Blind Proxy Voting Implementation

František Hakl

hakl@cs.cas.cz

Institute of computer science, Prague

Sep 2018

Motivation:

- long-term request of some colleagues for distance voting
- expected increase in the number of voters with a probable low participation in direct voting

Statutory requirements for the electoral process:

- $\mathbf{SR_{No341}}$: have to meet law №. 341/2005 (Zákon o v.v.i.)
- $\mathbf{SR_{\S 18(5)}}$: §18 (5) "přímá rovná tajná volba" (e.g. direct equal secret suffrage)

General requirement on electoral process:

- $\mathbf{GR_1}$: allows remote ballot
- $\mathbf{GR_2}$: subsequently check-able (after the voting)
- $\mathbf{GR_3}$: open-and-shut (easy to prepare, easy to vote, easy to evaluate)
- $\mathbf{GR_4}$: trustworthy and transparent
- $\mathbf{GR_5}$: private (should contain identity anti-disclosure mechanisms)
- $\mathbf{GR_6}$: resistant to sabotage and manipulation
- $\mathbf{GR_7}$: not demanding special infrastructure (including internet protocols and connections)
- $\mathbf{GR_8}$: the possibility of documenting the election result and recalculation of votes

Sources:

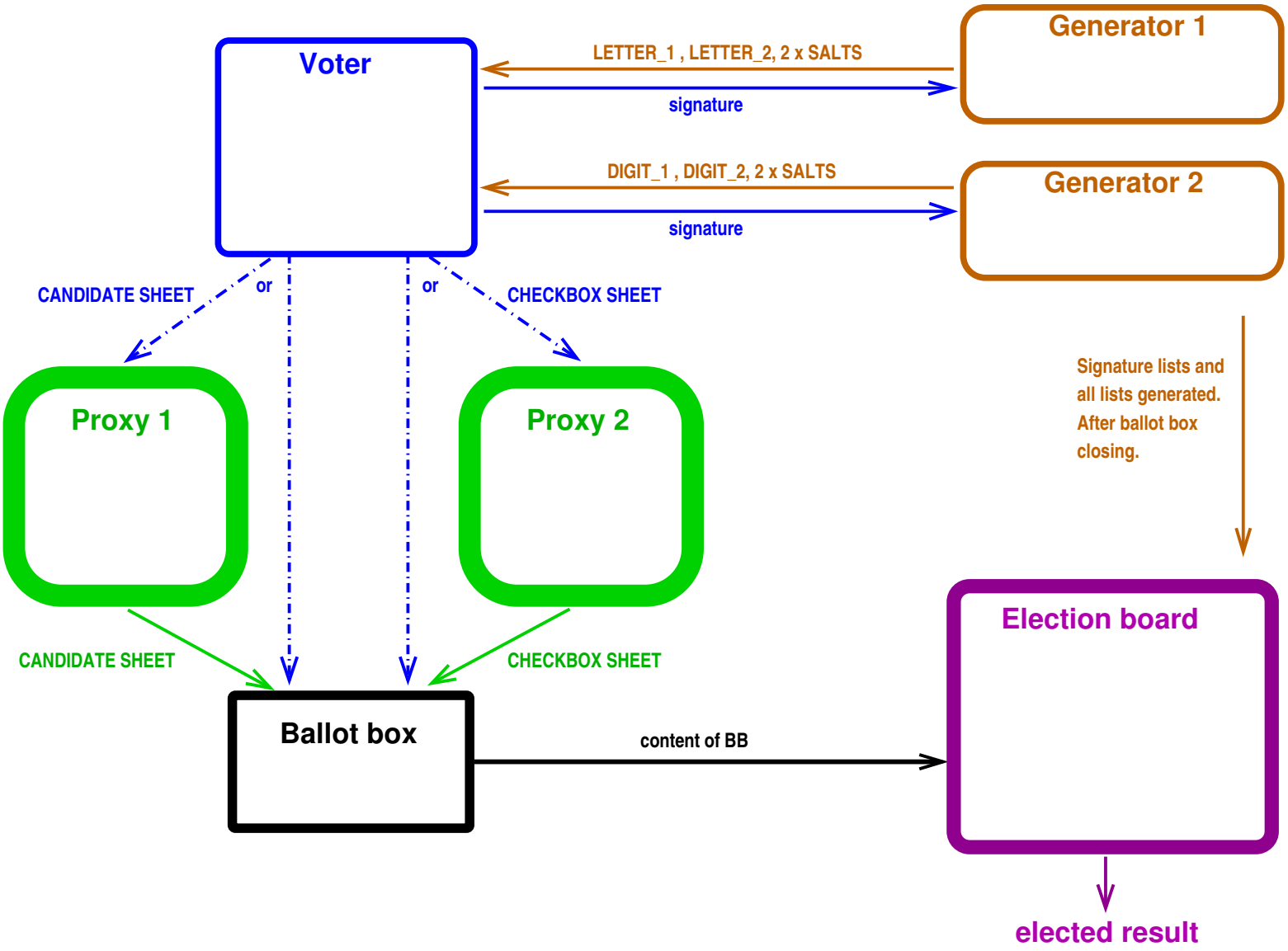
- Haniková Z., "Blind Proxy Voting", Tech. Rep. No. V-1250, ICS AS CR, 2017
- Wikipedia

