Some analysis of sVote, the Swiss Post/Scytl Internet Voting Protocol and some of the implications for New South Wales, Australia

Vanessa Teague, University of Melbourne
Joint work wih Olivier Pereira and Sarah Jamie Lewis

MSR August '19

The Swiss Post/Scytl Internet Voting System

Main goals:

- Internet Voting
- Supported by voting cards delivered by Post to voters

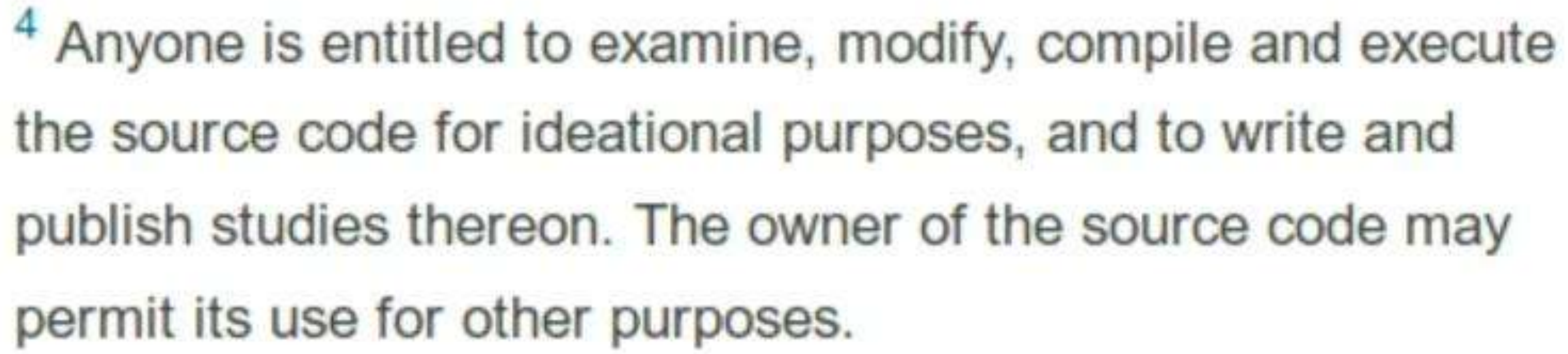
Intended for use by up to 100% of population after public review and pen test

Security requirements ($\approx$):

- Voting client trusted for privacy, not for integrity
- Server side operations verifiable, assuming at least one honest entity: "complete verifiability"
    - Actually the "printing service" is trusted for integrity. It also generates the keys.
    - Also the link between the votes that are received and the ones that are mixed/counted is tenuous.

The Review Process

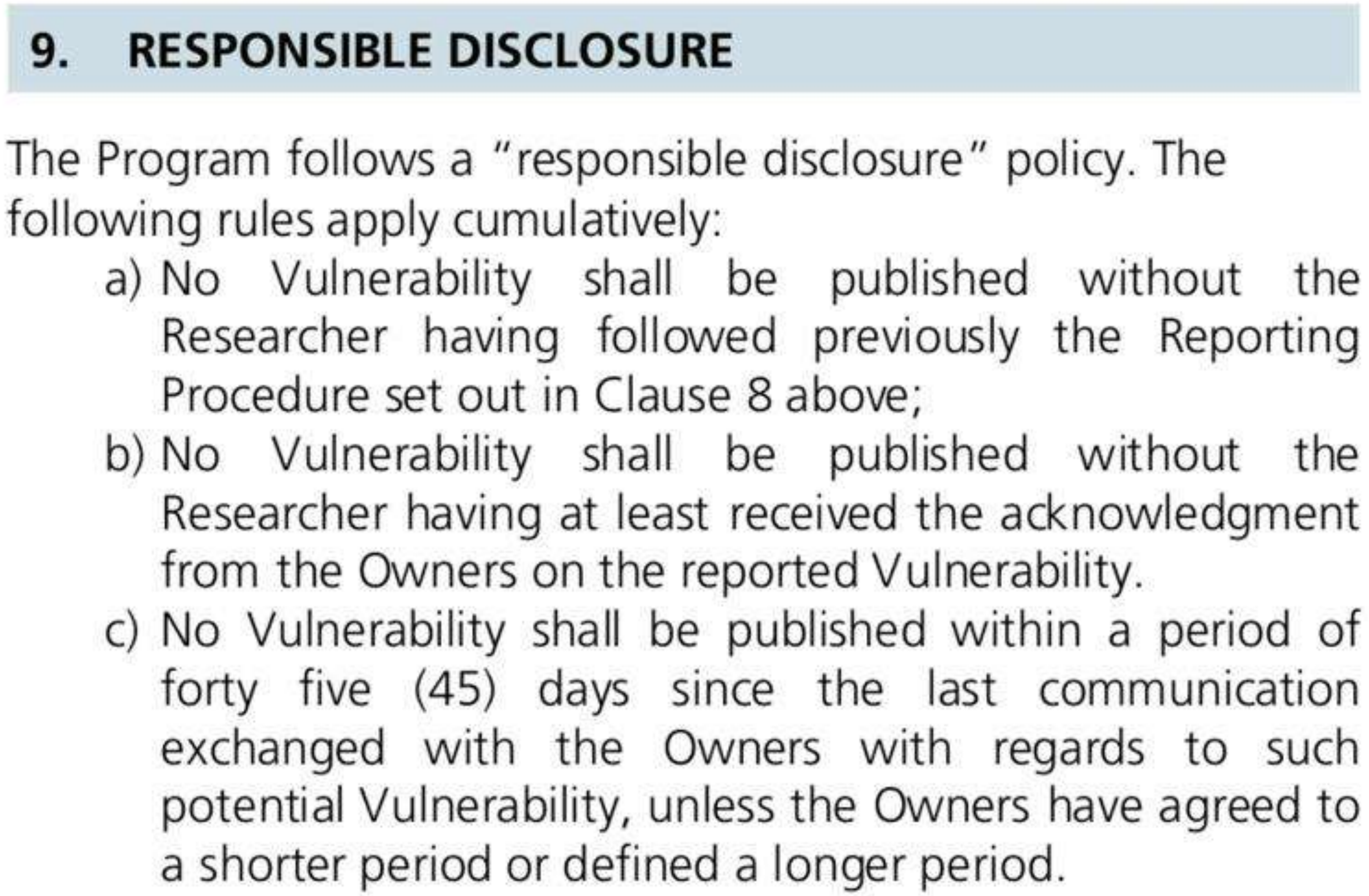
Swiss Chancellery Ordinance 161.116, Art. 7b:



[4] Anyone is entitled to examine, modify, compile and execute the source code for ideational purposes, and to write and publish studies thereon. The owner of the source code may permit its use for other purposes.

The Review Process

A twist in the Swiss Post “conditions of use”:



9. RESPONSIBLE DISCLOSURE

The Program follows a “responsible disclosure” policy. The following rules apply cumulatively:

a) No Vulnerability shall be published without the Researcher having followed previously the Reporting Procedure set out in Clause 8 above;

b) No Vulnerability shall be published without the Researcher having at least received the acknowledgment from the Owners on the reported Vulnerability.

c) No Vulnerability shall be published within a period of forty five (45) days since the last communication exchanged with the Owners with regards to such potential Vulnerability, unless the Owners have agreed to a shorter period or defined a longer period.

