

---

# SECURITY BY DESIGN IN SWISS POST'S NEW E-VOTING SYSTEM WELCOME TO OUR EXPERT WEBINAR

10 June 2021

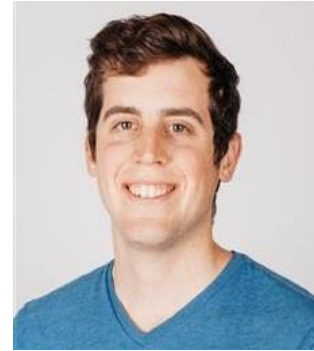
# SPEAKERS



Xavier Monnat  
Product manager e-voting  
Swiss Post



Olivier Esseiva  
Cryptographer  
Swiss Post



Hadrien Renold  
Software engineer  
Swiss Post

# ORGANIZATION

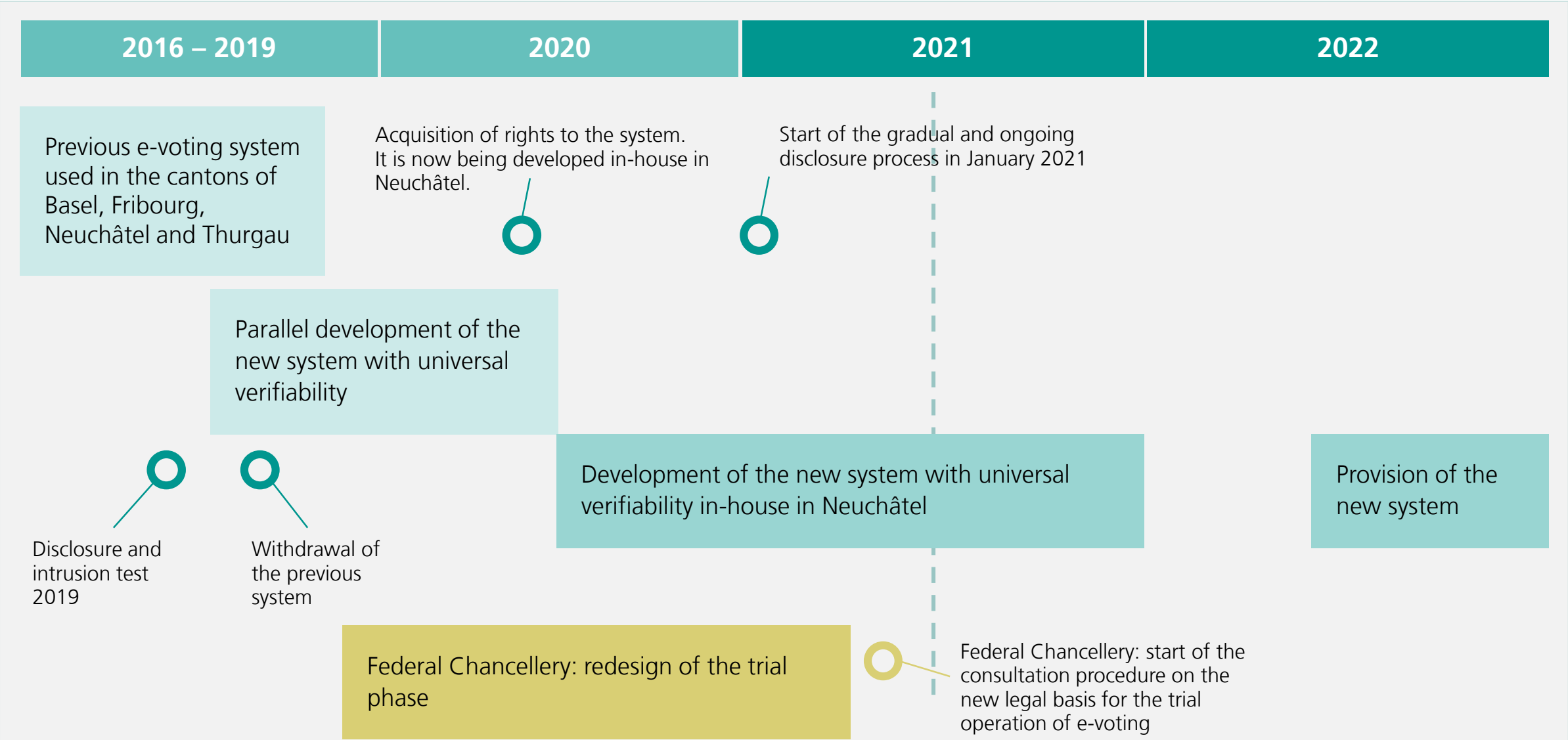
- **Microphone:** Please put your microphone on mute during the presentations.
- **Video:** Please switch your camera on so we can engage in lively discussion despite the virtual event format.
- **Questions:** Please submit your questions in the chat during the presentations or ask them directly in the Q&A session at the end.
- **Recording:** The webinar will be recorded and then made available to you afterwards.



# SWISS POST'S E-VOTING SYSTEM: CURRENT STATUS AND OUTLOOK

Xavier Monnat

# Review and outlook



# System disclosure

## Community programme

- Swiss Post is gradually disclosing the beta version of its new system.
- When preparing the community programme, Swiss Post incorporated feedback from experts on the planned approach.
- Anyone who is interested can take part in this community programme.
- A concise Code of Conduct governs the conditions of participation.
- No registration is required.
- Full information can be found at [www.swisspost.ch/e-voting-community](https://www.swisspost.ch/e-voting-community).

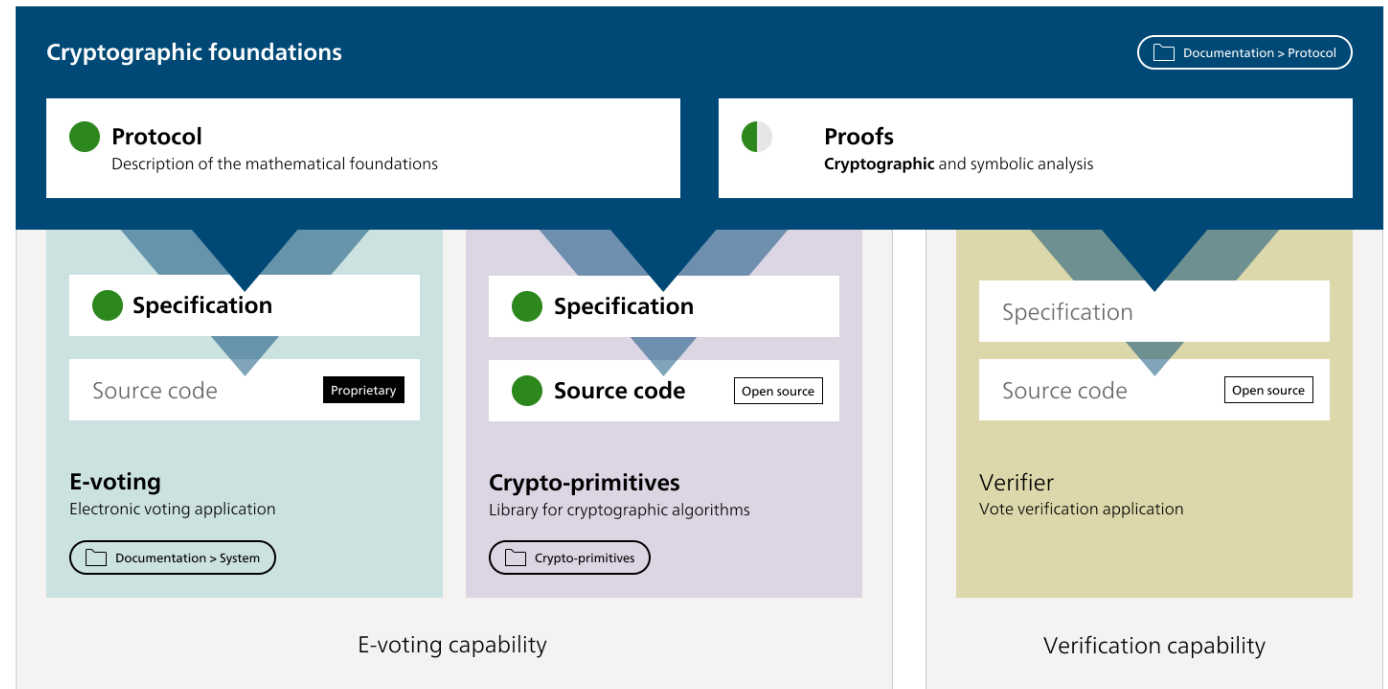
**The aim is to make it as easy as possible for independent experts to access and analyse the system, so that they can test it and report any vulnerabilities to us.**



# System disclosure

## Current status

- The cryptographic protocol, the cryptographic primitives and the specifications have already been published.
- Swiss Post is publishing all reports that it receives on GitLab.
- It is in contact with the experts who have submitted reports.
- Around 20 reports have been received since the start of disclosure.
- Community research programme (BugBounty).

