Sign Sign (sign2x)

Multi-sign documents locally (privacy) relying on mathematics for security and correctness

@opensourcedays 2017-03-18

Overview

- About me
- Recent Open Source projects
- Background
- The Math behind
- Sign Sign (sign2x) + Demo
- The missing parts and how to contribute
- Summary
- Q & A

About me (very shortly)

- Ramón Soto Mathiesen
- MSc. Computer Science ***DIKU/Pisa*** and minors in Mathematics ***HCØ***
- ***CompSci*** @ SPISE MISU ApS
  - ***"If I have seen further it is by standing on the shoulders of giants"***
    -- Isaac Newton (Yeah Science, Bitch … Mostly mathematics)
  - ***Elm*** (***JS*** due to ports) with a bit of ***Haskell*** and a bit of ***F#*** (fast prototyping)
- Elm / Haskell / TypeScript / F# / OCaml / Lisp / C++ / C# / JavaScript
- Blog: http://blog.stermon.com/

Recent Open Source projects

- Previous workplace (CTO of CRM @ Delegate A/S):
  - MS CRM Tools: http://delegateas.github.io/
- Current workplace (SPISE MISU ApS):
  - Pure Elm libraries (UTF8, SHA, Merkle Tree)
  - Syntactic Versioning (SynVer @ F# Community Projects)
  - Puritas, isolate side-effects at compile-time in F# (talk April, London)
  - Sign Sign (sign2x) (coming soon https://sign2x.org)
  - UniverΣ (UniverSum) (coming at some point https://universigma.org)

Background

- I recently joined the board of the housing cooperative (“andelsbolig”) where I live.
- Our accountant was complaining about that we are not always home when he needs our signatures, which results in several (paid) trips in non-business hours (overtime in Denmark is not worth it)
- He suggested we use Penneo ApS …

Background

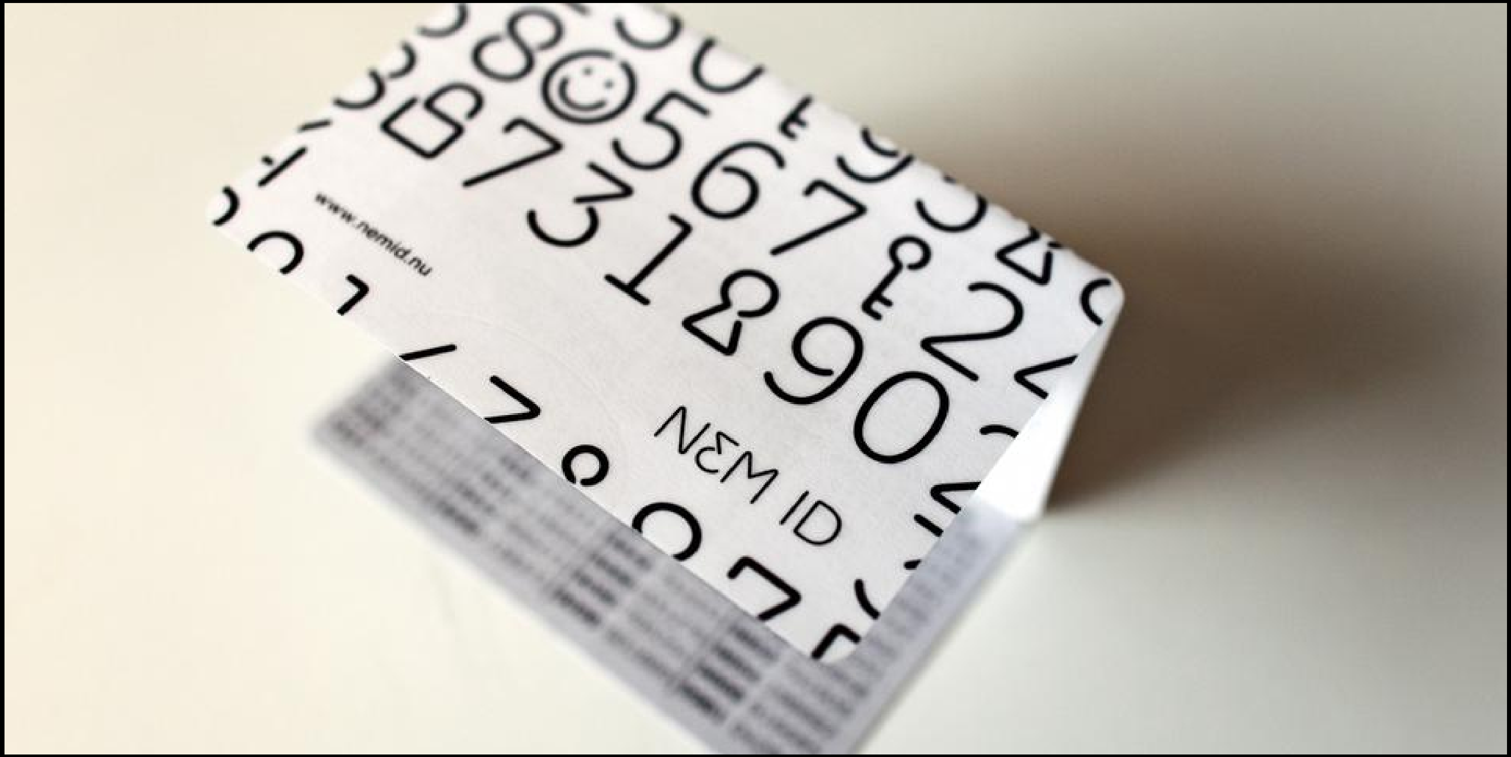
- Because of the pricing, I looked into a few other alternatives: Visma Addo, DocuSign, Cryptomathic, … They all had in common the same trend as Penneo:
  - Pricey
  - Closed and proprietary approach
    - Wannabe monopolistic?
- So we ended not choosing any of them …

