Blockchain-based Service Network User Manual

Version 1.4.0

BSN Foundation

CONTENTS

1 BSN Introduction 1
1.1 Brief Introduction 1
1.2 BSN Services 2
1.2.1 Permissioned Services 2
1.2.2 Permissionless Services 2
1.2.3 Interchain Services 3
1.3 Terminologies 3
2 Release Notes 5
3 Quick Start 7
3.1 Permissioned Blockchain 7
3.2 Permissionless Blockchain 8
3.3 Documentation 9
4 Registration and Activation 10
4.1 Registration 10
4.2 Login 12
4.3 Forgot Password 13
5 Permissioned Services 15
5.1 Overview 15
5.2 BSN Keys and Certificates Mechanism 16
5.2.1 BSN Keys and Certificates Mechanism 16
5.2.2 Locally generate the DApp access key pair 17
5.3 DApp Services Publication and Participation 18
5.3.1 Overview 18
5.3.2 DApp Services Publication 18
5.3.3 DApp Services Management 25
5.3.4 DApp Services Participation 28
5.4 Off-BSN system Access Guide 34
5.4.1 Overview 34
5.4.2 BSN Smart Contract Package Requirements 38
5.4.3 PCN Gateway Fabric API 43
5.4.4 PCN gateway FISCO API 66
5.5 Development SDK and Examples 90

5.5.1 BSN Gateway SDK Example 90
5.5.2 Off-BSN System Examples 91
5.6 BSN Testnet Services 91
5.6.1 Overview 91
5.6.2 Permissioned DApp Service Publication 91
5.6.3 Interchain Services on BSN Testnet 93
6 Permissionless Services 94
6.1 Overview 94
6.2 Select Plans 94
6.3 Create and Manage Projects 97
6.4 Off-BSN system Access Guide 99
6.4.1 Overview 99
6.4.2 Ethereum 100
6.4.3 EOS 101
6.4.4 Nervos 101
6.4.5 NEO 101
6.4.6 Tezos 102
6.4.7 IRISnet 102
6.4.8 dfuse-eos 103
6.4.9 Solana 103
6.4.10 ShareRing 104
6.4.11 Algorand 104
6.4.12 BTY 105
6.4.13 Oasis Network 106
6.4.14 Polkadot 106
7 Interchain Services 108
7.1 Interchain Service Management 109
7.1.1 Open Interchain Services 109
7.1.2 View Interchain Services 110
7.1.3 Deactivation and Activation of Interchain Services 112
7.2 Interchain Services based on Poly Enterprise 113
7.2.1 Overview 113
7.2.2 Interchain Services based on Hyperledger Fabric 114
7.2.3 Interchain Services based on FISCO BCOS 118

7.2.4 Interchain Services based on Ethereum Ropsten 120
7.2.5 Interchain Services based on Neo Testnet 121
7.3 Interchain Services based on IRITA 123
7.3.1 Overview 123
7.3.2 Interchain Architecture based on IRITA 124
7.3.3 Interchain Services in BSN Testnet 125
8 Account Management 130
9 Online Documentation 131
10 Contact Us 133

Preface

Blockchain-based Service Network (BSN or Service network) is a worldwide infrastructure network that provides a one-stop-shop solution for blockchain and distributed ledger technology (DLT) applications (DApp). BSN is a complex system that involves programming, software development, resource and environment configurations, application deployment, gateway APIs, local SDK, key certificates, etc. To facilitate utilization, BSN International (www.bsnbase.io) has prepared this document for developers and users to learn how to use BSN. We hope that BSN will become the first choice for developers to develop and run their DApps.

BSN provides developers three types of services: Permissioned, Permissionless, and Interchain services.

Permissioned services are divided into two parts. The first part demonstrates how developers can deploy smart contracts to the selected public city nodes through the BSN portal; the second part describes how developers can connect their off-BSN systems to the corresponding smart contracts through the public city node gateway and conduct data transaction processing.

Permissionless services determine how developers can choose the appropriate public city nodes, plans, and public chain frameworks, to deploy and publish their DApps.

BSN's "Interchain Communications Hub" (ICH) integrates two interchain solutions based on relay chain mechanism: Poly Enterprise developed by Onchain and IRITA developed by Bianjie AI. It enables cross-chain interoperability between standard permissioned chains, open permissioned chains and public chains. We will continue to integrate more cross-chain protocols to achieve interoperability of all blockchains adapted to the BSN.

