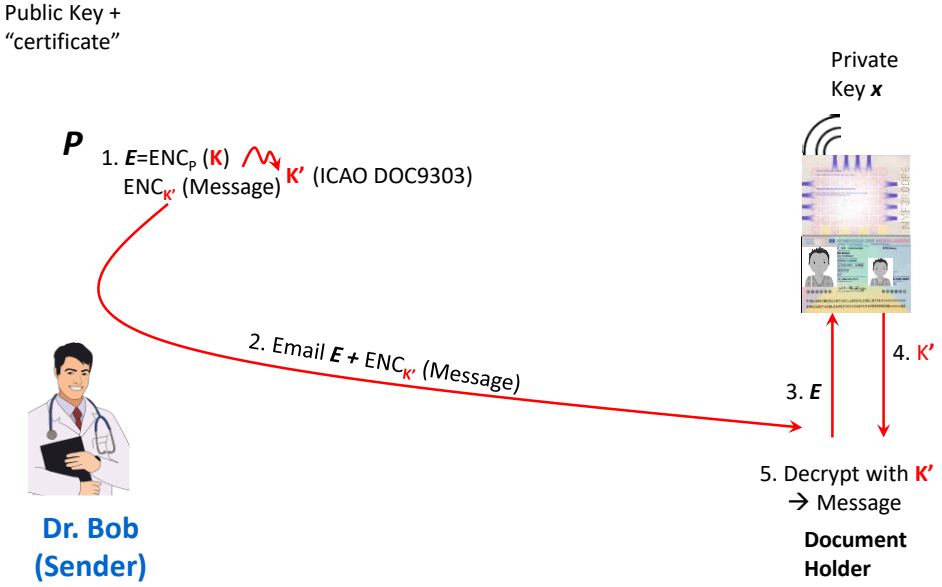
RDE encryption



Unfortunately: passport delivers *different* K than the original
But: sending party can predict what the holder passport will deliver, namely ***K'***.
RDE Crux: let sending party and holder use *that* key, i.e. ***K'***, as encryption key.

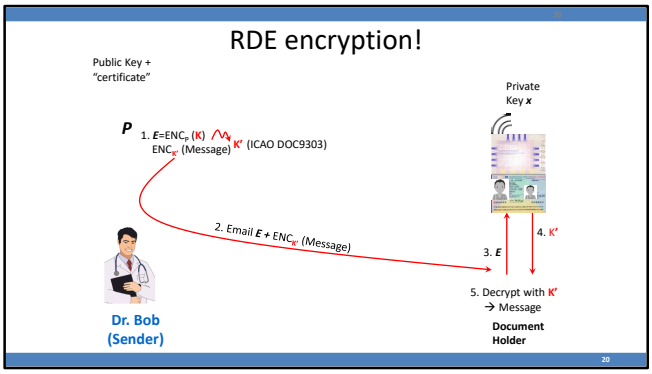
19

RDE encryption!



20

RDE PIN (two factor encryption)



- By additionally encrypting ENC_P (**K**) with a *Personal Encryption Number* (PEN) one gets two factor encryption (possession and knowledge)
- PEN can only be *brute forced* with the passport!
- By making PEN 'long enough' the brute force risk is controllable.
- PEN is an interesting intermediary form of PIN and password.

21

Example RDE applications

- **Secure email**
  RDE encrypt messages and send them through email. Already tested in pilot in 2018 with a developed APP.
- **Secure password managers, e.g. Keepass**
  Weak spot is the encryption of the password database. In practice this encryption is based on a guessable password making cloud archiving a bad idea. With RDE the password database can be adequately encrypted, allowing secure cloud archiving.
- **Secure personal health environments**
  Within Dutch healthcare it is facilitated that patients can have their medical records sent from their healthcare provider to a Personal Health Environment (PHE). With RDE the healthcare provider can encrypt the data ensuring that only the patient has access to them (and not the PHE).
- **End-to-end secure SURF FileSender (next slides)**
  See surffilesender.nl. SURF intents to implement RDE in its Filesender instance in a 2019 pilot. This pilot will be done in cooperation with Dutch government (RDW and RvIG).