The Review Process

The code leaked:

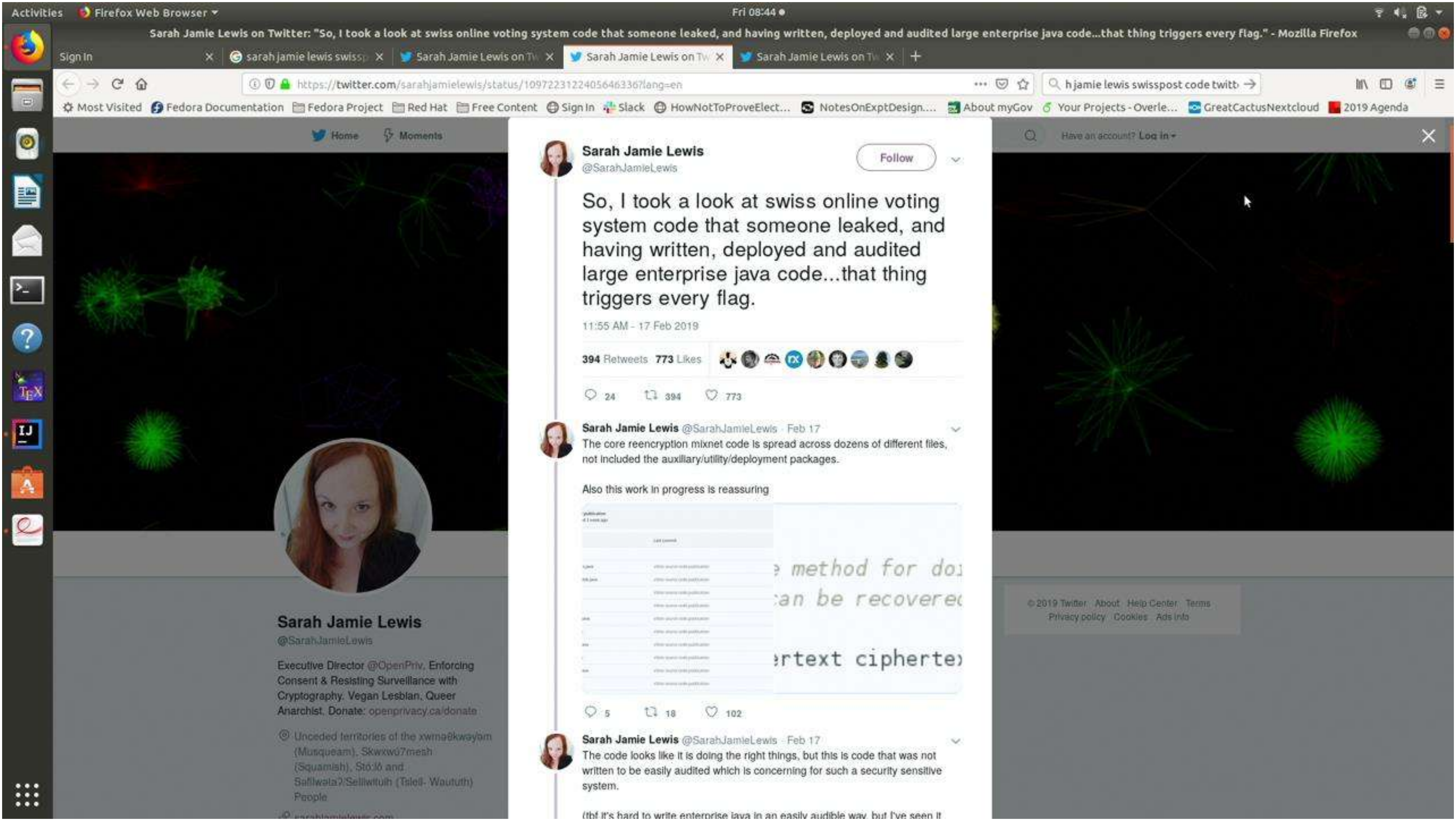
`https://gitlab.com/fickdiepost/`

(Repository taken down after a few days. . .)

Taking a first look

At Financial Crypto'19:

- Would any IDE be able to open this code?
- Let's open these `pdf` documentation files?


Activities
Firefox Web Browser
Fri 08:44
Sarah Jamie Lewis on Twitter: "So, I took a look at swiss online voting system code that someone leaked, and having written, deployed and audited large enterprise java code...that thing triggers every flag." - Mozilla Firefox
Sign In
sarah jamie lewis swissp
Sarah Jamie Lewis on Tw
Sarah Jamie Lewis on Tw
Sarah Jamie Lewis on Tw
https://twitter.com/sarahjamielewis/status/1097223122405646336?lang=en
h jamie lewis swisspost code twitt
Most Visited
Fedora Documentation
Fedora Project
Red Hat
Free Content
Sign In
Slack
HowNotToProveElect...
NotesOnExptDesign....
About myGov
Your Projects - Overle...
GreatCactusNextcloud
2019 Agenda
Home
Moments
Have an account? Log in
Follow
© 2019 Twitter About Help Center Terms Privacy policy Cookies Ads info

Sarah Jamie Lewis
@SarahJamieLewis

So, I took a look at swiss online voting system code that someone leaked, and having written, deployed and audited large enterprise java code...that thing triggers every flag.

11:55 AM - 17 Feb 2019

394 Retweets 773 Likes

24 394 773

Sarah Jamie Lewis @SarahJamieLewis · Feb 17
The core reencryption mixnet code is spread across dozens of different files, not included the auxiliary/utility/deployment packages.

Also this work in progress is reassuring

```
e method for do
can be recovered

ertext cipherte
```

5 18 102

Sarah Jamie Lewis @SarahJamieLewis · Feb 17
The code looks like it is doing the right things, but this is code that was not written to be easily audited which is concerning for such a security sensitive system.

(tbf it's hard to write enterprise java in an easily audible way, but I've seen it

Sarah Jamie Lewis
@SarahJamieLewis

Executive Director @OpenPriv. Enforcing Consent & Resisting Surveillance with Cryptography. Vegan Lesbian. Queer Anarchist. Donate: openprivacy.ca/donate

Unceded territories of the xwməθkwəyəm (Musqueam), Skwxwú7mesh (Squamish), Stó:lō and Səlilwətaɬ/Selilwitulh (Tsleil- Waututh) People

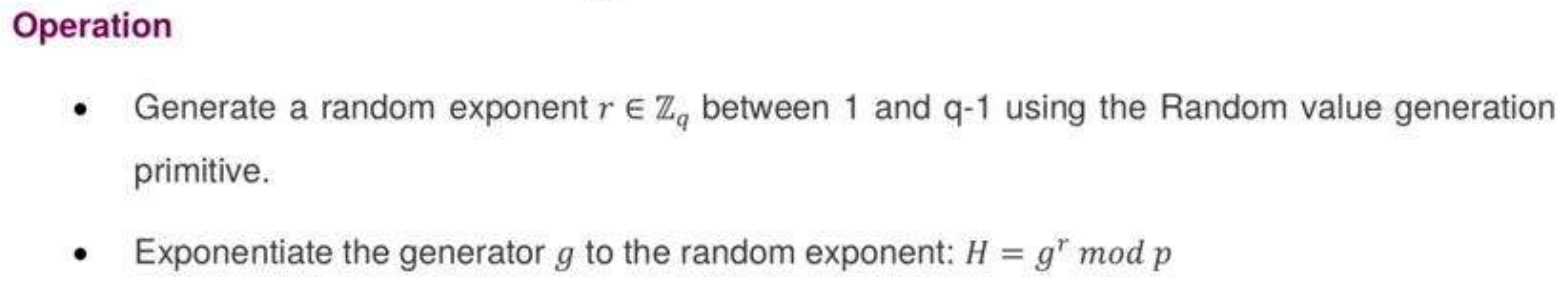
sarahjamielewis.com

Taking a first look

At Financial Crypto'19:

- Would any IDE be able to open this code?
- Let's open these `pdf` documentation files?

Wait!? What is this procedure for Pedersen commitment key generation????