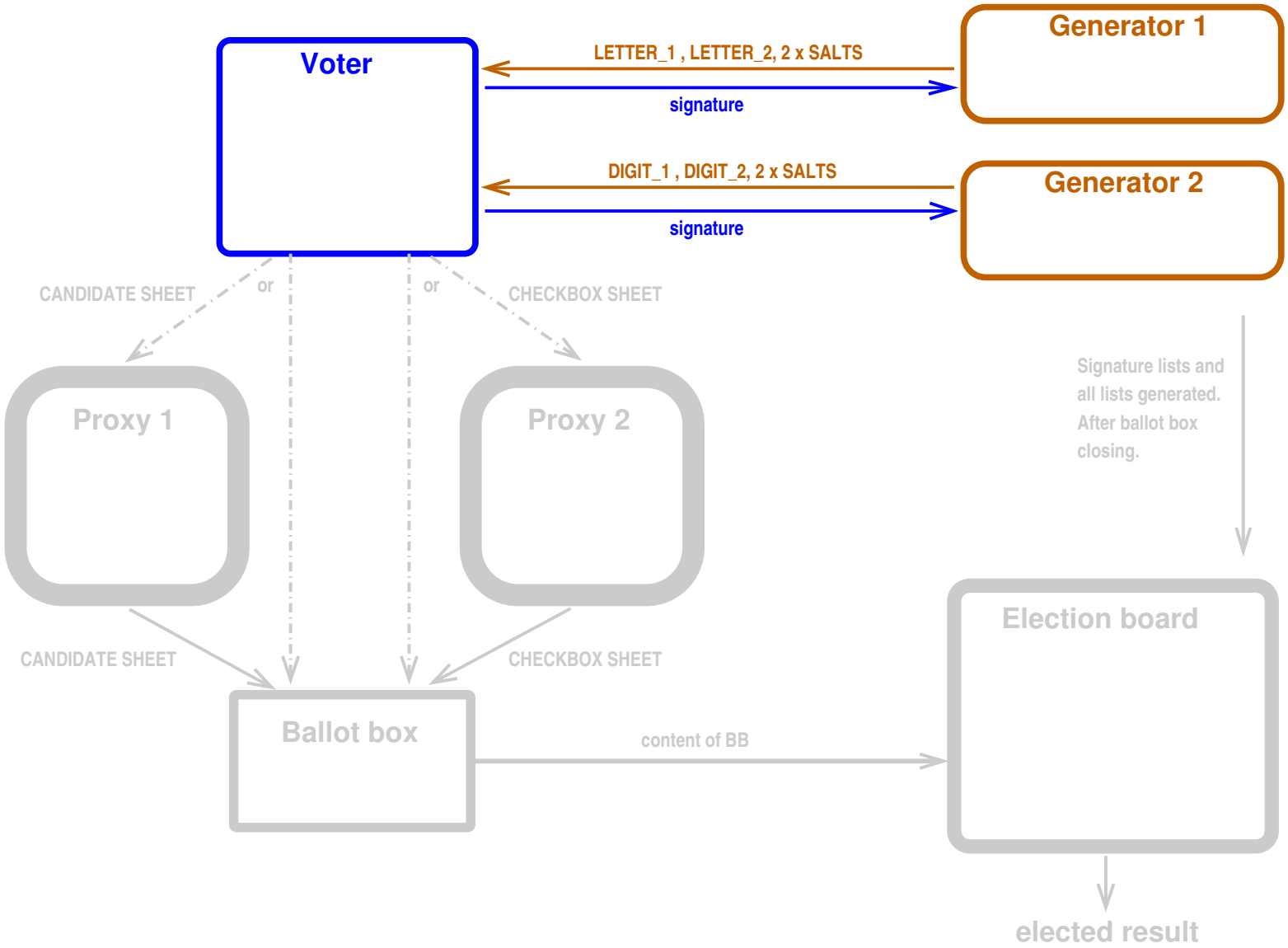
Human sets:

		defined by	role	minimal requirements
voters	$\bar{V}$	law	determine $\bar{G}_i$, $\bar{EB}$	meet law and internal regulations
two generators	$\bar{G}_1$ $\bar{G}_2$	$\bar{V}$	generate and distribute keys	$\lvert\bar{G}_1\rvert = \lvert\bar{G}_2\rvert = 1$ $(\bar{G}_1 \cup \bar{G}_2) \cap \bar{V} = \emptyset$
election board	$\bar{EB}$	$\bar{V}$	evaluation of elections	$\lvert\bar{EB}\rvert \geq 3$ (odd and >1)
two proxies	$\bar{P}_{1,v}$ $\bar{P}_{2,v}$	voter $v \in \bar{V}$	represent voter	$\bar{P}_{1,v}$ and $\bar{P}_{2,v}$ mutually unknown
candidates	$\bar{C}$	themselves	persons to be elected	meet law and internal regulations

- in addition all sets $\bar{G}_i$, $\bar{P}_{i,v}$, $v \in \bar{V}$, $\bar{EB}$, $\bar{C}$ must be mutually disjoint
- there is possible that proxy does not know the identity of his/her principal voter
- !!! proxy does not know who is the second proxy and vice-versa !!!

Data-flow sheet:





Cryptography hash functions:

- main properties of cryptography hash:
  - pre-image resistance:
    for hash h it is difficult to find m such that $h = hash(m)$
  - second pre-image resistance:
    for m_1, it is difficult to find a different m_2 such that
    $hash(m_1) = hash(m_2)$
  - collision resistance:
    it is difficult to find tuple $m_1 \neq m_2$ such that $hash(m_1) = hash(m_2)$
- in addition $\mathrm{HMAC}(K, m) = hash\Big((K' \oplus opad) \| hash\big((K' \oplus ipad) \| m\big)\Big)$ is resistant to length-extension attacks
- widely uses in electronic communication for password store & verification, file integrity check, proof-of-work, file or data identifier, pseudo-random generation, key derivation and other digest applications
- implemented in frequently used programming languages, including php and Python
  - Python 3.6 implements *hashlib* and *hmac* libraries: sha3_224(), sha3_256(), sha3_384(), sha3_512(), shake_128(), shake_256(), blake2b(), blake2s() (sha is developed and used by NSA)
  - php7.2: *print_r*(*hash_algos*()); lists approximately 50 hash functions (md5, sha, ripemd, whirlpool, tiger, snefru, gost, gost-crypto, adler, crc, fnv, joaat, haval)
- IMPORTANT: 2018 standard CPU (GPU) can compute approximately 10^6 hashes per second for an input of the length 8

Keys (suggested example):

$\bar{G}_1$ generate LETTER keys set

Key	Salt	hash
AfGkDT	f19f774a23ab46b89356f7ce77f6a203	hash(salt.AfGkDT.salt)
DsEgju	08609e5cb43d4b69ba48dd46d73303eb	hash(salt.DsEgju.salt)
DwEjKI	19dd4a6303824e6396b4b971c98fa3ee	hash(salt.DwEjKI.salt)
eERviA	436d52511b0646389f1ab45c0191d7c7	hash(salt.eERviA.salt)
HGEShY	23e28e51a9904b47a667220bf9847ec4	hash(salt.HGEShY.salt)
HSWEja	f5cb0c491729441d98ebf3a6224032aa	hash(salt.HSWEja.salt)
lahdFT	f25cebd3078b4512ad5cad33d502376b	hash(salt.lahdFT.salt)
ldfyFg	74cdb89d710c479e97aa952be9828e27	hash(salt.ldfyFg.salt)
lSiKaF	27c295a0406a4106a1a470a249281925	hash(salt.lSiKaF.salt)
sdgEda	aef5b316c04b47db85f86038bfb61108	hash(salt.sdgEda.salt)
sDhHda	6f8fb80daa944bca89e061b0051eb71c	hash(salt.sDhHda.salt)