22

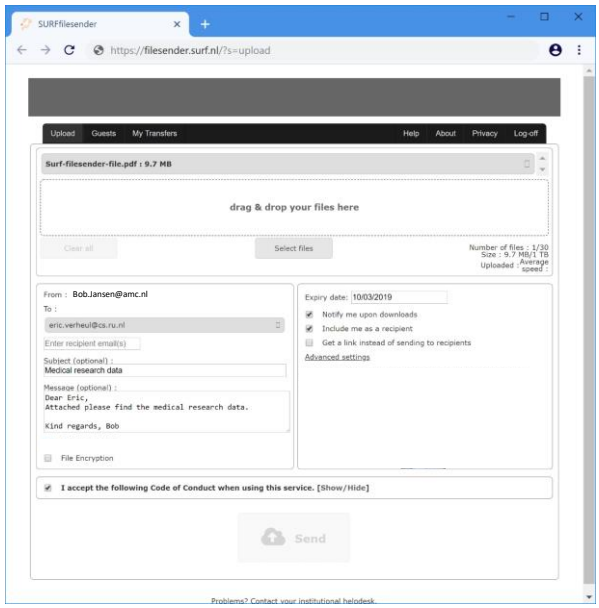
RDE application: end-to-end secure SURF FileSender

- FileSender:
  - is an open source, web based application for exchanging large files, see https://filesender.org/
  - currently supports the use of a password (decryption key) to encrypt the exchanged files.
  - encryption en decryption takes place in the browser (W3C Web Cryptography API JavaScript) providing end-to-end security.
- However, the (classical) problem is the password exchange among users in a secure and user-friendly way.
- To address this problem, SURF and Dutch government (RDW, RvIG) intent to supplement RDE to the FileSender open source project.
- In this way FileSender can conveniently support the secure exchange of personal data in education, research, healthcare. It also can facilitate the GDPR 'right of access' at institutions.

23

RDE application: end-to-end secure SURF FileSender

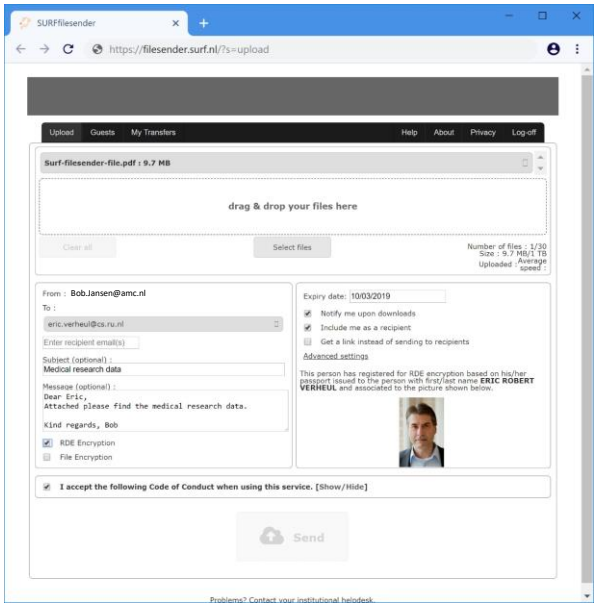
Current Interface Sender



24

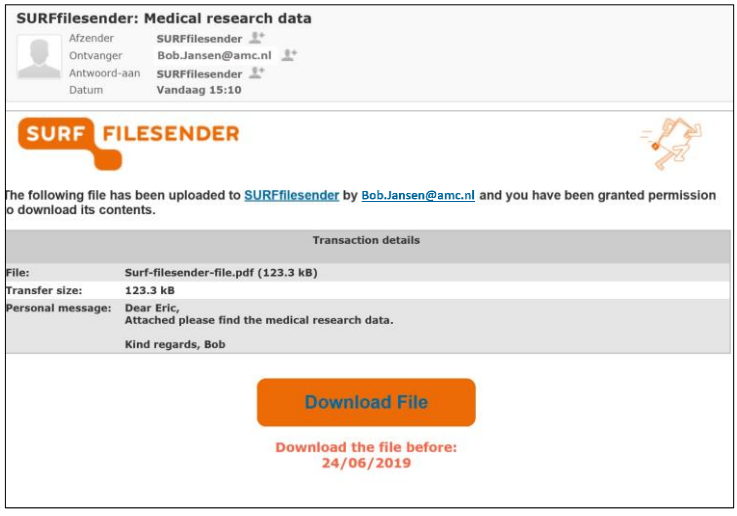
RDE application: end-to-end secure SURF FileSender