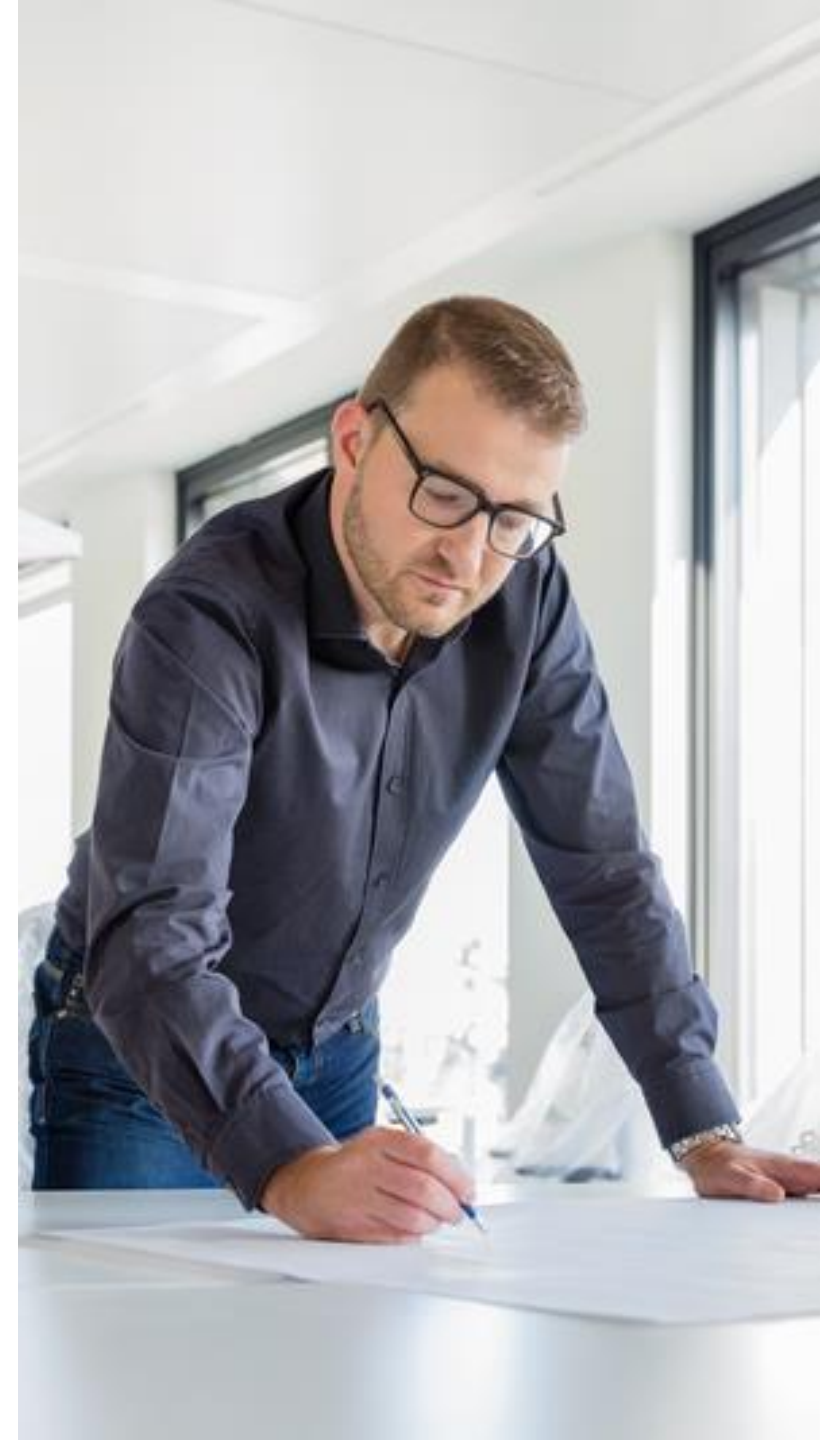


Swiss Post e-voting system

**The community programme is up and running. Swiss Post has made various improvements based on the reports received.**

# What happens next?

- The beta version of the e-voting system will be completely disclosed in a few months.
- The permanent public bug bounty programme for e-voting is also being launched.
- Swiss Post is set to carry out an intrusion test in the autumn.
- **The scope of the community research programme will gradually be expanded.**
- Our aim is to make the system available for use in the cantons during the course of 2022.







# SECURITY BY DESIGN AND PUBLIC SCRUTINY

Olivier Esseiva & Hadrien Renold

# Agenda

- Security by design
  - How do we define a “secure” e-voting system?
  - How do we make sure that the code matches the design?
- Public Scrutiny in practice
  - Current experiences in our public GitLab Repository
  - Open-Source crypto-primitives

