# Polyas Open Cred – a Tool for Voters Credential Generation

The purpose of this tools is to generate voting credentials for voters in a transparent and controlled way. This process is intended to be carried out by an entity independent from the e-voting provider (POLYAS), typically designated by the Election Council, in order to guarantee that the e-voting provider does not know the voters password and cannot add additional ballots to the ballot box.

The tool takes as its input a list of records containing voter identifiers and additional *delivery* information (information used later to deliver passwords to voters, such as postal addresses or e-mail).

The tool generates several files, some of them intended for the distribution facility (such as trusted printing facility) and the other are intende for the e-voting provider (POLYAS); the former contain, in particular, voters's passwords (in an encrypted form), the latter contain, public voter credentials (public voter's verification keys) which will be uploaded to the public registry.

## Prerequisits

In order to compile or run the tool JAVA 11 or higher must be installed. (Note that, there's no need to install Java Cryptography Extension (JCE) unlimited strength anymore since newer versions include it.)

## How to compile the tool

Change directory to parent folder and type

```
mvn install
```

This will build all sub-projects including the credential generation tool. The maven build tool is hosted at http://maven.apache.org.

After the build, the zip file `polyas-core3-open-cred-x.y.zip`, or `polyas-core3-open-cred-x.y-SNAPSHOT.zip` for snapshot versions containing the program is located in folder

```
polyas-core3-open-cred/target
```

The part `x.y` of the file name describes the version information.

## How to run the tool

Copy the zip file (created in the previous step) to a preferably empty directory and unzip it to use the program. Then execute command

```
java -jar polyas-core3-open-cred-x.y-SNAPSHOT.jar \
     DISTRIBUTOR_PUBKEYFILE REGISTRYFILE VOTER_ID_COLUMN_HEADER
```

where

- `DISTRIBUTOR_PUBKEYFILE` is the public PGP key from the clear text password distributing party.
- `REGISTRYFILE` is the input registry.
- `VOTER_ID_COLUMN_HEADER` is the name of the registry file column that contains the voter identifier

### Input data format

`DISTRIBUTOR_PUBKEYFILE` can be created using GPG. It should contain **only the public key**.

`REGISTRYFILE` is a UTF8 encoded, CSV formated file. It must be seperated by semicolon and must include a column header line. At least one column of this input registry must be unique and not empty. Each column that is unique and not empty may serve as voter id. The voter id is used at voting time by the voter to log in. It's also called some times PIN.

### Ouput files

The following files are generated and placed in a newly created folder of the form `YYYYMMDD-HHMISS_HASH`

- `dist_YYYYMMDD-HHMISS_HASH.asc`
- `polyas_YYYYMMDD-HHMISS_HASH.csv`
- `polyasSig_YYYYMMDD-HHMISS_HASH.csv.sig`
- `sigPubKey_YYYYMMDD-HHMISS_HASH.asc`

where `YYYYMMDD-HHMISS` is the current date and time, eg. 20190524-194559, and `HASH` are the first 5 bytes in a hexidecimal representation of SHA512-hash of the content of the file `polyas_YYYYMMDD-HHMISS_HASH.csv`. Tagging files with such a suffix is meant to provide a measure agains confusing files produced by different runs of the tool. Integrity of the process requires that the corresponding files (that is files from the same run and, hence, files with the same suffix) are send to the distribution facility and POLYAS.

#### Public verification key of the credential generation tool

`sigPubKey_YYYYMMDD-HHMISS_HASH.asc` contains the public verification key used by the tool to sign the output files. Every time the tool is executed a new key pair is produced. The secret key is ephemeral: it lives only in memory, is used for signing and then forgotten. The public key is persisted in the public signing key file. It is used for verification procedures using eg. GPG.

### Registry for the distributing party (printing facility)

`dist_YYYYMMDD-HHMISS_HASH.asc` is PGP encrypted file with the public key contained in DISTRIBUTOR_PUBKEYFILE and singed by the ephemeral signing key (the signature can be verified using the public key from `sigPubKey_YYYYMMDD-HHMISS_HASH.asc`).

The `dist` file contains

- the clear text password in column 'Password'
- all columns from the input registry except for voter id column

The distribution facility, which owns the secret key corresponding to the public key dist_YYYYMMDD-HHMISS_HASH.asc used in this process, may decrypt this file using:

```
gpg -d dist_YYYYMMDD-HHMISS_HASH.asc
```

If the signature should be checked during decryption one must import the public part of the signing key using

```
gpg --import sigPubKey_YYYYMMDD-HHMISS_HASH.asc
```

and edit the trust level to a suiting level.

### Registry for polyas

`polyas_YYYYMMDD-HHMISS_HASH.csv` is a semicolon separated CSV file containing

- all columns from the input registry
- the hashed password of the voter in column named 'Password SALTED_SHA_256'
- the public signing key of the voter in column named 'Public Signing Key'

### Detachted signature file for polyas

`polyasSig_YYYYMMDD-HHMISS_HASH.csv.sig` contains a detachted signature of the content of the registry file for polyas using the public key in sigPubKey_YYYYMMDD-HHMISS_HASH.asc.