Please feel free to contact us if there are any further questions. Our contact information can be found in Chapter 10. Contact Us. We strongly recommend users access the Online Documentation section to explore BSN technical details further.

1 BSN Introduction

1.1 Brief Introduction

The BSN design and concept as taken from the Internet, is a connected set of devices across data centers using the TCP/IP protocol. BSN is formed by the connection of the public city nodes using a set of blockchain operating environment protocols. Just like the Internet, BSN is also a cross-cloud, cross-portal, cross-framework, global infrastructure network.

With BSN, there are three types of participants: cloud service providers, blockchain framework providers, and portal operators.

Cloud service providers, through the installation of free BSN public city node software, can make their cloud service resources (computing power, storage, and bandwidth) accessible and sell through BSN to end-users.

Blockchain framework providers align with the BSN's framework adaptation standards and deploy them on BSN so developers can use it to develop and deploy applications. The Permissionless service only applies to the BSN international portal and international public city nodes.

Portal operators can easily and quickly build a Blockchain as a Service (BaaS) platform on their existing websites using BSN APIs. This allows them to provide BSN capabilities to their end users without users leaving their websites.

BSN is an open network that any cloud service provider, framework provider, or portal operator, that complies with BSN requirements and standards is free to use and stop using the service network at any time.

Similar to the Internet, most users of BSN are developers and technology companies. They can use any BSN portal to purchase cloud resources that charge based on transactions per second (TPS), storage quantity, and bandwidth from any public city node around the world. They select any pre-adapted framework to conveniently develop, deploy, and manage permissioned blockchain applications at a very low cost. Blockchain developers only need to deploy the application to one or more public city nodes on BSN so participants can connect to the application at no cost through any public city node gateway. All deployed applications share server resources in every public city node. For high-frequency applications, public city nodes can intelligently allocate a dedicated peer node with high processing capacity. For low-frequency applications, they share the same peer node. This resource-sharing mechanism allows BSN to reduce the resource cost to one-twentieth of the cost of traditional blockchain cloud services.

BSN is a blockchain infrastructure network. Just as households do not need to dig their own wells, but instead, enjoy the water supply services provided by public water plants in cities, BSN blockchain application publishers and participants do not need to buy physical servers or cloud resources to build their blockchain operating environment.

They use the public services provided by BSN and rent shared resources as needed, thus greatly reducing their costs. According to recent research, it takes about 20,000 USD per year for developers to build and deploy a traditional permissioned blockchain LAN-type environment. However, with BSN, the minimum cost to run such an application is as low as one dollar a day. Cost is a huge factor and will encourage a large number of small, medium, and micro enterprises and even individuals (including students) to innovate and start businesses through BSN. This will undoubtedly promote rapid development and popularization of blockchain technology. In general, the development from the closed architecture of the traditional blockchain to the resource-sharing architecture of BSN completely mimics the development process of the Internet, which gathered numerous isolated LANs in the early days to the global connectivity facilities we have today. We hope to make BSN the blockchain Internet.

1.2 BSN Services

As mentioned above, BSN provides a one-stop-shop solution for developers to deploy, operate, and manage DApps. BSN provides three types of services: Permissioned, Permissionless, and Interchain services.

1.2.1 Permissioned Services

BSN is continually adapting most of the mainstream permissioned blockchain frameworks. On the BSN portal users can deploy DApps on any public city nodes based on the type of selected framework and the number of peer nodes. The number of peer nodes per application can be up to 60 and can be distributed among public city nodes based on different cloud services. Users can easily complete the DApp deployment process by uploading smart contracts and configuring the corresponding parameters. This service mode allows developers to focus on business innovation, smart contract programming. All work related to environment construction, system maintenance, application deployment, node transmission, and network configuration is done by BSN.

The pricing strategy for the Permissioned service is based on three resource elements of each peer node of the published application. The three elements are TPS, storage, and data traffic. Among them, TPS and storage in the BSN portal are pre-paid, while data traffic will be charged based on actual usage. This pricing strategy is designed to minimize resource costs and provide users with the best services. Based on the data provided by the BSN portal, if a user deploys a three-peer Fabric DApp, and each peer node supports 10TPS and 10GB storage capacity, the monthly fee is only 20 USD.

