Formally Verified Verifiable Generator

Mina A. Cyrus (Swansea University)
Mukesh Tiwari (Swansea University)

Can you vote for political elections online?

Verified 09 July 2024 - Directorate for Legal and Administrative Information (Prime Minister)

The rules are different depending on whether you vote in France, or from the foreigner:

In France | From the foreigner

You can vote via the internet provided you meet the following 3 conditions:

- You live abroad
- You are registered on a consular voters list. You can check your voter registration using this online service.
- When you registered, you provided an email address (e-mail address) and a phone number. This data is necessary to communicate a username and password.

E-Voting

In Switzerland, e-voting means voting online via the internet. E-voting is part of the Swiss e-government strategy, which entails close collaboration between the Confederation and the cantons. Since 2004, 15 cantons have offered internet voting to a limited part of their electorate in over 300 trials. The Confederation and cantons have followed the principle of 'security before speed'. In Switzerland, e-voting is only permitted if strict requirements under federal law are met.

Estonia leads world in making digital voting a reality

Baltic nation has expanded its digital revolution to include free, fair and secure online ballots

Moscow's blockchain voting system cracked a month before election

French researcher nets $15,000 prize for finding bugs in Moscow's Ethereum-based voting system.

Congress manifesto for 2024 LS polls should promise to junk EVMs, restore paper ballots: Prithviraj Chavan

Chavan, a former Maharashtra chief minister, raised the issue during deliberations in the party's panel on political issues at the three-day 'Chintan Shivir'. He said many other leaders have supported the view.

15 May, 2022, 04:10 PM IST

CNN politics The Biden Presidency Facts First 2022 Mi

Federal review says Dominion software flaws haven't been exploited in elections


By Sean Lyngaas, Evan Perez and Whitney Wild, CNN

Updated 12:18 PM EDT, Fri June 3, 2022

MOTHERBOARD
TECH BY VICE

Experts Find Serious Problems With Switzerland's Online Voting System Before Public Penetration Test Even Begins

Flaws found in NSW iVote system yet again

Analysis of source code published at the request of the NSW Electoral Commission shows that the state's election system software was still vulnerable to attack.

Blind advocates allege NSW's removal of online voting system is a breach of human rights

State electoral commission accused of discrimination for suspending iVoting platform as Blind Citizens Australia takes case to watchdog

How India conducted the world's largest election

Poll workers reach India's most remote corners to set up polling

AEC Australian Electoral Commission

1. Computer recognition (OCR/ICR/OMR) reads the preferences and other marks from the image of a scanned ballot paper.

   1a) A data entry operator populates any preferences that were unable to be read with high confidence by the computer recognition in step 1. The data entry operator also manually checks the marks read by computer recognition (for example a voter potentially identifying themselves on a ballot paper) for AEC determination.

Bootstrapping an Election

Prime p, q such that $p = 2 \times q + 1$

$$G_p = \{g^1, g^2, \ldots g^{(q-1)}\}$$

How to compute g?

$$u \leftarrow \{1 \ldots p\}$$
$$g = u^2 \mod p$$

Proof (Fermat's Little Theorem)

$$
\begin{aligned}
g^q \mod p &= (u^2)^q \mod p \\
&= u^{2\times q} \mod p \\
&= u^{p-1} \mod p \\
&= 1
\end{aligned}
$$

If we need another generator h, we can repeat the same process or

$$h = g^k \mod p \text{ for some arbitrary } k$$

Problem

Both methods are bad but the second one is disastrous if used in a commitment scheme that demands independence of generator.

Swiss Post Public Intrusion Test

Undetectable Attack Against Vote Integrity and Secrecy

1. Manipulation of the Election Result:
   Instead of re-encrypting the shuffled encrypted votes, the malicious mixing component manipulates the input encryption list in an arbitrary way, for example such that the preferred candidate wins the election. Then a fake proof is constructed, which links the manipulated output encryption list to the correct input encryption list. The verification of the proof will succeed and the attack remains undetected.

