# Library Application for a Fair, Traceable, Auditable and Participatory Drawing Tool for Legal Systems

**Marcos Vinicius M. Silva**[1], **Marcos A. Simplicio Jr.**[1]

[1] Escola Politécnica – University of São Paulo (USP), Brazil

***Abstract.*** *Several applications require random sampling to be protected against a biased behavior. When conflicting parties have different interests in its result, the system must guarantee that collusion among any number of entities cannot influence the resulting computation. Such is the case of legal systems, in which jurors and judges must be randomly picked to ensure impartiality in judicial cases. However, when informational systems are used to generate randomness, it should also provide auditability of its mechanisms to promote confidence in its fairness. This article presents a tool of one such mechanism that combines the randomness provided by hash functions with active social engagement. Each stakeholder in a legal proceeding contribute with his/her own share to the drawing, so that fairness is achieved if at least one entity is honest. Any interested party can audit a drawing using only public information, and misconduct of any party can be traced back to the culprit as soon as the result is computed. Our open-source implementation provides security by design, not depending on the secrecy of its component to attain all the required security properties.*

## 1. Introduction

The process of randomization is largely used in several applications where the results can be biased towards a manipulated output or the confidence in its methods can be degraded, such as scientific experiments, medical trials, and operation of legal systems. Not only does it protect against a malicious adversary attempting to corrupt the execution, but also from spurious manipulations unintended by the system designer or its users. In medical trials, for example, knowledge of interference may lead participants to behave in unintended ways, leading to artificial inferences about the effects of a new drug or procedure [Solomon 1949]. Therefore, randomized interference provides a safeguard for achieving more reliable results, by not allowing the participants to know whether they are receiving the intervention or only a placebo.

Another scenario in which randomization is critical is that of judicial proceedings. Several countries employ randomized methods in a way to select jurors [Duxbury 2002] or judges [Eisenberg et al. 2012] to get unbiased decisions. When using informational systems to generate (pseudo)random values, however, fairness can only be achieved with two new requirements: auditability of the execution, and active social engagement. On one hand, auditability guarantees that the output of random functions can be verified by any entity outside of scope of the execution. Hence, it improves trust in the system where statistical deviations may result in suspicion in non-auditable procedures. On the other hand, an active, self-reflective and well-coordinated participation by pertinent members of a community can result in more engagement and inclusiveness, relevant aspects of social practices that also apply to the legal system [Stern 2018].

One proposed method for a fair drawing procedure is presented in [Silva et al. 2020], which provides both auditability and social participation. The presented methods are capable of electing a candidate from a row of participants, if the probability of sampling any participant is known. In this article, we present an implementation of such methods in the form of a library in the Java language. The correctness of the implementation (and of the proposed procedure) can be observed by reproducible tests, for both honest and corrupt behavior. A simple interface was also developed for better presentability of the execution of a drawing.

The remainder of this document is divided as follows: Section 2 summarizes the protocol as defined in [Silva et al. 2020], which is the source of our implementation. Section 3 details the library, together with two simple implementations that will be demonstrated. Section 4 presents the steps of the intended demonstration. Finally, we conclude in Section 5.

## 2. Auditable Random Draw

In this section, we describe the implemented methods for randomly drawing some entity among a list of eligible candidates. The algorithms here presented were proposed in [Silva et al. 2020], which not only defines the implemented methods, but also analyzes the security of the cryptographic methods and extends the discussion for non-uniform probability distribution of candidates. In this article, we focus on the two methods available in the library for uniform distribution: (1) a random drawing for a single legal proceeding, and (2) a random drawing for multiple proceedings with the same stakeholders. In both cases, the idea behind the protocol was discussed by M. Blum for solving the "coin-flipping by telephone" [Blum 1983]. In this scenario, two mutually untrusted parties plays a coin-flipping game, attempting to communally generate a random bit in a way that neither party can bias the outcome. The solution proposed is a two-step protocol known as commit-and-reveal [Naor 1990]: first, each party generates a random share and *commits* to it, sending only the commitment to the other party; then, once both commitments were received, each party *reveals* their share, which is verified regarding the commitment. The security is guaranteed if no share can be guessed from their commitments, and if it is infeasible to reveal a share different from the committed one. Although this method can be applied to a variety of applications, in this article we focus on the context of legal proceedings, assuming that entities like judge, juror(s) or rapporteur must be selected at random from a known row of candidates.

