GIOVANNI ABENI
JOÃO PAULO LINS

COMMIT AND REVEAL: APLICAÇÕES PARA SORTEIOS JUSTOS E CONTRATOS CONFIÁVEIS

São Paulo
2020

GIOVANNI ABENI
JOÃO PAULO LINS

COMMIT AND REVEAL: APLICAÇÕES PARA SORTEIOS JUSTOS E CONTRATOS CONFIÁVEIS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Elétrico, com ênfase em computação.

São Paulo
2020

GIOVANNI ABENI
JOÃO PAULO LINS

COMMIT AND REVEAL: APLICAÇÕES PARA SORTEIOS JUSTOS E CONTRATOS CONFIÁVEIS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Elétrico, com ênfase em computação.

Área de Concentração:
Engenharia Elétrica

Orientador:
Marcos Antonio Simplício Junior

São Paulo
2020

Dedicamos às nossas famílias e amigos,
que fizeram parte desse processo.

AGRADECIMENTOS

Ao professor Simplício e a sua equipe pela contribuição no desenvolvimento teórico do assunto.

À Escola Politécnica, por todo o conhecimento que nos foi dado.

RESUMO

Este documento explica e descreve a construção de uma biblioteca de *software* que fornece recursos para o desenvolvimento de aplicações de sorteios justos, auditáveis e distribuídos, de forma que possa ser utilizada em diversos contextos. Para isso, escolheu-se uma metodologia de sorteio em que, através da técnica de comunicação criptografada conhecida como *Commit-Reveal*, pudesse garantir ao processo seu caráter aleatório, confiável e imparcial. Além disso, o presente trabalho detalha como o sistema desenvolvido funciona e quais são as configurações necessárias para que ele seja integrado a outras aplicações. Por fim, para demonstrar uma aplicação da biblioteca, desenvolveu-se com sucesso um aplicativo *web* para que seu funcionamento pudesse ser testado e validado de maneira simples.

Palavras-Chave – Sorteio. *Commit-Reveal*. Geração de Números Aleatórios. Auditabilidade. *Software Development Kit*. *SDK*. Randomização.

ABSTRACT

This document explains and describes the building of a software library that provides the resources for the development of fair, auditable and distributed draw applications to be used in different contexts. To do so, great effort has been spent to choose a draw methodoly which, thru an encrypted communication technique know as Commit-Reveal, could guarantee its random, reliable and imparcial character. Furthermore, the current piece of work sheds light on how the developed system works and what is the necessary setup to integrate it to other applications. Lastly, to demonstrate an application of the library, a web app as developed in order to test and validate the final software.

Keywords – Draw, Commit-Reveal. Random number generator. Auditability. Software Development Kit. SDK. Randomization.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o avanço tecnológico, cada vez mais a sociedade constrói sistemas computacionais para serem responsáveis por processos cruciais de sua organização. Em muitos casos, tais sistemas surgem com objetivos que vão muito além da praticidade e eficiência, pois trazem para si a responsabilidade de serem justos, transparentes, seguros e confiáveis. Como exemplo, podemos citar sistemas de votações, de sorteios, de acesso à conteúdo protegido, de intermediação financeira, entre outros em que haja uma grande preocupação com integridade, confidencialidade e imparcialidade.

Entretanto, os sistemas computacionais são construídos e mantidos por seres humanos, de maneira que ainda estão sujeitos à atitudes de má fé durante sua construção. Isso nos permite levantar diversas questões à respeito do quanto podemos confiar nos sistemas que utilizamos hoje. Por exemplo: como garantimos que a urna eletrônica é imparcial? Ou como garantir que o código fonte compartilhado à comunidade é de fato o implantado nas máquinas? Para o caso de uma lotérica, como assegurar que os sorteios são de fato justos e imprevisíveis?

São inúmeros os gargalos passíveis de questionamento, de modo que se torna evidente a necessidade de processos que assegurem a confiança absoluta de todos os envolvidos em um sistema, especialmente àqueles onde haja conflitos de interesse entre os atores. Dessa forma, o principal objetivo deste trabalho é explorar o uso de técnicas alternativas para garantir a confiança em aplicações, particularmente àquelas que têm fortes requisitos de aleatoriedade, imparcialidade e integridade.

Diande desse cenário, uma técnica no ramo da criptografia conhecida como ***commit-reveal*** pode ser utilizada de modo a resolver o problema acima. Esta técnica consiste em um esquema de comprometimento, onde as partes podem trocar informações com garantias de integridade e confidencialidade, sendo possível, a partir disso, gerar números aleatórios com alta entropia.

O grande diferencial desse esquema de comprometimento é que se cria um **pro-**

tocolo de segurança colaborativo e distribuído entre as partes envolvidas. Assim, as aplicações podem passar a não mais depender de um intermediário, como entidades que realizam sorteios, ou que contam os votos, ou que processam os jogos etc. Com a técnica de *commit-reveal*, a confiança não é em uma ”figura certificada”, mas sim no processo em si, onde cada parte é capaz de validar as interações com as outras.

Assim, o projeto teve como norte a criação de um *software* que fosse capaz de gerar um sorteio justo e aleatório, segundo um modelo incremental e evolutivo, onde cada desafio forneceu insumos para a implementação e definição dos próximos. Houve um grande interesse em, uma vez bem estabelecida as bases do sorteio justo, criar um módulo que pudesse ser utilizado em diversas aplicações. No projeto em questão, desenvolveu-se um aplicativo *web* para a demonstração do sorteio.

2 OBJETIVO

O objetivo geral deste projeto consiste em explorar a técnica criptográfica de *commit-reveal* a fim de se estruturar uma ferramenta de aleatorização justa, distribuída, colaborativa e auditável, para atuar especialmente em processos onde haja conflitos de confiança entre as partes envolvidas, como, por exemplo, em sorteios.

Como objetivos específicos, tinha-se por objetivo, além da análise da segurança da ferramenta proposta, desenvolver uma biblioteca de *software* - *Software Development Kit* (*SDK*) - que fornecesse a organização e os recursos necessários para a construção de aplicações que usufruam da técnica, bem como exemplos de implementação.

O projeto contou ainda com a construção de uma aplicação que pudesse atuar na distribuição aleatória de processos entre ministros do Supremo Tribunal Federal brasileiro.

3 PROBLEMÁTICA

Uma situação comum estudada no campo da Teoria dos Jogos é a influência da ordem das ações nas decisões dos jogadores. Em diversos cenários, a primeira parte a revelar sua ação/decisão para as outras é prejudicada, pois fornece informações de forma que seja mais fácil superar sua estratégia.

Um exemplo trivial para entender a problemática acima é o caso de uma partida de "par ou ímpar". Partindo das premissas que cada jogador já escolheu sua aposta e que ambos os jogadores querem vencer, caso as revelação dos números seja de forma sequencial, o último jogador a revelar o número escolhido terá 100% de chances de vitória. Isso acontece porque o jogador final conhece o número escolhido pelo jogador anterior e, portanto, basta escolher um número que satisfaça sua condição de vitória (a soma dos números ser par ou ímpar de acordo com o rótulo escolhido).

Esse conflito pode ser resolvido fazendo as apostas de maneira simultânea, isto é, todos os jogadores revelando seus números ao mesmo tempo, não havendo maneiras de um previamente saber a escolha do outro. Entretanto, nem sempre isso é possível, seja por questões práticas/técnicas ou por características essenciais do negócio (às vezes os jogadores não estão disponíveis ao mesmo tempo ou um desconfia que outro possa atrasar um pouco sua jogada).

Sendo assim, uma alternativa para superar a necessidade da simultaneidade é esconder o número de cada jogador numa caixa segura, protegida por uma chave escondida, garantindo que uma vez inserido o valor, não há como alterá-lo. Assim, a ordem em que as jogadas são feitas deixa de ser uma variável importante, pois um não conseguirá saber a resposta do outro até que todos tenham feito suas jogadas.

Com as duas apostas feitas e protegidas, cada jogador pode compartilhar sua respectiva chave para que as caixas sejam abertas. Não é possível abrir caixas com outras chaves senão a original assim como não é possível mudar o conteúdo da caixa após seu trancamento e reabertura. Dessa forma, é garantido que:

- Nenhuma das partes escolheu seu número baseando-se na resposta de outra; e
- Nenhuma das partes alterou seu número após conhecer o número do adversário.

Ao fim de todo esse processo, quando calcula-se a soma dos números para determinar a paridade resultante, é possível confiar que a fonte de entropia (aleatoriedade base) não é gerenciada por um terceiro, mas sim pelo processo em si. Portanto, o método, além de eliminar o conflito de confiança entre as partes, também insere uma aleatoriedade genuína, com a possibilidade de verificação.

Apesar do exemplo do jogo de “par ou ímpar” não ter utilidade prática formal, existem diversas aplicações mais relevantes onde esse processo pode ser aproveitado. Situações em que haja conflito de confiança entre partes interessadas ou que exijam um alto grau de aleatoriedade com necessidade de verificação podem se beneficiar do uso desse método, implementando-o em um sistema computacional.

4 FUNCIONAMENTO DO *COMMIT-REVEAL*

No mundo da tecnologia, há procedimentos de segurança baseados em uma técnica análoga ao processo explicado na seção anterior, chamada ***Commit-Reveal***. Esta técnica é uma primitiva criptográfica onde as partes envolvidas se comprometem a um valor imutável, inicialmente escondido das outras partes para, posteriormente, revelá-lo mediante uma condição estabelecida. Esse esquema de comprometimento (*Commitment Scheme*) se estabelece em duas fases:

- ***Commit*** (comprometer): é a fase de escolha e compartilhamento do valor protegido. Análogo à entrega das caixas trancadas, no exemplo acima;
- ***Reveal*** (revelar): é a fase onde as partes revelam o conteúdo explícito de suas escolhas, após todos os *commitments* terem sido enviados. Análogo ao compartilhamento das chaves no exemplo acima.

A implementação dessa técnica consiste na combinação de funções de *hash* com números randômicos. Na fase de *commit*, a mensagem crua é combinada com um valor genérico (chamado de ”***nonce***”) e passada por uma função de ***hash***.

Figura 1: Passo 1 - Fluxo de criação do commitment.

O resultado disso é chamado de *digest* e é enviado para as outras partes, fazendo o papel do *commitment*. Nenhuma das partes consegue decifrar os *commitments* das outras, senão o seu, pois além de ser uma característica essencial de uma função de hash, apenas os remetentes originais conhecem seus respectivos *nonces*.

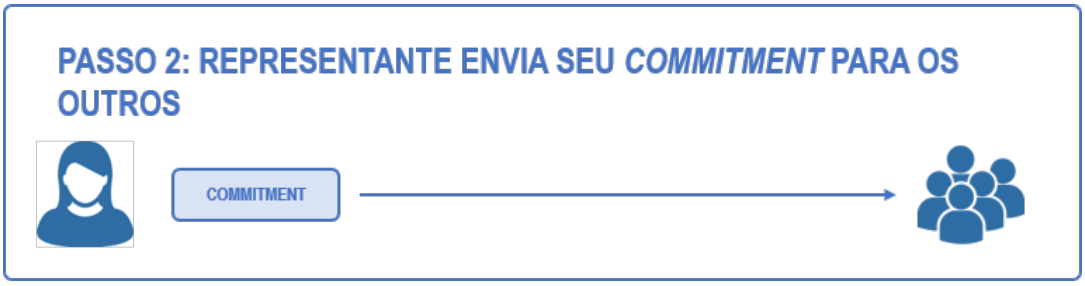

Figura 2: Passo 2 - Compartilhamento do commitment com o grupo.