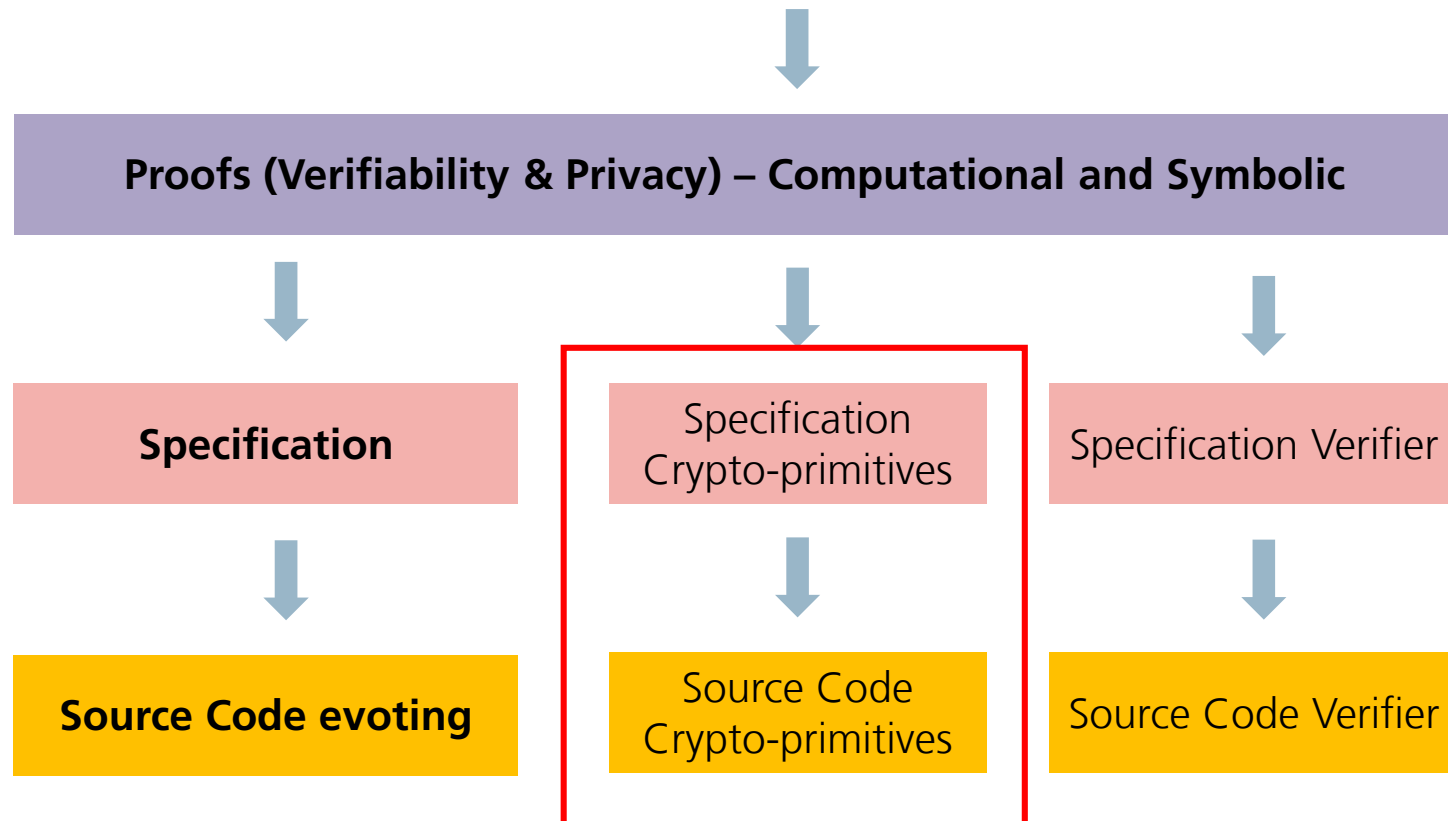


# SECURITY BY DESIGN

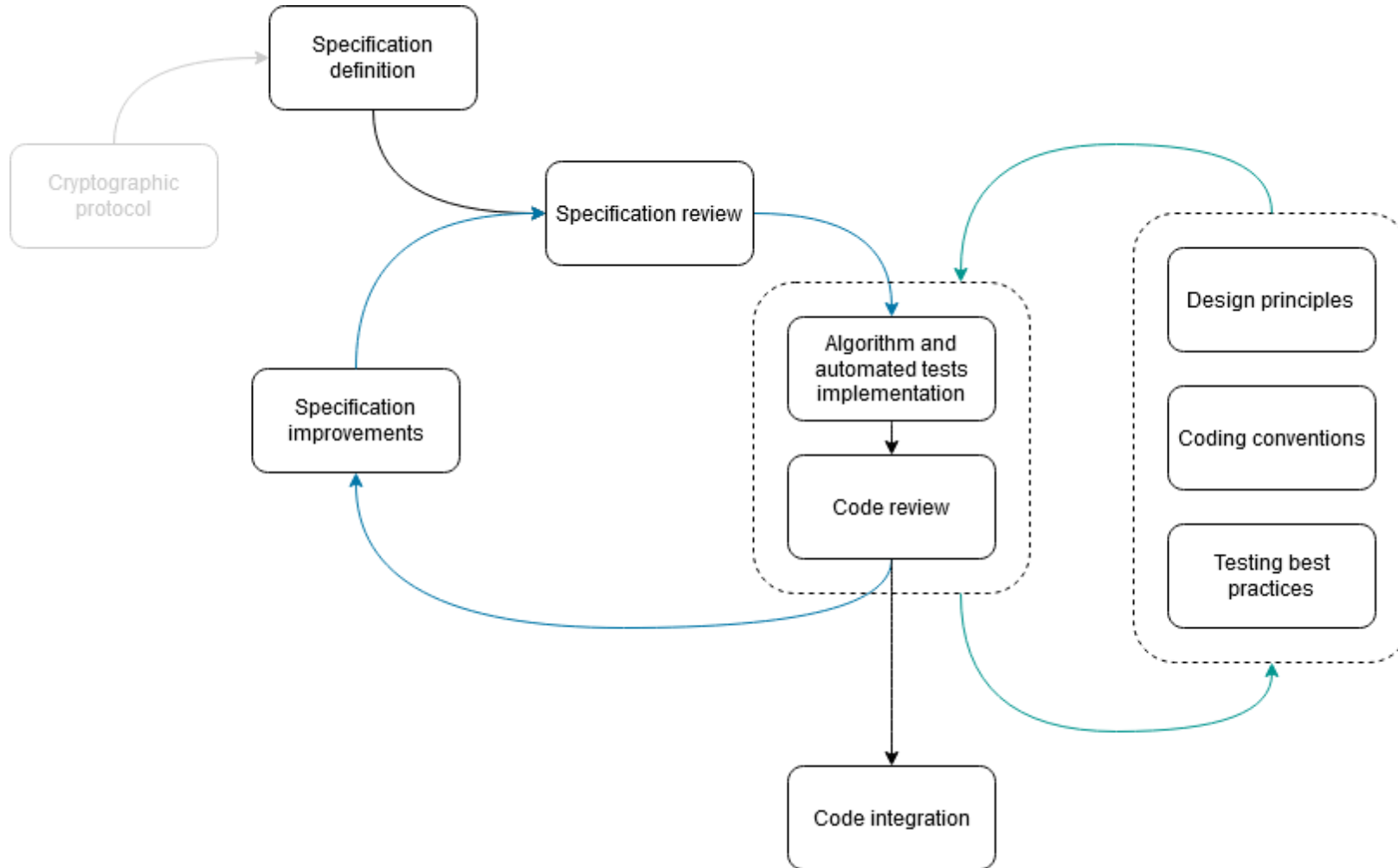
Olivier Esseiva & Hadrien Renold

# Hierarchy of artefacts

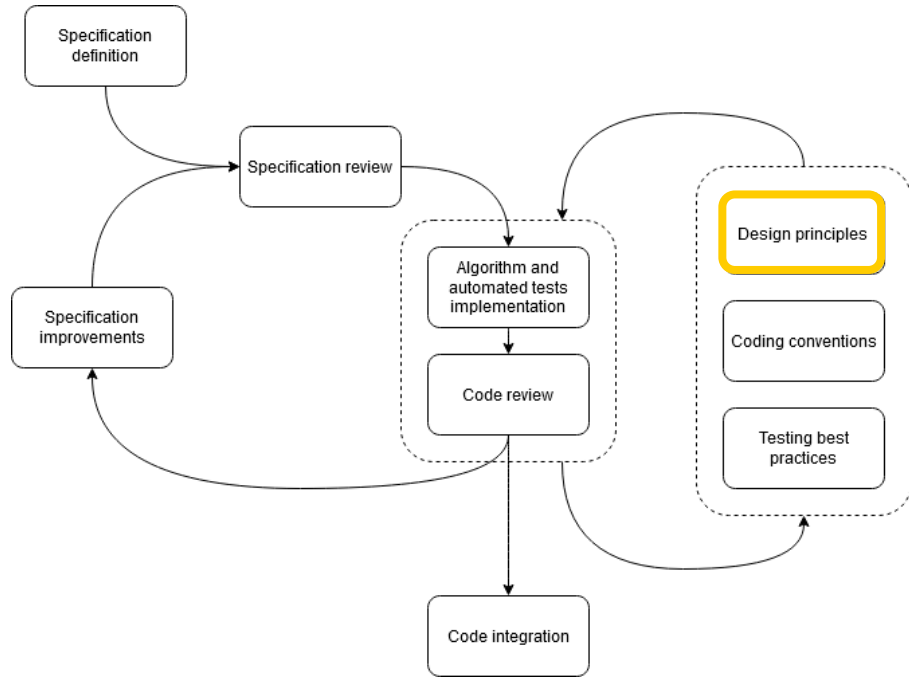
OEV – Federal Chancellery’s Ordinance on Electronic Voting: Precise trust model and security objectives