Operation

- Generate a random exponent $r \in \mathbb{Z}_q$ between 1 and q-1 using the Random value generation primitive.
- Exponentiate the generator g to the random exponent: $H = g^r \ mod \ p$

Pedersen Commitments

Pedersen commitments in a group G:

- Take a public key pk made of random generators g, h
- $Commit_{pk}(m)$ computed as $c = g^r h^m$ for random r

- Perfectly hiding: g^r completely hides m
- Computationally binding if DL of h in base g is unknown
  If $h = g^x$ for a known x then:

$$c = g^{r_1} h^{m_1} = g^{r_1 + xm_1} = g^{(r_1 + xm_1 - xm_2) + xm_2} = g^{r_1 + xm_1 - xm_2} h^{m_2}$$

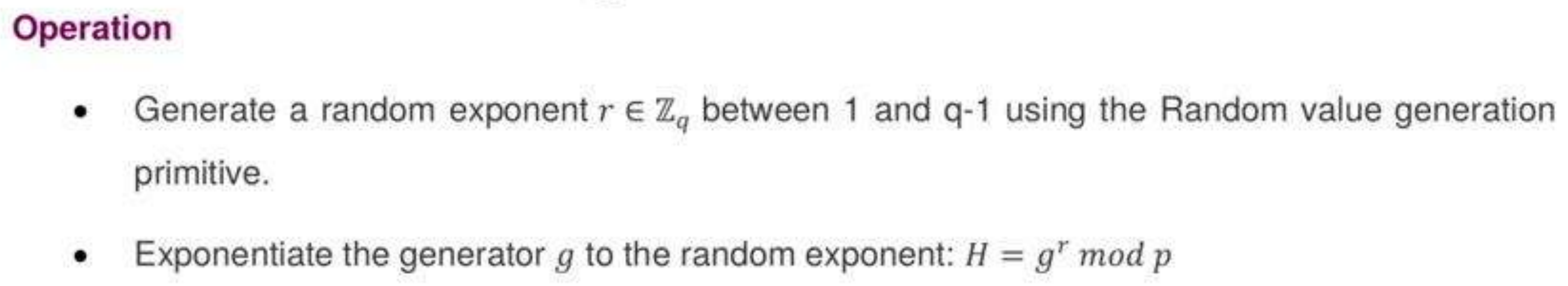
So, x is a trapdoor that makes it possible to re-open a commitment anyway I like

This is why Pedersen commitments are called:

Trapdoor Commitments

Taking a first look

Wait!? What is this procedure for Pedersen commitment key generation????



Operation

- Generate a random exponent $r \in \mathbb{Z}_q$ between 1 and q-1 using the Random value generation primitive.
- Exponentiate the generator g to the random exponent: $H = g^r \; mod \; p$

Why is Scytl actually generating the trapdoor???

- How do we know that no-one keeps track of r?

Standard/correct way of picking generators: $g = \mathscr{H}(1)$, $h = \mathscr{H}(2)$ where $\mathscr{H}$ is a random oracle pointing to group generators

OP's first reaction

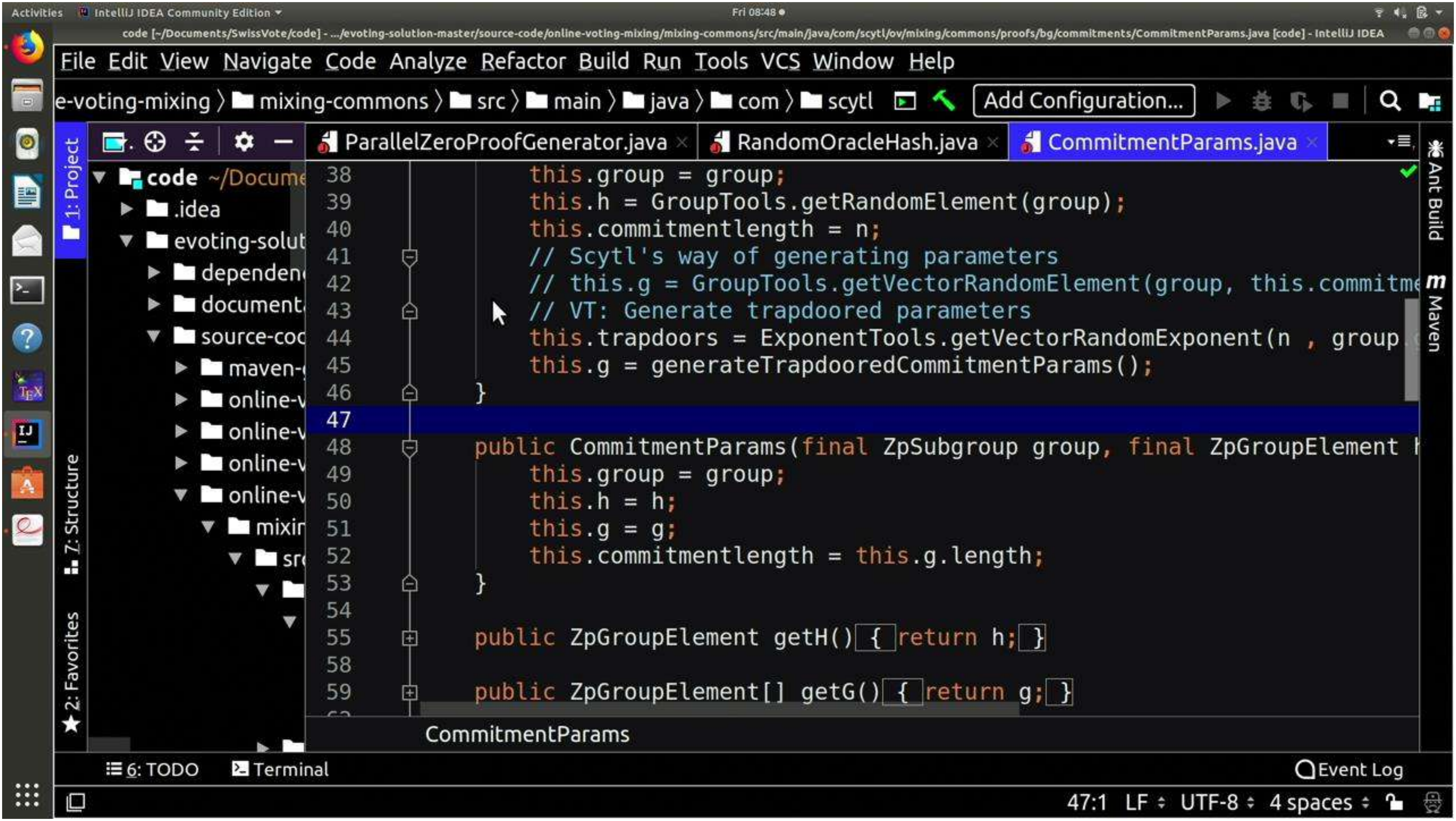
This is not possible:

- They implemented this huge sophisticated system, they surely understand what a Pedersen commitment is
- The system has been through several independent reviews
- Several security proofs have been published
- This must be a bug in the doc, but the code certainly does something different

OP's first reaction

This is not possible:

- They implemented this huge sophisticated system, they surely understand what a Pedersen commitment is
- The system has been through several independent reviews
- Several security proofs have been published
- This must be a bug in the doc, but the code certainly does something different
- VT's first reaction: I've met these people before. Let's look at the code.


Activities
IntelliJ IDEA Community Edition
Fri 08:48
code [~/Documents/SwissVote/code] - .../evoting-solution-master/source-code/online-voting-mixing/mixing-commons/src/main/java/com/scytl/ov/mixing/commons/proofs/bg/commitments/CommitmentParams.java [code] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
e-voting-mixing 〉 mixing-commons 〉 src 〉 main 〉 java 〉 com 〉 scytl
Add Configuration...
ParallelZeroProofGenerator.java
RandomOracleHash.java
CommitmentParams.java
1: Project
code ~/Docume
.idea
evoting-solut
dependen
document
source-coc
maven-
online-v
online-v
online-v
online-v
mixir
src
7: Structure
2: Favorites
Ant Build
Maven
CommitmentParams
6: TODO
Terminal
Event Log
47:1 LF UTF-8 4 spaces

```java
        this.group = group;
        this.h = GroupTools.getRandomElement(group);
        this.commitmentlength = n;
        // Scytl's way of generating parameters
        // this.g = GroupTools.getVectorRandomElement(group, this.commitme
        // VT: Generate trapdoored parameters
        this.trapdoors = ExponentTools.getVectorRandomExponent(n , group
        this.g = generateTrapdooredCommitmentParams();
    }

    public CommitmentParams(final ZpSubgroup group, final ZpGroupElement h
        this.group = group;
        this.h = h;
        this.g = g;
        this.commitmentlength = this.g.length;
    }

    public ZpGroupElement getH() { return h; }

    public ZpGroupElement[] getG() { return g; }
```

Testing an exploit

Let's build a basic exploit from this, code it, and see if it passes their verifier

- A few hours to fully specify an exploit on the Bayer-Groth mixnet
  Open a permutation commitment to a non-permutation
  Use this to change as many votes as desired (provided we have their randomness)
- 4 days to have it running
  - $\approx$ 20 lines of code modified
  - Need to go through dozens of files to understand any tech detail

What do we communicate about this?