RDE Interface Sender



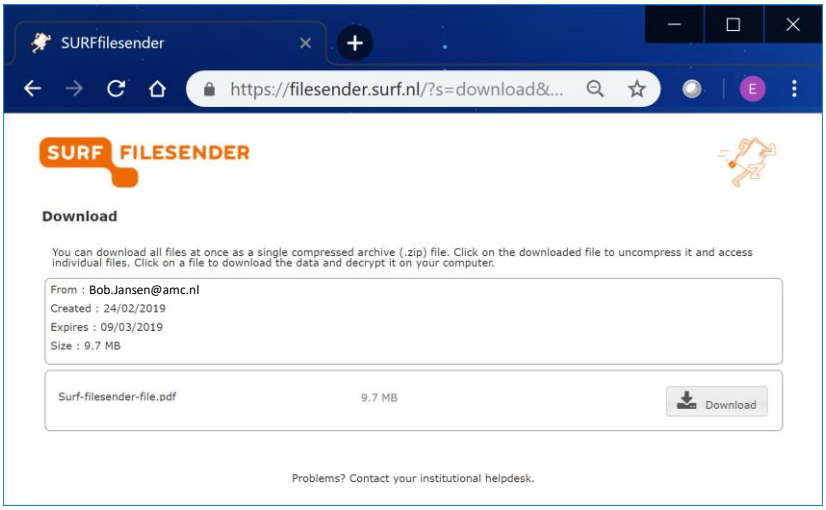
RDE application: end-to-end secure SURF FileSender

Notification Email



27

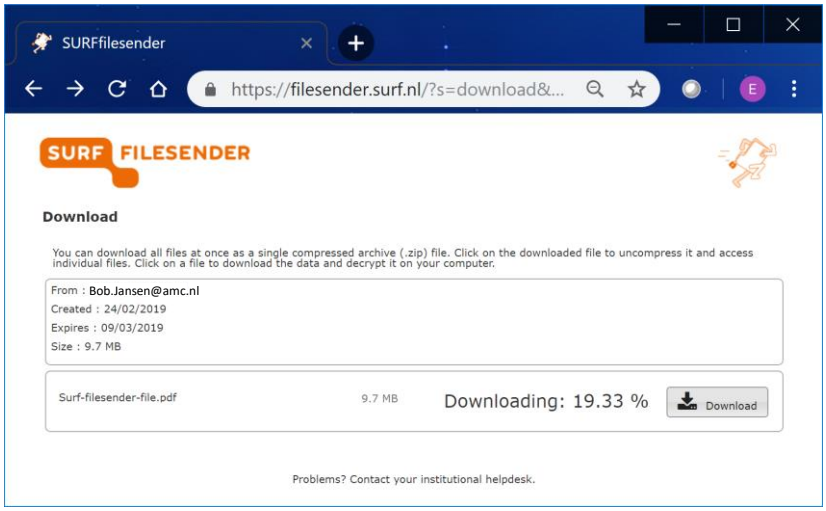
RDE application: end-to-end secure SURF FileSender