# Crypto primitives software development process

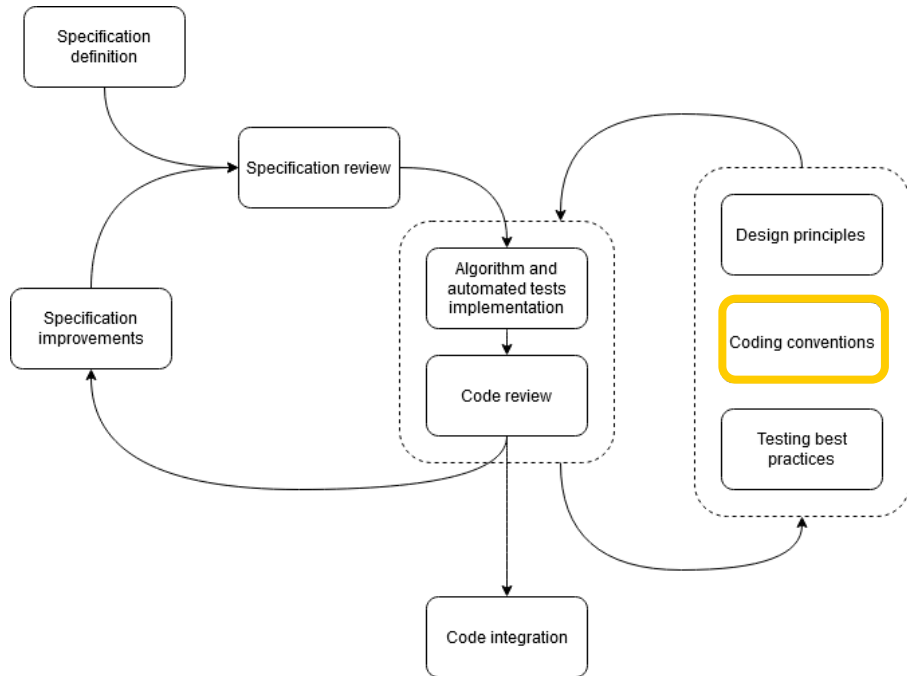


# Design principles



- Auditability
- Easy mapping between specification and code
- Defence in depth
- Misuse-resistant
- Consistency
- Maintainability
- Immutability

# Coding conventions



## Design decisions

Argument checking using *Guava Preconditions*

### Pseudo-code implementation conventions

Created by Renold Hadrien, I226, last modified on May 12, 2021

This page documents the conventions when implementing pseudo-code algorithms from the specification.

- Method names
- Method parameter names
- Domain checks
- One pseudo code operation, one code operation
- Strive for immutability
- Minimize getters
- Favor streaming over for-looping
- Unit tests
- Mathematical variables names
- Javadoc preconditions
- Consistent precondition checks
- Instance versus static methods
- Test Data
- Static imports
- Mocking Accessors

#### Method names

Method names should mirror the pseudo-code algorithm name.

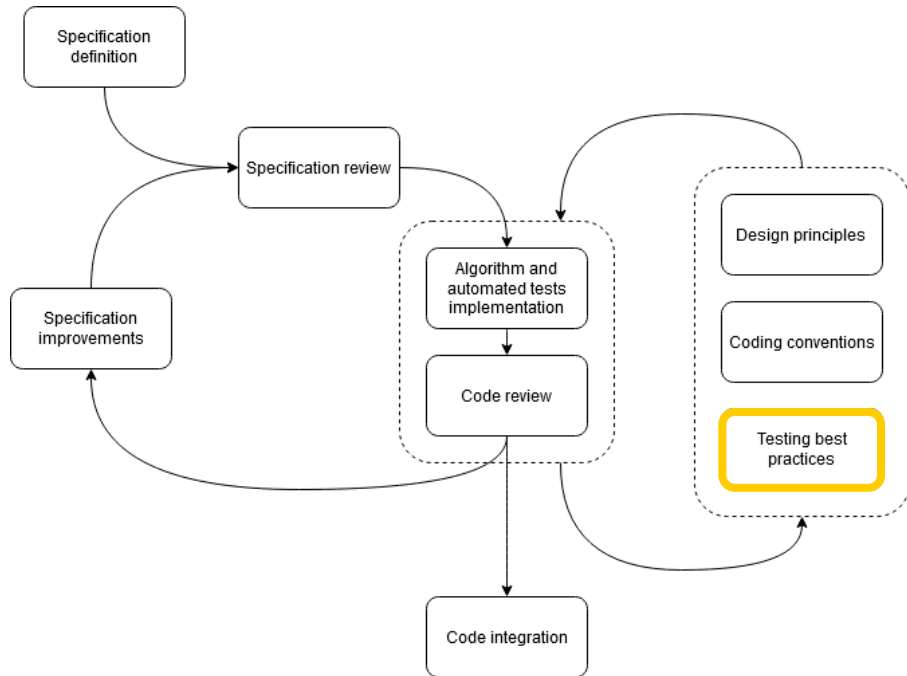
#### Method parameter names

Parameter names should follow Java best practices for naming conventions. When possible try to synthesize the name given in the pseudo-code in a parameter name. Additionally, the Java doc should associate the Java parameter name with the variable name used in the pseudo code.

For example, in the `genRandomIntegerWithinBounds` method, the lower bound argument is named `lowerBound` and the docstring is `@param lowerBound a, inclusive`. where `a` is the name used in the pseudo-code.

```
/**
```

# Testing best practices



Types of tests:

- Unit tests
- Property based tests
- Hand computed values
- Independent implementation test data



# Algorithm implementation Example

Cryptographic Primitives of the Swiss Post Voting System  
Pseudo-code Specification

© 2021 Swiss Post Ltd.  
Version 0.9.4

## Algorithm 5.11 GetShuffleArgument: compute a cryptographic argument for the validity of the shuffle

### Context:

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

### Input:

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_q)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_q)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{\rho} \in \mathbb{Z}_q^N$

- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

Ensure:  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{r}, \rho_i, \mathbf{pk}), \vec{C}_{\pi(i)})$

Ensure:  $N = mn$

### Operation:

- $r \leftarrow \text{GenRandomVector}(q, m)$  ▷ See algorithm 3.2
  - $A \leftarrow \text{Transpose}(\text{ToMatrix}(\{\pi(i)\}_{i=0}^{N-1}, m, n))$  ▷ Create a  $n \times m$  matrix. See algorithm 5.14 and algorithm 5.13
  - $c_A \leftarrow \text{GetCommitmentMatrix}(A, r, \mathbf{ck})$  ▷ See algorithm 5.8
  - $x \leftarrow \text{ByteArrayToInteger}(\text{RecursiveHash}(p, q, \mathbf{pk}, \mathbf{ck}, \vec{C}, \vec{C}', c_A))$
  - $s \leftarrow \text{GenRandomVector}(q, m)$
  - $b \leftarrow \{x^{\pi(i)}\}_{i=0}^{N-1}$
  - $B \leftarrow \text{Transpose}(\text{ToMatrix}(b, m, n))$
  - $c_B \leftarrow \text{GetCommitmentMatrix}(B, s, \mathbf{ck})$
  - $y \leftarrow \text{ByteArrayToInteger}(\text{RecursiveHash}(c_B, p, q, \mathbf{pk}, \mathbf{ck}, \vec{C}, \vec{C}', c_A))$
  - $z \leftarrow \text{ByteArrayToInteger}(\text{RecursiveHash}("1", c_B, p, q, \mathbf{pk}, \mathbf{ck}, \vec{C}, \vec{C}', c_A))$
- ▷ Both  $\vec{C}$  and  $\vec{C}'$  are passed in the vector forms here
- $\text{Zneg} \leftarrow \text{Transpose}(\text{ToMatrix}(\{-z\}_{i=1}^N, m, n))$  ▷ Vector of length  $N$ , with all values being  $q - z$
  - $c_{-z} \leftarrow \text{GetCommitmentMatrix}(\text{Zneg}, 0, \mathbf{ck})$  ▷ A vector of length  $m$ , with all 0 values
  - $c_D \leftarrow c_B^N c_{-z}$  ▷ Entry-wise product
  - $D \leftarrow yA + B$
  - $t \leftarrow yx + s$
  - $b \leftarrow \prod_{i=0}^{N-1} (y_i + x^i - z)$
  - $p\text{Statement} \leftarrow (c_D, c_{-z}, b)$
  - $p\text{Witness} \leftarrow (D + \text{Zneg}, t)$
  - $\text{productArgument} \leftarrow \text{GetProductArgument}(p\text{Statement}, p\text{Witness})$  ▷ See algorithm 5.18
  - $\rho \leftarrow q - (\vec{\rho} \cdot b)$  ▷ Standard inner product  $\sum_{i=0}^{N-1} \rho_i b_i$
  - $\vec{x} \leftarrow \{x^i\}_{i=0}^{N-1}$
  - $C \leftarrow \text{GetCiphertextVectorExponentiation}(\vec{C}, \vec{x})$  ▷ See algorithm 4.6
  - $m\text{Statement} \leftarrow (\text{ToMatrix}(\vec{C}', m, n), C, c_B)$  ▷ See algorithm 5.13
  - $m\text{Witness} \leftarrow (B, s, \rho)$
  - $\text{multiExponentiationArgument} \leftarrow \text{GetMultiExponentiationArgument}(m\text{Statement}, m\text{Witness})$  ▷ See algorithm 5.13