### 2.1. Overview

In this application, a drawing procedure `Draw` is defined as the set of fields $(\texttt{DID}, S, E, \texttt{info})$, in which:

- `DID` is the unique identifier of the procedure in the system and directly depends on the legal proceedings it represents. It is defined as a set $\texttt{DID} = \{\texttt{PID}||\texttt{cnt}\}$, where each element is defined as the legal proceeding unique identifier `PID` concatenated to a drawing counter `cnt`, which is incremented as new drawings are required for the same proceeding. In the implementation, the `PID` is specified as a character string, and is separated from `cnt` with the character #. For example, the first drawing to the proceeding identified by $\texttt{PID} = 123.456{-}7$ will define the drawing procedure $\texttt{DID} = 123.456{-}7\#0$.

- $S$ is the set of all stakeholders $s_j$ that must participate in the drawing procedure for all legal proceedings in the drawing. These entities are selected according to the proceedings, and may include public entities (e.g., Ministry of Justice, Supreme Court, and bar council) or proceeding-specific one (e.g., defense lawyer, prosecutor, and judge). Since these entities are responsible for generating the shares for a drawing, they are identified by their public key or associated digital certificate issued by a trusted Certificate Authority. In this way, their signature can be verified during the protocol execution, providing the necessary authenticity.
- $E$ are the lists of eligible candidates $e_j$, where each proceeding in the drawing has a list with its own candidates. It may refer, for example, to all judges with no conflict of interest to each proceeding, when one must be selected. Each candidate is uniquely identifiable (e.g., their social security number, functional identities, digital certificates) and, to avoid any ambiguity, the list must be sorted in a determined order (e.g., in lexicographic order).
- `info` defines additional information about the legal proceedings and is presented in a human-readable format. It may include the proceeding title, class, subject, last modification date. This field is *optional*, as the information provided by $\{\texttt{PID}||\texttt{cnt}\}$ is enough to uniquely identify the proceeding in the system.

### 2.2. Drawing procedure for a single proceeding

For electing a candidate from $E_i$ for a drawing procedure $\texttt{Draw}_i = \{\texttt{DID}_i, S_i, E_i, \texttt{info}_i\}$, each stakeholder $s_j \in S_i$ engages in the following two-phase procedure:

**Commitment** Each stakeholder $s_j$ generates a random share $0 \leqslant share_{i,j} < |E_i|$, that will be this stakeholder's contribution, and a mask $mask_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$ (for some security level $\lambda$), that will be used to hide the share in the commitment. The commitment is computed as $C_{i,j} \leftarrow \mathcal{H}(\texttt{Draw}_i, mask_{i,j}, share_{i,j})$, where $\texttt{Draw}_i$ is common to all stakeholders. By concatenating the mask in the input of the hash function, the potentially low-entropy share cannot be guessed from $C_{i,j}$. The commitment in then signed in $\sigma_{i,j} \leftarrow \mathcal{S}(sk_j, \{\texttt{Draw}_i, C_{i,j}\})$ by the stakeholder to provide both authenticity and non-repudiation. Finally, the set $(C_{i,j}, \sigma_{i,j})$ can be sent to all stakeholders of this drawing, and each receiver ensures that the commitment is valid by verifying the signature $\mathcal{V}(pk_j, \{\texttt{Draw}_i, C_{i,j}\}, \sigma_{i,j})$.