1.2.2 Permissionless Services

The Permissionless service is only applicable to the BSN international portal (www.bsnbase.io) and international public city nodes. Compared with the complexity of the Permissioned service, Permissionless service has the virtue of simplicity. The Permissionless service mainly provides developers who develop public chain DApps, with unified access service covering numerous public chain nodes. Developers may choose different plans on the BSN portal, and can simultaneously deploy DApps and process transactions on all BSN adapted public chain nodes through the selected public city nodes.

We offer a free plan and different premium plans. The free plan includes up to 2,000 requests per day. There are 3 types of premium plans, priced at $20, $100, and $500 per month. The premium plans include up to 40,000 requests, 250,000 requests, and 1,500,000 requests, respectively, per day. All requests can be assigned to any public chain freely.

Permissionless services only provide shared nodes and access environments and do not involve any business of the public chain itself. The gas fees incurred in publishing and running DApps on any public chain shall be borne by the developers themselves and have nothing to do with BSN.

1.2.3 Interchain Services

The vision of BSN is to become the Internet of blockchains. In the future, millions of DApps will be deployed and run on BSN. Both Permissioned and Permissionless DApps will be very easy to call and they can interact with each other just like applications currently do on the Internet. From this perspective, Interchain will become a very core part of the BSN technical architecture.

The BSN's "Interchain Communications Hub" (ICH) is now commercially available and integrates with Onchain's Poly Enterprise cross-chain solution. It supports cross-chaining between permissioned chains and cross-chaining between permissioned chains and ETH Ropsten testnet and NEO testnet. The IRITA-based cross-chain solution is also being adapted and is expected to be commercially available in the next iteration.

A demo version of ICH is also live on the BSN Testnet, integrating two interchain solutions based on the relay chain mechanism: (1) Poly Enterprise developed by Onchain and (2) IRITA developed by Bianjie AI. We welcome all developers to try out and provide feedback and suggestions to us and we will continue to improve the interchain functionality.

BSN's "Interchain Communications Hub" (ICH) integrates two interchain solutions based on relay chain mechanism: Poly Enterprise developed by Onchain and IRITA developed by Bianjie AI. It enables cross-chain interoperability between standard permissioned chains, open permissioned chains and public chains. We will continue to integrate more cross-chain protocols to achieve interoperability of all blockchains adapted to the BSN.

1.3 Terminologies

- **Public city node (PCN):** This is the core element of BSN but the "node" part doesn't refer to the blockchain nodes and BSN isn't a blockchain. With BSN, each PCN is a virtual data center used to allocate a portion of resources from the cloud service or data center on which it was deployed. An entire blockchain operating environment has been built within this resource pool and includes multiple blockchain frameworks, shared peer nodes, CA management, authority chain, PCN gateway, and PCN manager systems.
- **DApp:** This is a generic term for blockchain and distributed ledger technology

application.

- **DApp Service or Service:** This is a DApp that is already deployed and in use on BSN that users can access with an invitation from the DApp publisher. The invitation allows them to directly join and use the service.
- **Service Publisher:** This is the individual or enterprise who published and deployed the DApp service on BSN and is responsible for granting access to users who apply to participate in the service.
- **Service Participant:** This is a user that uses the BSN DApp service via a BSN portal or the publisher's system. Also, the user's off-BSN system can connect to the DApp service via the PCN gateway to execute transactions and query data.
- **Off-BSN system:** A business IT system developed and managed by a DApp service publisher or a service participant outside BSN.

