A melhor abordagem para renderizar o PDF com controle total é usar a biblioteca **PDF.js** da Mozilla, em vez de <iframe> ou <embed>, que oferecem pouquíssimo controle sobre a barra de ferramentas (download, impressão, etc.).

Estrutura do Projeto

Vamos criar três arquivos:

1. index.html: A estrutura da nossa página.
2. style.css: Os estilos para desabilitar a seleção e criar uma camada de proteção.
3. script.js: A lógica para carregar o PDF e desabilitar as ações do usuário.

Passo 1: O HTML (index.html)

O HTML será simples. Incluiremos a biblioteca PDF.js a partir de uma CDN (Content Delivery Network) para facilitar. Teremos um div que servirá como contêiner para as páginas do nosso PDF.

HTML

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Visualizador de PDF Seguro</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <h1>Visualizador de PDF Protegido</h1>
    <p>O conteúdo abaixo é apenas para visualização.</p>

    <div id="pdf-viewer-container">
        <div id="pdf-renderer"></div>
        <div class="no-copy-overlay"></div>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/pdf.js/2.10.377/pdf.min.js"></script>

    <script src="script.js"></script>
</body>
</html>
```

Passo 2: O CSS (style.css)

Aqui está a mágica para dificultar a vida do usuário que tenta copiar ou interagir indevidamente.

- user-select: none;: Impede que o usuário selecione o texto na página.
- .no-copy-overlay: Criamos uma camada transparente por cima do visualizador. Isso pode interferir com algumas ferramentas de captura de tela mais simples e ajuda a capturar eventos de clique.

CSS

```css
body {
    font-family: sans-serif;
    background-color: #f0f0f0;
    text-align: center;

    /* Desabilita a seleção de texto em toda a página */
    -webkit-user-select: none; /* Safari */
    -ms-user-select: none; /* IE 10+ */
    user-select: none;
}

#pdf-viewer-container {
    width: 80%;
    max-width: 900px;
    height: 80vh;
    margin: 20px auto;
    border: 1px solid #ccc;
    overflow-y: scroll; /* Permite rolar o conteúdo do PDF */
    position: relative; /* Necessário para o posicionamento da camada de sobreposição */
    background-color: #fff;
    box-shadow: 0 4px 8px rgba(0,0,0,0.1);
}

#pdf-renderer canvas {
    /* Garante que o canvas de cada página ocupe toda a largura */
    display: block;
    width: 100%;
```

```
    margin: 0 auto;
    margin-bottom: 10px; /* Espaço entre as páginas */
}

/* Camada de sobreposição para dificultar cópias e cliques */
.no-copy-overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    z-index: 10; /* Fica acima do conteúdo do PDF */
    /* Deixamos transparente para que o usuário veja o conteúdo */
    background: transparent;
}
```

Passo 3: O JavaScript (script.js)

Este é o coração da nossa aplicação. Ele fará duas coisas:

1. **Bloquear Ações:** Desabilitar o menu de contexto (botão direito) e atalhos de teclado comuns (Ctrl+C, Ctrl+P).
2. **Renderizar o PDF:** Usar o PDF.js para carregar o PDF de uma URL e desenhá-lo dentro do nosso div.

JavaScript

```
// --- PARTE 1: BLOQUEIO DE AÇÕES DO USUÁRIO ---

// 1. Bloquear o menu de contexto (botão direito do mouse)
document.addEventListener('contextmenu', function(e) {
    e.preventDefault();
    alert('Esta função foi desabilitada para proteger o conteúdo.');
});

// 2. Bloquear atalhos de teclado (Copiar, Imprimir, etc.)
document.addEventListener('keydown', function(e) {
    // Bloquear Ctrl+C (Copiar) e Ctrl+U (Ver código fonte)
    if (e.ctrlKey && (e.key === 'c' || e.key === 'u' || e.key === 's' || e.key === 'p')) {
        e.preventDefault();
        alert('Esta função foi desabilitada para proteger o conteúdo.');
    }
    // Tentar bloquear a tecla Print Screen (funciona em alguns navegadores/cenários)
    if (e.key === 'PrintScreen') {
        navigator.clipboard.writeText(''); // Limpa a área de transferência
        e.preventDefault();
        alert('Capturas de tela foram desabilitadas.');
    }
}, false);
```