### Output:

shuffleArgument  $(c_A, c_B, \text{productArgument}, \text{multiExponentiationArgument}) \in \mathbb{G}_q^m \times \mathbb{G}_q^m \times \dots \times \dots$   
▷ See algorithm 5.18 and algorithm 5.13 for their respective domains



```

87 * Compute a cryptographic argument for the validity of the shuffle. The statement and witness must comply with the following:
88 *
89 *
90 *
91 *
92 *
93 *
94 *
95 *
96 *
97 *
98 *
99 *
100 *
101 *
102 *
103 *
104 *
105 *
106 *
107 *
108 *
109 *
110 *
111 *
112 *
113 *
114 *
115 *
116 *
117 *
118 *
119 *
120 *
121 *
122 *
123 *
124 *
125 *
126 *
127 *
128 *
129 *
130 *
131 *
132 *
133 *
134 *
135 *
136 *
137 *
138 *
139 *
140 *
141 *
142 *
143 *
144 *
145 *
146 *
147 *
148 *
149 *
150 *
151 *
152 *
153 *
154 *
155 *
156 *
157 *
158 *
159 *
160 *
161 *
162 *
163 *
164 *
165 *
166 *
167 *
168 *
169 *
170 *
171 *
172 *
173 *
174 *
175 *
176 *
177 *
178 *
179 *
180 *
181 *
182 *
183 *
184 *
185 *
186 *
187 *
188 *
189 *
190 *
191 *
192 *
193 *
194 *
195 *
196 *
197 *
198 *
199 *
200 *
201 *
202 *
203 *
204 *
205 *
206 *
207 *
208 *
209 *
210 *
211 *
212 *
213 *
214 *
215 *
216 *
217 *
218 *
219 *
220 *
221 *
222 *
223 *
224 *
225 *
226 *
227 *
228 *
229 *
230 *
231 *
232 *
233 *
234 *
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *
770 *
771 *
772 *
773 *
774 *
775 *
776 *
777 *
778 *
779 *
780 *
781 *
782 *
783 *
784 *
785 *
786 *
787 *
788 *
789 *
790 *
791 *
792 *
793 *
794 *
795 *
796 *
797 *
798 *
799 *
800 *
801 *
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 *
819 *
820 *
821 *
822 *
823 *
824 *
825 *
826 *
827 *
828 *
829 *
830 *
831 *
832 *
833 *
834 *
835 *
836 *
837 *
838 *
839 *
840 *
841 *
842 *
843 *
844 *
845 *
846 *
847 *
848 *
849 *
850 *
851 *
852 *
853 *
854 *
855 *
856 *
857 *
858 *
859 *
860 *
861 *
862 *
863 *
864 *
865 *
866 *
867 *
868 *
869 *
870 *
871 *
872 *
873 *
874 *
875 *
876 *
877 *
878 *
879 *
880 *
881 *
882 *
883 *
884 *
885 *
886 *
887 *
888 *
889 *
890 *
891 *
892 *
893 *
894 *
895 *
896 *
897 *
898 *
899 *
900 *
901 *
902 *
903 *
904 *
905 *
906 *
907 *
908 *
909 *
910 *
911 *
912 *
913 *
914 *
915 *
916 *
917 *
918 *
919 *
920 *
921 *
922 *
923 *
924 *
925 *
926 *
927 *
928 *
929 *
930 *
931 *
932 *
933 *
934 *
935 *
936 *
937 *
938 *
939 *
940 *
941 *
942 *
943 *
944 *
945 *
946 *
947 *
948 *
949 *
950 *
951 *
952 *
953 *
954 *
955 *
956 *
957 *
958 *
959 *
960 *
961 *
962 *
963 *
964 *
965 *
966 *
967 *
968 *
969 *
970 *
971 *
972 *
973 *
974 *
975 *
976 *
977 *
978 *
979 *
980 *
981 *
982 *
983 *
984 *
985 *
986 *
987 *
988 *
989 *
990 *
991 *
992 *
993 *
994 *
995 *
996 *
997 *
998 *
999 *
1000 *

```

# Algorithm implementation Example

**Algorithm 5.11 GetShuffleArgument:** compute a cryptographic argument of the shuffle

**Context:**

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

**Input:**

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{r} \in \mathbb{Z}_q^N$
- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

**Ensure:**  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{C}, \rho_i, \mathbf{pk}))$   
**Ensure:**  $N = mn$



```

86 /**
87  * Computes a cryptographic argument for the validity of the shuffle. The statement and witness must comply with the following:
88  *
89  * <ul>
90  *   <li>be non null</li>
91  *   <li>the statement and witness values must have the same size</li>
92  *   <li>the statement's ciphertexts group and the witness's randomness group must be of the same order</li>
93  *   <li>re-encrypting and shuffling the statement ciphertexts C with the witness randomness and permutation must give the statement
94  *   ciphertexts C'</li>
95  *   <li>the size N of all inputs must satisfy N = m * n</li>
96  * </ul>
97  *
98  * @param statement the {@link ShuffleStatement} for the shuffle argument.
99  * @param witness    the {@link ShuffleWitness} for the shuffle argument.
100 * @param m          the number of rows to use for ciphertext matrices. Strictly positive integer.
101 * @param n          the number of columns to use for ciphertext matrices. Strictly positive integer.
102 * @return a {@link ShuffleArgument}.
103 */
104 ShuffleArgument getShuffleArgument(final ShuffleStatement statement, final ShuffleWitness witness, final int m, final int n) {

```

# Algorithm implementation Example

**Algorithm 5.11** **GetShuffleArgument**: compute a cryptographic argument of the shuffle

**Context:**

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

**Input:**

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{r} \in \mathbb{Z}_q^N$
- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

**Ensure:**  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{r}, \rho_i, \mathbf{pk}))$   
**Ensure:**  $N = mn$



```

86 /**
87  * Computes a cryptographic argument for the validity of the shuffle. The statement and witness must comply with the following:
88  *
89  * <ul>
90  *   <li>be non null</li>
91  *   <li>the statement and witness values must have the same size</li>
92  *   <li>the statement's ciphertexts group and the witness's randomness group must be of the same order</li>
93  *   <li>re-encrypting and shuffling the statement ciphertexts C with the witness randomness and permutation must give the statement ciphertexts C'</li>
94  *   <li>the size N of all inputs must satisfy N = m * n</li>
95  * </ul>
96  *
97  *
98  * @param statement the {@link ShuffleStatement} for the shuffle argument.
99  * @param witness    the {@link ShuffleWitness} for the shuffle argument.
100 * @param m          the number of rows to use for ciphertext matrices. Strictly positive integer.
101 * @param n          the number of columns to use for ciphertext matrices. Strictly positive integer.
102 * @return a {@link ShuffleArgument}.
103 */
104 ShuffleArgument getShuffleArgument(final ShuffleStatement statement, final ShuffleWitness witness, final int m, final int n) {