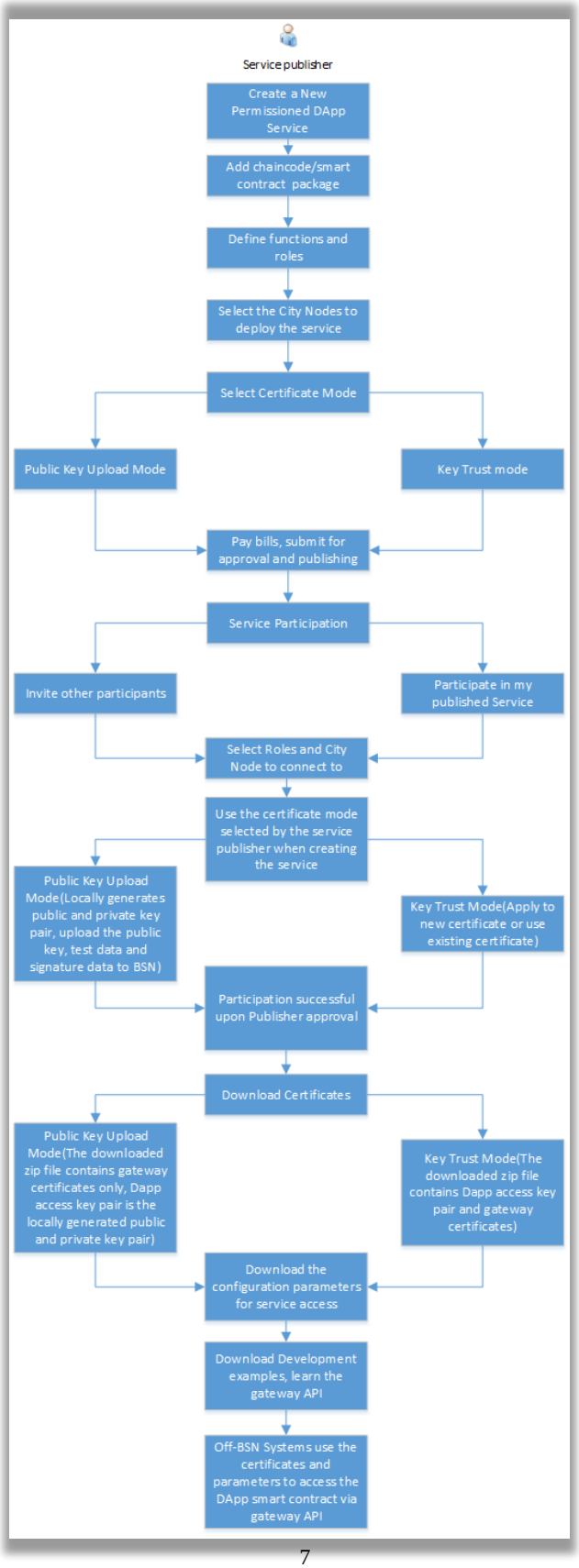
2 Release Notes

Release date	Version	Notes
2021/01/31	1.4.0	• Iterative optimization and technical optimization of the BSN international website (www.bsnbase.io) to enhance user experience. • Launched the commercial service of Interchain Communications Hub based on Poly Enterprise. • Fixed some bugs and enhanced the stability of the system.
2020/11/30	1.3.1	• Added public chain main net and test net nodes along with native API access services. Including: BTY, Oasis and Polkadot.
2020/10/31	1.3.0	• Optimized the BSN International website (www.bsnbase.io) to improve user experience. • Launched BSN Permissioned Blockchain Testnet, providing developers with a free testing environment supporting: Hyperledger Fabric R1, FISCO BCOS K1 DApp Services publication -Interchain testing services • Launched the BSN Interchain Communications Hub on BSN Testnet based on Poly Enterprise and IRITA. • Added the BSN empowerment platform APIs to allow third-party portals to access BSN Permissionless Services. • Added the TPD (Transactions Per Day) limit control function in the Permissionless Services • Fixed some bugs and enhances the stability of the system.
2020/9/24	1.2.1	• Updated the BSN Global website address to https://www.bsnbase.io. • Added public chain main net and test net nodes along with native API access services. Including: Algorand, ShareRing and Solana. • Added Enable Key function in the public chain project.
2020/8/10	1.2.0	• Redesigned the user interface toprovide better navigation and user experience. • Added public chain main net and test net nodes along with native API access services. Including: Nervos, Neo, ETH, Tezos, EOS, IRISnet, etc. • Added commercial functionality for Hyperledger Fabric and FISCO BCOS frameworks. • Updated FISCO BCOS framework to support SECP256 K1 encryption algorithm. • Added the following functionality to Permissioned services: recurring payment mechanism for service charge and data usage charge, service configuration upgrade. • Added Permissionless service plan purchase and upgrade. • Added "My Account" in User Center to make it easier for users