Background

- … until I recently had to sign a document for my bank by using e-Boks + NemID.
- Let’s go through the process but before I start, lets explain what e-Boks and NemID are (taken from WP):
  - **E-Boks**: The product is an online mailbox where individuals and companies with a Danish CPR or CVR number may choose to receive and then store the mail from Danish companies, municipalities and public institutions, which is connected to the product. Owned by Nets DanID A/S and Post Danmark A/S.
  - **NemID**: (literally: EasyID) is a common log-in solution for Danish Internet banks, government websites and some other private companies. It also requires a Danish CPR or CVR number. Owned by Nets DanID A/S.

Note: You see the trend right?

Background



(Digitalisering + papirløse samfund? Say what?)

Background

- So, the process …
  - First I receive an e-mail from my bank with regard of having to log-in to e-Boks to sign a document
    - I use one number code from Paper Card
  - I can read electronically (**PDF**) the document I need to sign.
  - After signing the document, I get white page
    - I have used another number code from the Paper Card
  - I read it can take up to 20 minutes to go through …

Background

- … process continues …
  - I log in half an hour after later to see status, which is a useless as the blank page after signing
    - I have used another number code from the Paper Card
  - It seems that I don’t have signed the document
  - I contact the bank, which refers me to Nets Support
    - I refuse, It took them around 2 month to send me the Paper Cards in the first place after infinite many e-mails back and forth

Background

- … process continues II …
  - So I ended up printing the electronic **PDF**, signing it with a good old pen, scanning it and then sending it to the bank … Good old Dell 1600n saved the day (love you !!!)

Background

- Nets DanID A/S are ***not*** known for producing high quality products, just an example.
- They are more known for using monopolistic approach to deliver software “that’s mandatory to be used” and to “enrich their executives” (IPO).
  Here are a few in their top-8:
  - First, Bo Nilsson (CEO): ***679*** mDKK
  - Next, Klaus Pedersen (CFO): **175** mDKK
  - ...
  - And last but not least, Frode Åsheim (Group EVP, BU Corporate Services): **11** mDKK
- ***Long-Story-Short***, we need somekind of standard that allow os to multi-sign locally (privacy matters) and that could be validated by anybody, also locally, who had access to the documents. Therefore it needs to be based on ...

The Math behind

- … some sound and robust mathematics !!!
- I have based the solution on the following concepts:
  - **SHA-256** (checksum hashing)
  - **OpenPGP** (standard for e-mail and text encryption)
    - **RSA** (public-key crypto system)
    - **AES** (specification for the encryption of electronic data)
  - Threshold Scheme (secret sharing)

    Note: A discussion with Ximin Luo yesterday at speakers dinner made me realize that this component might be superfluous (argue at Summary)

The Math behind

- I’m not going to explain **SHA-256**, **OpenPGP**, **RSA**, **AES** due to time limit. Anyhow, those concepts should be know by most of you attending an Open Source Conference anyway right?
- So I will only focus only on the Threshold Scheme

The Math behind

- Threshold Scheme for secret sharing
  - I was already using this piece of mathematics in one of my other projects (**UniverΣ**) so it was obvious to reuse it here as it solves the problem which is: “How do we ensure that ***some***, ***any*** or ***all*** of the signers comply with the requirements of the document”?
    - A few examples:
      - The father or the mother needs to sign the document (***any***)
      - The board needs to sign this document (***all***)
      - At least the CEO needs to sign or any two of the owners (***some***)

The Math behind

- Threshold Scheme for secret sharing (src: WP)
  - The used approach is described in Adi Shamir (yes he is the the S in **RSA**) ***Turing*** awarded paper: “How to Share a Secret” where a secret ***S***, is divided into ***n*** pieces of data: $S_1, S_2, \ldots S_n$ so that:
    - Knowledge of any ***k*** or more S_i pieces makes ***S*** easily computable
    - Knowledge of any ***k-1*** or fewer S_i pieces makes ***S*** completely undetermined (in the sense that all its possible values are equally likely)

The Math behind

- Threshold Scheme for secret sharing (src: WP)
  - The essential idea of the threshold scheme is that ***2 points*** are sufficient to define a ***line***, ***3 points*** are sufficient to define a parabola, ***4 points*** to define a ***cubic curve*** and so forth. It takes ***k*** points to define a polynomial of degree ***k – 1***

The Math behind

- Threshold Scheme for secret sharing (src: WP)
  - Suppose we want to use a (k,n) threshold scheme to share our secret S without loss of generality assumed to be an element in a finite field F of size P where $0 < k \le n < P$; $S < P$ and P is a prime number.