```
// --- PARTE 2: RENDERIZAÇÃO DO PDF COM PDF.js ---

// URL do arquivo PDF que você quer exibir.
// IMPORTANTE: O PDF precisa estar em um local que permita CORS (Cross-Origin Resource Sharing)
// ou no mesmo domínio do seu site. Um link direto do GitHub (raw) funciona para teste.
const pdfUrl = 'sua URL';

// Configura o "worker" do PDF.js. É necessário para ele rodar em background.
pdfjsLib.GlobalWorkerOptions.workerSrc =
`https://cdnjs.cloudflare.com/ajax/libs/pdf.js/2.10.377/pdf.worker.min.js`;

const loadingTask = pdfjsLib.getDocument(pdfUrl);
const pdfViewer = document.getElementById('pdf-renderer');

loadingTask.promise.then(function(pdf) {
  console.log('PDF carregado com sucesso!');

  // Itera por todas as páginas do PDF
  for (let pageNum = 1; pageNum <= pdf.numPages; pageNum++) {
    pdf.getPage(pageNum).then(function(page) {
      const scale = 1.5;
      const viewport = page.getViewport({ scale: scale });

      // Cria um elemento <canvas> para cada página
      const canvas = document.createElement('canvas');
      const context = canvas.getContext('2d');
      canvas.height = viewport.height;
      canvas.width = viewport.width;

      // Adiciona o canvas ao nosso container
      pdfViewer.appendChild(canvas);

      // Renderiza a página do PDF no canvas
      const renderContext = {
        canvasContext: context,
        viewport: viewport
      };
      page.render(renderContext);
    });
  }
}, function (reason) {
  // Tratamento de erro
```

```
    console.error('Erro ao carregar o PDF: ', reason);
});
```

Explicação e Considerações

1. **Impossibilidade da Proteção Total**:
    - **Captura de Tela (Print Screen)**: O bloqueio da tecla PrintScreen via JS é **pouco confiável**. Softwares de terceiros (como Snipping Tool, Lightshot, etc.) ou combinações de teclas do sistema operacional (como Win + Shift + S no Windows) **não podem ser bloqueados** por um site. A camada CSS (.no-copy-overlay) pode atrapalhar, mas não impedir.
    - **Download**: O PDF.js precisa **baixar** o arquivo para o navegador para poder exibi-lo. Um usuário com a aba "Rede" (Network) das Ferramentas de Desenvolvedor abertas verá a requisição para o helloworld.pdf e poderá acessá-lo diretamente.
    - **Foto da Tela**: A proteção mais básica pode ser burlada simplesmente com um celular.
2. **A Solução Full-Stack (Mais Robusta)**:
    - Para realmente proteger o acesso ao arquivo PDF, a lógica deve estar no **back-end**.
    - **URL Temporária**: Em vez de um link direto, o front-end solicitaria ao back-end uma URL assinada e de curta duração para o PDF. Isso impede que o link seja compartilhado.
    - **Servir o Arquivo por Partes (Streaming)**: O servidor pode ler o arquivo PDF e enviá-lo em "pedaços" para o cliente, sem nunca expor o arquivo completo de uma vez.
    - **Marca d'Água Dinâmica**: A solução mais segura envolve o back-end renderizando o PDF como imagens no servidor e aplicando uma marca d'água com o nome do usuário ou IP em cada página antes de enviá-las para o front-end. Isso desincentiva o compartilhamento de capturas de tela, pois elas seriam rastreáveis.

O exemplo fornecido é a implementação mais forte que você pode fazer **apenas com tecnologias front-end**. Ele cria uma barreira de usabilidade eficaz contra a maioria dos usuários leigos, que é geralmente o objetivo principal para esse tipo de requisito.