Após todos *commitments* serem enviados, as partes podem revelar suas respostas, compartilhando sua mensagem original junto do nonce usado na fase de commit.

Figura 3: Passo 3 - Fluxo de reveal.

Considerando que função de *hash* é a mesma para todos envolvidos, então para validar se uma mensagem exposta é a mesma que fora prometida, basta cada parte aplicar a função de *hash* sobre a mensagem crua combinada com o *nonce* e comparar se o resultado é idêntico ao *digest* anteriormente compartilhado.

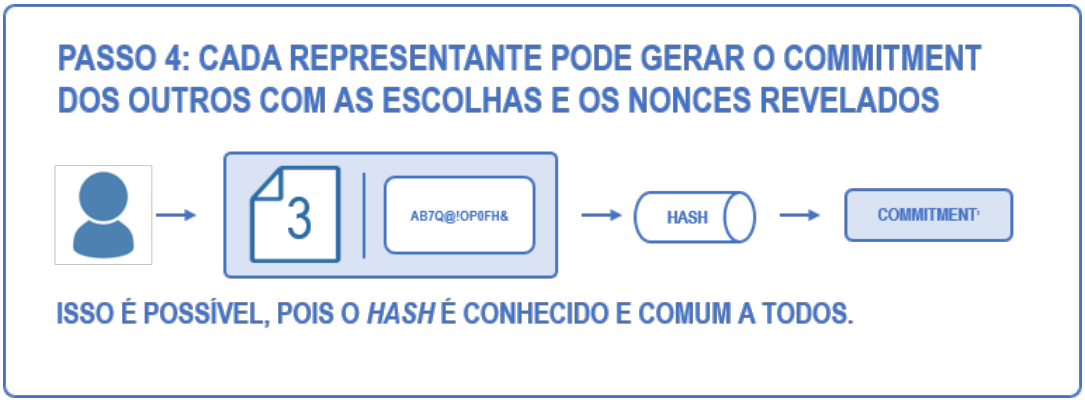

Figura 4: Passo 4 - Fluxo de geração do digest a partir do reveal enviado.

Caso não seja idêntico, significa que aquela mensagem revelada não é a mesma que fora prometida, evidenciando uma trapaça (”*cheat*”). Isto é computacionalmente garantido pois a função de *hash* escolhida deve obedecer as propriedades fundamentais de resistir à primeira inversão, à segunda inversão e à colisão. De forma geral, a

função é considerada suficientemente segura caso as tarefas de inverte-la ou encontrar colisões forem computacionalmente inviáveis.

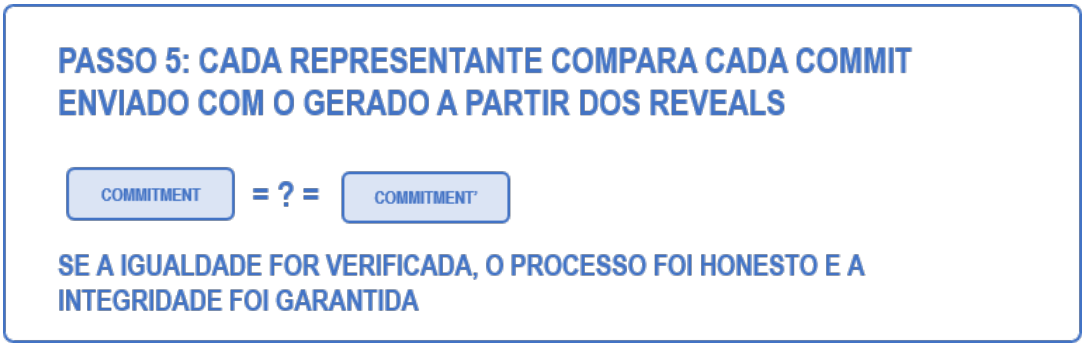

Figura 5: Passo 5 - Fluxo de validação do reveal.

Fazendo uma analogia com exemplo acima, a mensagem equivale ao número apostado, o *digest* à caixa segura, o *nonce* à chave e a *hash* aos sistemas chave-fechadura.

5 PROCESSO DE SORTEIO

5.1 Inspiração

Uma brincadeira muito comum entre grupos de amigos que precisam sortear uma pessoa para, por exemplo, ser o primeiro a realizar uma tarefa é a chamada ”Dedos Iguais”. Nela, todos os participantes se posicionam em uma roda e indicam simultaneamente um número de 0 a 10, usando seus dedos da mão. Após isso, soma-se os números escolhidos por cada participante e, partindo de uma pessoa qualquer, conta-se sequencialmente e ciclicamente, de 1 até o valor da soma. O sorteado é a última pessoa da contagem.

O princípio de funcionamento desse sistema de sorteio é chegar a um número aleatório, a partir de contribuições individuais que impactam na soma final, de modo que, se executado corretamente, o resultado é imprevisível. Tendo essa somatória randômica, para encontrar o sorteado, basta calcular o resto da divisão inteira entre a soma e o número de participantes.

Este mesmo modelo de sorteio pode ser implementado tecnologicamente, através da técnica de *commit-reveal*, como descrito na próxima seção.

5.2 Formalização

Considerando um processo de sorteio qualquer, envolvendo N participantes numerados de 0 a $N-1$, para se escolher um deles, aleatoriamente, divide-se o processo em três fases: ***Commit***, ***Reveal*** e ***Resultado***.

Na fase de *Commit*, cada um escolhe um número inteiro de 1 a $(Z.N-1)$, onde Z é um inteiro positivo, e o envia de forma protegida para os outros participantes (fazem o *commit*).

Uma vez que todos escolheram e enviaram seus números secretos, inicia-se a

segunda fase (Reveal) onde todos revelam suas escolhas. O conjunto de todos os valores revelados forma o vetor values.

Caso todos reveals tenham sido enviados e validados, parte-se para a fase final (Resultado) onde o vencedor é calculado segundo a seguinte função, que retorna a posição (index) do participante sorteado:

$$winnerIndex(N, values) = (\sum_{i=0}^{N-1} values[i])\%N$$

Exemplo:

Index	Participante	Value
0	A	3
1	B	8
2	C	6

$$values = [3, 8, 6]$$

$$winnerIndex(3, values) = (3 + 8 + 6)\%3$$

$$winnerIndex(3, values) = (17)\%3 = 2$$

Portanto o participante C é o vencedor.

5.3 Análise da Distribuição de Probabilidade

O modelo de sorteio exposto acima garante a distribuição uniforme da probabilidade de um participante ser sorteado, desde que sejam respeitados os intervalos de números passíveis de escolha e a numeração dos participantes.

Não será explorado neste documento a demonstração matemática formal, porém, segue uma breve explicação, tomando um exemplo como base.

Exemplo:

Seja D um sorteio com $N = 2$ participantes (p_0 e p_1) e considerando que cada um deles escolhe um número inteiro qualquer entre 0 e $N-1 = 1$. O conjunto das $N^N = 4$ possíveis combinações de escolhas (C) é dado pelas tuplas a seguir, cuja ordem dos elementos é respectiva à ordem dos participantes:

$$C_{N=2} = (0, 0); (0, 1); (1, 0); (1, 1);$$

Somando-se os elementos de cada tupla, obtêm-se o seguinte conjunto S de possíveis somas resultantes:

$$S_{N=2} = 0; 1; 1; 2;$$

Nota-se que as somas se limitam entre 0 e $N * (N - 1) = 2$.

Ao encontrar o resto da divisão inteira das somas por N = 2, obtêm-se o conjunto R dos possíveis restos, ou seja, das possíveis posições sorteadas:

$$R_{N=2} = 0; 1; 1; 0;$$

Dessa forma, a probabilidade P(i) de um participante p_i ser sorteado é de:

$$P(0) = 2/4 = 50\%$$

$$P(1) = 2/4 = 50\%$$

O que caracteriza uma distribuição uniforme para este caso particular.

É possível ainda estender o intervalo de números possíveis, sem afetar a distribuição, permitindo que se escolha qualquer número inteiro entre 0 e $(Z * N - 1)$, onde Z é um número inteiro maior que 0. Por exemplo, aplicando $Z = 2$ no caso anterior, teríamos:

$$C_{N=2;Z=2} = (0,0)...(0,3); (1,0)...(1,3); (2,0)...(2,3); (3,0)...(3,3);$$

$$S_{N=2;Z=2} = 0; 1; 2; 3; 1; 2; 3; 4; 2; 3; 4; 5; 3; 4; 5; 6]$$

$$R_{N=2;Z=2} = 0; 1; 0; 1; 1; 0; 1; 0; 0; 1; 0; 1; 1; 0; 1; 0$$

O que manteria as probabilidades:

$$P(0) = 8/16 = 50\%$$

$$P(1) = 8/16 = 50\%$$

A partir da demonstração formal, é possível generalizar o observado nesse caso particular para N sendo qualquer número natural.

É interessante ressaltar que é possível adotar um modelo onde deseja-se uma distribuição não uniforme, um sorteio ponderado, em que pessoas podem ter probabilidades diferentes de serem escolhidas. Para isso, bastaria que um participante ocupasse não necessariamente uma posição, mas sim o número de posições relativas às chances de ser sorteado, como um sorteio de rifas.

6 ANÁLISE DE SEGURANÇA DO PROTOCOLO

A seguir, analisa-se como o mecanismo proposto se comporta mediante a diferentes tentativas de ataque.

6.1 Tentativa de obter as escolhas antes da revelação

Como descrito anteriormente, para que o sorteio seja justo, é necessário que o número resultante da soma dos *commits* dos usuários seja aleatório. Ele não o seria caso o sistema não garantisse confidencialidade aos participantes e um dos usuários soubesse das escolhas alheias antes de enviar o seu *commit*.

No entanto, para que um dos participantes obtenha todos os números enviados através dos *commits*, seria necessário inverter ou encontrar colisões para o *digest* resultante da função *hash*. Sendo a função de *hash* suficientemente validada e sendo os nonces sequências escolhidas de forma pseudo-aleatória e suficientemente grandes, esta tarefa não é computacionalmente viável.

6.2 Alteração da escolha

Na seção 4, foi explicado que a técnica de *commit-reveal* permite que os usuários testem se os dados enviados pelos outros participantes foram alterados ou não. Isso porque o sistema revela os *nonces* utilizados, de modo que cada usuário possa validar se o *commit* recebido é o mesmo que um *commit'* gerado utilizando o número revelado e o *nonce* disponibilizado. Por isso, caso alguma inconsistência seja notada, é possível invalidar o processo.

6.3 Resistência à revelação

Apesar do sistema não impedir que um dos usuários deixe de revelar sua escolha, é possível detectar qual dos integrantes que está pendente. Assim é possível tomar as medidas cabíveis para resolver a situação.

6.4 Envio de múltiplas escolhas

Um usuário pode querer enviar diferentes valores para diferentes participantes, o que resultaria em um valor final diferentes em cada uma das situações. O sistema previne que isso ocorra uma vez que não haveria consenso entre os participantes. Assim como no caso anterior, não é possível evitar que isso seja feito porém é possível identificar qual usuário que está agindo de maneira imprópria.

6.5 Combinação de valores entre participantes

Alguns usuários podem querer combinar valores para influenciar o valor do sorteio. Porém basta que um usuário não integre o conluio para que seu efeito seja irrelevante. Isso porque o grupo mal intencionado não terá controle sobre uma das escolhas e portanto não o terá sobre o valor final somado.