1. Summary
   - Error gives a trapdoor in the system, breaking verifiability, a core requirement by the Chancellery
   - Exploit of the trapdoor is undetectable
   - Is it on purpose? (good cover-up as a mistake) Just total misunderstanding of basic crypto? Or total negligence?

What do we communicate about this?

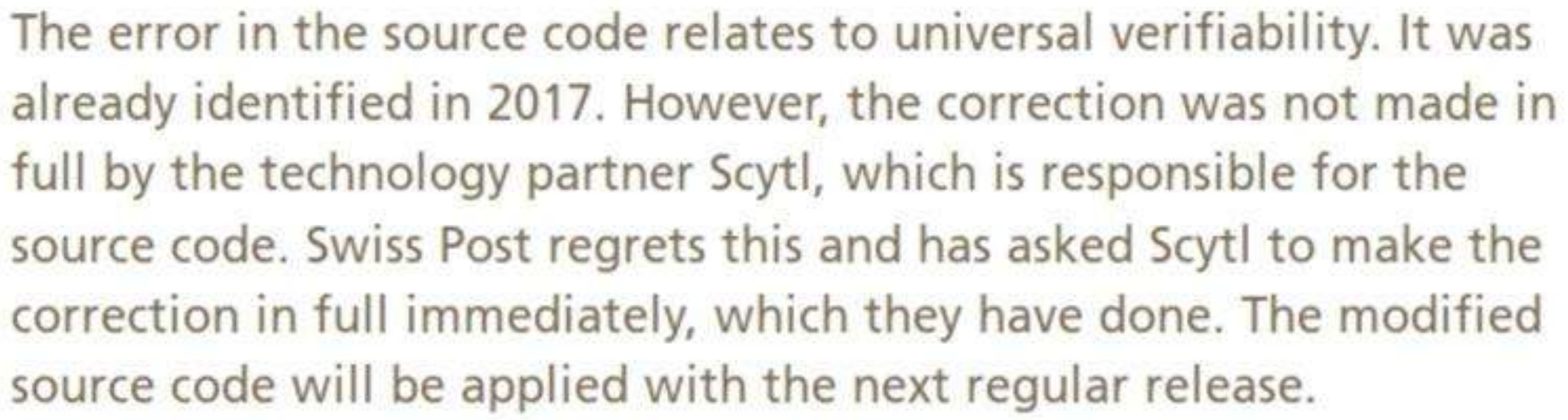
1. Summary
   - Error gives a trapdoor in the system, breaking verifiability, a core requirement by the Chancellery
   - Exploit of the trapdoor is undetectable
   - Is it on purpose? (good cover-up as a mistake) Just total misunderstanding of basic crypto? Or total negligence?
2. We wrote

   Nothing in our analysis suggests that this problem was introduced deliberately. It is entirely consistent with a naive implementation of a complex cryptographic protocol by well-intentioned people who lacked a full understanding of its security assumptions and other important details. Of course, if someone did want to introduce an opportunity for manipulation, the best method would be one that could be explained away as an accident if it was found.

Swiss Post's Press release

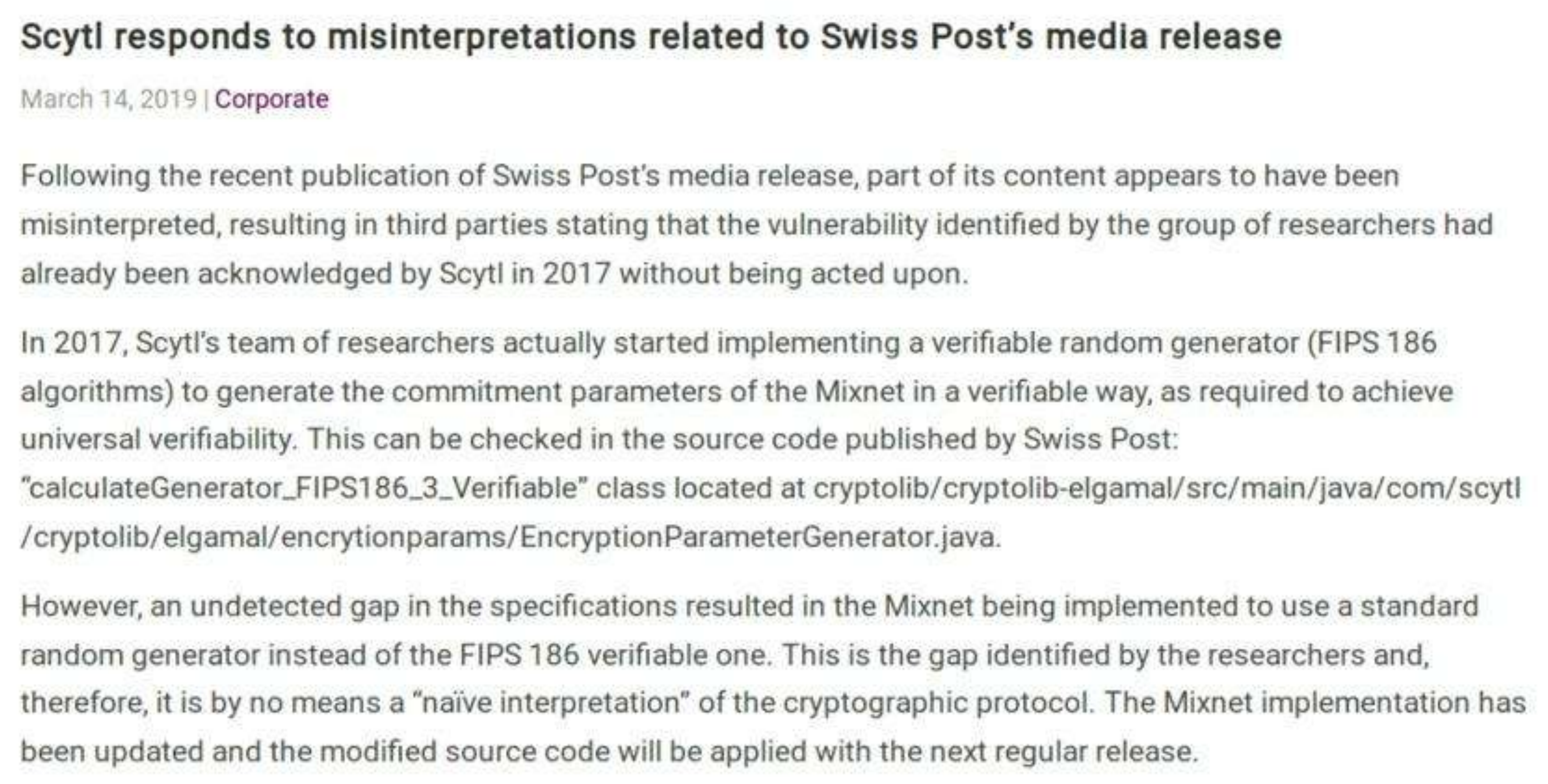
What we learned:

- During the exploit review process at Swiss Post, the incorrect generator selection problem had been pointed by an anonymous researcher (Thomas Haines) and Rolf Haenni
  We quickly acknowledged them in our report.
- Scytl knew about the problem:



The error in the source code relates to universal verifiability. It was already identified in 2017. However, the correction was not made in full by the technology partner Scytl, which is responsible for the source code. Swiss Post regrets this and has asked Scytl to make the correction in full immediately, which they have done. The modified source code will be applied with the next regular release.

