Privacy in the Right To Ask project

Vanessa Teague[†] and Chris Culnane

July 15, 2022

[†]vanessa@thinkingcybersecurity.com and the ANU

https://github.com/RightToAskOrg

With thanks to these contributors: Andrew Conway, Rosey Conway, Charmaine Chew, Ishan Goyal, Matt Lefurge, Chuanyuan Liu, Lillian McCann, Tim McCann, Eleanor McMurtry, Hanna Navissi, Pedro Rosas, Miguel Wood

Outline

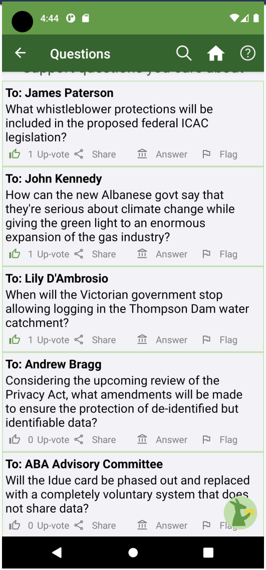
Right To Ask

How it will work

RightToAsk lets people suggest and up- and down-vote questions, which could be:

- directed to an MP to ask for an answer (e.g. from a constituent), or
- suggested for an MP to ask someone else (e.g. in a committee).

RightToAsk shows MPs which questions are popular and relevant to their role.



Cryptographic Tools

We have some tools

Microsoft's ElectionGuard crypto library includes:[1]

- additive-homomorphic encryption (based on El Gamal)
- threshold key generation
- distributed decryption (so the key is never recombined)
- proofs of proper decryption (based on Chaum-Pedersen)

`https://www.electionguard.vote/`

So people can express approval or disapproval (upvotes or downvotes), which can be aggregated homomorphically, decrypted in the aggregate, and proven correct, without exposing individual votes.

[1] This project has received a research grant from Microsoft

Homomorphic addition for ordinary elections

Voters verifying
their votes with
code they control

EncrVote = Encr(V,r)?
EncrVote = (g^r,g^m h^r) mod p?

Proofs of honest
vote recording
for the voter

A public bulletin board
so everyone can check
that their vote is there

Voter1: EncrVote1
Voter2: EncrVote2
Voter3: EncrVote3
Voter4: EncrVote4
Voter5: EncrVote5
Voter6: EncrVote6
....
....
....

EncrSum

Publicly computable
homomorphic addition

A set of decryption authorities
so no individual can decrypt
individual votes

Decrypt1
Decrypt2
Decrypt3
....
....
....

Proofs of honest
decryption on the
bulletin board

Sum

Publicly computable
combination of decrypted
shares

RightToAsk does *not* include

- Proofs of honest vote recording (cast-as-intended verification)
- ... though ElectionGuard does offer this
- Voter Authentication
- Receipt-freeness / defence against coercion
- ... so you can prove how you voted

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q|p-1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \bmod p$
- Private key s

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q|p-1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \bmod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1} h^{r_1}) \bmod p$$

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q|p-1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \bmod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1} h^{r_1}) \bmod p$$

- Encrypted votes can be added

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q | p - 1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \mod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1} h^{r_1}) \mod p$$

- Encrypted votes can be added
- $C_1 \circ C_2 = (g^{r_1} \times g^{r_2}, g^{v_1} h^{r_1} \times g^{v_2} h^{r_2}) \mod p$

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q | p - 1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \mod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1} h^{r_1}) \mod p$$

- Encrypted votes can be added
- $C_1 \circ C_2 = (g^{r_1} \times g^{r_2}, g^{v_1} h^{r_1} \times g^{v_2} h^{r_2}) \mod p$
- $= (g^{r_1 + r_2}, g^{v_1 + v_2} h^{r_1 + r_2}) \mod p$ which is an encryption of $v_1 + v_2$.

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q|p-1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \mod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1}h^{r_1}) \mod p$$

- Encrypted votes can be added
- $C_1 \circ C_2 = (g^{r_1} \times g^{r_2}, g^{v_1}h^{r_1} \times g^{v_2}h^{r_2}) \mod p$
- $= (g^{r_1+r_2}, g^{v_1+v_2}h^{r_1+r_2}) \mod p$ which is an encryption of $v_1 + v_2$.
- We can do this over and over again for millions of votes.

