@push.rocks/smartkey

Handles the creation and management of private/public key pairs.

readme.md for @push.rocks/smartkey

Handle private/public key creation efficiently and securely with this TypeScript-based library, offering seamless integration and robust functionality for managing cryptographic key pairs.

Install

To get started with `@push.rocks/smartkey` in your project, you'll need to have Node.js and npm installed. Then, you can install the package using npm:

```
npm install @push.rocks/smartkey --save
```

This command adds `@push.rocks/smartkey` to your project's dependencies, ensuring that your project is ready to leverage the power of key management.

Usage

Getting Started with SmartKey

To use `@push.rocks/smartkey` in your TypeScript project, initially import the main class `SmartKey` and relevant interfaces from the package:

```
import { SmartKey, KeyPair } from '@push.rocks/smartkey';
```

`SmartKey` is your gateway to creating and managing key pairs, while `KeyPair` represents the public and private keys.

Creating a New Key Pair

Generating a new key pair is simple and straightforward with `@push.rocks/smartkey`. You can create a key pair with an optional passphrase for the private key encryption:

```
async function generateKeyPair() {
  const smartKeyInstance = new SmartKey();
  try {
    const keyPair: KeyPair = await smartKeyInstance.getKeypair('YourPassPhraseHere');

    console.log('Public Key:', keyPair.publicKey);
    console.log('Private Key:', keyPair.privateKey);
  } catch (error) {
    console.error('Error generating key pair:', error);
  }
}

generateKeyPair();
```

In this example, we instantiate `SmartKey` and call `getKeypair` with an optional passphrase. The resulting `keyPair` object contains both the public and private keys. Note that including a passphrase is optional but recommended for additional security of the private key.

Understanding Key Formats

The keys generated by `@push.rocks/smartkey` are in the PEM format, widely used and compatible with many cryptographic operations and libraries. This format ensures that your keys can be easily integrated into a variety of applications and systems.

Secure Key Storage

After generating your keys, it's crucial to store them securely. The public key can be distributed or stored openly, as it's used to encrypt data that only the matching private key can decrypt. However, the private key should be kept secure and confidential. If you're using a passphrase, it adds an extra layer of protection, encrypting your private key to prevent unauthorized access even if the key itself is compromised.

Integration with Cryptographic Operations

`@push.rocks/smartkey` focuses on key management, and you may want to use the keys generated for operations such as data encryption, decryption, or creating digital signatures. This usually involves integrating with other cryptographic libraries or modules, such as Node.js's built-in `crypto` module or other npm packages that provide higher-level cryptographic functionalities.

For example, you might use the public key from a `KeyPair` to encrypt data with another library, ensuring that only the holder of the corresponding private key can decrypt and access the original information.

Advanced Usage and Considerations

- **Key Rotation**: Regularly rotating keys is a fundamental security practice. `@push.rocks/smartkey` makes generating new key pairs simple, so periodic rotation can be easily implemented in your application's security strategy.
- **Passphrase Management**: If you opt to use passphrases, ensure they are securely managed and stored independently from the keys themselves. Leveraging a secure storage solution, such as a dedicated secrets manager, is advisable.
- **Cryptography Best Practices**: Always adhere to current cryptography best practices, including using strong passphrases, securely storing keys, and understanding the cryptographic principles underlying the operations you perform with the keys.

`@push.rocks/smartkey` provides a robust foundation for managing cryptographic keys within your TypeScript applications, streamlining the creation, management, and secure storage of key pairs. Through its integration-friendly design and adherence to best practices, it's an essential tool for any security-conscious developer.

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.