Scytl's communication



Scytl responds to misinterpretations related to Swiss Post's media release

March 14, 2019 | Corporate

Following the recent publication of Swiss Post's media release, part of its content appears to have been misinterpreted, resulting in third parties stating that the vulnerability identified by the group of researchers had already been acknowledged by Scytl in 2017 without being acted upon.

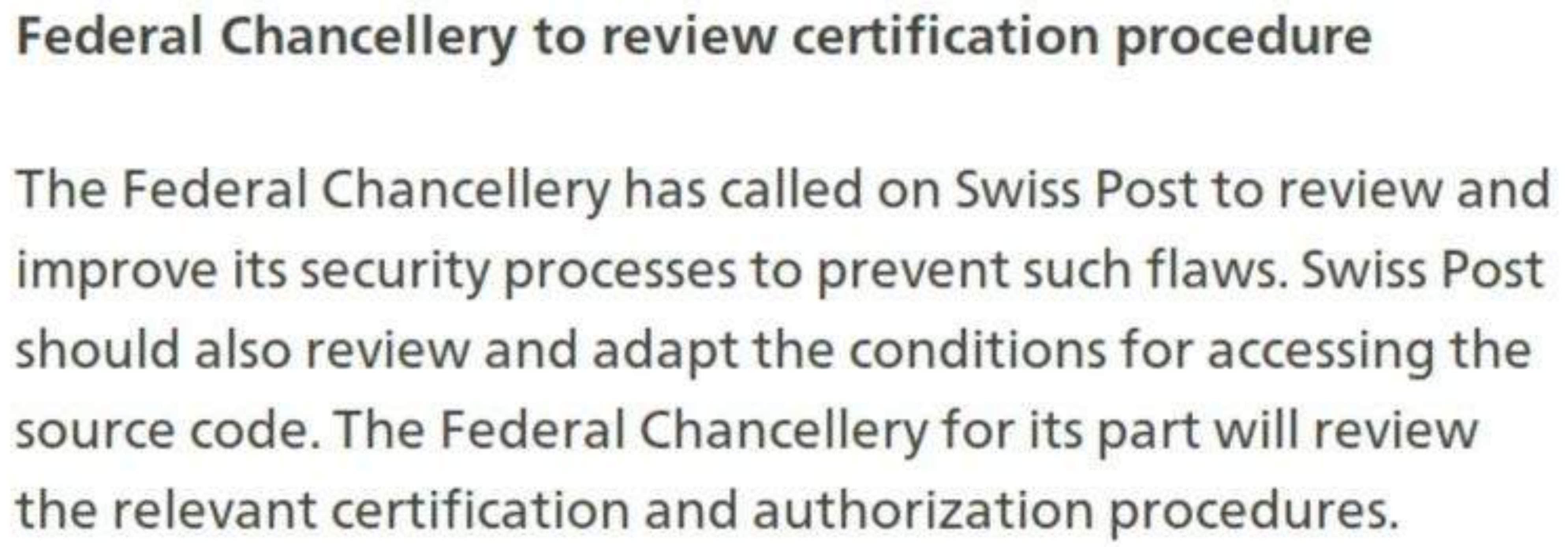
In 2017, Scytl's team of researchers actually started implementing a verifiable random generator (FIPS 186 algorithms) to generate the commitment parameters of the Mixnet in a verifiable way, as required to achieve universal verifiability. This can be checked in the source code published by Swiss Post: "calculateGenerator_FIPS186_3_Verifiable" class located at cryptolib/cryptolib-elgamal/src/main/java/com/scytl /cryptolib/elgamal/encrytionparams/EncryptionParameterGenerator.java.

However, an undetected gap in the specifications resulted in the Mixnet being implemented to use a standard random generator instead of the FIPS 186 verifiable one. This is the gap identified by the researchers and, therefore, it is by no means a "naïve interpretation" of the cryptographic protocol. The Mixnet implementation has been updated and the modified source code will be applied with the next regular release.

Swiss Chancellery's Press release

What we learned:

- Chancellery acknowledges that this issue should not have been there

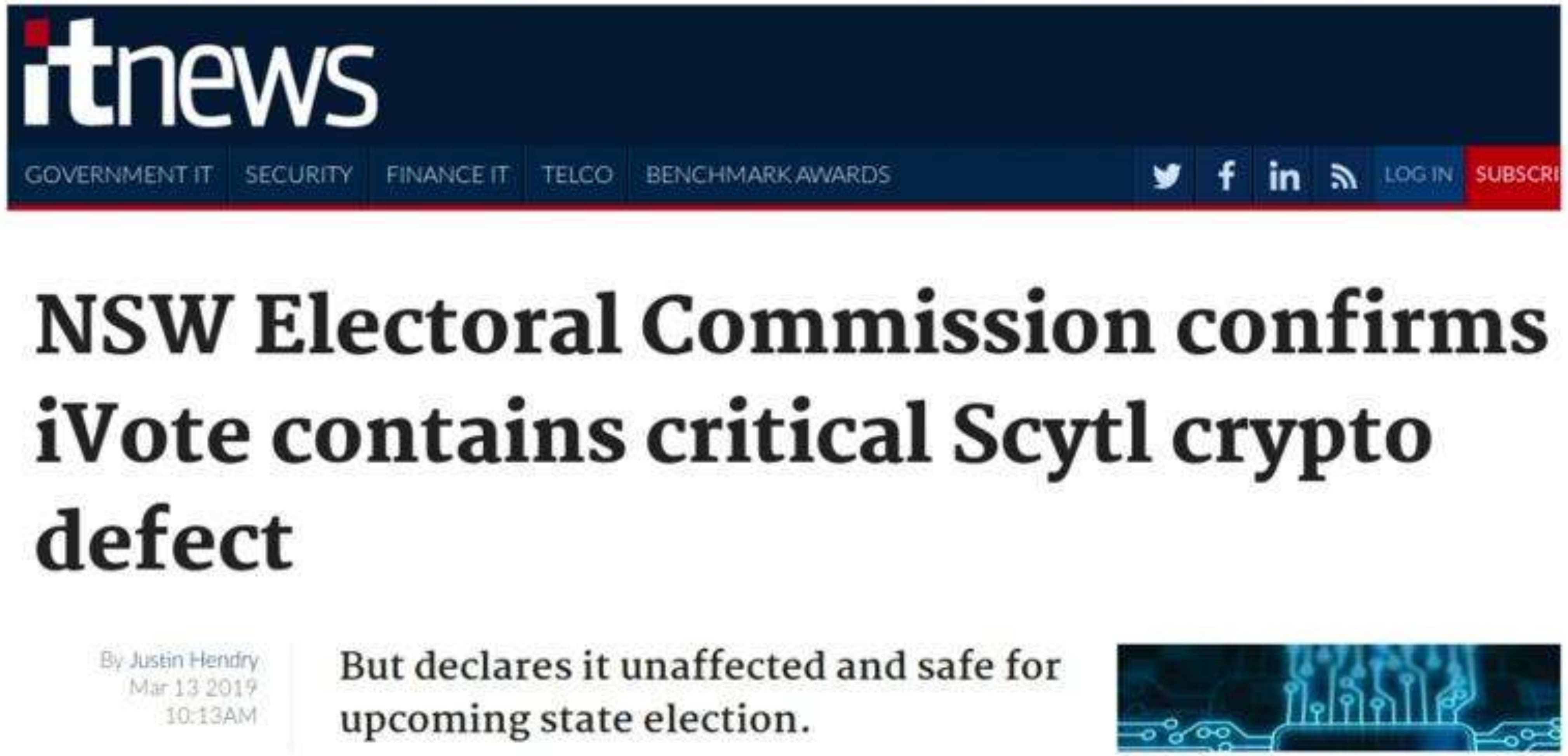


Federal Chancellery to review certification procedure

The Federal Chancellery has called on Swiss Post to review and improve its security processes to prevent such flaws. Swiss Post should also review and adapt the conditions for accessing the source code. The Federal Chancellery for its part will review the relevant certification and authorization procedures.