**Reveal** After the commitments were received from all stakeholders, any party can reveal their share for all other participants. Each stakeholder, then, sends the set $(share_{i,j}, mask_{i,j})$, that can be verified by recomputing the hash and checking if $C_{i,j} \stackrel{?}{=} \mathcal{H}(\texttt{Draw}_i, mask_{i,j}, share_{i,j})$ holds true. Since this hash was signed in the commitment phase, it is not necessary to sign it again, as the share was indirectly signed when computing $\sigma_{i,j}$. After all shares were received, the result of the draw is computed as $d = (\sum_{j=0}^{|S_i|-1} share_{i,j}) \bmod |E_i|$, which corresponds to the index of the candidade in the ordered list $E_i$ of the drawing.

### 2.3. Drawing procedure for multiple proceedings with the same stakeholders

The process from Section 2.2 can be extended for executing the drawings simultaneously for several proceedings that share the same set of stakeholders. Considering $\Delta = \{\texttt{Draw}_i\}$ a list of random drawing procedures $\{\texttt{DID}_i, S, E_i, \texttt{info}_i\}$ with the same set $S$ of stakeholders, we follow the same structure of a two-phase procedure:

**Commitment** Each stakeholder $s_j$ generates a random share $0 \leqslant share_{i,j} < |E_i|$ for each $\texttt{Draw}_i \in \Delta$, that will be this stakeholder's contribution to each drawing, and a single mask $mask_{0,j} \xleftarrow{\$} \{0,1\}^\lambda$ (for some security level $\lambda$), that will be used to hide the shares in the commitments. The commitment is computed interactively (for $0 \leqslant i < |\Delta|$) by $C_{i,j} \leftarrow \mathcal{H}(\texttt{Draw}_i, mask_{i,j}, share_{i,j})$, and updating the subsequent mask $mask_{i+1,j} = C_{i,j}$. The last commitment $C_{|\Delta|-1,j}$ is then signed and broadcast to all other stakeholders in $S$.

**Reveal** After the commitments were received from all stakeholders and their signatures verified, the interested parties can reveal their share for all other participants. Each stakeholder, then, sends the mask $mask_{0,j}$ and all the shares $share_{i,j}$ (for $i \geqslant 0$). This allows every entity to recompute interactively the commitments up to the last $C_{|\Delta|-1,j}$, which will only hold true if all shares (and the mask) were the originally committed. After all shares were received, the result of the draw is computed for each share as $d_i = (\sum_{j=0}^{|S_i|-1} share_{i,j}) \bmod |E_i|$ for each $i \geqslant 0$.

## 3. Implementation Details

The proposed library was developed in Java, using the version 14 of the OpenJDK compiler with compliance with Java language version 8. It is currently available at `https://doi.org/10.24433/CO.6108166.v1` under the MIT License, with three main projects:

- The *library* code in the `br.usp.larc.securedraw` package contains the basic code for the execution of the fair drawing protocol. For using the library, it is required to implement communication and storage interfaces, depending on the actual architecture it will be built upon.
- The *automatic tests* in the `securedrawsimple` package contains a naive implementation of the library interfaces, relying in the Java Virtual Machine (JVM) execution stack for communication. This was used to test both correctness and some malicious attempts to subvert the execution.
- The *graphical interface* in the `securedrawgui` package contains a simple graphical user interface (GUI) to demonstrate how an actual system may be designed for human interaction. This interface can run standalone, but have the automatic tests incorporated to a more user-friendly execution. This is the package used in the demonstration.

In what follows, we present more details of the components of these projects.

### 3.1. The library

The library is divided in three packages, depending on the functionalities presented in its classes: the default, `comm`, and `model` packages. The default package contains the implementation of the fair drawing procedure and of the underlying cryptographic algorithms. A stakeholder handler controls the execution of the drawing procedure as defined in Sections 2.2 and 2.3. The cryptographic methods were designed implemented using the standard Java Security API. Therefore, the hash function for generating commitments was instantiated using the SHA256 algorithm, and the signature scheme was the ECDSA on the elliptic curve NIST P-256 (signing a SHA256 digest). This choice of parameters leads to 128 bits of security.