		to update credit card information, check bills, pay bills and download invoices (we process all credit card activities directly on Stripe). • Added Online Help Manual to provide developers and portal users easy-to-follow instructions. • Added PCN gateway SDK and all examples on Github: https://github.com/bsnda. • Fixed a few bugs to enhance the stability of the system.
2020/4/25	1.1.0	• The BSN global portal has officially launched. Beta testing will be held from April 25th, 2020 to June 25th, 2020. • Developers can deploy one three-peer DApp (service) at up to three public city nodes (PCNs) free of charge during beta testing. • There is a total of 10 available PCNs during beta testing. They are deployed on AWS, Microsoft Azure, Google Cloud, China Mobile Cloud, and Huawei Cloud. • During beta testing, there are two frameworks to choose from, Hyperledger Fabric V1.4.3 or FISCO BCOS V2.4.0. • Developers can choose "Key Trust Mode" or "Public-Key Upload Mode" to manage their service users' certificates and keys. • Basic information and chaincode/smart contracts in deployed services can be modified anytime. PCNs, however, cannot be changed once chosen. • Published services are private by default. Developers will need to apply for a public listing. After approval, they will be available on the App Store. • Developers will need to grant permissions to other users to participate in their services. The participants then follow the services' instructions to generate service access keys and user transaction keys by using either "Key Trust Mode" or "Public-Key Upload Mode". • The PCN gateway provides a set of user registration APIs for deployed services. Developers can register service users via these APIs from their off-BSN systems. Developers do not need to log in to the BSN global portal. • The PCN gateway APIs support "Key Trust Mode" for both Fabric and FISCO BCOS. "Public Key Upload Mode" is only supported for Fabric. • For more info on gateway APIs, please refer to the developer's manual.