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q | p - 1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \bmod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1} h^{r_1}) \bmod p$$

- Encrypted votes can be added
- $C_1 \circ C_2 = (g^{r_1} \times g^{r_2}, g^{v_1} h^{r_1} \times g^{v_2} h^{r_2}) \bmod p$
- $= (g^{r_1 + r_2}, g^{v_1 + v_2} h^{r_1 + r_2}) \bmod p$ which is an encryption of $v_1 + v_2$.
- We can do this over and over again for millions of votes.
- Decrypt the sum, not the individual votes.

Exponential El Gamal over a prime field

El Gamal encryption (exponential form):

- Public parameters:
  - p, q large primes s.t. $q|p-1$
  - g with order q in $\mathbb{Z}_p^*$
  - public key $h = g^s \bmod p$
- Private key s
- Encrypt v_1 by generating random r_1,

$$C_1 = (g^{r_1}, g^{v_1}h^{r_1}) \bmod p$$

- Encrypted votes can be added
- $C_1 \circ C_2 = (g^{r_1} \times g^{r_2}, g^{v_1}h^{r_1} \times g^{v_2}h^{r_2}) \bmod p$
- $= (g^{r_1+r_2}, g^{v_1+v_2}h^{r_1+r_2}) \bmod p$ which is an encryption of $v_1 + v_2$.
- We can do this over and over again for millions of votes.
- Decrypt the sum, not the individual votes.
- Use ElectionGuard's proofs of proper decryption.

Everything goes on The Bulletin Board

Bulletin Board Library

A general purpose library allowing an entity to publish things continuously on a public bulletin board, enforcing historical transparency. Based on Merkle trees.

- Assume that citizens have some out-of-band way of comparing the root hash.

API

Written by Andrew Conway in rust. Available on crates.io as merkle-tree-bulletin-board and at `https://github.com/RightToAskOrg/bulletin-board`

- submit_leaf(string) $\rightarrow$ HashValue
- order_new_published_root() $\rightarrow$ HashValue
- get_hash_info(HashValue) $\rightarrow$ ...
- get_proof_chain(HashValue) $\rightarrow$ ...
- censor_leaf(HashValue)

WHAT DOES THE PROOF LOOK LIKE?

013cf9d2e26f0714b37bb1551a2d56bf30ad2b62a0d04bf7786f2113deac2f4c

Parent e4d533d4e7c356b2b11f5c120ce4465f70ca1f9b238b41c65aa58e431c119c1e

Leaf

Timestamp : 1628666742 which means Wed Aug 11 2021 17:25:42 GMT+1000 (Australian Eastern Standard Time)
Data : A

How the hash value was computed

Leaf prefix	00 (1 hex bytes)
Timestamp	0000000061137b76 (8 hex bytes)
Posted Data	A (1 string bytes)

The Sha256 hash of the above elements concatenated is 013cf9d2e26f0714b37bb1551a2d56bf30ad2b62a0d04bf7786f2113deac2f4c
This can be checked by the Linux command :

```
echo -n 000000000061137b76 | xxd -r -p | cat - <(echo -n "A") | sha256sum
```

Censor!

Full text inclusion proof

The purpose of this is to demonstrate that this hash value is included in the bulletin board. This is done by showing a chain of hash values leading up to a published hash value. Reversing the Sha256 hash function is (as far as we can tell) impractical. This means that other people who see the same published hash value as you, can tell if something nefarious is attempted with this node. The above explanation of the hash value proves that this hash value represents the values it is claimed for at the top of this page.

This node's parent is e4d533d4e7c356b2b11f5c120ce4465f70ca1f9b238b41c65aa58e431c119c1e

Branch

Left 013cf9d2e26f0714b37bb1551a2d56bf30ad2b62a0d04bf7786f2113deac2f4c
Right b8ba295e3ef5979d8eb1aebfab225f3b5aa1a81da1ff78b567189d2c84d01cc1

How the hash value was computed

Branch prefix	01 (1 hex bytes)
Left hash	013cf9d2e26f0714b37bb1551a2d56bf30ad2b62a0d04bf7786f2113deac2f4c (32 hex bytes)
Right hash	b8ba295e3ef5979d8eb1aebfab225f3b5aa1a81da1ff78b567189d2c84d01cc1 (32 hex bytes)