The `model` package contains the serializable implementation of all entities and artifacts that are either stored or transmitted to other parties. It represents the drawing procedure (for either a single or multi-proceeding), its stakeholders and eligible candidates, and the generated shares and commitments. These classes contains only the minimum information required as presented in this article, but they can be extended if additional information is required to be represented.

The `comm` package contains the interfaces that require specific implementations depending on the architecture choices, whose methods are called directly from the stakeholder handler. The communication interface is a simple interface used to allow two parties to exchange messages. The storage manager is responsible for storing the drawing information (namely, stakeholders, commitments and shares) of running procedures. Finally, the handler listener which is responsible for observing the execution of the methods in the stakeholder handler, and notifying when relevant steps are completed or errors found.

To compile the library, generating a JAR file, the following commands must be executed in the terminal at the root of the project:

```
javac -d ./build/ br/usp/larc/securedraw/*.java br/usp/larc/
    securedraw/comm/*.java br/usp/larc/securedraw/model/*.java
jar cvf SecureDraw.jar -C build .
```

The source code also presents an embedded javadoc, which could by generated by the command:

```
javadoc -d ./docs/ br/usp/larc/securedraw/*.java br/usp/larc/
    securedraw/comm/*.java br/usp/larc/securedraw/model/*.java
```

### 3.2. Automatic tests

The automatic tests were created in order to generate reproducible runs that could verify the correctness of the scheme proposed. First, it verifies the correct execution of drawings for single or multiple proceedings. Then, it simulates some malicious behaviors of an attacker that tries to subvert the scheme. Namely, (1) the attacker tries to reveal a share outside the specific interval for the number of candidates, (2) he/she tries to sign a message different from the commitment, (3) he/she tries to reveal a share different of the committed one, and (4) he/she tries to reveal a different mask for a committed share.

It can be compile together with the library, executing the following command in the terminal at the root of the project:

```
javac -d ./build/ br/usp/larc/securedraw/*.java br/usp/larc/
    securedraw/comm/*.java br/usp/larc/securedraw/model/*.java
    securedrawsimple/*.java securedrawsimple/comm/*.java
    securedrawsimple/malicious/*.java
jar cvfe SecureDrawSimple.jar securedrawsimple.SecureDraw -C
    build .
```

### 3.3. Graphical Interface

In this project, we present an implementation of a simple graphical user interface for testing purposes that simulates the behavior of real stakeholders participating in some draw-

ing in a local environment. Figure 1 illustrates the creation of a new drawing, in which the user can input the drawing identification and can select, from a list of pre-registered entities, the stakeholders that will participate in the drawing and the eligible candidates. And Figure 2 illustrates the commitment phase, where the user can choose a value to commit to the proceeding, and commitments received from other stakeholders can be seen. For the presentation, each stakeholder receives a new tab, which represents an instance of their execution on their devices. This simplification was made for better presentability, since an actual interface would depend on the target platforms (such as desktops, smartphones, or dedicated hardware) and the specific scenario (how stakeholders and eligible candidates are defined). We note, however, that it does not limit the use of the library in a distributed environment, as the interfaces can be easily adapted.

The graphical interface can be compiled together with the library and requires the compilation of the automatic tests, since they can be called from the GUI. For that, it can be compile to generate a runnable JAR by the command:

```
javac -d ./build/ br/usp/larc/securedraw/*.java br/usp/larc/
    securedraw/comm/*.java br/usp/larc/securedraw/model/*.java
    securedrawsimple/*.java securedrawsimple/comm/*.java
    securedrawsimple/malicious/*.javasecuredrawsimple/*.java
    securedrawsimple/comm/*.java securedrawsimple/malicious/*.
    java securedrawgui/*.java securedrawgui/comm/*.java
jar cvfe SecureDrawGUI.jar securedrawgui.JFrameDraw -C build .
jar uf SecureDrawGUI.jar securedrawgui/img/*
```