Vale mencionar que caso todos os participantes estejam envolvidos em um conluio, é possível dar a falsa impressão a um juiz que esteja acompanhando o processo de justiça. Sendo assim, é recomendável que se inclua participantes cujos interesses são opostos..

6.6 Alteração no envio de outro participante

Um usuário malicioso poderia tentar alterar o envio de outro usuário. Como esse novo *commit* teria sido gerado genuinamente pelo processo descrito na seção 4, sua validade seria confirmada pelos outros usuários.

A fim de evitar essas ocorrências, o sistema utiliza criptografia assimétrica, de modo que os *commits* tenham assinatura dos participantes. Assim, além da checagem da integridade, os *commits* tem suas autenticidades analisadas.

6.7 Servidor não confiável

O protocolo desenvolvido permite a possibilidade de um servidor coordenar o processo, o qual poderia tentar manipular os valores enviados pelos usuários a fim de obter vantagem do sorteio. Contudo, o projeto impede que isso ocorra, uma vez que os *commits* são assinados pelos remetentes. Portanto, para evitar esse tipo de comportamento, deve-se confirmar que os dados enviados pelo possível servidor são provenientes dos participantes do sorteio. Assim como descrito na seção anterior, a autenticidade precisa ser confirmada.

7 IMPLEMENTAÇÃO

O sistema foi implementado de forma modular, a fim de permitir que diferentes aplicações aproveitem o código como em um formato de *SDK (Software Development Kit)* ou *framework* (conjunto de bibliotecas e scripts para fornecer uma estrutura base no desenvolvimento de aplicações).

Dado que o grande diferencial do sistema está no processamento distribuído do sorteio, além da modularidade, o projeto exige a definição protocolos de comunicação entre os diversos elementos que o compõem, de forma que cada um seja agnóstico em relação aos detalhes de implementação do outro.

Dessa forma, todo o processo é baseado na troca de “mensagens” entre os usuários finais, participantes dos sorteios. O conjunto de todas essas mensagens unidas às regras de utilização definem o que chamaremos de **Protocolo Commit-Reveal**, cujos detalhes serão especificados posteriormente.

7.1 Arquitetura

As mensagens são trocadas entre as partes segundo uma estrutura de 4 camadas, onde cada camada tem suas responsabilidades e seus atributos.

7.1.1 Estrutura de Camadas

- **Camada 4 - Frontend**: é a camada onde está implementada a interface humano-computador, responsável por apresentar o status do processo, bem como fornecer meios para envio de mensagens;
- **Camada 3 - SDK**: sendo foco do projeto, é a camada que fornece os módulos de softwares necessários para comunicação entre a camada 2 e 4, transformando os comandos do frontend em mensagens formatadas segundo o Protocolo e

vice-versa;

- **Camada 2 - Comunicação**: é a camada que converte as mensagens do Protocolo Commit-Reveal para protocolos de comunicação de mais baixo nível, permitindo a transferência desses dados por uma tecnologia já existente; Esta camada pode abranger diferentes níveis de rede, de acordo com a implementação. Exemplos: HTTPS (aplicação) para acesso à um servidor WebSockets, comunicação serial (enlace), Bluetooth (física) etc;
- **Camada 1 - Transmissão**: é a camada que abstrai o meio pelo qual uma mensagem que sai de um dispositivo é direcionada e transmitida para outro; O que a determina é sua função de direcionar as mensagens dos remetentes para os destinatários corretos e não suas características. Portanto, essa camada pode ser representada por um servidor WebSockets, por um cabo, por frequências de ondas eletromagnéticas etc.

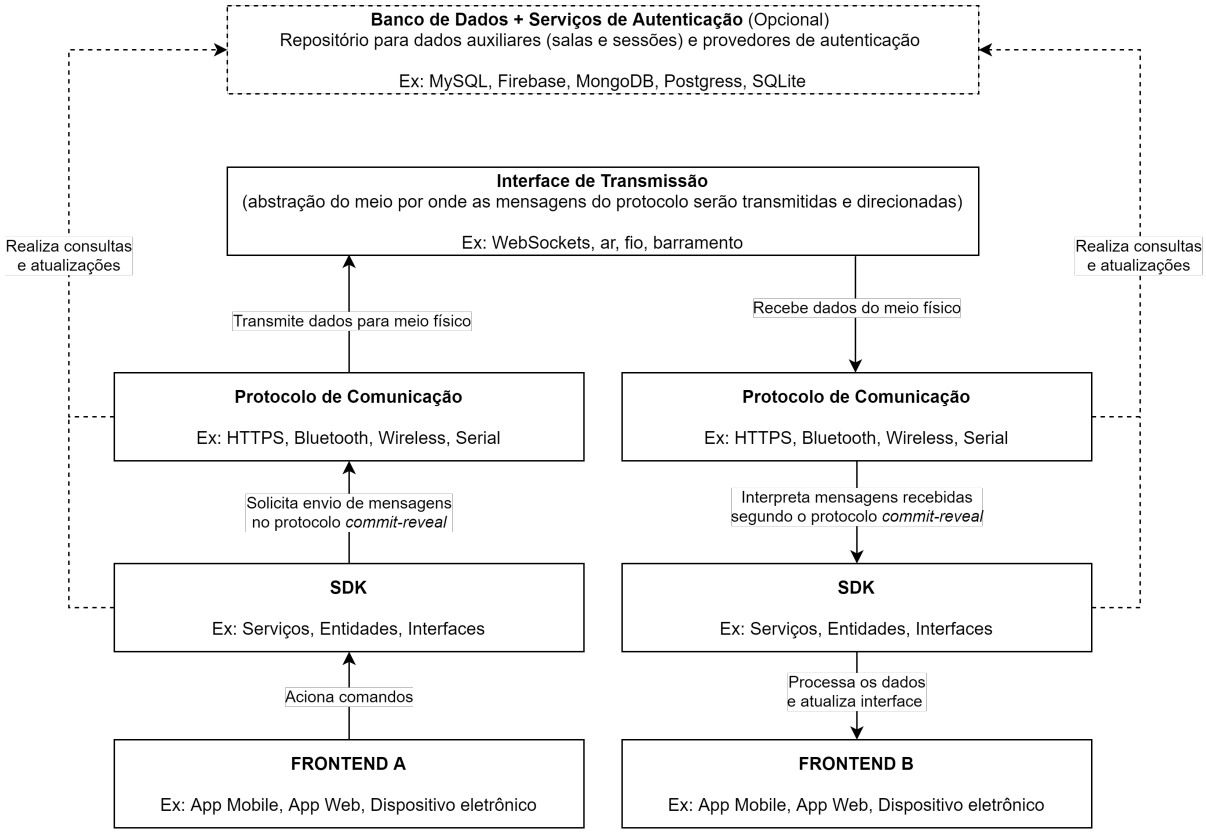

Figura 6: Estrutura de Camadas.

Adotando esse esquema de camadas é possível adaptar o SDK para comunicar-se com aplicações que não necessariamente tenham sido implementadas em uma

mesma tecnologia. Por exemplo, é possível que um usuário interaja no processo a partir de uma aplicação *mobile*, construída para sistema operacional *Android*, e outro interaja no mesmo processo através de uma interface física, construída em um *Arduino*.

Para tal, ambos os contextos devem possuir uma adaptação do SDK para as linguagens em que são programados (no caso, Java e C++ adaptado, respectivamente) e, especialmente, terem uma implementação da camada 2, de Comunicação, que permita a conversa entre os dois dispositivos. No exemplo, essa implementação de camada 2 poderia utilizar uma comunicação *Wireless* segura para converter as conversas em mensagens transimitidas pelo ar.

7.2 Componentes do Sistema

7.2.1 Módulos da biblioteca

O ***Frontend*** tem acesso a 3 principais módulos:

- ***Draw***: módulo que fornece entidades e funções relacionadas a sorteios;
- ***Commit-Reveal***: módulo que fornece interfaces e métodos para transformação de dados puros em modelos de Commits e Reveals, bem como validá-los;
- ***Security***: módulo que fornece métodos criptográficos importantes para segurança da aplicação, como geração de chaves, números aleatórios, assinaturas, etc.

Além disso, cada um desses módulos interagem entre si, de forma a reduzir ao máximo às tarefas necessárias para que o cliente atinja seus objetivos, concentrando tudo o que for possível no módulo ***Draw***.

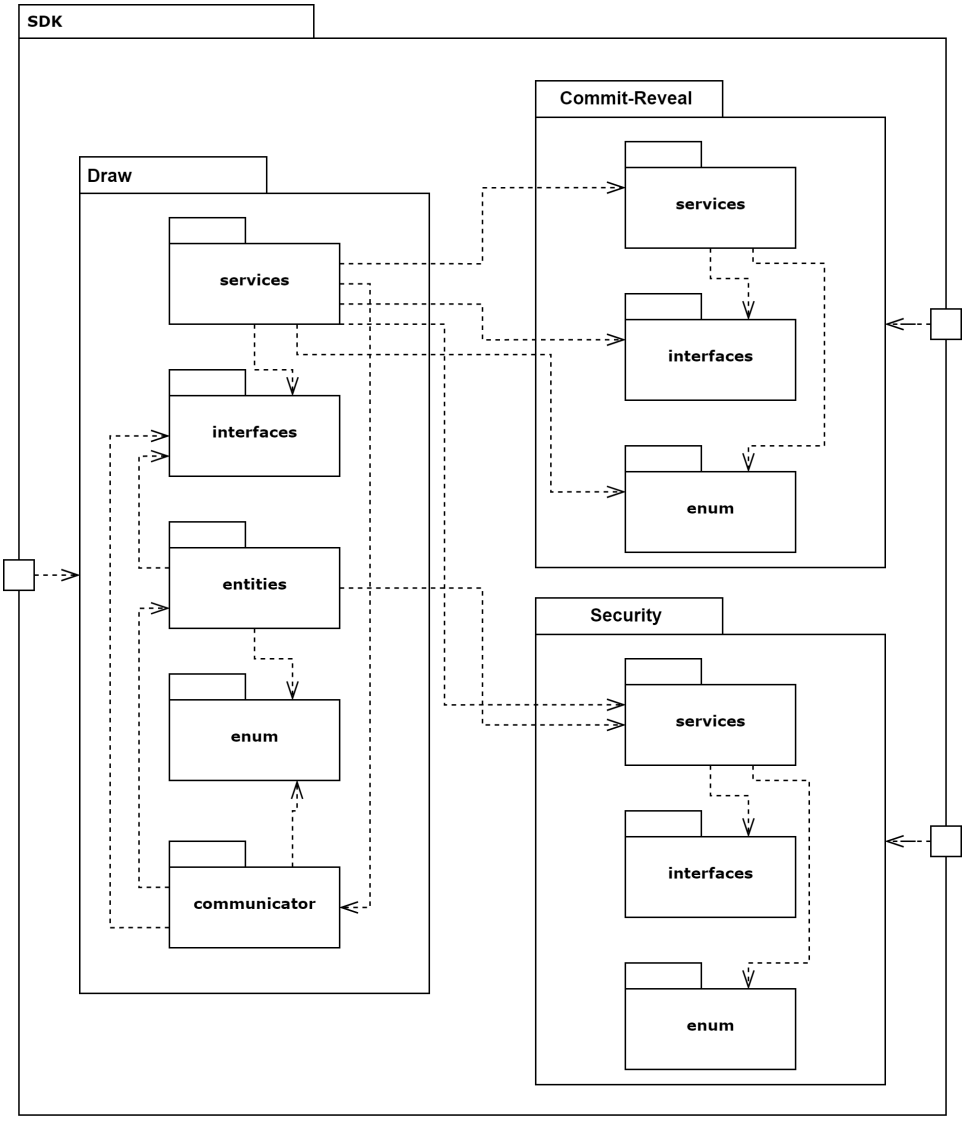

Figura 7: Divisão de módulos do sistema.