You may import the public key into gpg by

```
gpg --import sigPubKey_YYYYMMDD-HHMISS_HASH.asc
```

After that you might want to asign a trust level.

Use

```
gpg --verify polyasSig_YYYYMMDD-HHMISS_HASH.csv.sig polyas_YYYYMMDD-HHMISS_HASH.csv
```

to verify signature.

## Shipping

Please ship to POLYAS the following files

- `polyas_YYYYMMDD-HHMISS_HASH.csv`
- `polyasSig_YYYYMMDD-HHMISS_HASH.csv.sig`
- `sigPubKey_YYYYMMDD-HHMISS_HASH.asc`

Please ship to the PRINTING FACILITY (password distributing party) the following files

- `dist_YYYYMMDD-HHMISS_HASH.asc`
- `sigPubKey_YYYYMMDD-HHMISS_HASH.asc`

Preferably the file `sigPubKey_YYYYMMDD-HHMISS_HASH.asc` should be shipped separately, if possible. Also it's a good idea to verify a fingerprint of the key file eg. by a phone call.

## Process

We describe now the recommended process of the password generation. The process is carried out by three entities:

- **EP** – the election provider (Polyas),
- **REG** – the registrar (GI),
- **DIST** – the distribution facility (such as trusted printing facility).

Additionally, the Election Council (**EC**) is responsible to oversee the whole process and defines the ballot content.

The process as described below contains some extra steps (not strictly necessary for the set up of an election) in order to increase robustness of the process (prevent potential misconfigurations).

Steps 1-5 are preparation steps and should be done, if possible, ahead of time.

0. **[DIST/EC]** DIST obtains a template of the election invitation letter. The only missing fields of this template should be: address and password.

1. **[DIST/REG]** DIST sends its public PGP encryption key (in the .asc format) to REG (over, for instance, e-mail).

2. **[DIST/REG]** Using a different channel (for instance phone), DIST gets in touch with REG in order to confirm the fingerprint of its public key.

3. **[REG]** REG makes sure that it is able to run the credential too.

   REG may compile the tool from the sources (see above), which is the recommended option. Alternatively, REG may request compiled JARs form the EP.

4. **[REG]** REG prepares the input CSV file with the voters' data.

In the simpler case, the input data can only contain two columns: the voter identifier (with header `ID`) and voter's delivery address (e-mail or postal address, depending on the used password delivery channel). An example input file may look like this:

```
ID;Address
V101;voter101@example.com
V102;voter102@example.com
V103;voter103@example.com
```

REG tries if the tool works correctly on such input data, running command like this (up the the version number and file names):

```
java -jar polyas-core3-open-cred-0.5-SNAPSHOT/polyas-core3-open-cred-0.5-
SNAPSHOT.jar \
     printing_facility.asc inputRegistry.csv ID
```

where `printing_facility.asc` is the file with the public key of the printing facility (for testing any public key may be used) and `inputRegistry.csv` is the input registry file. Such a command creates several files; all the created files have the same tag consisting of the timestamp and public credential content hash, as described above, and all the files are created in a new folder whose name is the same tag. After this step, the created data should be deleted in order to avoid future confusion (even though the files are fingerprinted and time-stamped).

*At this moment REG should be able to run the password generation tool and should be in possession of two files: the .asc file with the public key of the DIST and the input registry CSV file.*

5. **[REG] Credential generation**. REG runs the credential generation tool, as in the step above, making sure that the proper input files are used: proper file with the public key of DIST and the final input registry file.

6. **[REG] File distribution**.

   REG sends files

   - `sigPub...asc` and
   - `dist...asc`

   to DIST (the printing facility) and files

   - `sigPub...asc`,
   - `polyas_...csv`, and
   - `polyasSig_...csv.sig`

   to EP (Polyas), where `...` is a tag (timestamp + hash).

   *Note that the file with passwords is encrypted*

   REG keeps the files for future reference.

7. **[REG/EP] Integrity check 1**: EP check that the received files are in the expected format and correctly signed.

   Using an independent channel (phone), EP and REG get in touch in order to confirm that the files have the expected data tag (suffix in all the shipped files) and to confirm the fingerprint of the ephemeral public key (the fingerprint of `sigPub...asc`).

8. **[REG/DIST] Integrity check 2**: Similarly, DIST check that the received file can be decrypted, is in expected format and is correctly signed.

   Using an independent channel (phone), DIST and REG get in touch in order to confirm the expected data tag (suffix in all the shipped files) and the fingerprint of the ephemeral public key (the fingerprint of `sigPub...asc`).

9. **[EP/DIST] Integrity check 3**: EP and DIST get in touch to make sure that they obtained files with the same tag and signed with the same ephemeral key (they exchange the key fingerprint).

10. **[EC/REG] Final confirmation**: REG/EC gets in touch with DIST and EP and gives the final confirmation and the green light to proceed.

11. **[DIST]** Having done the integrity checks as above and having obtained the final confirmation, DIST (prints and) distributes the credentials.

12. **[EP]** Having done the above integrity checks and having obtained the final confirmation, EP sets up the election with the obtained registry.