    Note: Example of finite fields is when integers mod p when p is a prime number.

The Math behind

- Threshold Scheme for secret sharing (src: WP)
  - Choose at random $k-1$ positive integers $a_1 , \dots , a_{k-1}$ with $a_i < P$ and let $a_0 = S$. Build the polynomial $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{k-1}x^{k-1}$. Let us construct any n points out of it, for instance set $i = 1 , \dots , n$ to retrieve $(i,f(i))$. Every participant is given a point (an integer input to the polynomial, and the corresponding integer output). Given any subset of k of these pairs, we can find the coefficients of the polynomial using ***interpolation***. The secret is the constant term a_0.

The Math behind

- Threshold Scheme for secret sharing (src: WP)
  - Properties of Shamir's ***(k,n)*** threshold scheme are (3/5):
    - **Secure**: Information theoretic security
      - Can't be broken even with unlimited computing power
    - **Minimal**: The size of each piece does not exceed the size of the original data.
    - **Extensible**: When ***k*** is kept fixed, $\boldsymbol{D_i}$ pieces can be dynamically added or deleted without affecting the other pieces.

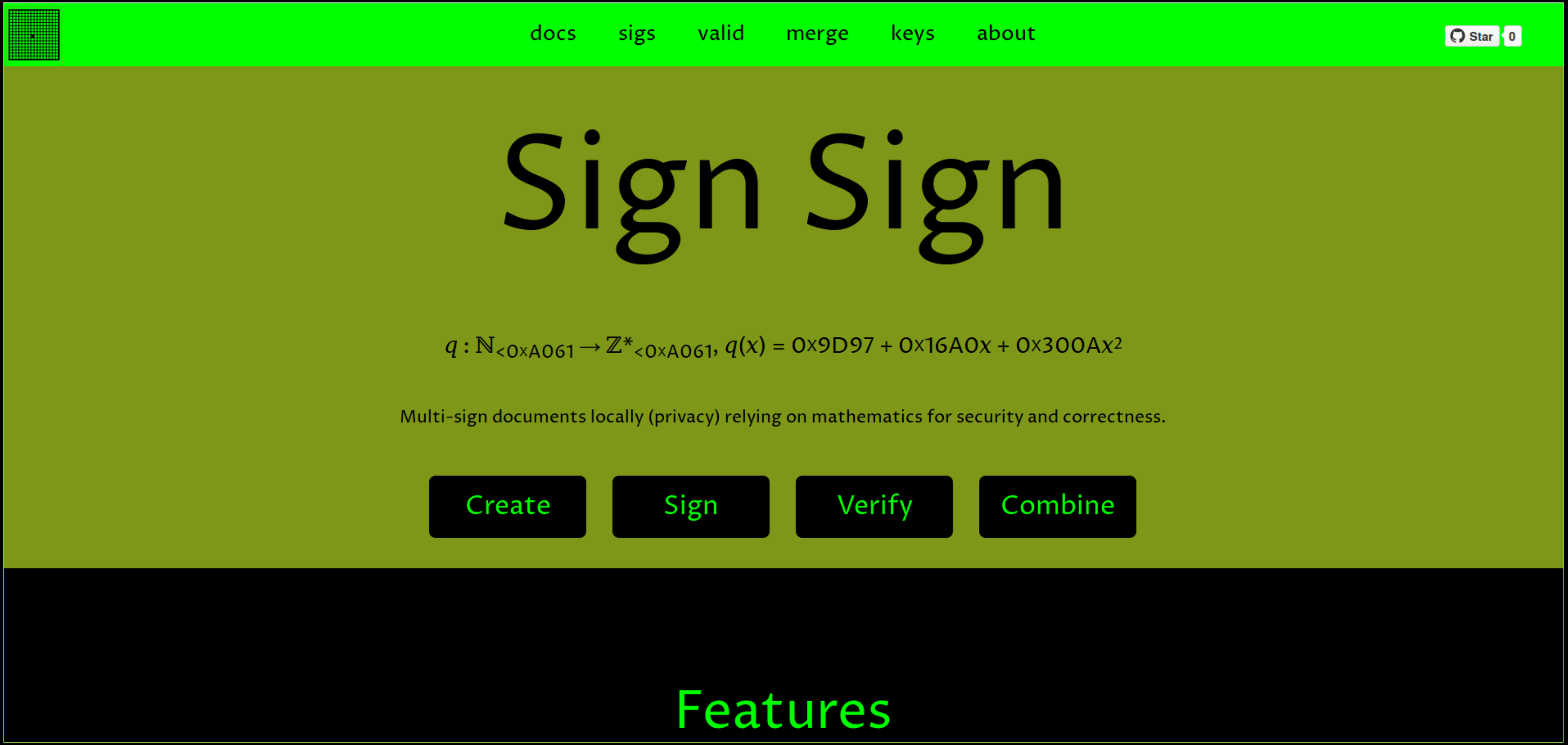
The Math behind

- Threshold Scheme for secret sharing (src: WP)
  - Properties of Shamir's ***(k,n)*** threshold scheme are (5/5):
    - **Dynamic**: Security can be easily enhanced without changing the secret, but by changing the polynomial occasionally (keeping the same free term) and constructing new shares to the participants.
    - **Flexible**: In organizations where hierarchy is important, we can supply each participant different number of pieces according to their importance inside the organization. For instance, the president can unlock the safe alone, whereas 3 secretaries are required together to unlock it.

The Math behind

- Threshold Scheme for secret sharing
  - A few implementation details:
    - ***Horner*** Method to calculate polynomials efficiently
    - ***Big Integers*** to support **256 bits** random primes and numbers
    - ***Lagrange*** polynomial ***interpolation*** to calculate the free term