$\bar{G}_2$ generate DIGIT keys set

Key	Salt	hash
136471	a343e926a85740f9b1fc21b1537c1d29	hash(salt.136471.salt)
156434	56cabdf213ca4d0aa9ac26a6fb083a6f	hash(salt.156434.salt)
451587	55e441d45523432cb771f441bf90b681	hash(salt.451587.salt)
458365	3a8b4a6afd0e48bbb49b742c10343a94	hash(salt.458365.salt)
658745	e15dc2e41b3540b19368f25d5a8a91ef	hash(salt.658745.salt)
712732	eb83bc95c1014e6592fac8bb739f2cbc	hash(salt.712732.salt)
746212	98a0c0e7d9fc4440b0d21928e23b3b15	hash(salt.746212.salt)
918396	2e8e013cb8e44d6991479a5382355533	hash(salt.918396.salt)
925319	dd8129a63c944cd5952ad707185105b5	hash(salt.925319.salt)

Keys:

- ordered tables of unique keys, salts and hashes are generated by $\bar{G}_1$ and $\bar{G}_2$ independently and in secret
- salts should be at least of 32 hex number
- number of keys generated in each set is roughly four-fold than the number of voters
- both lists of hashes are published
- hash method used is published
- both lists of keys and salts remain secret
- keys and corresponding salts are put into envelopes (separately by both $\bar{G}_1$ and $\bar{G}_2$, one corresponding tuple (key,salt) per envelope)
- voter randomly chooses just four envelopes with two LETTER ($L1$, $L2$) and two DIGIT ($D1$, $D2$) keys against the signature
- the voter can check to get the keys from the lists (hash method is published, salt is known by voter)

Code generating keys

Python3

```
#!/usr/bin/python3
import uuid, hashlib, random, time, string

key_array = []; salt_array = []; key_salt_array= []; hash_array = [];
sal = string.ascii_letters

random_seed = input('Please enter any randomize string:')
random.seed(random_seed+str(time.time())) # time prevents retake keys
pool_size = input('Please enter pool size (int):')

for v in range(int(pool_size)) :
#    key_v = str(random.randint(100000, 999999))  # DIGIT keys
    key_v = ''.join([random.choices( sal )[0] for x in range(6)]) # LETTER keys
    salt_v = uuid.uuid4().hex
    en_salt = salt_v.encode()
    hash_v = hashlib.sha256( en_salt + key_v.encode() + en_salt ).hexdigest()
    hash_array.append(hash_v + '\n')
    key_salt_array.append('\perpage{' + key_v + '}{' + salt_v + '}\n')

key_salt_array = list( set( key_salt_array))
sksa = sorted( key_salt_array); sh = sorted(hash_array)

fh = open("./keys/keysandsalts_unsorted_privat.tex","w");
fh.writelines(key_salt_array) ; fh.close()
fh = open("./keys/hashes_sorted_public.txt","w");
fh.writelines(sh) ; fh.close()
fh = open("./keys/keys_sorted_for_eb.txt","w");
fh.writelines(sksa) ; fh.close()

# the following tuple of rows check key validity
# salt = "put salt here"; my_key = "put key here";
# print(hashlib.sha256(salt.encode()+my_key.encode()+salt.encode()).hexdigest())
```