Phew. Good thing they had some genuine independent review before they used it

Phew. Good thing they had some genuine independent review before they used it



NSW Electoral Commission confirms iVote contains critical Scytl crypto defect

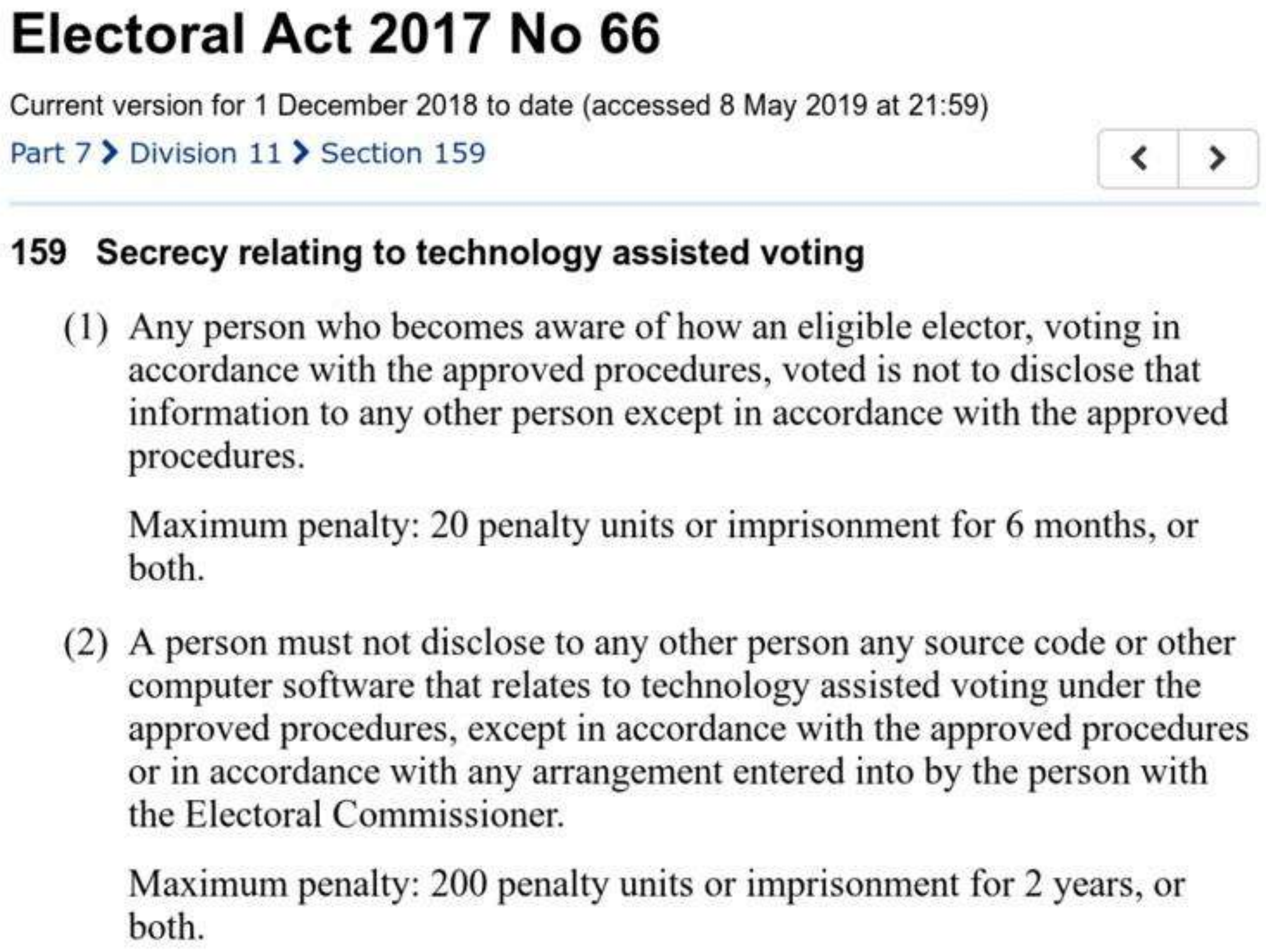
By Justin Hendry
Mar 13 2019
10:13AM

But declares it unaffected and safe for upcoming state election.

NSW was already using it for early voting. Decryption March 23rd.

But doesn't NSW have a law mandating open public access to the source code?

But doesn't NSW have a law mandating open public access to the source code?



Electoral Act 2017 No 66

Current version for 1 December 2018 to date (accessed 8 May 2019 at 21:59)

Part 7 > Division 11 > Section 159

159 Secrecy relating to technology assisted voting

(1) Any person who becomes aware of how an eligible elector, voting in accordance with the approved procedures, voted is not to disclose that information to any other person except in accordance with the approved procedures.

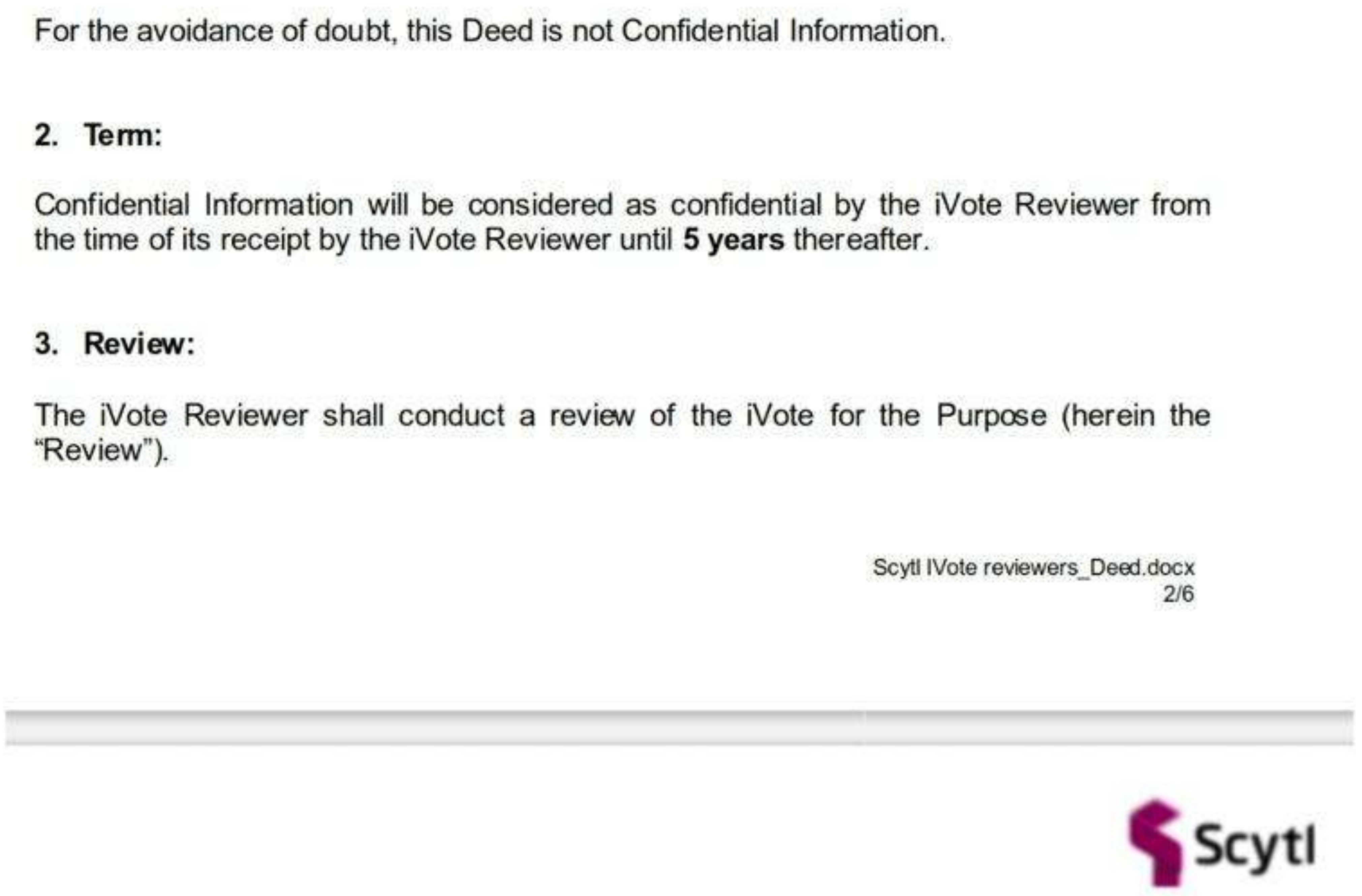
Maximum penalty: 20 penalty units or imprisonment for 6 months, or both.