7.2.2 Nomenclatura de componentes

Os componentes são categorizados e nomeados segundo sua função principal e suas características, conforme descrito a seguir:

- **Entidades**: classes instanciáveis que representam modelos de dados importantes no processo, estruturadas em atributos e métodos. São passíveis de persistência (mesmo que parcial), tanto durante a execução quanto em repositórios não voláteis. Como exemplo, temos as entidades de Sorteio e de Participante;
- **Interfaces**: modelos de dados não persistentes e não instanciáveis, servindo apenas como formatação para transferência de informações. Por exemplo, um

evento de sorteio ou um *commit* protegido;

- **Serviços**: classes (geralmente estáticas - não instanciáveis) que fornecem métodos para manipular objetos (entidades ou interfaces) e acionar funções de outras camadas ou de outros módulos. Por exemplo: serviço de sorteios ou de o segurança;
- **Enumeradores**: estrutura de dados que definem opções fixas de valores que uma determinada propriedade pode assumir. Exemplos: estado de um sorteio e tipo de evento.

7.2.3 Interação entre Serviços e Camadas

O diagrama abaixo resume como os principais componentes da biblioteca interagem entre si e entre camadas.

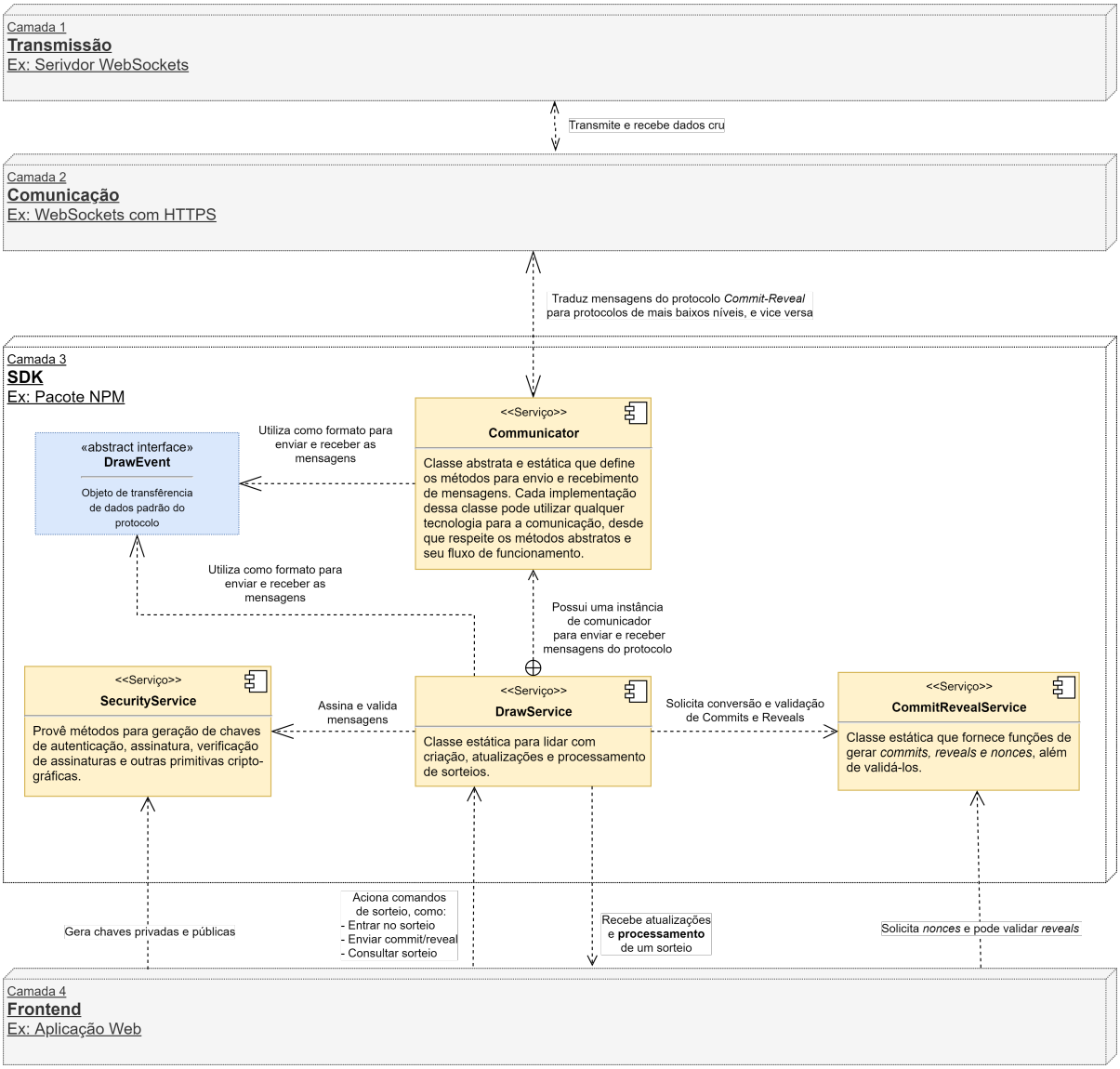

Figura 8: Fluxo de interação entre componentes e camadas.

7.3 Entidades

As entidades do sistemas concentram-se no módulo de Sorteios, sendo elas definidas abaixo.

7.3.1 *Draw <D>*

É a entidade que representa um processo único de sorteio, guardando suas informações de estado e fornecendo as interfaces corretas para consultá-las e alterá-las.

Propriedade	Tipo	Descrição	Acesso
uuid	*String*	Identificador único global do sorteio	Leitura
status	*DrawStatus*	Fase atual do sorteio	Leitura
spots	*Int*	Número de vagas disponíveis no sorteio	Leitura
data	*D* (genérico)	Informações gerais do sorteio	Leitura
stakeholders	*Stakeholder*[ ]	Lista de participantes (elegíveis ou não)	Leitura
candidates	*Candidate*[ ]	Lista de participantes elegíveis	Leitura
commits	*Commit*[ ]	Lista de commits registrados no processo	Leitura
reveals	*Reveal*[ ]	Lista de reveals registrados no processo	Leitura
winner	*Candidate*	Objeto do candidato sorteado	Leitura

Tabela 1: Propriedades da Entidade *Draw*

Além desses atributos públicos, a classe *Draw* também fornece diversos métodos para sua manipulação. Tais métodos permitem a leitura do estado atual do sorteio e também suas atualizações, como alteração do status, adicionar/remover um candidato, adicionar/remover commits e reveals, checar erros etc.

No geral, os métodos são mais acessados pela implementação do *DrawService*, de forma que para o desenvolvedor, apenas os métodos *getters* devem ser de maior interesse. Sendo assim, não será exposto nesse documento a especificação dos seus métodos, mas esta pode ser encontrada no repositório do projeto.

7.3.1.1 Status do Sorteio (*DrawStatus*)

Um sorteio pode estar em 5 diferentes estados, que são representados pelo enumerador *DrawStatus*.

Status	Valor	Descrição
PENDING	0	Sorteio ainda não pode começar pois há vagas não preenchidas.
COMMIT	1	Sorteio está na fase de commit e todos participantes devem enviar o seu.
REVEAL	2	Sorteio está na fase de reveal e todos participantes devem enviar o seu.
FINISHED	3	Sorteio foi concluído com sucesso.
INVALIDATED	-1	Sorteio foi invalidado devido alguma inconsistência ou desistência no processo.

Tabela 2: Opções de Status de um Sorteio - Enumerador *DrawStatus*

O status atual do sorteio é calculado pelo método ***draw****.updateStatus()* que retorna *true* caso haja mudança de estado e falso caso contrário. A lógica do método segue o diagrama de estados a seguir:

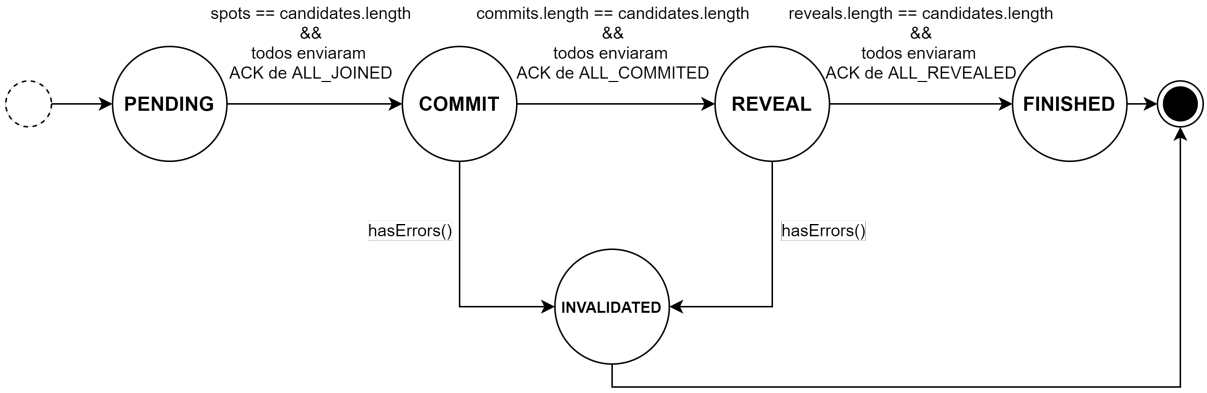

Figura 9: Diagrama de Estados de um Sorteio.

O funcionamento das *ACKs* será explicado mais adiante neste documento.

7.3.2 *Stakeholder <P>*

Representa um participante do sorteio. Seu *id* é fornecido pela instância de *Communicator* corrente e é um meio de associar o participante com algum banco de dados externo.

Propriedade	Tipo	Descrição	Acesso
id	*String*	Identificador único global do participante	Leitura/ Escrita
profile	*P* (genérico)	Informações de perfil do participante	Leitura/ Escrita
eligible	*Boolean*	Se participante é um candidato (true) do sorteio, ou se está apenas assistindo (false)	Leitura/ Escrita
publicKey	*JsonWebKey*	Chave pública compartilhada para verificação de assinatura	Leitura/ Escrita

Tabela 3: Propriedades da Entidade *Stakeholder*.

7.3.3 *Candidate*

Sendo classe filha de *Stakeholder*, representa um participante elegível do sorteio (candidato).

Propridade.	**Tipo**	**Descrição**	**Acesso**
		Estende *Stakeholder*	
indexes	int[ ]	Lista de números inteiros positivos que indicam as posições do sorteio pela qual o candidato concorre	Leitura

Tabela 4: Propriedades da Entidade *Candidate*

7.4 Interfaces

A seguir estão descritas as principais interfaces envolvidas no processo.

7.4.1 *RawCommit*

Commit antes de ser protegido.

Propridade.	**Tipo**	**Descrição**
data	any	Dados puros antes de serem criptografados
metadata?	*M (genérico)*	Meta informações associadas ao *commit*
nonce	*ArrayBuffer*	Sequência de bytes que serão o segredo usado para gerar a hash do *commit*
userId	string	Identificador externo do usuário que fez o commit. (Mesmo que o *id* do Stakeholder)

Tabela 5: Propriedades da Interface *RawCommit*

7.4.2 *Commit*

Commit após conversão e antes da assinatura.