## 4. Presentation

The only requirement for the presentation is a computer capable of executing a Java Virtual Machine of version 8 or newer. It will be divided in two models: a drawing of a single proceeding, and a drawing for multiple proceedings. In both cases, it will reflect the following steps:

- **Creation:** The drawing procedure will be initiated by defining the proceeding number and info, and then by selecting stakeholders and candidates from a preloaded list. The interface generates a standard set of attributes, but they can be modified by the user.
- **Commit:** Each stakeholder generates a random value in the valid interval, which can be modified by the user. When submitted, it will be committed and sent to the other stakeholders. This step can be observed by the other stakeholders as soon they receive the commitments.
- **Reveal:** After receiving all commitments, the stakeholders can reveal their committed values to the others, which is verified in background.
- **Result:** After all values are revealed, the software automatically computes the result, electing a candidate for each proceeding in the drawing. This result can be audited by the drawing's stakeholders, which have access to the commitments and revealed values.

## 5. Final Remarks

As next steps, we plan to develop a full fledged, mobile meta-application. The goal is to enable auditable random drawings in arbitrary contexts: users could register the drawing

**Figure 1. Creation of a new drawing procedure**

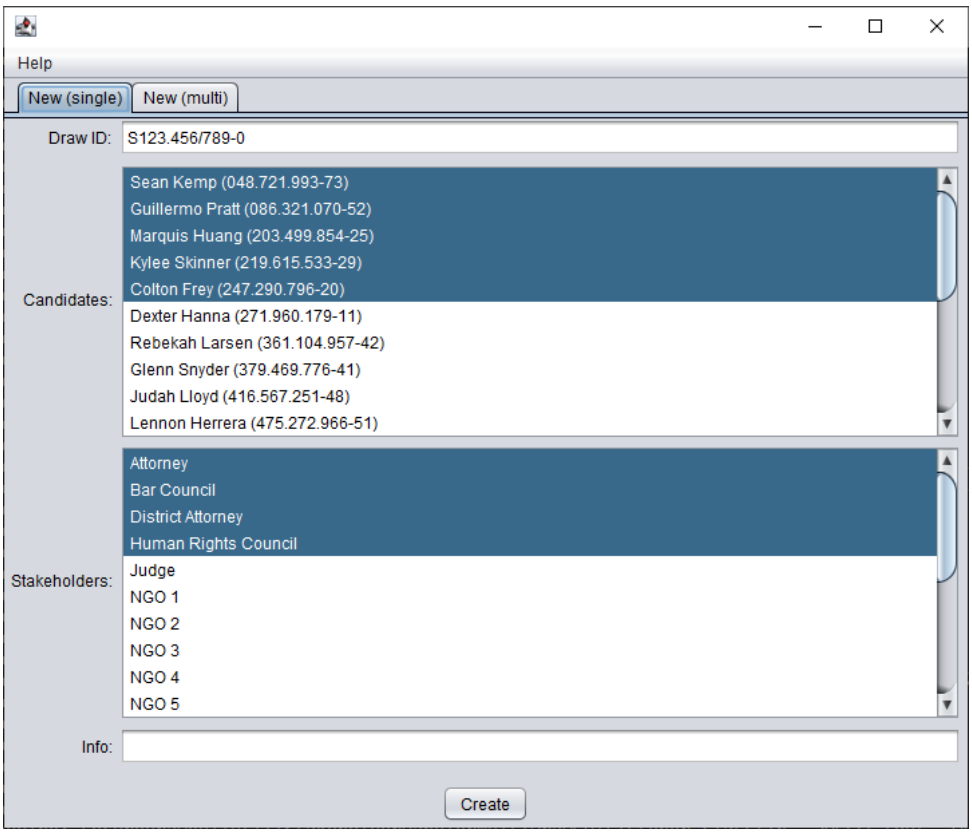
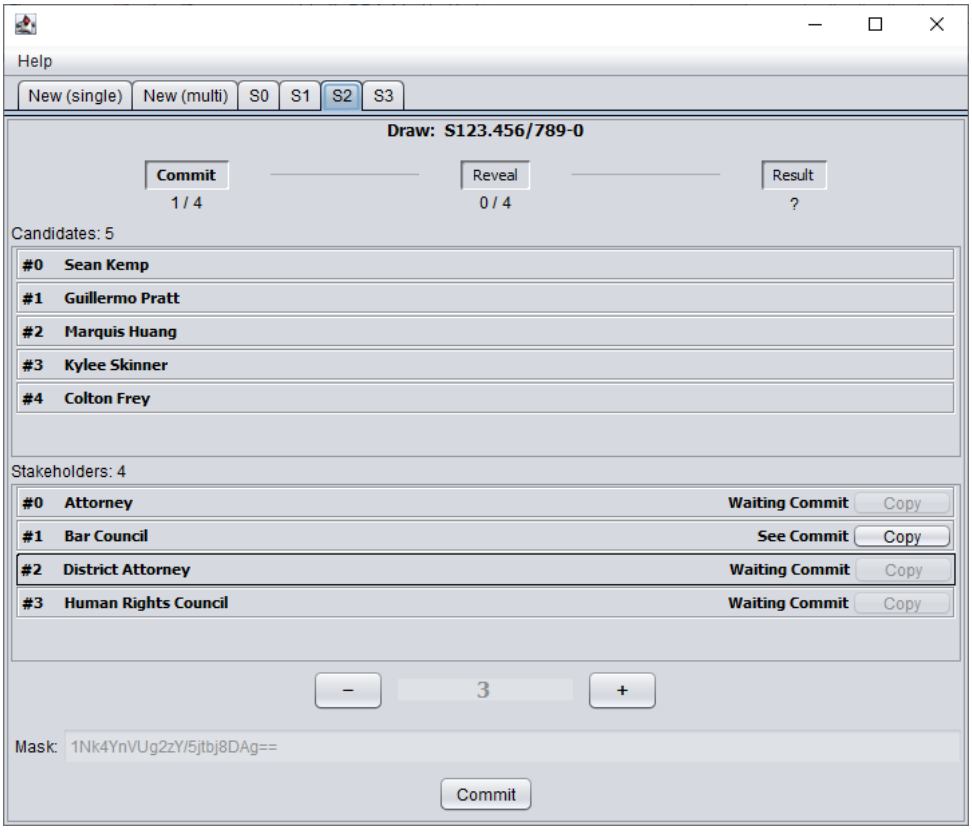