(2) A person must not disclose to any other person any source code or other computer software that relates to technology assisted voting under the approved procedures, except in accordance with the approved procedures or in accordance with any arrangement entered into by the person with the Electoral Commissioner.

Maximum penalty: 200 penalty units or imprisonment for 2 years, or both.

Or an NDA that allows the public to learn about problems in a reasonable time?

Or an NDA that allows the public to learn about problems in a reasonable time?



For the avoidance of doubt, this Deed is not Confidential Information.

2. Term:

Confidential Information will be considered as confidential by the iVote Reviewer from the time of its receipt by the iVote Reviewer until **5 years** thereafter.

3. Review:

The iVote Reviewer shall conduct a review of the iVote for the Purpose (herein the "Review").

Scytl IVote reviewers_Deed.docx
2/6

Scytl

OK good. Now we've fixed the bug so everything is perfectly secure, right?

OK good. Now we've fixed the bug so everything is perfectly secure, right?

- Actually the shuffle proof is really a shuffle and decryption proof
- Let's have a look at the decryption part

A decryption proof is a Chaum-Pedersen proof of equality of discrete logs

Given generator g and public key $pk = g^x$
To prove that (C_0, C_1') is a valid El Gamal encryption of 1
(i.e. that $C_1' = C_0^x$):

1. Pick a random a.
2. set $B_0 = g^a$ and $B_1 = C_0^a$.
3. **Compute $\mathbf{c} = \mathbf{H}(\mathbf{pk}, \mathbf{C_1'}, \mathbf{B_0}, \mathbf{B_1})$,** where H is a cryptographic hash function.
4. Compute $z = a + cx$.

The proof is (c, z).
Verification: check $B_0 = g^z(pk)^{-c}$, $B_1 = C_0^z(C_1')^{-c}$
and $c = H(pk, C_1', B_0, B_1)$.

A decryption proof is a Chaum-Pedersen proof of equality of discrete logs

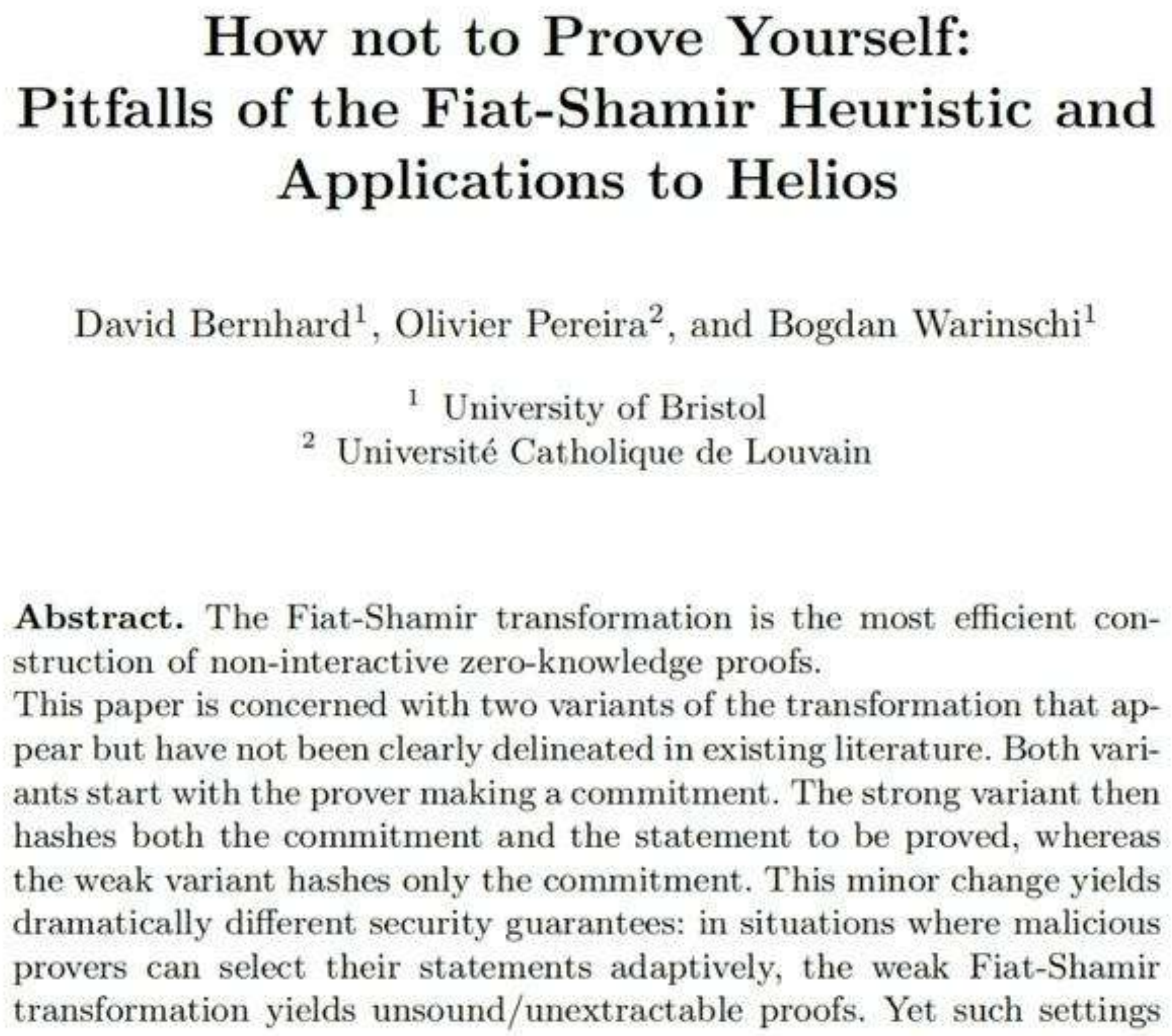
Given generator g and public key $pk = g^x$
To prove that (C_0, C_1') is a valid El Gamal encryption of 1
(i.e. that $C_1' = C_0^x$):

1. Pick a random a.
2. set $B_0 = g^a$ and $B_1 = C_0^a$.
3. **Compute** $\mathbf{c = H(pk, C_1', B_0, B_1)}$**,** where H is a cryptographic hash function.
4. Compute $z = a + cx$.

The proof is (c, z).
Verification: check $B_0 = g^z (pk)^{-c}$, $B_1 = C_0^z (C_1')^{-c}$
and $c = H(pk, C_1', B_0, B_1)$.
The problem is that C_0 is not hashed—it can be made up afterwards.

A decryption proof is a Chaum-Pedersen proof of equality of discrete logs

If only someone had already written a paper explaining how bad this is.