Propridade.	**Tipo**	**Descrição**
digest	*ArrayBuffer*	Sequência de bytes resultante da função de hash.
hashFunction	*HashOptions*	Função de hash escolhida ao se fazer o commit.
timestamp	number	Momento em que o commit foi gerado.
userId	string	Identificador externo do usuário que fez o commit. (Mesmo que o *id* do Stakeholder)

Tabela 6: Propriedades da Interface *Commit*

7.4.3 *Reveal*

Revelação do *commit*. Estende *RawCommit*, adicionando-se o *timestamp* do momento de geração do *reveal*.

Propridade.	**Tipo**	**Descrição**
	Estende *RawCommit*	
timestamp	number	Momento em que o *reveal* foi gerado

Tabela 7: Propriedades da Interface *Reveal*

7.4.4 *SignedCommit*

Conjunto de um *Commit* e a assinatura digital do *Stakeholder* que o gerou.

Propridade.	**Tipo**	**Descrição**
commit	*Commit*	Commit gerado e criptografado.
signature	*ArrayBuffer*	Assinatura digital do commit.

Tabela 8: Propriedades da Interface *SignedCommit*

7.4.5 *SignedReveal*

Conjunto de um *Reveal* e a assinatura digital do *Stakeholder* que o gerou.

Propridade.	**Tipo**	**Descrição**
reveal	*Reveal*	Reveal de um participante.
signature	*ArrayBuffer*	Assinatura digital do reveal.

Tabela 9: Propriedades da Interface *SignedReveal*

7.5 Protocolo Commit-Reveal

As atualizações em um sorteio serão acionadas e recebidas através da troca de mensagens entre os dispositivos finais. Isso se dará conforme um motor de eventos, onde cada atualização irá disparar um evento pré-definido, de modo a executar suas respectivas funções de resposta. Esses eventos junto às fases de funcionamento constituem o Protocolo Commit-Reveal.

7.5.1 Jornada de um Evento

O processo se inicia de forma ativa, com os seguintes passos:

1. **Ação**: o usuário executa alguma ação, através do Frontend;

   Exemplo: clique no botão de enviar commit;

2. **Acionamento**: o Frontend aciona o respectivo método no DrawService;

   Exemplo: Aplicativo A chama o método ***DrawService****.sendSignedCommit(...)*;

3. **Tradução**: o DrawService gera o objeto de evento correspondente àquela ação e o envia para o Communicator;

   Exemplo: DrawService gera o evento COMMIT_RECEIVED, acoplando os dados do commit, e solicita que o Communicator o envie;

4. **Conversão**: o Communicator recebe o objeto de evento e o adequa para o envio segundo o seu protocolo de baixo nível;

   Exemplo: Communicator de WebSockets cria a requisição HTTPS que envia o evento para o servidor, adicionando as informações de autenticação;

5. **Transmissão**: o Communicator solicita efetivamente o envio da mensagem para os outros participantes, usando sua própria tecnologia.

   Exemplo: Communicator de WebSockets executa o HTTPS Request que é recebido pelo servidor, que, por sua vez, faz o *broadcast* para todos do canal de WebSockets.

Já do ponto de vista passivo, de quem recebe as mensagens, os passos seriam os inversos dos passos anteriores:

6. **Recebimento**: o protocolo de transmissão capta uma nova mensagem e direciona para ser tratada pelo Communicator

   Exemplo: recebimento de broadcast no canal de WebSockets;

7. **Desconversão**: o Communicator entende qual é a mensagem sendo passada e a transforma de volta em um objeto de evento, passando-o para o DrawService.

   Exemplo: Communicator lê broadcast recebido, interpreta como um evento de COMMIT_RECEIVED e o envia para DrawService;

8. **Tratamento**: o DrawService dispara as respectivas ações necessárias para tratar o evento recebido e informa Frontend da atualização.

   Exemplo: DrawService atualiza instância de Draw com o novo commit e notifica o Frontend da atualização;

9. **Exibição**: o Frontend recebe a atualização e a exibe para o usuário;

   Exemplo: Aplicativo mostra que outro usuário fez o commit;

10. **Confirmação**: o DrawService reconhece que o evento fora recebido e tratado e envia de volta uma mensagem de Acknowledge (ACK).

    Exemplo: Publicação do evento ACK no canal de WebSockets.

7.5.2 Interface *DrawEvent*

Para comunicação entre os diferentes Communicators e o DrawService, usa-se a seguinte interface que define os eventos e seus dados.

Propridade.	**Tipo**	**Descrição**
type	*DrawEventType*	Tipo de evento
timestamp	*Date*	Momento em que o commit foi gerado
drawUuid	*string*	Identificador único do sorteio
from	*Stakeholder*	Participante que gerou o evento
data	*Draw, Stakeholder, Commit, Reveal*, ...	Dados associados ao evento. Cada tipo de evento tem um respectivo tipo de dado

Tabela 10: Propriedades da Interface *DrawEvent*

7.5.2.1 *DrawEventType* - Tipos de Evento

Os tipos de eventos possíveis podem ser divididos em duas categorias: Normais e de Erros.

EVENTOS NORMAIS		
Tipo de Evento	**Descrição**	**Tipo de Dado**
DRAW_CREATED	Sorteio criado	*Draw*
DRAW_DELETED	Sorteio excluído	*Draw*
CANDIDATE_SUBSCRIBED	Novo participante em um sorteio	*Candidate*
COMMIT_RECEIVED	Novo commit em um sorteio	*Commit*
REVEAL_RECEIVED	Novo reveal em um sorteio	*Reveal*
STATUS_CHANGED	Status do sorteio alterado	*DrawStatus*
ACK	Confirmação de fim de etapa	*DrawAck*

Tabela 11: Tipos de Eventos Normais

EVENTOS DE ERRO		
Tipo de Evento	**Descrição**	**Tipo de Dado**
CANDIDATE_UNSUBSCRIBED	Participante saiu do sorteio	*Candidate*
WRONG_COMMIT_FORMAT	Commit mal formato identificado	*SignedCommit*
WRONG_REVEAL_FORMAT	Reveal mal formato identificado	*Reveal*
DUPLICATE_COMMIT	Commit recebido mais de uma vez	*SignedCommit*
DUPLICATE_REVEAL	Reveal recebido mais de uma vez	*Reveal*
INVALID_REVEAL_MASK	Reveal inconsistente com commit identificado	*Reveal*
FORBIDDEN_COMMIT_USER_ID	Commit feito por usuário não presente no sorteio	*SignedCommit*
FORBIDDEN_REVEAL_USER_ID	Commit feito por usuário não presente no sorteio	*Reveal*
UNAUTHORIZED_COMMIT_SIGNATURE	Commit com assinatura não aceita	*SignedCommit*
UNAUTHORIZED_REVEAL_SIGNATURE	Reveal com assinatura não aceita	*Reveal*

Tabela 12: Tipos de Eventos de Erro

7.5.2.2 ACK - Consenso de Estado

Faz parte do escopo das camadas de Comunicação e Transmissão garantir que os eventos emitidos por um participante seja recebido pelos outros, solicitando reenvio quando há perdas. Entretanto, o SDK também deve lidar com confirmações de recebimentos quando em momentos de mudança de estado, assegurando que todos participantes estão sincronizados e processando os eventos da mesma forma.

Sendo assim, há um tipo de evento (“*ACK*”) disparado automaticamente pelo

DrawEventEngine sempre que um participante detecta que sua instância de sorteio atingiu uma condição passiva de mudança de estado. Por exemplo, quando um participante nota que recebeu todos os commits do sorteio e que todos estão válidos, ele emite uma *ACK* do tipo "***ALL_COMMITED***" informando que está pronto para ir para a próxima etapa do processo. Junto ao evento, é inserido os registros de todos commits que recebera, para que os outros participantes possam conferir com os registros deles.

Completando o processo de consenso, sempre que um participante recebe um evento do tipo *ACK*, ele deve registrá-lo em sua instância de sorteio e checar se os dados informados no conteúdo do evento conferem com os que ele recebeu previamente. Caso haja inconsistência, um evento de *INVALID_ACK* é disparado.

Quando todos participantes tiverem enviado suas *ACKs* e caso todas elas estiverem válidas, a instância de sorteio é atualizada para o novo estado e o evento de *STATUS_CHANGED* é disparado ao grupo.

As possibilidades de eventos de *ACK* estão listadas na tabela a seguir.

Tipo de Ack.	**Tipo de Dados**	**Descrição**
ALL_JOINED	*Candidate*[ ]	Todas vagas foram preenchidas
ALL_COMMITED	*Commit*[ ]	Todos commits recebidos e validados
ALL_REVEALED	*Reveal*[ ]	Todas reveals recebidos e validados
FINISHED	*Candidate*	Sorteio foi finalizado

Tabela 13: Possibilidades de Eventos *ACK*

7.5.3 Motor de Eventos

Dentro do módulo de de Sorteios, existe uma classe estática auxiliar chamada *DrawEventEngine*. Esta classe define as ações a serem tomadas com a instância de *Draw* após o recebimento de cada tipo de evento.

O diagrama a seguir exibe de forma simplificada como que os principais eventos do processo são tratados.

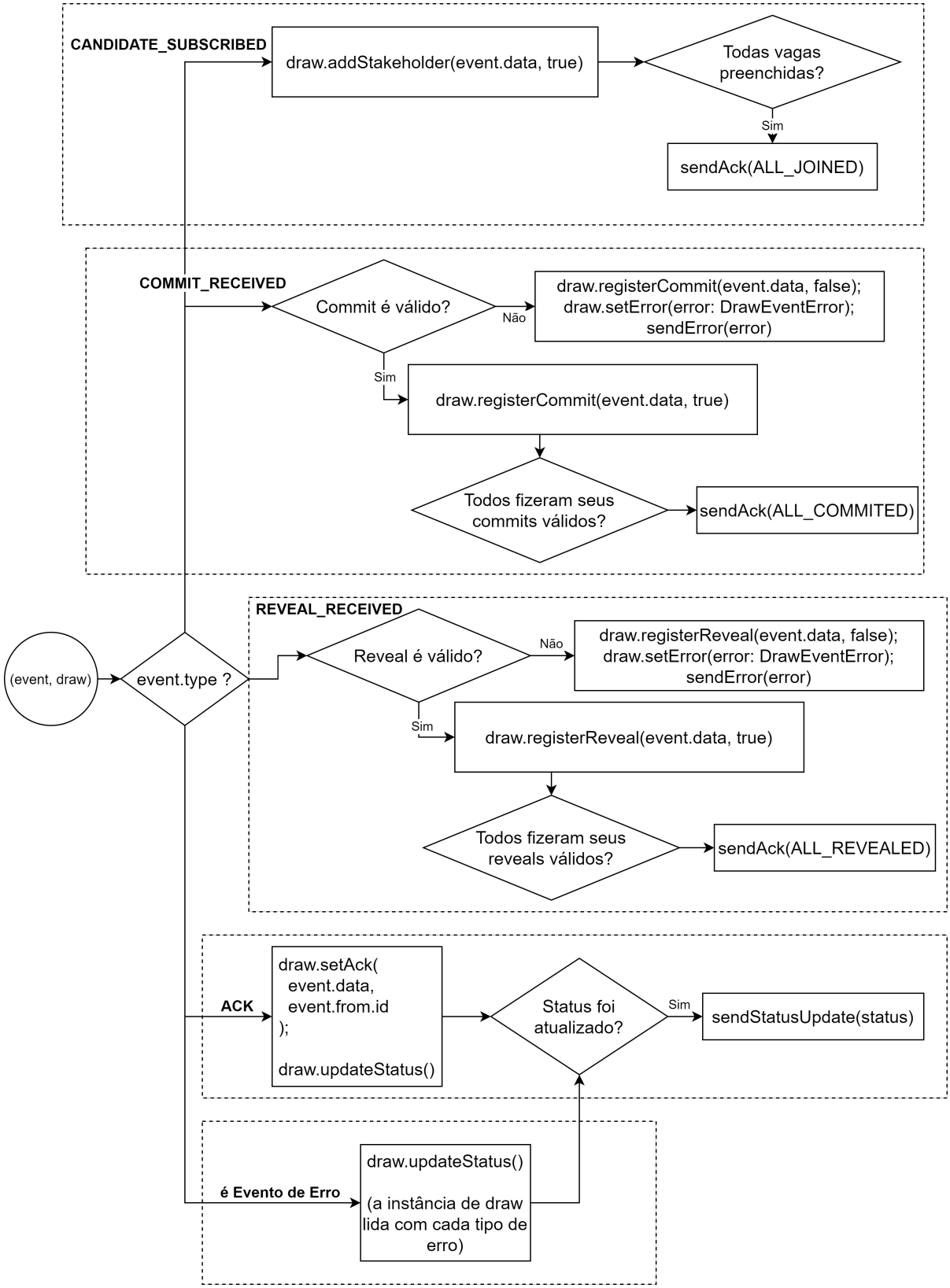

Figura 10: Tratamento de Eventos na classe *DrawEventEngine*

7.6 Serviços

O principal meio de acesso aos recursos fornecidos pelo SDK é através dos métodos definidos e implementados nos serviços. Sendo assim, entender o funcionamento de cada um deles e como interagem entre si é de suma importância para o uso correto da biblioteca.