3 Quick Start

3.1 Permissioned Blockchain

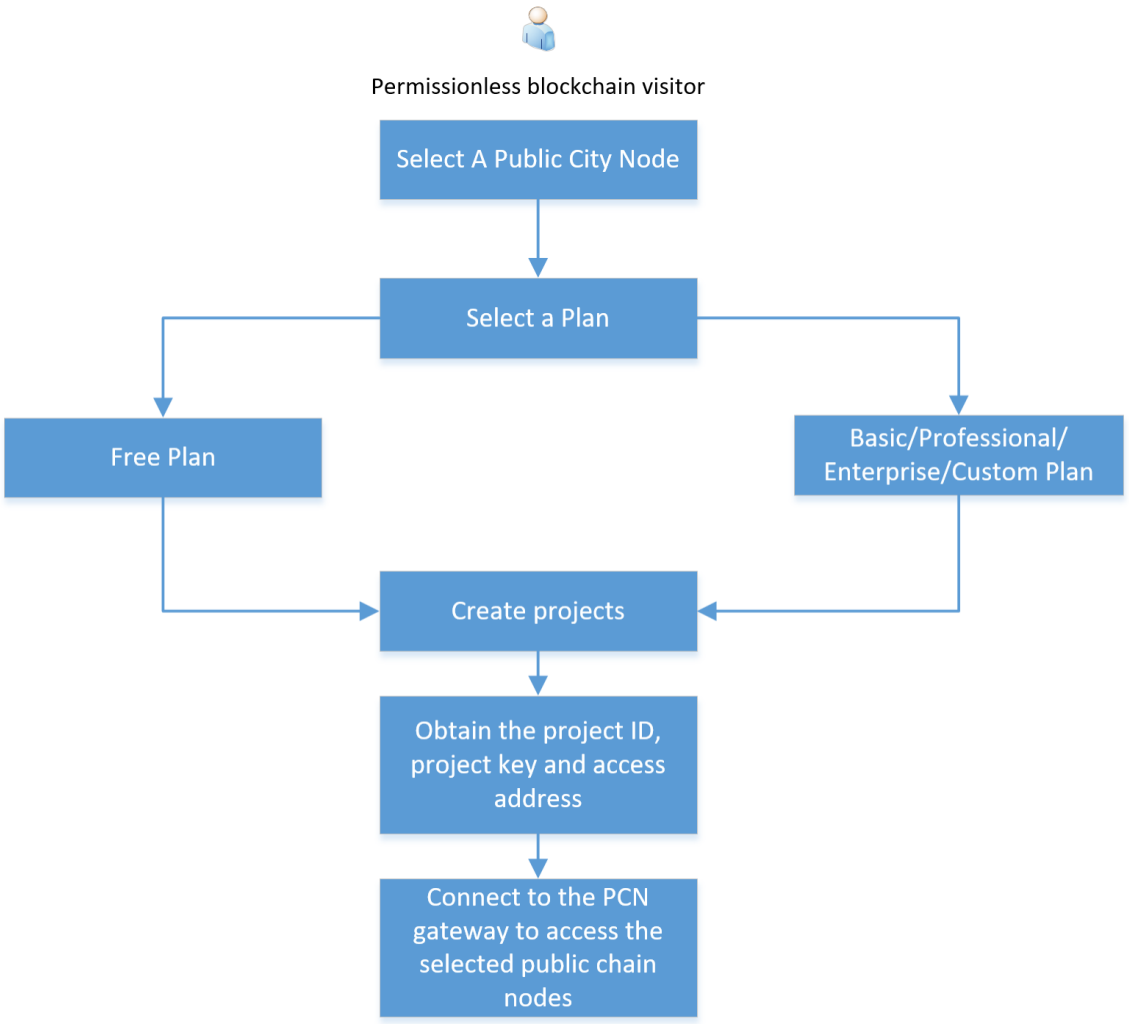


Permissioned DApp Service publishers can create DApp services in the BSN portal. To create the service, it is necessary to upload the smart contract/chaincode package, define the service functions and roles, select the public city nodes and select the participant certificate mode (including Key Trust mode and Uploaded Public Key mode). After that, publishers pay the bills and submit the service deployment request to the network operator for approval and publishing.

After the successful publication of the service, publishers can participate in their service or invite other users to participate in the service. To participate in the service, participants should select designated roles and the access public city node, then generate the certificates according to the certificate mode set by publishers. Participation will be successful after being approved by service publishers.

Once successfully participating in the service, participants can download the certificate, and use the certificate and service access configuration parameters to access the chaincodes/smart contracts through the gateway API.

3.2 Permissionless Blockchain



Permissionless services allow visitors to select public city nodes and plans to participate in a service. There are 2 types of plans, the free plan, and premium plans. Visitors can choose plans according to their business requirements. To connect to the public chain nodes, users can create projects to obtain project IDs, project keys, and access addresses to access the public chain services.

3.3 Documentation

The direct users of the BSN portal are developers. As the environment and tools of the blockchain application's development, deployment, and operation, BSN is relatively complex in its overall operation. We strongly recommend that all developers start by examining the documentation and examples so that they will be able to master the use of BSN within a day or two.

For your convenience, all examples we've provided are available on Github. We hope that developers with serious interest can help us optimize and enrich the examples so that other developers are able to adapt and learn about blockchain development. Developers who share their samples, will receive small gifts and be invited to BSN's internal technical seminar.

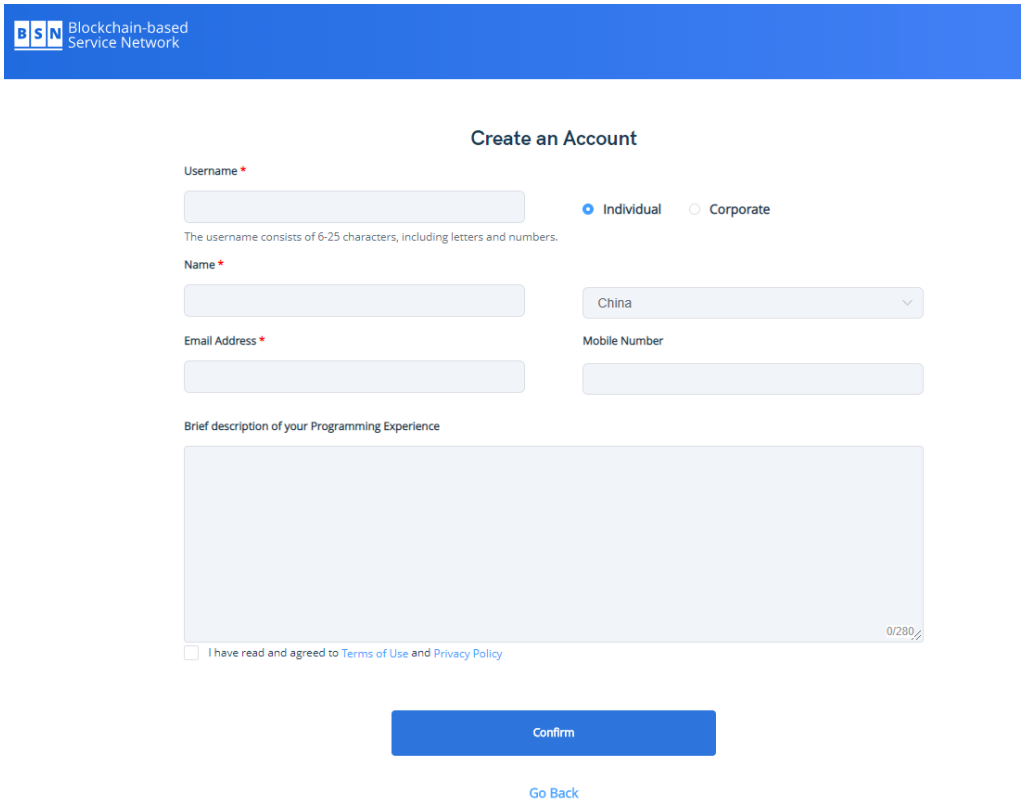
For links to all documents and examples, please visit Chapter 9. Online Documentation.