      - With precomputed values

Sign Sign (sign2x) + Demo



docs sigs valid merge keys about

Sign Sign

$q : \mathbb{N}_{<0\text{xA061}} \rightarrow \mathbb{Z}^*_{<0\text{xA061}}, q(x) = 0\text{x9D97} + 0\text{x16A0}x + 0\text{x300A}x^2$

Multi-sign documents locally (privacy) relying on mathematics for security and correctness.

Features

2017-03-18

Sign Sign (sign2x) + Demo

- Basics:
  - Accepts
    - Documents of type **PDF**
    - Public keys of type **ASCII-armor**
  - Producing a single **SIGN2X** file:
    - Underlaying container is just an uncompressed **ZIP** file
  - Each added public key will get an assigned fraction of the total of polynomial parts

Sign Sign (sign2x) + Demo

- Basics:
  - Information of documents, keys, signers and parts are stored in the **ZIP** file as a **JSON** file
  - A Certificate Of Authenticity is added by signing the above **JSON** file in combination with the creators public key. Mandatory before sending **SIGN2X** to be signed.

Note: Signing and adding **COA** is not limited to the **SPA**, can be done with basic *nix terminal commands in combination with gpg (**GnuPG**) as for people who don’t want to add their **RSA** private keys to a web app

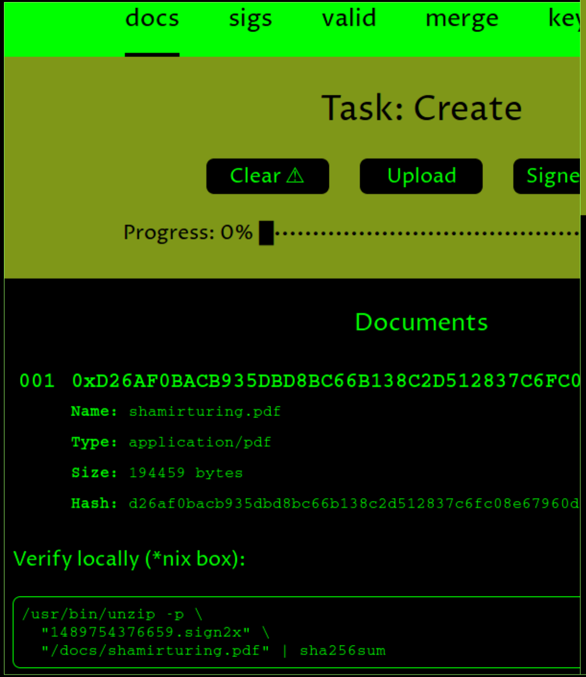
Sign Sign (sign2x) + Demo

- Demo script for creation:
  - Adding files:
    - Click the **Create** button on the main page
    - Click the **Upload** button and select some PDF files
  - Adding signers:
    - Click the **Signees →** button (notice you are in the sigs section)
    - Click the **Add** button (notice how you are in the keys section)
    - Click the **Upload** button and select ASCII-armor files
    - Click the **← Add All** button (notice how you are back at sigs)

Sign Sign (sign2x) + Demo

- Demo script for creation (continuation):
  - Specifying fractions:
    - Click the **+** and/or **-** on each of the signers.
    - Click on the **Calculate** button so the Threshold Scheme is calculated
    - Click on **Validate →** (notice how you are in the valid section)
  - Adding Certificate-Of-Authenticity (optional):
    - Click the **Add Coa** button
  - Finally, download the file, Click the **Download** button