Elections to the ICS Institution Board 2021
Election round №: 2

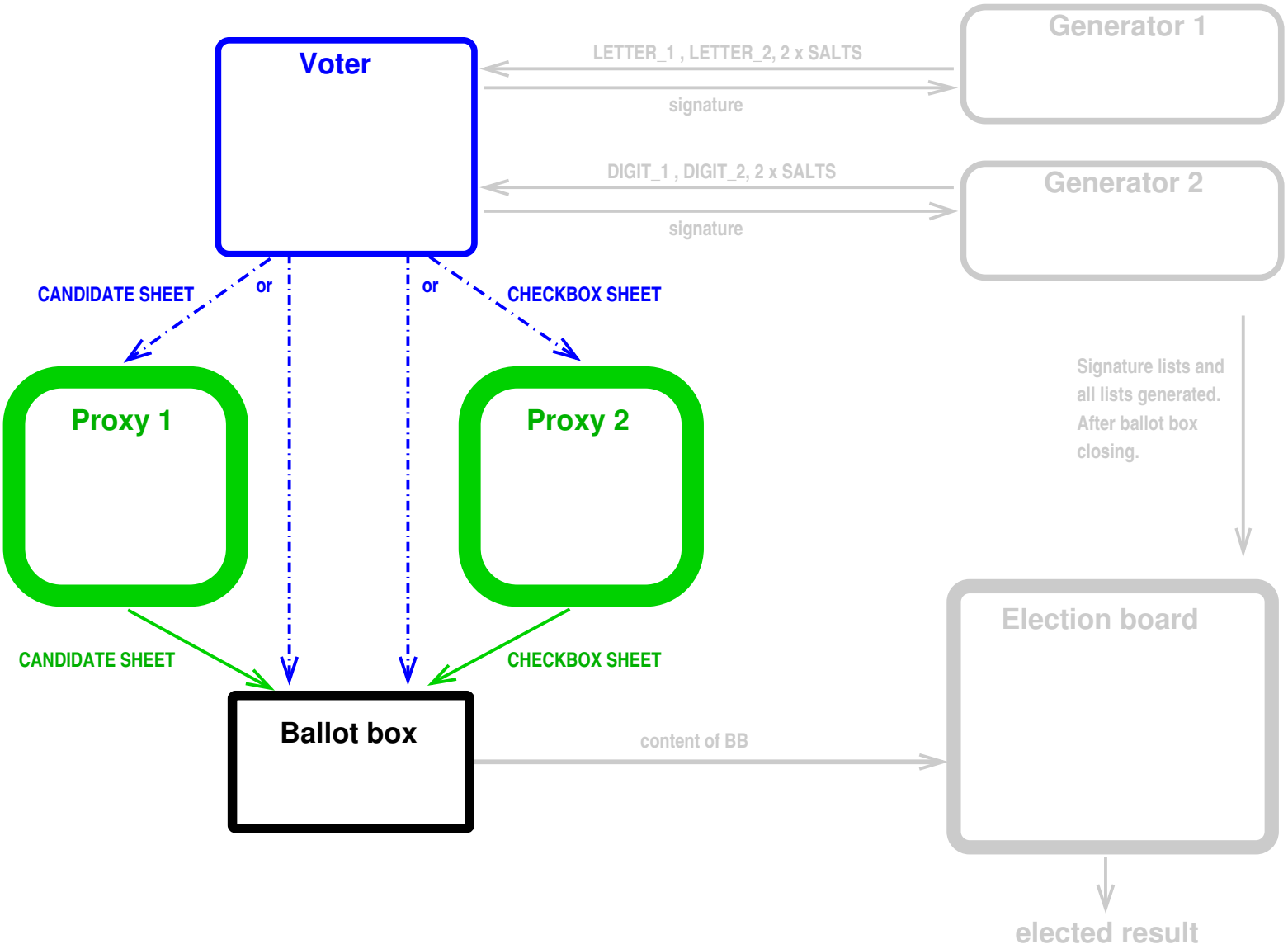
SALT: 7360f21fb824400f974d1954769fa018

KEY: coLvPR

Use the following Python code to check validity of the key obtained:

```
import hashlib, string
salt = "7360f21fb824400f974d1954769fa018"
mykey = "coLvPR"
print(hashlib.sha256(salt.encode()+mykey.encode()+salt.encode()).hexdigest())
```

List of hash values is available at:
http://url.to.hash.list



Ballot forms example (minimal handwriting, fixed number of fields):

(odt/docx blank files will be available, blue text is filled by voter)

CANDIDATE part

L2-D1 keys

HGEShY-458365

Fill in 10 cells with 'x' exactly!

Karel	Honza	Libor	Josef	Alena	
				x	1
					2
		x			3
					4
			x		5
	x				6
					7
				x	8
					9
					10
	x				11
x					12
	x				13
			x		14
				x	15

L1 key: **sdgEda**

CHECKBOX part

L2-D1 keys

HGEShY-458365

Fill in 10 rows with 'x' exactly!

1	**x**
2	
3	**x**
4	**x**
5	
6	**x**
7	**x**
8	
9	**x**
10	**x**
11	**x**
12	
13	**x**
14	
15	**x**

D2 key: **136471**

(Honza, Libor, Alena selected)

Ballot forms example (text version with variable number of fields):

(odt/docx blank files will be available, blue text is filled by voter)

CANDIDATE sheet

L2-D1 keys

HGEShY-458365
A=number of filled fields in rows 1–15
B = 6232
X ∗ 14131 − **Y** ∗ 2092 + 9785 = Ω

Alena	Nobody	1
Honza	Beatles	2
Josef		3
Karel	blah blah	4
Libor		5
	Karel, Libor	6
	Alena	7
	Karel	8
	Ferrari	9
	Libor	10
		11
	Josef	12
	Honza	13
	Mozart	14
	Libor	15