RDE Interface Receiver

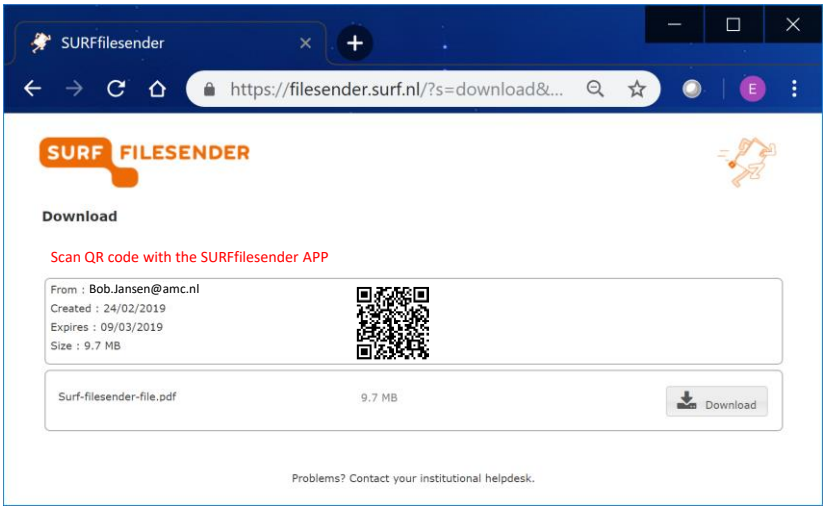
28

RDE application: end-to-end secure SURF FileSender



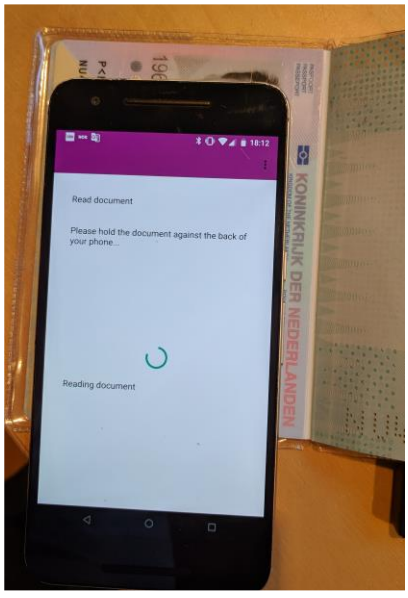
RDE Interface Receiver

RDE application: end-to-end secure SURF FileSender

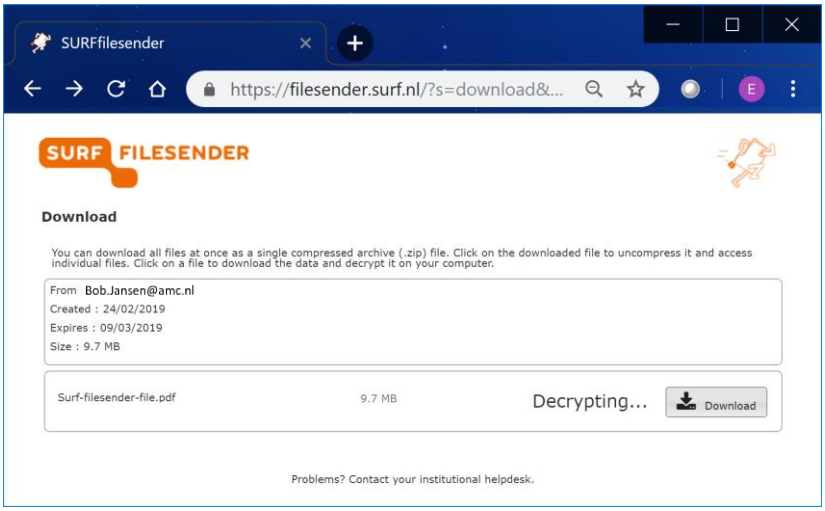


RDE Interface Receiver

RDE application: end-to-end secure SURF FileSender

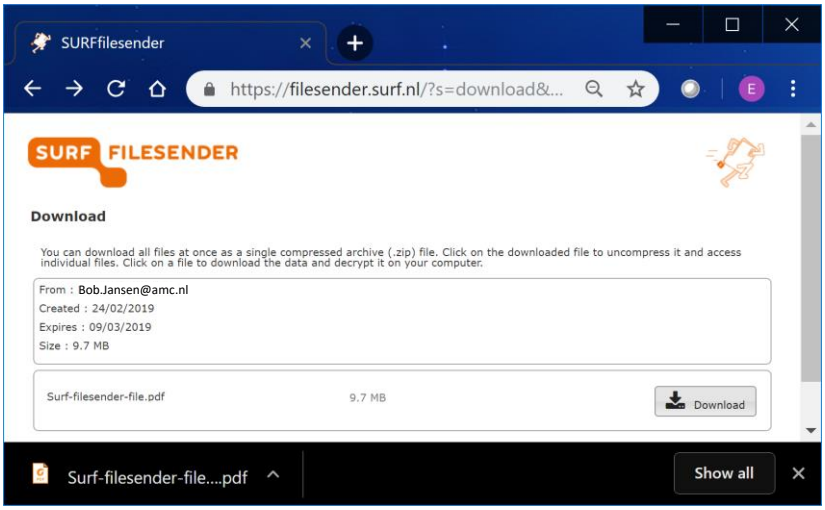


RDE application: end-to-end secure SURF FileSender



RDE Interface Receiver

RDE application: end-to-end secure SURF FileSender



RDE Interface Receiver

Implementation of RDE in SURF FileSender

- RDE cryptographic basis is symmetrical encryption based on AES in GCM mode. The AES-GCM operations take place within the internet browsers of the users.
- This is completely supported by the W3C Web Cryptography API (https://www.w3.org/TR/WebCryptoAPI/): AES-GCM can be called *natively* through JavaScript without requiring extra JavaScript *libraries*.
- By using a suitable AES-GCM configuration, encryption and decryption in (large) chunks is also easily possible; this is relevant for very large files.
- For the essential part of RDE (ECDH) support for so-called Brainpool elliptic curves is required. Alas W3C only supports NIST curves as P-256.
- It is therefore required that Brainpool based ECDH is separately implemented, e.g. by limited use of the Stanford Javascript Crypto Library (http://bitwiseshiftleft.github.io/sjcl/).
- RDE chunk based setup also usable for current password based Filesender, allowing sending very large file (>2GB).

33

Conclusion

34