How not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios

David Bernhard[1], Olivier Pereira[2], and Bogdan Warinschi[1]

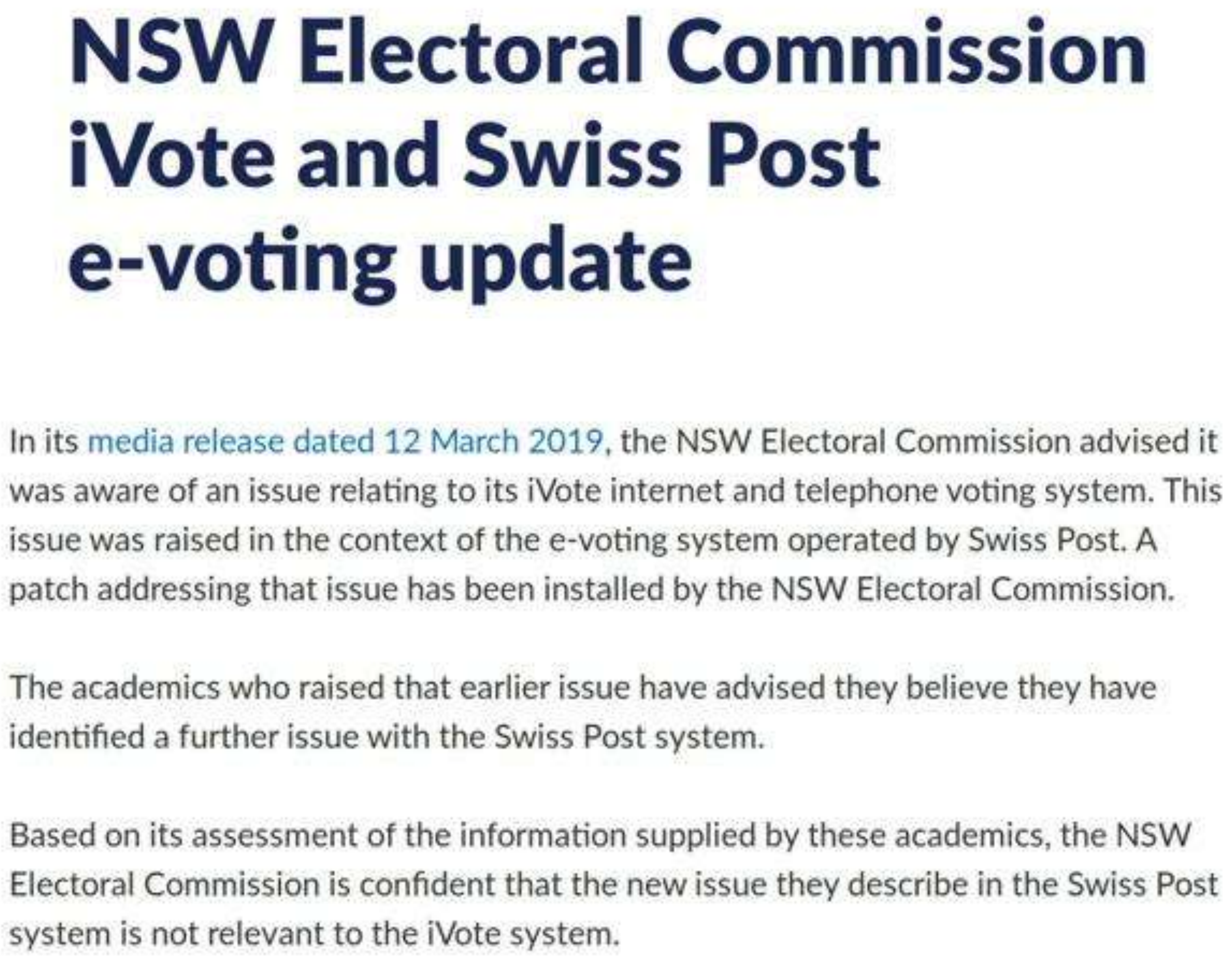
[1] University of Bristol
[2] Université Catholique de Louvain

Abstract. The Fiat-Shamir transformation is the most efficient construction of non-interactive zero-knowledge proofs.
This paper is concerned with two variants of the transformation that appear but have not been clearly delineated in existing literature. Both variants start with the prover making a commitment. The strong variant then hashes both the commitment and the statement to be proved, whereas the weak variant hashes only the commitment. This minor change yields dramatically different security guarantees: in situations where malicious provers can select their statements adaptively, the weak Fiat-Shamir transformation yields unsound/unextractable proofs. Yet such settings

Implications

- A cheating mixer/decrypter can turn a valid vote into nonsense while providing a perfectly verifying (but false) proof that it decrypted properly
- It took us a few days to do the maths and a few more to code the exploit and check that it passed verification
- Informally, it would be obvious something had gone wrong

Never yet used in Switzerland, but used this year in NSW?



NSW Electoral Commission iVote and Swiss Post e-voting update

In its media release dated 12 March 2019, the NSW Electoral Commission advised it was aware of an issue relating to its iVote internet and telephone voting system. This issue was raised in the context of the e-voting system operated by Swiss Post. A patch addressing that issue has been installed by the NSW Electoral Commission.

The academics who raised that earlier issue have advised they believe they have identified a further issue with the Swiss Post system.

Based on its assessment of the information supplied by these academics, the NSW Electoral Commission is confident that the new issue they describe in the Swiss Post system is not relevant to the iVote system.

These proofs are used for cast-as-intended verification too

- A cheating voting client can submit a nonsense vote, but the voter receives the right return codes

Cast-as-intended verification

The voting client submits two items of data:

1. An encrypted vote
$$E_1 = (g^r, \Pi_{i=1}^m v_i \cdot EL^r)$$
where EL is the election public key and $v_1, \ldots, p_m$ are small primes.
2. For each choice v_i $(i = 1, \ldots, \psi)$, it computes a partial choice code
$$pCC_i = v_i^k.$$
It encrypts each pCC_i with a separate element of the multi-element key PK, as
$$E_2 = (g^{r'}, pCC_1 \cdot (PK^{(1)})^{r'}, pCC_2 \cdot (PK^{(2)})^{r'}, \ldots, pCC_m \cdot (PK^{(m)})^{r'}).$$

The partial choice codes are used to compute the return codes. Need to prove they're consistent with the vote, i.e. that
$$\{pCC_i\}_{i=1}^m = \{v_i^k\}_{i=1}^m$$
consistent with public key $K = g^k$.

Proving vote consistency

Remember we have an encrypted vote $E_1 = (g^r, \Pi_{i=1}^m v_i \cdot EL^r)$ and encrypted partial choice codes
$E_2 = (g^{r'}, v_1^k \cdot (PK^{(1)})^{r'}, v_2^k \cdot (PK^{(2)})^{r'}, \ldots, v_m^k \cdot (PK^{(m)})^{r'})$.
To prove consistency:

1. π_s: a Schnorr proof of knowledge of r used in E_1.
2. It computes F_1 as E_1^k, that is,
$$F_1 = (g^{rk}, (\Pi_{i=1}^m v_i \cdot EL^r)^k).$$
3. π_e: a proof of exponentiation, *i.e.*
$$(K, F_1) = (g^k, E_1^k) \text{ for a secret } k \text{ \& public } K$$
4. It multiplies all but the first element of E_2 together to form a standard El Gamal encryption.
$$\tilde{E}_2 = (g^{r'}, \Pi_{i=1}^m PK(i)^{r'} v_i^k).$$
5. π_p: a plaintext equality proof that
F_1 and $\tilde{E}_2$ encrypt the same value (that is, $\Pi_{i=1}^m v_i^k$),
w.r.t. EL in the first case and w.r.t. $\Pi_{i=1}^m PK(i)$ in the second case.

So the client-side ZKPs are simply insufficient

- A cheating client can make all but one of the choice codes right, then fudge the last one to make the product work out.
- *e.g.* cheat on the codes for President & Senator, hope the voter doesn't check the code for dog catcher.

Consequences in Switzerland

- This flaw affects systems that have already been used in Switzerland
- SwissPost decided not to offer their e-voting system in the May elections
- They now say they will fix the bugs and continue development of their universally-verifiable system.

Consequences in NSW?

- NSW doesn't use the Swiss code-return system
- Voters use a closed-source Scytl app to vote...
- and a different closed-source Scytl app to verify

Conclusion

This code should not be trusted.
This can only just be the tip of the iceberg:

- We need more theory to analyze this kind of system
- We need more reviews, as an ongoing process

Not weeks, but years of it.
TLS 1.3: 4.3 years between 1st and final draft.
The SwissPost/Scytl code had been under non-public assessment for years, but within a few weeks of going public was shown to have serious unnoticed errors.

Conclusions

I agree with Aleks.
The greatest risk to democracy is people's inclination to trust without feeling the need to verify.