L1 key: sdgEda

CHECKBOX sheet

L2-D1 keys

HGEShY-458365
X=number of "x" in rows 1–15
Y = 3291
Ω = **A** ∗ 15220 − **B** ∗ 1124 + 46058

1	x
2	
3	x
4	
5	
6	
7	x
8	
9	
10	x
11	x
12	
13	x
14	
15	x

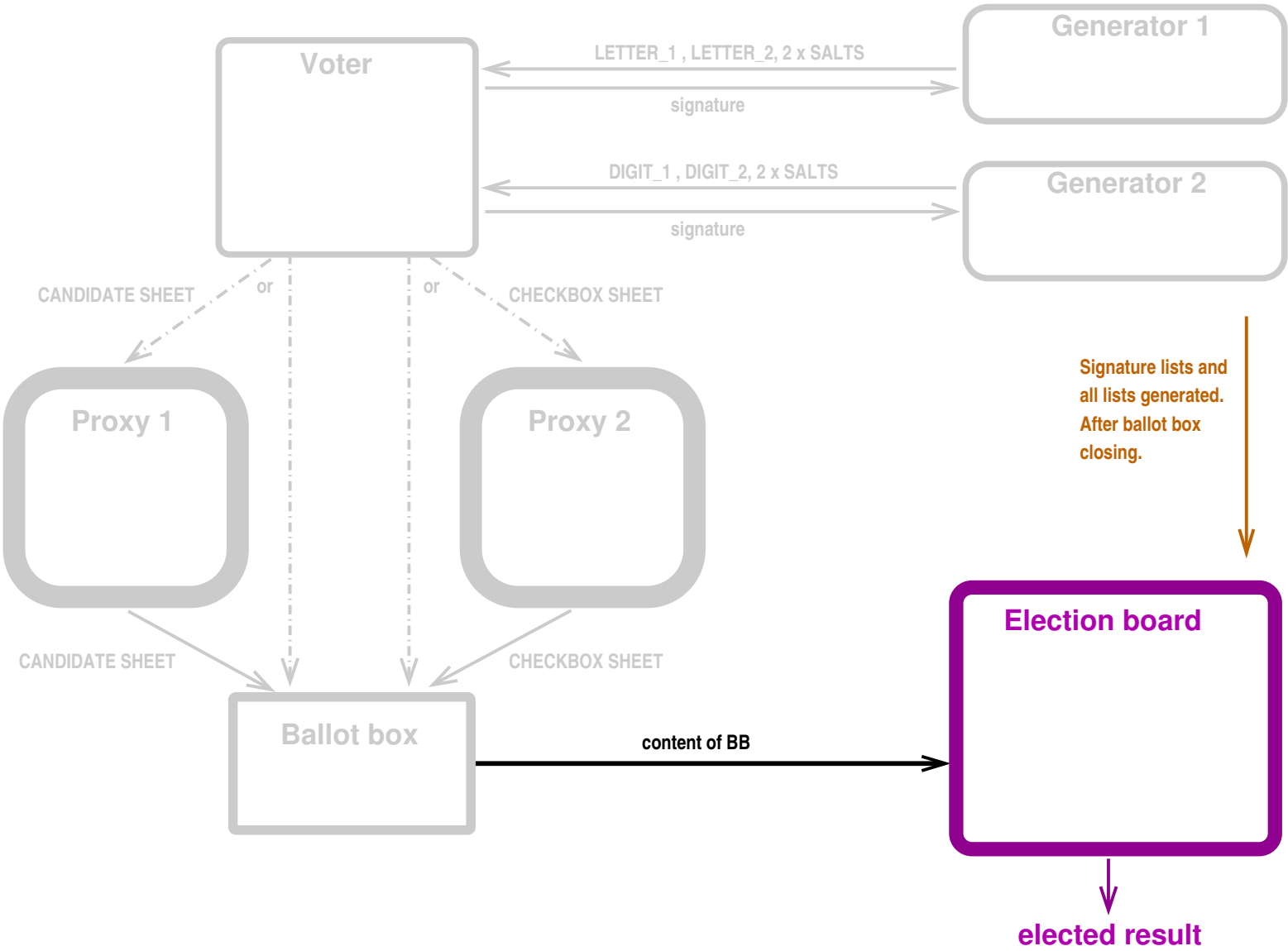
D2 key: 136471

(Honza, Libor, Alena selected, $\Omega = -6776070$)