The Sha256 hash of the above elements concatenated is e4d533d4e7c356b2b11f5c120ce4465f70ca1f9b238b41c65aa58e431c119c1e
This can be checked by the Linux command :

```
echo -n 01013cf9d2e26f0714b37bb1551a2d56bf30ad2b62a0d04bf7786f2113deac2f4cb8ba295e3ef5979d8eb1aebfab225f3b5aa1a81da1ff78b567189d2c84d01cc1 | xxd -r -p | sha256sum
```

This node's parent is 1fd7b0aa49f523783fff87778ea7d167d6910c75457ee8025aed02b3c2dc2d52

Branch

Left e4d533d4e7c356b2b11f5c120ce4465f70ca1f9b238b41c65aa58e431c119c1e
Right ef57232d3efda36dde0e79bc90ef967575e5e901293af800a21221410e4623a4

How the hash value was computed

Branch prefix	01 (1 hex bytes)
Left hash	e4d533d4e7c356b2b11f5c120ce4465f70ca1f9b238b41c65aa58e431c119c1e (32 hex bytes)
Right hash	ef57232d3efda36dde0e79bc90ef967575e5e901293af800a21221410e4623a4 (32 hex bytes)

The Sha256 hash of the above elements concatenated is 1fd7b0aa49f523783fff87778ea7d167d6910c75457ee8025aed02b3c2dc2d52
This can be checked by the Linux command :

```
echo -n 01e4d533d4e7c356b2b11f5c120ce4465f70ca1f9b238b41c65aa58e431c119c1eef57232d3efda36dde0e79bc90ef967575e5e901293af800a21221410e4623a4 | xxd -r -p | sha256sum
```

This node is listed in the published root node dbe3ab351b5b7e43226443195c4a665c0e4a5f54fa284ae105a34984ef98310f

Published Root

A privacy model

PRIVACY MODEL

- **Your *writing* is public**
  - **and named**
- **Your *votes* are private**
  - **and only decrypted in the aggregate**
  - **decryption key is 2-out-of-3 secret shared and never explicitly recombined**

Is that good enough?

In this talk we'll look at the privacy implications of repeated exact aggregates in batches.

In different work (Litos, Kiayias, T : IACR eprint 760) we examined how to share the decryptor role among participants.

Right To Ask is different from ordinary elections

- Good...
  - It’s less important than real elections
  - Some perturbation might be acceptable
- Bad...
  - Small sizes make unanimity (or large biases) more likely
  - The system reveals who voted on what (not whether it was +1 or 0)
  - Ongoing decryption in batches makes privacy analysis hard

Plan A: Just do it

- Tally in batches of size B
- Let p be the fraction of votes that are up-votes
- In the best case, up- and down-votes are iid
- Then

$$Pr(\text{unanimity}) = p^B + (1-p)^B$$

If you participate a lot, some of your contributions will be in a unanimous batch, but most won't.

Voter1:	V1
Voter2:	V2
Voter3:	V3
Voter4:	V4
Voter5:	V5
....	....
....	....
....	....
VoterB:	VB

Publicly computable

E(Tally)	T

Distributed decryption

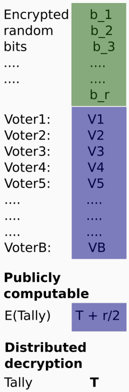
Tally **T**

Plan B: Add some random padding bits

- Group in batches of size B
- Add r encrypted random bits
- Tally the batch of $B + r$
- Then

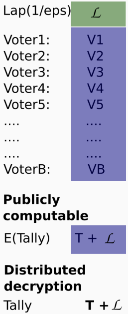
$$Pr(\text{unanimity}) = (p^B + (1-p)^B)/2^r$$

- This is $(\epsilon, \delta) - DP$ with $\delta = 1/2^r$
- There will still be some exposed unanimous batches, but this reduces the frequency.
- Need to subtract $r/2$ to preserve average relative rankings.



PLAN C: LAPLACE MECHANISM ON EACH QUESTION

- Sensitivity $\Delta f = 1$
- Add value from $Lap(x|1/\epsilon)$ with pdf $\frac{\epsilon}{2}\exp(-\epsilon|x|)$
- achieves $(\epsilon, 0)$-Differential Privacy



Plan D: Considering each person's complete list of contributions

No idea how to deal with this...

Verifiably generating the random padding

Lots of ways to do this in various trust models. Suggestions for efficient protocols welcome.

Discussion & future work

GETTING INVOLVED

- See the code and technical docs here
  `https://github.com/RightToAskOrg/`
- email me if you'd like to join the chat channel.
  `vanessa[at]democracydevelopers.org.au`

Questions?