4 Registration and Activation

BSN requires its users to register and confirm their registration before they can access the network to carry out services and actions across the network. As a first-time user, follow these steps to register:

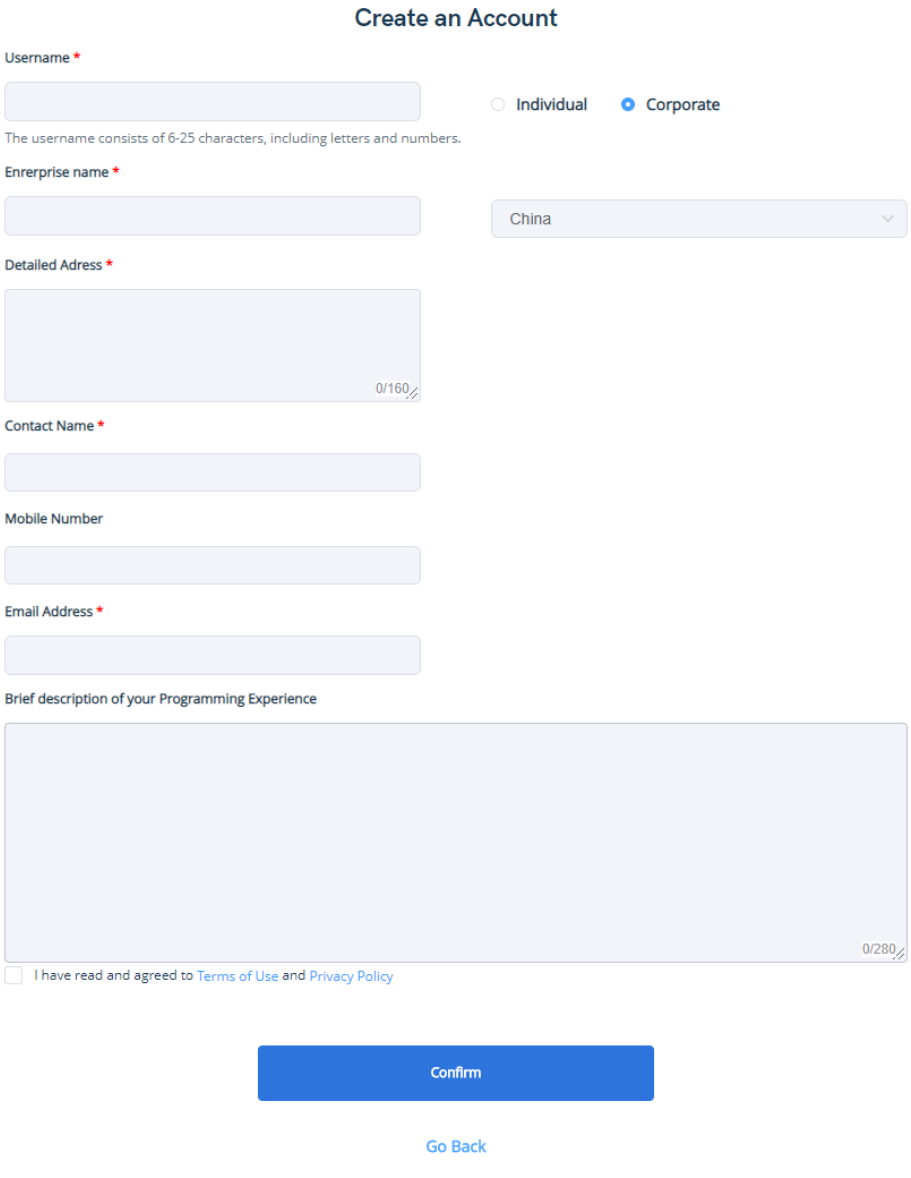
4.1 Registration

1. Click here to access the website at www.bsnbase.io.
2. With the blockchain-based service network, you can access the system either as an Individual or a Corporate entity.
3. To register as an Individual, enter or select the following:
   - **Username** – Enter a preferred username
   - **Nationality** – Click the dropdown to select your country from the list of countries
   - **Name** – Enter your real name, different from the username
   - **Mobile Number** (Optional) – Enter your mobile number
   - **Email address** – Enter an email address you have access to
   - **Brief description of your programming experience** (Optional) – If you have some experience in programming, we would love to hear about it
   - Check the **I have read and agree to Terms of User and Privacy Policy** box
   - Click **Confirm** to finish the registration.