7.6.1 *Communicator <P, C>*

O *Communicator* é a classe que fará a interface entre o *DrawService* (principal representante da camada 3 do SDK) e a camada 2 de Comunicação. Sendo assim, dependendo do sistema de comunicação sendo adotado no processo, o comunicador deve assumir implementações diferentes. Portanto, a classe *Communicator* é uma classe abstrata, isto é, apenas uma definição restrita de quais métodos, parâmetros e retornos todas e quaisquer implementações dela devem seguir.

Sendo assim, para todo projeto que for usufruir do SDK, deve-se desenvolver (ou utilizar alguma implementação pronta de) uma classe filha de *Communicator*, que estende as propriedades da classe pai, seguindo rigidamente as suas definições. Como exemplo, caso se utilize um serviço de WebSockets como camadas 1 e 2, deve-se criar uma classe *WebSocketsCommunicator*, filha da classe *Communicator*.

Esta **classe filha** de *Communicator* deverá ser acoplada ao *DrawService*, como será exposto mais adiante.

A seguir, explica-se as responsabilidades de cada método do *Communicator*, bem como seus parâmetros e retornos:

7.6.1.1 *openConnection(userId: string, params?: P)*

```
abstract async openConnection(userId: string, params?: P): Promise<
```

Este método deve configurar e iniciar a conexão com as camadas abaixo, a partir dos parâmetros de entrada, podendo retornar uma instância de conexão quando necessário. É o momento para o primeiro handshake entre o dispositivo e a rede em que vai se comunicar. Caso haja necessidade, é também ele que deve cuidar da autenticação de usuário, negando aqueles com credenciais inválidas.

Neste método também deve-se receber e registrar o *userId* que será usado para

identificar o *Stakeholder* do dispositivo durante toda comunicação. Caso não seja passado esse identificador, o eventos enviados não serão associados a um usuário e o processo falhará.

7.6.1.2 *closeConnection()*

```
abstract async closeConnection(): Promise<boolean>;
```

Este método encerra a conexão estabelecida previamente, apagando os tokens e credenciais de autenticação.

Retorna *true* caso a conexão tenha sido fechada com sucesso e *false* caso contrário.

7.6.1.3 *getDrawsList(page: number, perPage: number)*

```
abstract async getDrawsList(
    page: number,
    perPage: number
): Promise<PaginationResponse<Draw>>;
```

Faz uma consulta síncrona à lista de sorteios armazenadas nas camadas baixo, retornando uma lista de Draws, subconjunto do total.

Retorna um objeto de paginação com os sorteios selecionados.

Parâmetros:

- ***page*** - página de consulta da lista
- ***perPage*** número de itens por página

7.6.1.4 *subscribeToDrawsList()*

```
abstract subscribeToDrawsList(): Promise<Observable<Draw[]>>;
```

Retorna um observável (emitidor de eventos) que traz a lista de sorteio atualizada sempre que ela sofre alteração. Útil para conexões persistentes.

7.6.1.5 *createDraw(draw: Draw)*

```
abstract async createDraw(draw: Draw): Promise<DrawEvent>;
```

Cria (persiste) ou comunica os outros participantes de um novo objeto de sorteio. Retorna um promessa de evento de *DRAW_CREATED*.

Parâmetros:

- ***draw*** - Objeto com dados obrigatórios para criação do sorteio

7.6.1.6 *getDraw(uuid: string)*

```
abstract async getDraw(uuid: string): Promise<Draw>;
```

Retorna os dados públicos atuais do sorteio com determinado *uuid*.

Parâmetros:

- ***uuid*** - Identificador único do sorteio

7.6.1.7 *joinDraw(uuid: string)*

```
abstract async joinDraw(uuid: string): Promise<true>;
```

Adiciona o participante atual como candidato no sorteio com determinado *uuid*.

Retorna *true* se obteve sucesso, e *false* caso contrário.

Parâmetros:

- ***uuid*** - Identificador único do sorteio

7.6.1.8 *leaveDraw(draw: Draw)*

```
abstract async leaveDraw(draw: Draw): Promise<true>;
```

Remove o participante atual de um processo de sorteio.

Retorna *true* se obteve sucesso, e *false* caso contrário.

Parâmetros:

- ***uuid*** - Identificador único do sorteio

7.6.1.9 *broadcast(event: DrawEvent)*

```
abstract async broadcast(event: DrawEvent): Promise<boolean>;
```

Envia um evento para todos dispositivos na conexão atual.

Retorna *true* se obteve sucesso, e *false* caso contrário.

Parâmetros:

- ***event*** - DrawEvent a ser enviado

7.6.1.10 *post(event: DrawEvent, uuid: string)*

```
abstract async post(event: DrawEvent, uuid: string): Promise<boolea
```

Envia um evento para todos dispositivos inscritos em um determinado processo de sorteio.

Retorna *true* se obteve sucesso, e *false* caso contrário.

Parâmetros:

- ***event*** - DrawEvent a ser enviado
- ***uuid*** - Identificador único do sorteio

7.6.1.11 *listen(uuid: string)*

```
abstract async listen(uuid: string): Promise<Observable<DrawEvent |
```

Retorna um observável (emitidor de eventos) que traz todos DrawEvents recebidos em um canal de sorteio.

Parâmetros:

- ***uuid*** - Identificador único do sorteio

7.6.2 *DrawService <D>*

Sendo o serviço central do SDK, o *DrawService* permite que o cliente solicite e receba atualizações dos sorteios, integrando os outros serviços da biblioteca. Por ser uma classe estática, o serviço de sorteios é o mesmo durante todo ciclo de vida da aplicação, de forma que o único estado que ele guarda é a instância de *Communicator* responsável por traduzir os comando adivindos do *DrawService*.

De modo geral, após a inicialização de qualquer aplicação que implemente o SDK, deve-se chamar o método *DrawService.setCommunicator(communicator)*, que acopla

uma instância de *Communicator* com conexão já estabelecida para todo escopo da aplicação. Sem fazer isso, qualquer chamada a outros métodos do serviço jogará uma exceção.

Tendo acoplada a conexão, o fluxo de uso do DrawService é bastante simples. No geral, a ordem de chamadas ao serviço se dará da seguinte maneira:

1. Acopla-se um Communicator, usando o método ***setCommunicator(communicator)***
2. Obtêm-se a lista de sorteios, usando o método ***subscribeToDrawList()***
3. Cria-se um sorteio, usando o método ***createDraw(draw)***
4. Obtêm-se as informações de um sorteio, usando o método ***getDraw(draw)***
5. Inscreve-se no sorteio, usando o método ***joinDraw(draw)***
6. Envia-se um commit assinado, usando o método ***sendSignedCommit(draw, rawCommit, privateKey)***
7. Envia-se um reveal assinado, usando o método ***sendSignedReveal(draw, reveal, privateKey)***
8. Sai-se do soretio, usando o método ***leaveDraw(draw, reveal, privateKey)***

Além desses, existem alguns outros métodos públicos auxiliares que são usados pelo próprio *DrawService* em sua implementação. Porém, não é necessário lidar com eles para implementar os fluxos principais do processo de sorteio, já que o *DrawService* faz por si só a ponte entre o *Communicator*, o *DrawEventEngine* e a instância de *Draw*.

A referência completa dos métodos pode ser encontrada no repositório.

7.6.3 *CommitRevealService*

O *CommitRevealService* fornece os métodos para se lidar com a geração e validação de Commits e Reveals.

Como seu funcionamento é bem simples e o *DrawService* já lida internamente com ela, não passando essa responsabilidade para o desenvolvedor, ela não será detalhada neste documento. Porém, pode-se entender seus detalhes visitando o repositório do projeto.

7.6.4 *SecurityService*

O *SecurityService* é basicamente uma interface para se acessar os métodos criptográficos necessário no processo.

Como assinatura de mensagens é essencial para garantir a segurança do processo, o desenvolvedor deverá chamar o método ***generateKeyPair()*** do serviço para obter um par de chaves privada e pública. Tendo gerado essas chaves, deve-se armazenar a chave privada de forma segura no dispositivo, usando o método ***exportKey(key: CryptoKey)*** para converter o os *bytes* da chave em um formato armazenável.

Ao inicializar a aplicação, deve-se obter a chave privada armazenada para que sejam assinados os *commits* e *reveals* no *DrawService*. Além disso, deve-se transmitir de alguma forma confiável a chave pública do participante para os outros, permitindo que eles verifiquem as assinaturas recebidas. Para tal, recomenda-se utilizar alguma entidade certificadora ou algum provedor de autenticação com esse recurso, facilitando o processo.

Um bom lugar para se inserir essa lógica de divulgação da chave pública é no método *openConnection* do *Communicator* sendo empregado.

8 RESULTADOS

Ao fim do processo de desenvolvimento, obteve-se com sucesso a definição de um SDK base que validasse o processo de sorteio, tendo como resultados um pacote de software extensível e uma implementação dele num contexto de aplicativo mobile, comunicando-se através de um servidor de WebSockets.

8.1 Tecnologias e ferramentas

No projeto em questão, o SDK foi construído em JavaScript utilizando a *engine* Node.js. Isso porque a comunidade envolvendo Node.js é uma das maiores da atualidade e existe uma grande estrutura para compartilhamento de pacotes prontos para uso, através de gerenciadores de pacotes como NPM (Node Package Manager). Fora isso, é possível adotar TypeScript (uma versão tipada do JavaScript) que é muito versátil e possui várias ferramentas interessantes.

Para a camada de Frontend (aplicativo), adotou-se a *framework* híbrida (*mobile* e *web*) Ionic 5, também construída em JavaScript. Esta escolha se dá pela versatilidade em poder gerar um aplicativo para celular e para navegador a partir de um mesmo código fonte.