```

# Algorithm implementation

## Example

**Algorithm 5.11 GetShuffleArgument:** compute a cryptographic argument of the shuffle

**Context:**

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

**Input:**

- The **statement** composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{r} \in \mathbb{Z}_q^N$
- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

**Ensure:**  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{r}, \rho_i, \mathbf{pk}))$   
**Ensure:**  $N = mn$



```
86 /**
87  * Computes a cryptographic argument for the validity of the shuffle. The statement and witness must comply with the following:
88  *
89  * <ul>
90  *   <li>be non null</li>
91  *   <li>the statement and witness values must have the same size</li>
92  *   <li>the statement's ciphertexts group and the witness's randomness group must be of the same order</li>
93  *   <li>re-encrypting and shuffling the statement ciphertexts C with the witness randomness and permutation must give the statement
94  *   ciphertexts C'</li>
95  *   <li>the size N of all inputs must satisfy  $N = m * n$ </li>
96  * </ul>
97  *
98  * @param statement the {@link ShuffleStatement} for the shuffle argument.
99  * @param witness    the {@link ShuffleWitness} for the shuffle argument.
100 * @param m          the number of rows to use for ciphertext matrices. Strictly positive integer.
101 * @param n          the number of columns to use for ciphertext matrices. Strictly positive integer.
102 * @return a {@link ShuffleArgument}.
103 */
104 ShuffleArgument getShuffleArgument(final ShuffleStatement statement, final ShuffleWitness witness, final int m, final int n) {
```

# Algorithm implementation

## Example

**Algorithm 5.11 GetShuffleArgument:** compute a cryptographic argument of the shuffle

**Context:**

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

**Input:**

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{r} \in \mathbb{Z}_q^N$
- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

**Ensure:**  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{C}, \rho_i, \mathbf{pk}))$

**Ensure:**  $N = mn$



```
86  /*
87  * Computes a cryptographic argument for the validity of the shuffle. The statement and witness must comply with the following:
88  *
89  * <ul>
90  *   <li>be non null</li>
91  *   <li>the statement and witness values must have the same size</li>
92  *   <li>the statement's ciphertexts group and the witness's randomness group must be of the same order</li>
93  *   <li>re-encrypting and shuffling the statement ciphertexts C with the witness randomness and permutation must give the state
94  *   ciphertexts C'</li>
95  *   <li>the size N of all inputs must satisfy  $N = m * n$ </li>
96  * </ul>
97  *
98  * @param statement the {@link ShuffleStatement} for the shuffle argument.
99  * @param witness    the {@link ShuffleWitness} for the shuffle argument.
100 * @param m          the number of rows to use for ciphertext matrices. Strictly positive integer.
101 * @param n          the number of columns to use for ciphertext matrices. Strictly positive integer.
102 * @return a {@link ShuffleArgument}.
103 */
104 ShuffleArgument getShuffleArgument(final ShuffleStatement statement, final ShuffleWitness witness, final int m, final int n) {
```

# Algorithm implementation Example

## Algorithm 5.11 GetShuffleArgument: compute a cryptog of the shuffle

### Context:

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

### Input:

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{r} \in \mathbb{Z}_q^N$

- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

Ensure:  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{C}_i, \rho_i, \mathbf{pk}))$

Ensure:  $N = mn$



```
105 checkNotNull(statement);
106 checkNotNull(witness);
107
108 checkArgument(m > 0, "The number of rows for the ciphertext matrices must be strictly positive.");
109 checkArgument(n > 0, "The number of columns for the ciphertext matrices must be strictly positive.");
110
111 final GroupVector<ElGamalMultiRecipientCiphertext, GqGroup> ciphertextsC = statement.getCiphertexts();
112 final GroupVector<ElGamalMultiRecipientCiphertext, GqGroup> shuffledCiphertextsCPrime = statement.getShuffledCiphertexts();
113 final Permutation permutation = witness.getPermutation();
114 final GroupVector<ZqElement, ZqGroup> randomness = witness.getRandomness();
115
116 // Cross dimensions checking.
117 checkArgument(ciphertextsC.size() == permutation.getSize(),
118             "The statement ciphertexts must have the same size as the permutation.");
119
120 // Cross group checking.
121 checkArgument(ciphertextsC.getGroup().hasSameOrderAs(randomness.getGroup()),
122             "The randomness group must have the order of the ciphertexts group.");
123
124 // Ensure the statement corresponds to the witness.
125 final GqGroup gqGroup = ciphertextsC.getGroup();
126 final ZqGroup zqGroup = randomness.getGroup();
127 final int N = permutation.getSize();
128 final int l = ciphertextsC.get(0).size();
129
130 checkArgument(l <= publicKey.size(), "The ciphertexts must be smaller than the public key.");
131
132 final ElGamalMultiRecipientMessage ones = ElGamalMultiRecipientMessage.ones(gqGroup, l);
133 final List<ElGamalMultiRecipientCiphertext> encryptedOnes = randomness.stream()
134     .map(rho -> getCiphertext(ones, rho, publicKey))
135     .collect(toList());
136 final List<ElGamalMultiRecipientCiphertext> shuffledCiphertexts = permutation.stream()
137     .mapToObj(ciphertextsC::get)
138     .collect(toList());
139
```

# Algorithm implementation Example

## Algorithm 5.11 GetShuffleArgument: compute a cryptog of the shuffle

### Context:

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

### Input:

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\vec{r} \in \mathbb{Z}_q^N$

The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$   
The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

Ensure:  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{C}, \rho_i, \mathbf{pk}))$

Ensure:  $N = mn$



```
105 checkNotNull(statement);
106 checkNotNull(witness);
107
108 checkArgument(m > 0, "The number of rows for the ciphertext matrices must be strictly positive.");
109 checkArgument(n > 0, "The number of columns for the ciphertext matrices must be strictly positive.");
110
111 final GroupVector<ElGamalMultiRecipientCiphertext, GqGroup> ciphertextsC = statement.getCiphertexts();
112 final GroupVector<ElGamalMultiRecipientCiphertext, GqGroup> shuffledCiphertextsCPrime = statement.getShuffledCiphertexts();
113 final Permutation permutation = witness.getPermutation();
114 final GroupVector<ZqElement, ZqGroup> randomness = witness.getRandomness();
115
116 // Cross dimensions checking.
117 checkArgument(ciphertextsC.size() == permutation.getSize(),
118             "The statement ciphertexts must have the same size as the permutation.");
119
120 // Cross group checking.
121 checkArgument(ciphertextsC.getGroup().hasSameOrderAs(randomness.getGroup()),
122             "The randomness group must have the order of the ciphertexts group.");
123
124 // Ensure the statement corresponds to the witness.
125 final GqGroup gqGroup = ciphertextsC.getGroup();
126 final ZqGroup zqGroup = randomness.getGroup();
127 final int N = permutation.getSize();
128 final int l = ciphertextsC.get(0).size();
129
130 checkArgument(l <= publicKey.size(), "The ciphertexts must be smaller than the public key.");
131
132 final ElGamalMultiRecipientMessage ones = ElGamalMultiRecipientMessage.ones(gqGroup, l);
133 final List<ElGamalMultiRecipientCiphertext> encryptedOnes = randomness.stream()
134     .map(rho -> getCiphertext(ones, rho, publicKey))
135     .collect(toList());
136 final List<ElGamalMultiRecipientCiphertext> shuffledCiphertexts = permutation.stream()
137     .mapToObj(ciphertextsC::get)
138     .collect(toList());
```

# Algorithm implementation Example

## Algorithm 5.11 GetShuffleArgument: compute a cryptog of the shuffle

### Context:

- Group modulus  $p \in \mathbb{P}$
- Group cardinality  $q \in \mathbb{P}$  s.t.  $p = 2q + 1$
- Group generator  $g \in \mathbb{G}_q$
- A multi-recipient public key  $\mathbf{pk} \in \mathbb{G}_q^m$
- A commitment key  $\mathbf{ck} = (h, g_1, \dots, g_m) \in (\mathbb{G}_q \setminus \{1, g\})^{m+1}$