Sign Sign (sign2x) + Demo

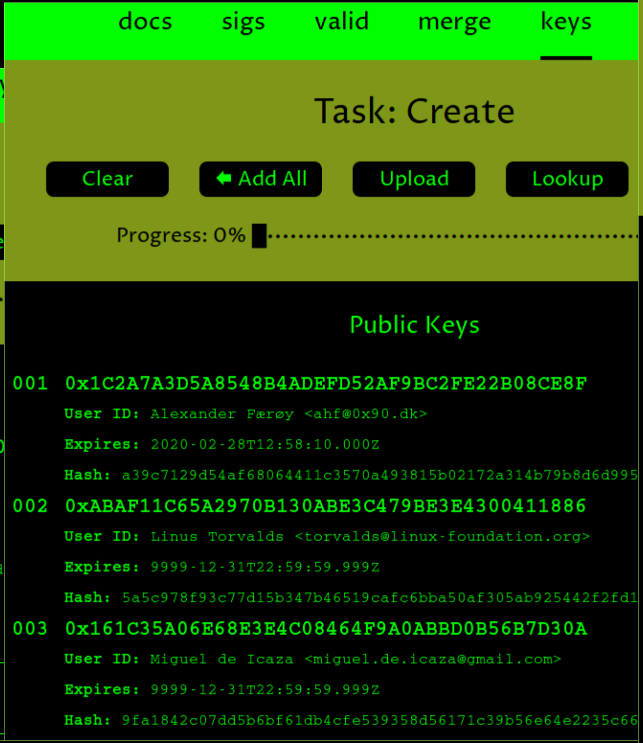


Documents

```
001  0xD26AF0BACB935DBD8BC66B138C2D512837C6FC0
     Name: shamirturing.pdf
     Type: application/pdf
     Size: 194459 bytes
     Hash: d26af0bacb935dbd8bc66b138c2d512837c6fc08e67960d
```

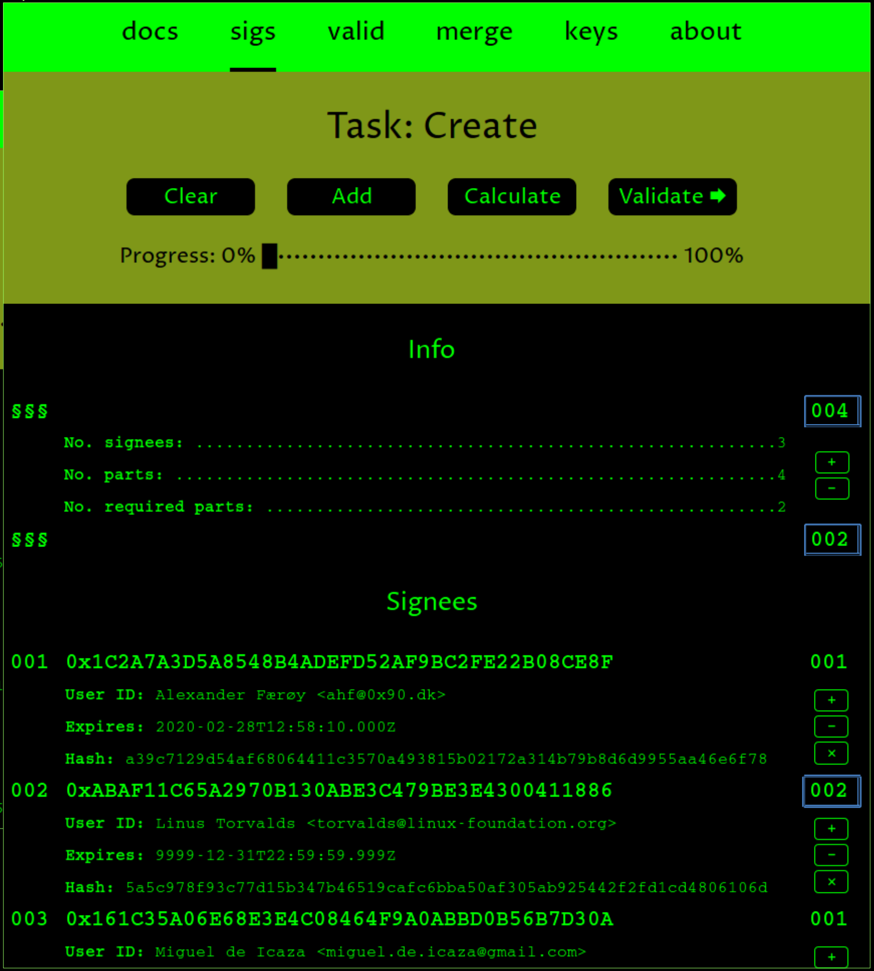
Verify locally (*nix box):

```
/usr/bin/unzip -p \
  "1489754376659.sign2x" \
  "/docs/shamirturing.pdf" | sha256sum
```