Code generating OG vectors

Python3

```
#!/usr/bin/python3
import random, time, string

random_seed = input('Please enter any randomize string: ')
random.seed(random_seed+str(time.time())) # time prevent recalculation
A = int(input('Please enter value of "A" (int): '))
X = int(input('Please enter value of "X" (int): '))

B = random.randint(1000, 10000); Y = random.randint(1000, 10000)
AA = random.randint(10000, 50000); BB = random.randint(1000, 10000)
XX = random.randint(10000, 50000); YY = random.randint(1000, 10000)
Z  = random.randint(1000, 10000)

omega = X*XX-Y*YY+Z
C = omega - A*AA +B*BB
while( C < 3000 ) :
    B = B + 1
    C = omega - A*AA +B*BB

while( C > 50000 ) :
    BB = BB - 1
    C = omega - A*AA +B*BB

left_str = "X * %d - Y * %d + %d = Omega"%(XX,YY,Z)
right_str = "Omega = A * %d - B * %d + %d"%(AA,BB,C)

print("A = %d"%(A)); print("B = %d"%(B)); print( left_str+"\n" )
print("X = %d"%(X)); print("Y = %d"%(Y)); print( right_str+"\n" )
print("Omega = %d"%(omega))
```



Evaluation rules:

- $\mathbf{R_K}$: $\bar{G}_{1,2}$ publish lists of all keys with salts and size of $\bar{S}_1 \cap \bar{S}_2$, where $\bar{S}_{1,2}$ are signature lists corresponding to $\bar{G}_{1,2}$
- $\mathbf{R_V}$: a tuple (*CANDIDATE*, *CHECKBOX*) of sheets is valid iff
  - $\mathbf{R_{VA}}$: all keys $L1, L2, D1, D2$ are in key lists and
  - $\mathbf{R_{VK}}$: $L2 - D1$ keys tuple is the same on both sheets and
  - $\mathbf{R_{VO}}$: numbers of filled fields on the opposite sheets are correct and
  - $\mathbf{R_{V1}}$: for a given $L2 - D1$ keys only one such *CANDIDATE* sheet and one such *CHECKBOX* sheet are in the ballot box
- $\mathbf{R_R}$: any sheets which does not form a valid tuple will be removed
- $\mathbf{R_2}$: if $L1, L2, D1, D2$ and $L1^*, L2^*, D1^*, D2^*$ are keys for two valid tuples and $\{L1, L2, D1, D2\} \cap \{L1^*, L2^*, D1^*, D2^*\} \neq \emptyset$ remove both valid tuples
- $\mathbf{R_N}$: the candidate received a vote in a valid tuple if his/her name is in the row in which "x" is present in the *CHECKBOX* sheet
- $\mathbf{R_{PL}}$: finally the following items will be published:
  - $\mathbf{R_{PL1}}$: list of $L2$ and $D1$ keys in all valid tuples will be published (without bounds between $L2$ and $D1$)
  - $\mathbf{R_{PL2}}$: list of all keys in invalid sheets will be also published
  - $\mathbf{R_{PL3}}$: for all valid tuples of sheets both of them will be published but WITHOUT upper parts of tables containing $L2 - D1$ keys and number of filled fields in the second sheet

Axioms:

- $\mathbf{A_{EB}}$: election board is undoubtedly credible and trustworthy
- $\mathbf{A_G}$: each generator is undoubtedly credible and trustworthy
- $\mathbf{A_{keys}}$: the probability of keys matching is negligible
- $\mathbf{A_{hash}}$: it is infeasible to generate a key from its hash value except by trying all possible salt.key.salt
- $\mathbf{A_{VPP}}$: each voter know and trust his/her proxy(ies)
- $\mathbf{A_{PP}}$: proxies does not know each other
- $\mathbf{A_P}$: each proxy has electoral intentions similar to that of his/her principal voter or does not know who is
- $\mathbf{A_{PI}}$: proxy identity is known to his/her principal voter only
- $\mathbf{A_{disj}}$: $\bar{G}_i$, $\bar{P}_{i,v}$, $v \in \bar{V}$, $\bar{EB}$, $\bar{C}$ are mutually disjoint

Validity of election process:

- $\mathbf{V_1}$: election process is invalid if the number of valid tuples is greater than size of signature lists intersection
- $\mathbf{V_2}$: election process is invalid if the number of valid tuples is less than predefined number (mainly one half of all voters)