Para funcionar como camada 1 (de transmissão), criou-se *backend* simples com um servidor Web Sockets em Node.js, utilizando-se a *framework* NestJS como base. O motivo da escolha é remover limitações de espaço, e facilitar a distribuição das informações para vários participantes.

8.2 Código fonte

O código sendo desenvolvido pode ser acompanhado em: https://github.com/giabeni/fair-drawings-sdk.

8.3 Aplicação de Demonstração

Para ilustrar o funcionamento base do *commit-reveal* e guiar o desenvolvimento das funções principais, um aplicativo *mobile* de demonstração foi desenvolvido.

8.3.1 Objetivo

O aplicativo tem como objetivo principal permitir sorteios justos entre usuários, utilizando a técnica de *commit-reveal*.

8.3.2 Descrição

Os usuários devem se autenticar através de uma conta Google para acessar o sistema. Após isso, é possível criar grupos de sorteios (”salas”), especificando seu nome, descrição e número de participantes. Com as salas criadas, outros usuários poderão entrar. Ao atingir o número de participantes necessários, o processo de sorteio é iniciado e o *app* fornecerá uma interface para cada passo envolvido no processo: *commit*, *reveal*, validação e definição do vencedor. Em cada fase do processo, os usuários podem acompanhar o status da sala e verificar o *commit/reveal* de cada participante.

Para testar a integridade do sistema, será permitido pelo app enviar um *reveal* falso (não correspondente ao *commit* enviado). Se isso ocorrer, os outros participantes serão alertados imediatamente e o sorteio será cancelado.

8.3.3 Protótipo

Um protótipo de média fidelidade foi elaborado para ilustrar as telas do aplicativo e guiar o desenvolvimento do SDK. Ele pode ser navegado dinamicamente através do link https://marvelapp.com/e2i0437.

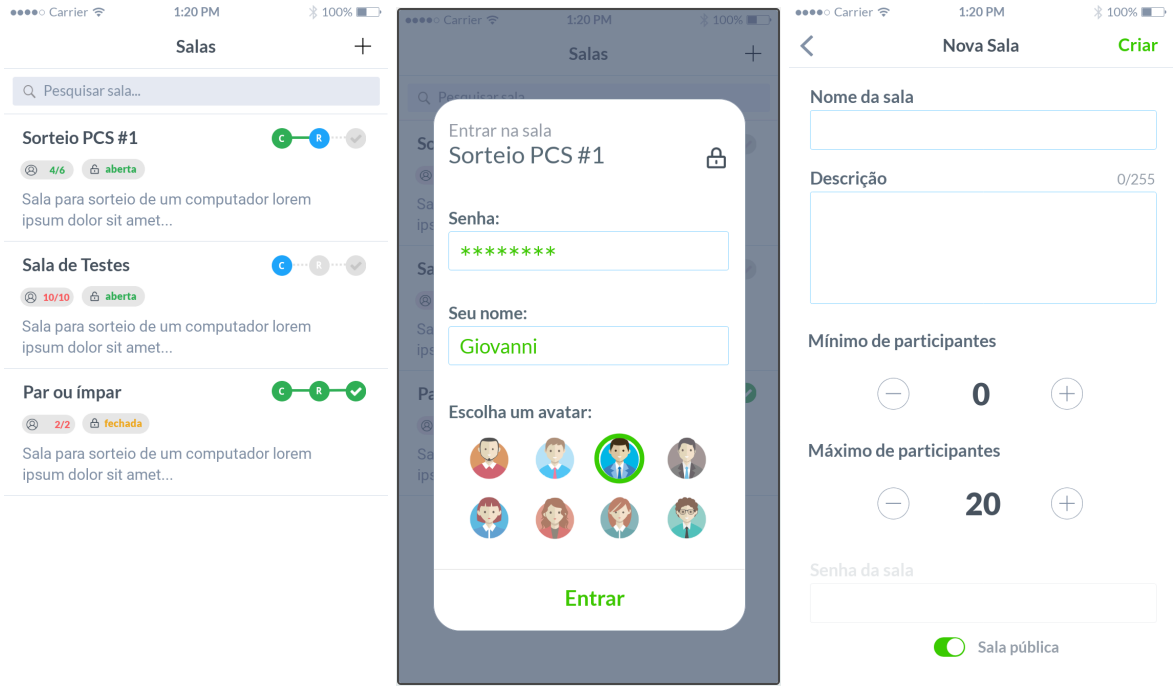

Figura 11: Protótipo - Telas de listar sala, entrar em sala e criar sala.

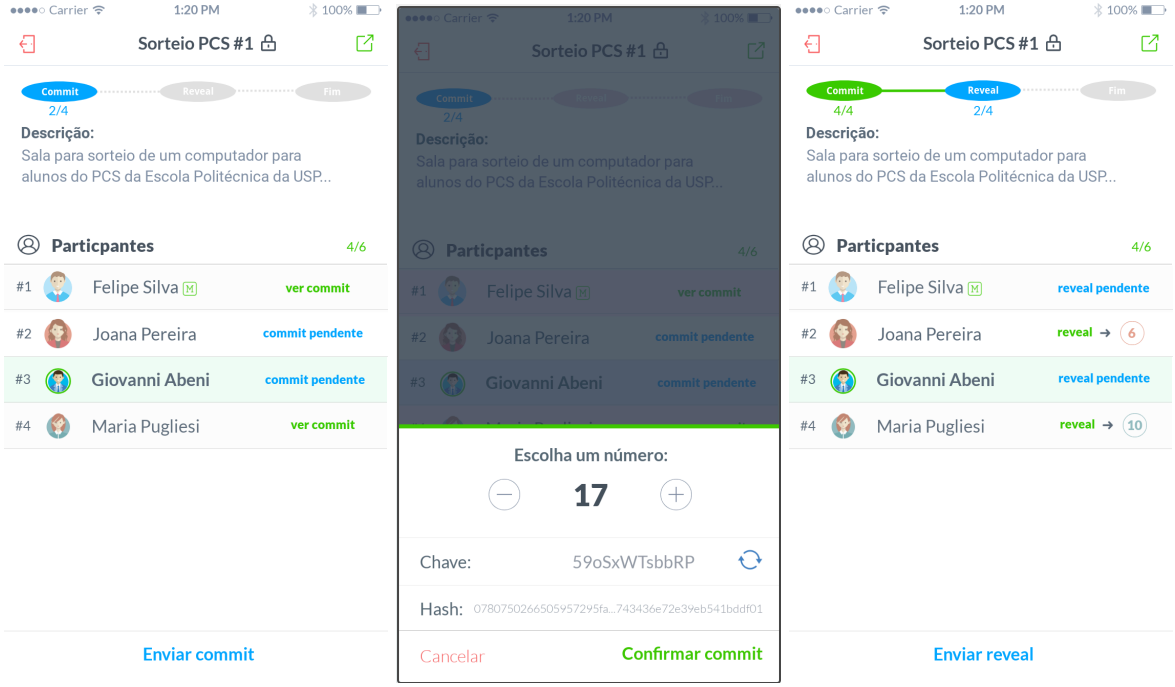

Figura 12: Protótipo - Telas de fase de commit, enviar commit e fase de reveal.

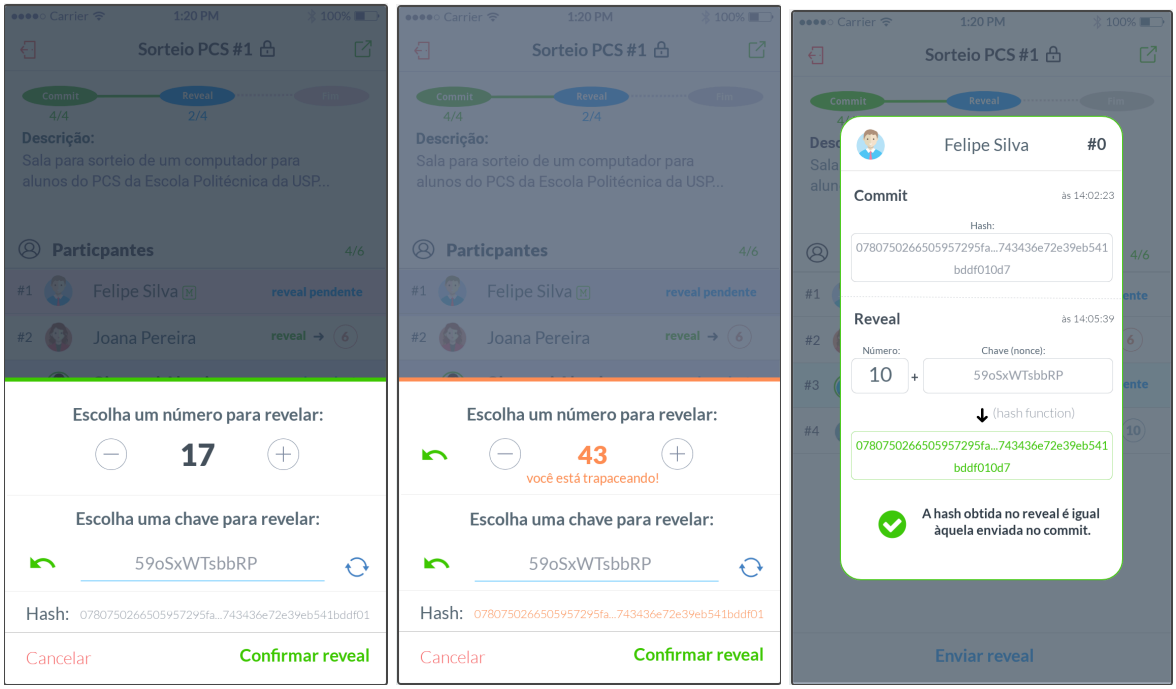

Figura 13: Telas de enviar reveal correto, enviar reveal falso e checar reveal.

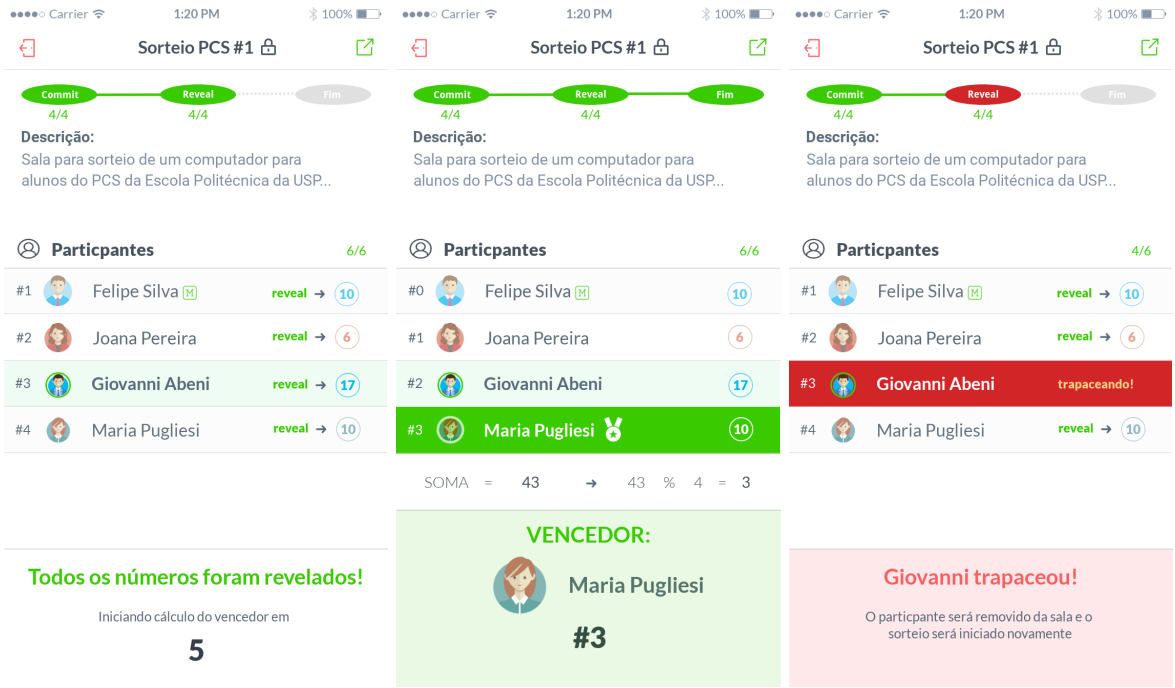

Figura 14: Telas de contagem regressiva, determinação do vencedor e trapaça.