Public Keys

```
001  0x1C2A7A3D5A8548B4ADEFD52AF9BC2FE22B08CE8F
     User ID: Alexander Færøy <ahf@0x90.dk>
     Expires: 2020-02-28T12:58:10.000Z
     Hash: a39c7129d54af68064411c3570a493815b02172a314b79b8d6d995
002  0xABAF11C65A2970B130ABE3C479BE3E4300411886
     User ID: Linus Torvalds <torvalds@linux-foundation.org>
     Expires: 9999-12-31T22:59:59.999Z
     Hash: 5a5c978f93c77d15b347b46519cafc6bba50af305ab925442f2fd1
003  0x161C35A06E68E3E4C08464F9A0ABBD0B56B7D30A
     User ID: Miguel de Icaza <miguel.de.icaza@gmail.com>
     Expires: 9999-12-31T22:59:59.999Z
     Hash: 9fa1842c07dd5b6bf61db4cfe539358d56171c39b56e64e2235c66
```



Info

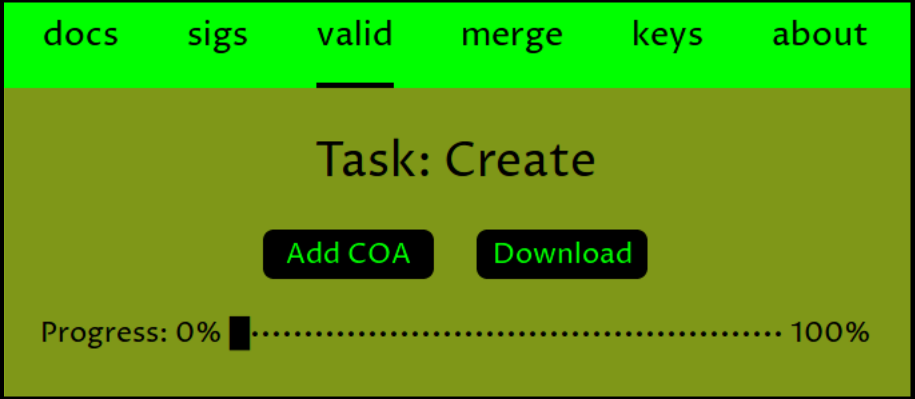
```
§§§                                                               004
    No. signees: ...............................................3
    No. parts: .................................................4
    No. required parts: ........................................2
§§§                                                               002
```

Signees

```
001  0x1C2A7A3D5A8548B4ADEFD52AF9BC2FE22B08CE8F                   001
     User ID: Alexander Færøy <ahf@0x90.dk>
     Expires: 2020-02-28T12:58:10.000Z
     Hash: a39c7129d54af68064411c3570a493815b02172a314b79b8d6d9955aa46e6f78
002  0xABAF11C65A2970B130ABE3C479BE3E4300411886                   002
     User ID: Linus Torvalds <torvalds@linux-foundation.org>
     Expires: 9999-12-31T22:59:59.999Z
     Hash: 5a5c978f93c77d15b347b46519cafc6bba50af305ab925442f2fd1cd4806106d
003  0x161C35A06E68E3E4C08464F9A0ABBD0B56B7D30A                   001
     User ID: Miguel de Icaza <miguel.de.icaza@gmail.com>
```