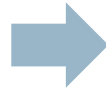
### Input:

- The statement composed of
  - The incoming list of ciphertexts  $\vec{C} \in (\mathbb{H}_1)^N$
  - The shuffled and re-encrypted list of ciphertexts  $\vec{C}' \in (\mathbb{H}_1)^N$
- The witness composed of
  - permutation  $\pi \in \Sigma_N$
  - randomness  $\rho \in \mathbb{Z}_q^m$

- The number of rows to use for ciphertext matrices  $m \in \mathbb{N}^*$
- The number of columns to use for ciphertext matrices  $n \in \mathbb{N}^*$

Ensure:  $\forall i \in [0, N) : \vec{C}'_i = \text{GetCiphertextProduct}(\text{GetCiphertext}(\vec{C}_i, \rho_i, \mathbf{pk}))$

Ensure:  $N = mn$



```

105 checkNotNull(statement);
106 checkNotNull(witness);
107
108 checkArgument(m > 0, "The number of rows for the ciphertext matrices must be strictly positive.");
109 checkArgument(n > 0, "The number of columns for the ciphertext matrices must be strictly positive.");
110
111 final GroupVector<ElGamalMultiRecipientCiphertext, GqGroup> ciphertextsC = statement.getCiphertexts();
112 final GroupVector<ElGamalMultiRecipientCiphertext, GqGroup> shuffledCiphertextsCPrime = statement.getShuffledCiphertexts();
113 final Permutation permutation = witness.getPermutation();
114 final GroupVector<ZqElement, ZqGroup> randomness = witness.getRandomness();
115
116 // Cross dimensions checking.
117 checkArgument(ciphertextsC.size() == permutation.getSize(),
118             "The statement ciphertexts must have the same size as the permutation.");
119
120 // Cross group checking.
121 checkArgument(ciphertextsC.getGroup().hasSameOrderAs(randomness.getGroup()),
122             "The randomness group must have the order of the ciphertexts group.");
123
124 // Ensure the statement corresponds to the witness.
125 final GqGroup gqGroup = ciphertextsC.getGroup();
126 final ZqGroup zqGroup = randomness.getGroup();
127 final int N = permutation.getSize();
128 final int l = ciphertextsC.get(0).size();
129
130 checkArgument(l <= publicKey.size(), "The ciphertexts must be smaller than the public key.");
131
132 final ElGamalMultiRecipientMessage ones = ElGamalMultiRecipientMessage.ones(gqGroup, l);
133 final List<ElGamalMultiRecipientCiphertext> encryptedOnes = randomness.stream()
134     .map(rho -> getCiphertext(ones, rho, publicKey))
135     .collect(toList());
136 final List<ElGamalMultiRecipientCiphertext> shuffledCiphertexts = permutation.stream()
137     .mapToObj(ciphertextsC::get)
138     .collect(toList());
139

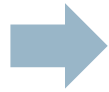
```



# Algorithm implementation

## Example

- 1:  $r \leftarrow \text{GenRandomVector}(q, m)$
- 2:  $A \leftarrow \text{Transpose}(\text{ToMatrix}(\{\pi(i)\}_{i=0}^{N-1}, m, n))$  ▷ Create a  $n \times m$   $\pi$
- 3:  $c_A \leftarrow \text{GetCommitmentMatrix}(A, r, \text{ck})$
- 4:  $x \leftarrow \text{ByteArrayToInteger}(\text{RecursiveHash}(p, q, \text{pk}, \text{ck}, \vec{C}, \vec{C}', c_A))$
- 5:  $s \leftarrow \text{GenRandomVector}(q, m)$

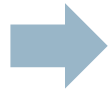


```
147 // Algorithm operations.
148
149 final BigInteger p = gqGroup.getP();
150 final BigInteger q = gqGroup.getQ();
151
152 // Compute vector r, matrix A and vector c_A
153 final GroupVector<ZqElement, ZqGroup> r = randomService.getRandomVector(q, m);
154 final GroupVector<ZqElement, ZqGroup> permutationVector = permutation.stream()
155     .mapToObj(BigInteger::valueOf)
156     .map(value -> ZqElement.create(value, zqGroup))
157     .collect(toGroupVector());
158 final GroupMatrix<ZqElement, ZqGroup> matrixA = permutationVector.toMatrix(m, n).transpose();
159 final GroupVector<GqElement, GqGroup> cA = getCommitmentMatrix(matrixA, r, commitmentKey);
160
161 // Compute x.
162 final byte[] xHash = hashService.recursiveHash(
163     HashableBigInteger.from(p),
164     HashableBigInteger.from(q),
165     publicKey,
166     commitmentKey,
167     ciphertextsC,
168     shuffledCiphertextsCPrime,
169     cA
170 );
```

# Algorithm implementation

## Example

- 1:  $r \leftarrow \text{GenRandomVector}(q, m)$
- 2:  $A \leftarrow \text{Transpose}(\text{ToMatrix}(\{\pi(i)\}_{i=0}^{N-1}, m, n))$  ▷ Create a  $n \times m$   $\pi$
- 3:  $c_A \leftarrow \text{GetCommitmentMatrix}(A, r, \text{ck})$
- 4:  $x \leftarrow \text{ByteArrayToInteger}(\text{RecursiveHash}(p, q, \text{pk}, \text{ck}, \vec{C}, \vec{C}', c_A))$
- 5:  $s \leftarrow \text{GenRandomVector}(q, m)$




```
147 // Algorithm operations.
148
149 final BigInteger p = gqGroup.getP();
150 final BigInteger q = gqGroup.getQ();
151
152 // Compute vector r, matrix A and vector c_A
153 final GroupVector<ZqElement, ZqGroup> r = randomService.getRandomVector(q, m);
154 final GroupVector<ZqElement, ZqGroup> permutationVector = permutation.stream()
155     .mapToObj(BigInteger::valueOf)
156     .map(value -> ZqElement.create(value, zqGroup))
157     .collect(toGroupVector());
158 final GroupMatrix<ZqElement, ZqGroup> matrixA = permutationVector.toMatrix(m, n).transpose();
159 final GroupVector<GqElement, GqGroup> cA = getCommitmentMatrix(matrixA, r, commitmentKey);
160
161 // Compute x.
162 final byte[] xHash = hashService.recursiveHash(
163     HashableBigInteger.from(p),
164     HashableBigInteger.from(q),
165     publicKey,
166     commitmentKey,
167     ciphertextsC,
168     shuffledCiphertextsCPrime,
169     cA
170 );
```

# Algorithm implementation

## Example

```
1 r ← GenRandomVector(q, m)
2 A ← Transpose(ToMatrix( $\{\pi(i)\}_{i=0}^{n-1}$ , m, n)) ▷ Create a n × m  $\pi$ 
3: cA ← GetCommitmentMatrix(A, r, ck)
4: x ← ByteArrayToInteger(RecursiveHash(p, q, pk, ck,  $\vec{C}$ ,  $\vec{C}'$ , cA))
5: s ← GenRandomVector(q, m)
```




```
147 // Algorithm operations.
148
149 final BigInteger p = gqGroup.getP();
150 final BigInteger q = gqGroup.getQ();
151
152 // Compute vector r, matrix A and vector c_A
153 final GroupVector<ZqElement, ZqGroup> r = randomService.getRandomVector(q, m);
154 final GroupVector<ZqElement, ZqGroup> permutationVector = permutation.stream()
155     .mapToObj(BigInteger::valueOf)
156     .map(value -> ZqElement.create(value, zqGroup))
157     .collect(toGroupVector());
158 final GroupMatrix<ZqElement, ZqGroup> matrixA = permutationVector.toMatrix(m, n).transpose();
159 final GroupVector<GqElement, GqGroup> cA = getCommitmentMatrix(matrixA, r, commitmentKey);
160
161 // Compute x.
162 final byte[] xHash = hashService.recursiveHash(
163     HashableBigInteger.from(p),
164     HashableBigInteger.from(q),
165     publicKey,
166     commitmentKey,
167     ciphertextsC,
168     shuffledCiphertextsCPrime,
169     cA
170 );
```