**Figure 2. Commitment to a drawing procedure**



information on a cloud server, and allow the participation of any interested (and, possibly, authorized) parties.

**Acknowledgements**

This work was supported by: Ripple's University Blockchain Research Initiative; CNPq (Brazilian National Council for Scientific and Technological Development – grant PQ 301198/2017-9).

## References

Blum, M. (1983). Coin flipping by telephone: a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27.

Duxbury, N. (2002). *Random Justice: On Lotteries and Legal Decision-Making*. Oxford University Press.

Eisenberg, T., Fisher, T., and Rosen-Zvi, I. (2012). Does the judge matter? exploiting random assignment on a court of last resort to assess judge and case selection effects. *Journal of Empirical Legal Studies*, 9(2):246–290.

Naor, M. (1990). Bit commitment using pseudo-randomness. In *Advances in Cryptology (CRYPTO'89)*, pages 128–136, New York, NY. Springer New York.

Silva, M. V. M., Jr., M. A. S., Pfeiffer, R. A. C., and Stern, J. M. (2020). A fair, traceable, auditable and participatory randomization tool for legal systems.

Solomon, R. L. (1949). An extension of control group design. *Psychological bulletin*, 46(2):137.

Stern, J. M. (2018). Verstehen (causal/ interpretative understanding), erklaeren (law-governed description/ prediction), and empirical legal studies. *Journal of Institutional and Theoretical Economics*, 174(1):105–114.