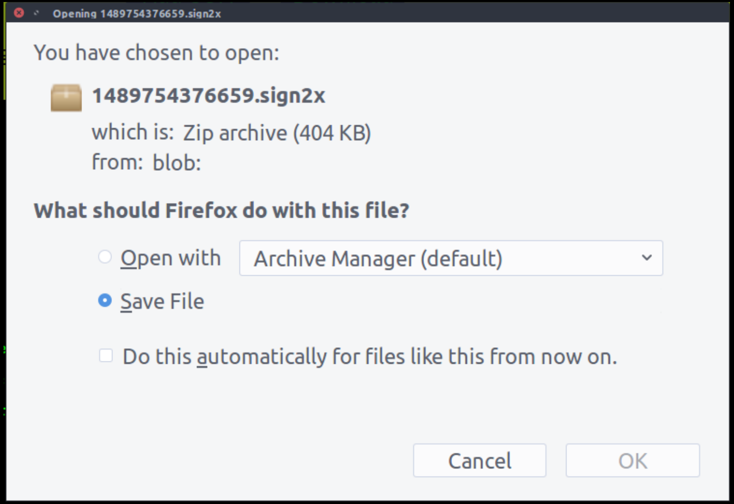
2017-03-18

Sign Sign (sign2x) + Demo



Task: Create

Progress: 0% ··· 100%



Opening 1489754376659.sign2x

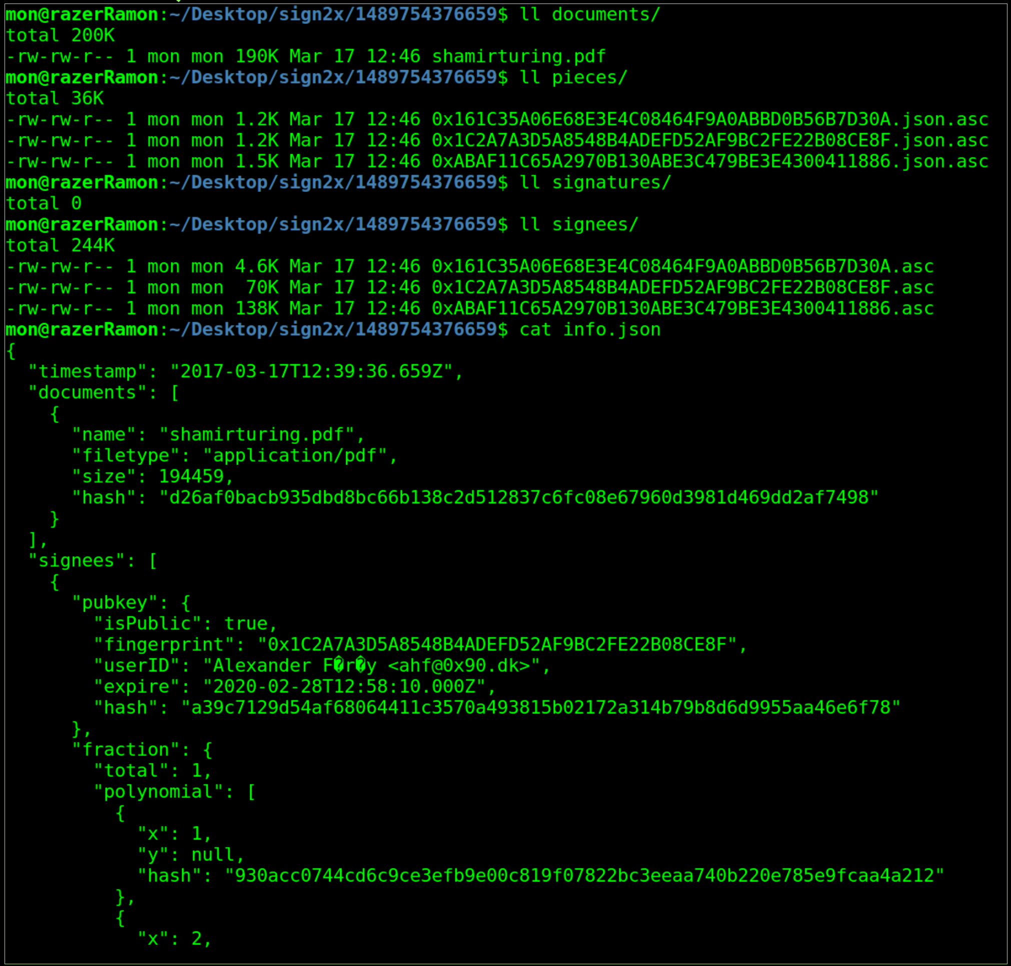
You have chosen to open:

1489754376659.sign2x

which is: Zip archive (404 KB)
from: blob:

What should Firefox do with this file?

Open with [Archive Manager (default)]
[✓ Save File]
Do this automatically for files like this from now on.



```
mon@razerRamon:~/Desktop/sign2x/1489754376659$ ll documents/
total 200K
-rw-rw-r-- 1 mon mon 190K Mar 17 12:46 shamirturing.pdf
mon@razerRamon:~/Desktop/sign2x/1489754376659$ ll pieces/
total 36K
-rw-rw-r-- 1 mon mon 1.2K Mar 17 12:46 0x161C35A06E68E3E4C08464F9A0ABBD0B56B7D30A.json.asc
-rw-rw-r-- 1 mon mon 1.2K Mar 17 12:46 0x1C2A7A3D5A8548B4ADEFD52AF9BC2FE22B08CE8F.json.asc
-rw-rw-r-- 1 mon mon 1.5K Mar 17 12:46 0xABAF11C65A2970B130ABE3C479BE3E4300411886.json.asc
mon@razerRamon:~/Desktop/sign2x/1489754376659$ ll signatures/
total 0
mon@razerRamon:~/Desktop/sign2x/1489754376659$ ll signees/
total 244K
-rw-rw-r-- 1 mon mon 4.6K Mar 17 12:46 0x161C35A06E68E3E4C08464F9A0ABBD0B56B7D30A.asc
-rw-rw-r-- 1 mon mon  70K Mar 17 12:46 0x1C2A7A3D5A8548B4ADEFD52AF9BC2FE22B08CE8F.asc
-rw-rw-r-- 1 mon mon 138K Mar 17 12:46 0xABAF11C65A2970B130ABE3C479BE3E4300411886.asc
mon@razerRamon:~/Desktop/sign2x/1489754376659$ cat info.json
{
  "timestamp": "2017-03-17T12:39:36.659Z",
  "documents": [
    {
      "name": "shamirturing.pdf",
      "filetype": "application/pdf",
      "size": 194459,
      "hash": "d26af0bacb935dbd8bc66b138c2d512837c6fc08e67960d3981d469dd2af7498"
    }
  ],
  "signees": [
    {
      "pubkey": {
        "isPublic": true,
        "fingerprint": "0x1C2A7A3D5A8548B4ADEFD52AF9BC2FE22B08CE8F",
        "userID": "Alexander F�r�y <ahf@0x90.dk>",
        "expire": "2020-02-28T12:58:10.000Z",
        "hash": "a39c7129d54af68064411c3570a493815b02172a314b79b8d6d9955aa46e6f78"
      },
      "fraction": {
        "total": 1,
        "polynomial": [
          {
            "x": 1,
            "y": null,
            "hash": "930acc0744cd6c9ce3efb9e00c819f07822bc3eeaa740b220e785e9fcaa4a212"
          },
          {
            "x": 2,
```