8.3.4 Capturas de Telas do Resultado

O resultado final da implementação do aplicativo foi bastante satisfatório e interessante para ilustrar o processo de sorteio. Obteve-se uma fidelidade muito alta aos protótipos desenvolvidos, exceto algumas melhorias em fluxos.

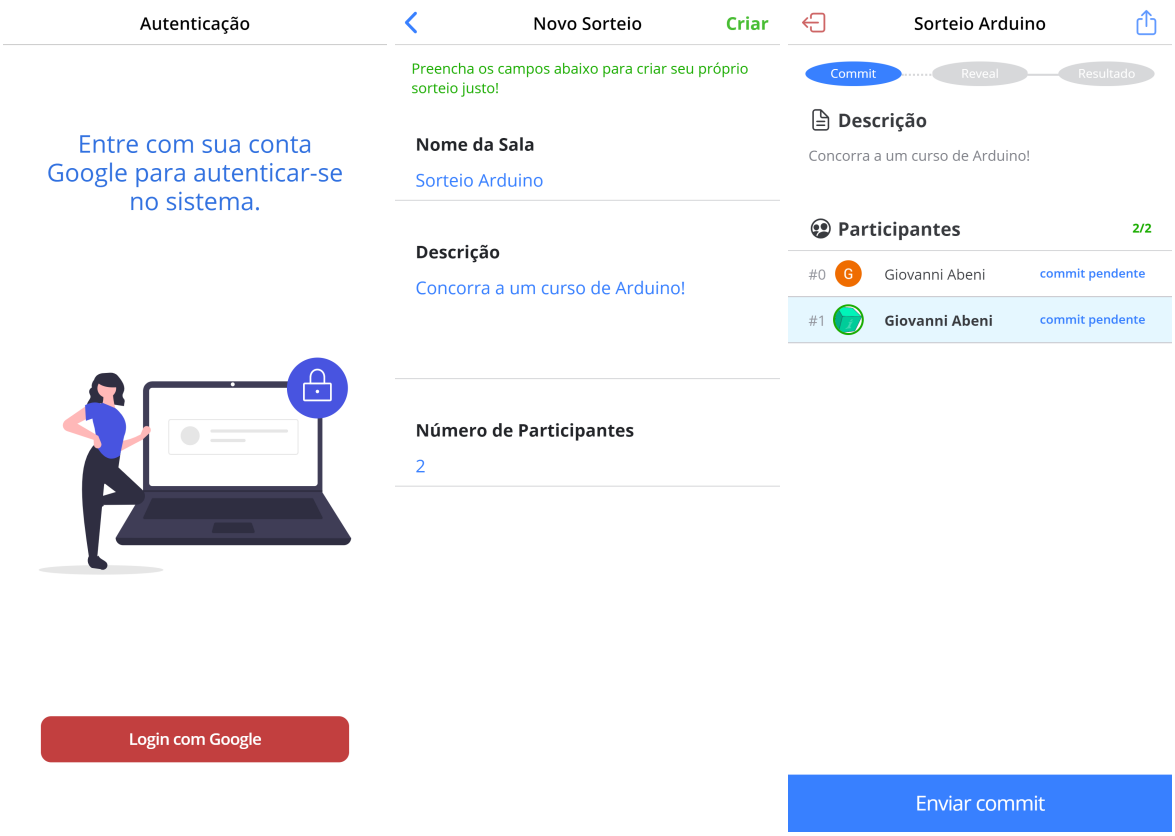

Figura 15: Telas de login, criar sorteio e página do sorteio

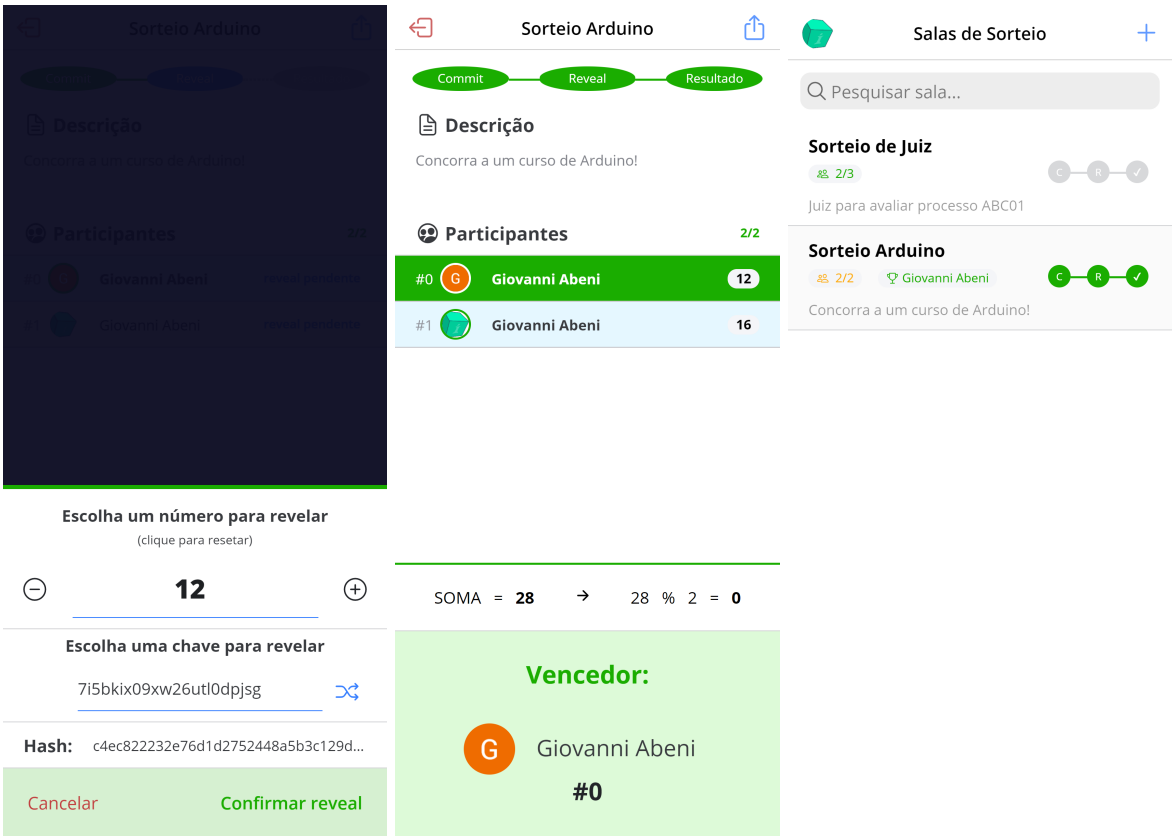

Figura 16: Telas de enviar reveal, ver vencedor e lista de salas

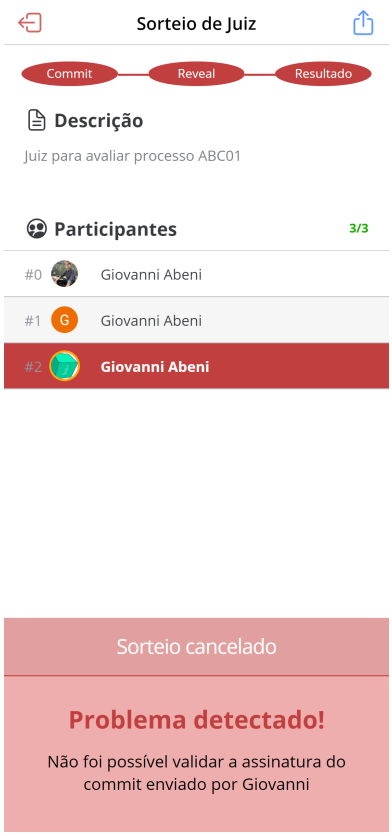

Figura 17: Tela de erro detectado

9 CONCLUSÃO

Ao longo do projeto, foi possível perceber a importância da imparcialidade e da segurança em processos de sistemas do cotidiano. Estes os quais podem ser alvo de programações traiçoeiras, passíveis de inserção de *backdoors* que possam alterar seus resultados de modo a favorecer partes envolvidas nos processos. O SDK desenvolvido teve seu objetivo atingido por: i) conseguir criar um sorteio verdadeiramente aleatório (justo); ii) sem que haja manipulações dos resultados objetivos, o que abriria a possibilidade de favorecimento para alguma parte envolvida na situação; e iii) de uma maneira auditável, que apesar de ter seu código aberto não tem sua segurança questionada.

Essas características puderam ser comprovadas pelo aplicativo *web* de demonstração, através do qual foi possível realizar testes do funcionamento da aplicação de sorteio do SDK. Além de trazer uma visão mais visual do passo a passo do sorteio baseado na técnica de *commit-reveal*, o aplicativo ainda possibilitou a simulação de situações em que houvesse um participante-atacante que tentasse manipular os resultados do sorteio. Como era de se esperar, esse tipo de ataque foi detectado pelo sistema, cuja reação foi avisar aos outros participantes do ocorrido e invalidar o sorteio.

Além disso, no projeto explicou-se detalhadamente as configurações necessárias para que o módulo do SDK possa ser integrado em aplicações de terceiros. Com isso espera-se que o processo de inclusão da ferramenta em futuros sistemas seja o mais simples possível. Vale destacar o potencial do *software* criado para aplicações legais, como o sorteio de processos judiciais entre os ministros que compões o Supremo Tribunal Federal. Por se tratar de uma aplicação justa que pode ser analisada e não depende de uma entidade reguladora, sua utilização poderia ser explorada nessa situação.

Por fim, vale ressaltar a possibilidade de extensão e revisão do SDK desenvolvido, a fim de promover melhorias e maior aplicabilidade. Como próximos passos, o SDK

desenvolvido será revalidado e terá sua documentação mais detalhada, para adaptá-lo a uma publicação na comunidade, facilitando sua divulgação e acesso.

10 REFERÊNCIAS

AGGARWALO, Sunny. **A Beginner's Guide to Ethereum and Dapp Development.** [*S. l.*: *s. n.*], 2017. *E-book*.

SILVA, Marcos *et al.* **A Fair, Traceable, Auditable and Participatory Randomization Tool for Legal Systems.** Universidade de São Paulo, 2020.

YAN, Katherine. **Commit and Reveal**: Adapting an Old Scheme for New Tricks. Medium, 2019. Disponível em: https://medium.com/polyswarm/commit-and-reveal-adapting-an-old-scheme-for-new-tricks-e3eafa0b3714. Acesso em: 5 jul. 2020.

GRIFFITH, Austin Thomas. **Commit Reveal Scheme on Ethereum**. Gitcoin, 2018. Disponível em: https://gitcoin.co/blog/commit-reveal-scheme-on-ethereum/. Acesso em: 5 jul. 2020.