Objectionable secret behavior:

("secret" behavior means that the originator(s) of the action will remain(s) unknown for everybody and forever)

- secret **Sabotage** of the electoral process
  (any action which results in the invalidity of the electoral process)
- secret intentional **Manipulation** of voting result
  (somebody has the possibility to change voting of someone else in a specific manner)
- secret voter's identity **Disclosure**
  (somebody knows the voting of somebody else voter or provides an information leading to such knowledge)
- secret **Randomization** of voting result
  (somebody has the possibility randomly change voting of someone else)
- secret **Targeted Invalidation** of voter's vote
  (any action which results in the invalid voting of known someone else)
- secret **Random Invalidation** of voter's vote
  (any action which results in the invalid voting of unknown someone else)

Analysis of **secret** violation of election: $\left(\frac{\text{why can do / why can not do}}{\text{a reason that violates the secrecy or impediments to action}}\right)$

	$\bar{G}_1$	$\bigcup_i \bar{G}_i$	$\bar{P}_{1,v}$	$\bigcup_i \bar{P}_{i,v}$	$m\bar{E}B \cup \bar{P}_{1,v}$	$m\bar{E}B \cup (\bigcup_i \bar{P}_{i,v})$
Sabotage	UOK $\mathbf{A_G}, \mathbf{R_{VA}}$	KK $\mathbf{A_G}, \mathbf{R_{PL}}$ (+1)	UOK $\mathbf{R_{VA}}$	UOK no info	ex post ex post $\mathbf{A_{EB}}$	ex post ex post $\mathbf{A_{EB}}$
Manipulation	UOK $\mathbf{A_G}, \mathbf{R_{VA}}$	MII $\mathbf{A_G}$	UOK $\mathbf{A_P}$,$\mathbf{A_{PP}}$	KK $\mathbf{A_P}$,$\mathbf{R_{PL3}}$	ex post ex post $\mathbf{A_{EB}}$	ex post ex post $\mathbf{A_{EB}}$
Disclosure	MII $\mathbf{A_G}$	MII $\mathbf{A_G}$	KK $\mathbf{R_{VPP}}$	KK $\mathbf{A_{VPP}}$	KK $\mathbf{A_{EB}}, \mathbf{A_{VPP}}$ $\mathbf{A_{PI}}$	KK $\mathbf{A_{EB}}, \mathbf{A_{VPP}}$ $\mathbf{A_{PI}}$
Randomization	UOK $\mathbf{A_G}, \mathbf{R_{VA}}$	MII $\mathbf{A_G}$	KK $\mathbf{A_P}$,$\mathbf{R_{PL3}}$	KK $\mathbf{A_P}$,$\mathbf{R_{PL3}}$	ex post ex post $\mathbf{A_{EB}}$	ex post ex post $\mathbf{A_{EB}}$
Targeted Inval.	UOK $\mathbf{A_G}, \mathbf{R_{VA}}$	MII $\mathbf{A_G}$	KK $\mathbf{A_{VPP}}$,$\mathbf{R_{PL3}}$	KK $\mathbf{A_{VPP}}$,$\mathbf{R_{PL3}}$	ex post ex post $\mathbf{A_{EB}}$	ex post ex post $\mathbf{A_{EB}}$
Random Inval.	UOK $\mathbf{A_G}, \mathbf{R_{VA}}$	KK $\mathbf{A_G}, \mathbf{R_2}$ (+1)	UOK $\mathbf{R_{VA}}$	UOK $\mathbf{R_{VA}}$	ex post ex post $\mathbf{A_{EB}}$	ex post ex post $\mathbf{A_{EB}}$

UOK - unknown other keys, **MII** - missing identity information, **KK**- known keys, $m\bar{E}B$ - member of $\bar{E}B$

Conclusion:

Suggested process:

- in the case of axiom validity no one person or tuple of persons can do objectionable secret action
- meets general requirements $\mathbf{GR_{1-8}}$
- ??? meets statutory requirement $\mathbf{SR_{No341}}$??? - legal analysis is needed

Practical notes:

- paper version of sheets is recommended due to lack of meta-info (which is included in electronic formats like PDF, jpeg, doc(x), ...)
- practical realization of sheets should be the same for all voters in order to keep privacy of distant voters
- public printers in ICS are accessible for everyone - use your own local printer or print directly via USB stick on printer with USB input port

Electronical (www) implementation:

??? trustworthy and transparent ???