Sign Sign (sign2x) + Demo

- A few design/implementation details:
  - Privacy-by-design, get used to it as General Data Protection Regulation (**GDPR**) arrives next year:
    - Doom-day: **2018-05-28**
  - Easiest way to do this is by isolate your side-effect. Languages supporting this at the moment are Haskell, COQ, Idris, PureScript and **Elm** (and soon F#, due to Puritas)
  - There is no remote state (web server + database) and local state is reset when F5 is pressed. Only keys are stored in keyring due to UX, but as stated before, keys are not necessary to create a **SIGN2X** file (it's to signing and adding **COAs**)
  - For example, password are never persisted in the model, only sent around as side-effects (onchange binded password form field)

    Note: Let's stop local web server and show that SPA still works !!!

Sign Sign (sign2x) + Demo

- A few design/implementation details:
  - Due to lack of Big Integers, crypto module and **PGP** implementation in **Elm**, I'm doing a lot of ports to ~~good~~ old **JavaScript**:
    - Going for **Elm** to **JS**, performance fall like 1000%
    - Without OpenPGP.js, this **SPA** would not be possible
    - Ditto with Tom Wu JSBN library
    - And with JSZip library as well

Missing parts and how to contribute

- The missing parts:
  - ***Highest priority***: GL Pages force HTTPS custom domains
    - GitLab is Accepting Merge Requests
  - Signatures and validate are almost done. Merge is not started yet. No biggy to implement this parts.
  - Signing of keys was recently added to OpenPGP.js and the same day it was implemented in **Sign Sign** but, at the moment it’s to slow but ...

Missing parts and how to contribute

- The missing parts:
  - … it would be ideal to use the trust model built-into **PGP**. For example, if the ***B4*** (***Deloitte***, ***PwC***, ***E&Y*** and ***KPMG***) had their own key **SKS Keyservers** on which they would have all the public keys of their employees which would be signed with their main key (public available from website). That way, whenever an employee would need some signatures, they would follow the guidelines of a "***Key signing party***" to request the neccesary information to validate the signers identity before signing their keys.
  - That would allow everybody else to see a trace of trust from ***signers → lawyer → B4***. It would be like an electronic version of a ***Notary***.
  - The point is, by using this approach, we wouldn't only rely on a **SPF** (centralized an monopolistic vs peer-to-peer network)

Missing parts and how to contribute

- How to contribute:
  - In order for this not to be a one-man-show, the basic concepts will be defined as a protocol for others to be able to implement in other languages or platforms. I myself are going to make a version in F# to calculate fractions of polynomials and for the rest rely on **GnuPG**. It's OK to have web apps, but I live for the most of my time in a terminal.
  - Licenses:
    - Protocol and media content: CC BY-SA 4.0
    - SPA: GPL-3.0
    - Underlying libraries: LGPL-3.0
    - ***Free*** as in "***free speech***", not as in "***free beer***"

Missing parts and how to contribute

- How to contribute (there are only three ways):
  - Donate well written code or create media content
  - Donate design/code reviews with constructive criticism
  - Donate BTCs:
    - 1NakxMC1DbSGtrzM7NgQtRVZc6qpB6JYvA

Summary

- A multi-way security by relying on mathematics to make problems very hard to solve instead of using obfuscation
- Create, sign and verify locally without any 3rd party knowing and tracking your behavior
- Distributed infrastructure avoids monopoly and SPF
- Aiming for a international cross-country standard
- Support for NemID? Sure, Nets DanID A/S should just run a script converting x.509 to PGP as they are storing all our private certificates. I would just leave that there … A “private” company can impersonate me electronically …
- Threshold Scheme superfluous? What it brings to the table (***some***,***all***,***any***) can be done with ***very basic math*** + **JSON** + **PGP** … (thank you Ximin Luo)

Q & A

Any Questions?