# Algorithm implementation

## Example

```
1 r ← GenRandomVector(q, m)
2 A ← Transpose(ToMatrix( $\{\pi(i)\}_{i=0}^{n-1}$ , m, n)) ▷ Create a n × m  $\pi$ 
3 cA ← GetCommitmentMatrix(A, r, ck)
4 x ← ByteArrayToInteger(RecursiveHash(p, q, pk, ck,  $\vec{C}$ ,  $\vec{C}'$ , cA)
5 s ← GenRandomVector(q, m)
```



```
147 // Algorithm operations.
148
149 final BigInteger p = gqGroup.getP();
150 final BigInteger q = gqGroup.getQ();
151
152 // Compute vector r, matrix A and vector c_A
153 final GroupVector<ZqElement, ZqGroup> r = randomService.getRandomVector(q, m);
154 final GroupVector<ZqElement, ZqGroup> permutationVector = permutation.stream()
155     .mapToObj(BigInteger::valueOf)
156     .map(value -> ZqElement.create(value, zqGroup))
157     .collect(toGroupVector());
158 final GroupMatrix<ZqElement, ZqGroup> matrixA = permutationVector.toMatrix(m, n).transpose();
159 final GroupVector<GqElement, GqGroup> cA = getCommitmentMatrix(matrixA, r, commitmentKey);
160
161 // Compute x.
162 final byte[] xHash = hashService.recursiveHash(
163     HashableBigInteger.from(p),
164     HashableBigInteger.from(q),
165     publicKey,
166     commitmentKey,
167     ciphertextsC,
168     shuffledCiphertextsCPrime,
169     cA
170 );
```



# **PUBLIC SCRUTINY IN PRACTICE**

Olivier Esseiva

# Gitlab issues

## Public scrutiny

- Swiss Post is publishing all reports received on GitLab.
- It is in contact with the experts who have submitted reports.

## Reports received

18 reports have been received since the start of disclosure.

- 1 high-priority finding
- 1 medium-priority finding
- 13 improvements
- 2 comments
- 1 question

# Hierarchy of artefacts

OEV – Federal Chancellery’s Ordinance on Electronic Voting: Precise trust model and security objectives

**Proofs (Verifiability & Privacy) – Computational and Symbolic**

**Specification**

Specification  
Crypto-primitives

Specification Verifier

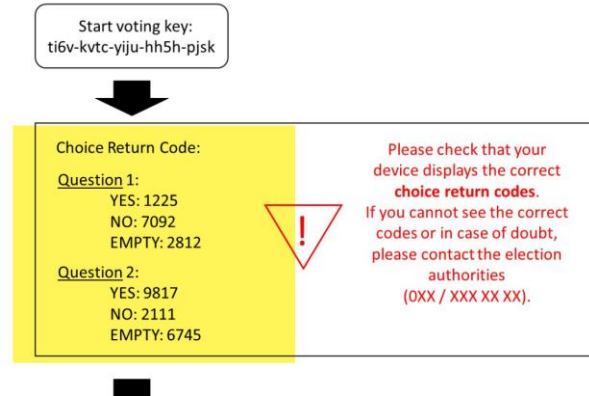
**Source Code evoting**

Source Code  
Crypto-primitives

Source Code Verifier

# Gitlab issues

## #2 Issue by Thomas Haines



### 6. Generate the Return Codes Mapping table CMtable:

- Compute hashes  $h1CC_{id,i} \leftarrow H(1CC_{id,i})$  and  $h1VCC_{id} \leftarrow H(1VCC_{id})$ . Then derive the symmetric encryption keys  $skcc_{id,i}$ ,  $skvcc_{id}$  using the key derivation function KDF (see section 6).

$$skcc_{id,i} \leftarrow KDF(h1CC_{id,i} || 16) \quad \forall i \in (1, \dots, n), \forall id \in \mathcal{ID}$$

- Symmetrically encrypt the short Choice Return Codes  $cc_{id}$  with the Choice Return Code encryption symmetric keys  $skcc_{id}$  and the short Vote Cast Return Code  $vcc_{id}$  with the Vote Cast Return Code encryption symmetric key  $skvcc_{id}$ .

$$ctCC_{id,i} \leftarrow Enc_s(cc_{id,i}; skcc_{id,i}) \quad \forall i \in (1, \dots, n), \forall id \in \mathcal{ID}$$

$$ctVCC_{id} \leftarrow Enc_s(vcc_{id}; skvcc_{id})$$

- Map the long Return Codes (long Choice Return Codes  $ICC$  and long Vote Cast Return Code  $1VCC_{id}$ ) to the encrypted short Return Codes (short Choice Return Codes  $cc_{id}$  and short Vote Cast Return Code  $vcc_{id}$ )

$$CMtable_{id} \leftarrow \left\{ \left[ H(1CC_{id,i}), ctCC_{id,i} \right]_{i=1}^n, [H(1VCC_{id}), ctVCC_{id}] \right\}$$

- Set the Return Codes Mapping table  $CMtable \leftarrow \{CMtable_{id}\}_{id \in \mathcal{ID}}$ .
- Shuffle the table's entries to avoid trivial correlation.



SwissPost Dev @Swisspost-DEV · 4 weeks ago

Reporter

Version 0.9.9 of the computational proof fixes the issue as described above:

Protocol of the Swiss Post Voting System  
Computational Proof of Complete Verifiability and Privacy

Version 0.9.9  
© 2021 Swiss Post Ltd.

- Derive the symmetric encryption keys  $skcc_{id,i}$ ,  $skvcc_{id}$  using the key derivation function KDF (see section 6).

$$skcc_{id,i} \leftarrow KDF(1CC_{id,i}, 16) \quad \forall i \in (1, \dots, n), \forall id \in \mathcal{ID}$$

$$skvcc_{id} \leftarrow KDF(1VCC_{id}, 16) \quad \forall id \in \mathcal{ID}$$

- Symmetrically encrypt the short Choice Return Codes  $cc_{id}$  with the Choice Return Code encryption symmetric keys  $skcc_{id}$  and the short Vote Cast Return Code  $vcc_{id}$  with the Vote Cast Return Code encryption symmetric key  $skvcc_{id}$ .

$$ctCC_{id,i} \leftarrow Enc_s(cc_{id,i}; skcc_{id,i}) \quad \forall i \in (1, \dots, n), \forall id \in \mathcal{ID}$$

$$ctVCC_{id} \leftarrow Enc_s(vcc_{id}; skvcc_{id})$$

- Map the long Return Codes (long Choice Return Codes  $ICC$  and long Vote Cast Return Code  $1VCC_{id}$ ) to the encrypted short Return Codes (short Choice Return Codes  $cc_{id}$  and short Vote Cast Return Code  $vcc_{id}$ )

$$CMtable_{id} \leftarrow \left\{ \left[ H(1CC_{id,i}), ctCC_{id,i} \right]_{i=1}^n, [H(1VCC_{id}), ctVCC_{id}] \right\}$$





# Q&A SESSION



# FEEDBACK

# Summary

We are developing our e-voting system at the IT center in Neuchâtel, adopting an iterative approach and working with external specialists.

Parts of the system will be open source.

The development of the system is based on Security by Design in cryptography and software engineering.

We started the gradual disclosure of a beta version of the system in January 2021, with the aim of identifying vulnerabilities and to continually improving the system.

We have opted for permanent disclosure with a bug bounty programme and annual intrusion tests.

As part of the community programme, we engage in direct dialogue with the experts who submit reports.

---

# THANK YOU!

[www.swisspost.ch/e-voting-community](http://www.swisspost.ch/e-voting-community)

**DIE POST** 