Solution (FIPS 186-4 and ROCQ)

1: **procedure** VERIFIABLE-GENERATOR-PROCEDURE($p, q, domain\_parameter\_seed, index$)
2: **Result**: *status*, g
3: **if** *index* is incorrect **then** return **INVALID**
4: **end if**
5: $N = \textbf{len}(q)$
6: $k = (p-1)/q$
7: $count = 0$
8: $count = count + 1$
9: **if** $count = 0$ **then** return **INVALID**
10: **end if**
11: $U = domain\_parameter\_seed \,||\, \text{“ggen”} \,||\, index \,||\, count$
12: $W = \textbf{Hash}(U)$
13: $g = W^k \bmod p$
14: **if** $g < 2$ **then**, go to step 8
15: **end if**
16: Return **VALID** and the value of g
17: **end procedure**

ROCQ

A trustworthy, industrial-strength interactive theorem prover and dependently-typed programming language for mechanised reasoning in mathematics, computer science and more.

- Implementation of A.2.3 with a correctness proof that it always produces correct generators.
- Implementation of A.2.4 to validate generators computed according to A.2.3.
- Implementation of the SHA-256 (FIPS 180-4) hash algorithm, required in A.2.3 and A.2.4.
- Implementation of the Fermat's little theorem, needed to prove the correctness of A.2.3 and A.2.4.

```
(* proof that it generates a correct generator *)
(* g is the generator of order q *)
(* g ^ q mod p = 1 *)
Lemma correct_compute_gen : forall n m g, compute_gen_fast n m = Valid g ->
  Zpow_mod (Z.of_N g) (Z.of_N q) (Z.of_N p) = 1%Z.
Proof.
```

```
Definition sha256_string (s : string) :=
  concat_bytes (List.flat_map big_endien_list_N (sha256 (list_byte_of_string s))).


Lemma sha256_4th : sha256_string ("abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghij" ++
 "klmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu") =
  0xcf5b16a778af8380036ce59e7b0492370b249b11e8f07a51afac45037afee9d1.
Proof.
  Time vm_compute; reflexivity.
Qed.
```

```
generated by using "domain_parameter_seed" *)
Module Type Prime.
  Parameters (p q k domain_parameter_seed ggen index : N).
  Axiom prime_p : prime (Z.of_N p).
  Axiom p_len : 1024 <= N.size p.
  Axiom prime_q : prime (Z.of_N q).
  Axiom q_len : 160 <= N.size q.
  Axiom k_gteq_2 : 2 <= k.
  Axiom safe_prime : p = k * q + 1.
  Axiom ggen_hyp : ggen = 0x6767656e.
  Axiom index_8bit : 0 <= index < 0xff.
End Prime.
```

https://github.com/mukeshtiwari/Formally_Verified_Verifiable_Group_Generator

```
Module Pins <: Prime.
Definition p : N :=
0xff600483db6abfc5b45eab78594b3533d550d9f1bf2a992a7a8daa6dc34f8045ad4e6e0c429d334eeeaaefd7
Definition q : N := 0xe21e04f911d1ed7991008ecaab3bf775984309c3.
Definition k : N := Eval vm_compute in N.div (p - 1) q.
Definition domain_parameter_seed : N :=  0x180180ee2f0ae4a7b3a1ab1b8414228913ef2911.
Definition ggen : N := 0x6767656e.
Definition index : N := 0x79.
```

```
(* get a concrete instance *)
Module genr := Comp Pins.
```

```
Time Eval vm_compute in genr.compute_generator.
```

Experiment

```
COQC src/Generator_1.v
Finished transaction in 51.48 secs (51.393u,0.088s) (successful)
     = genr.Valid
         13845410670820047905573533543088887716582050631568180852981258877489129426674596593
     : genr.Tag
```

```
COQC src/Generator_12.v
Finished transaction in 1417.487 secs (1415.836u,1.591s) (successful)
     = genr.Valid
         514226904661129875105333447171201838095238165999414906106964331078086341111775683038
     : genr.Tag
```

Conclusion

- Implementation of A.2.3 with a correctness proof that it always produces correct generators.
- Implementation of A.2.4 to validate generators computed according to A.2.3.
- Implementation of the SHA-256 (FIPS 180-4) hash algorithm, required in A.2.3 and A.2.4.
- Implementation of the Fermat's little theorem, needed to prove the correctness of A.2.3 and A.2.4.