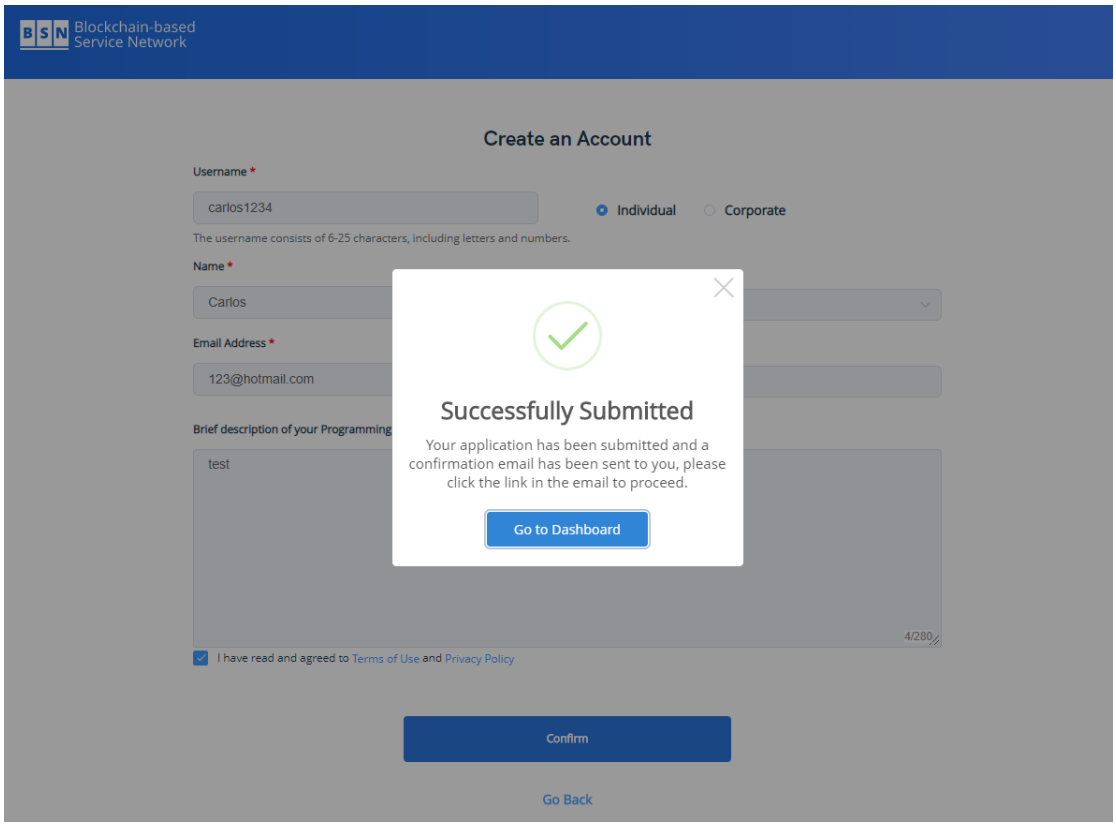


4. To register as a Corporate entity, enter or select the following:

- **Username** – Enter a preferred username
- **Nationality** – Click the dropdown to select your country from the list of countries
- **Enterprise Name –** Enter the legal name of your corporate body or company name
- **Detailed Address –** Enter a verifiable address of the company location
- **Contact Name –** Enter a contact name that represents the company
- **Mobile Number** (Optional) – Enter your corporate mobile number
- **Email address –** Enter a corporate email address that you have access to
- **Brief description of your programming experience** (Optional) **–**If you have some experience in programming, we would love to hear about it
- Check the **I have read and agree to Terms of User and Privacy Policy** box
- Click **Confirm** to finish the registration.



5. A confirmation dialog box will be displayed confirming your registration. Click **Go to Dashboard**.

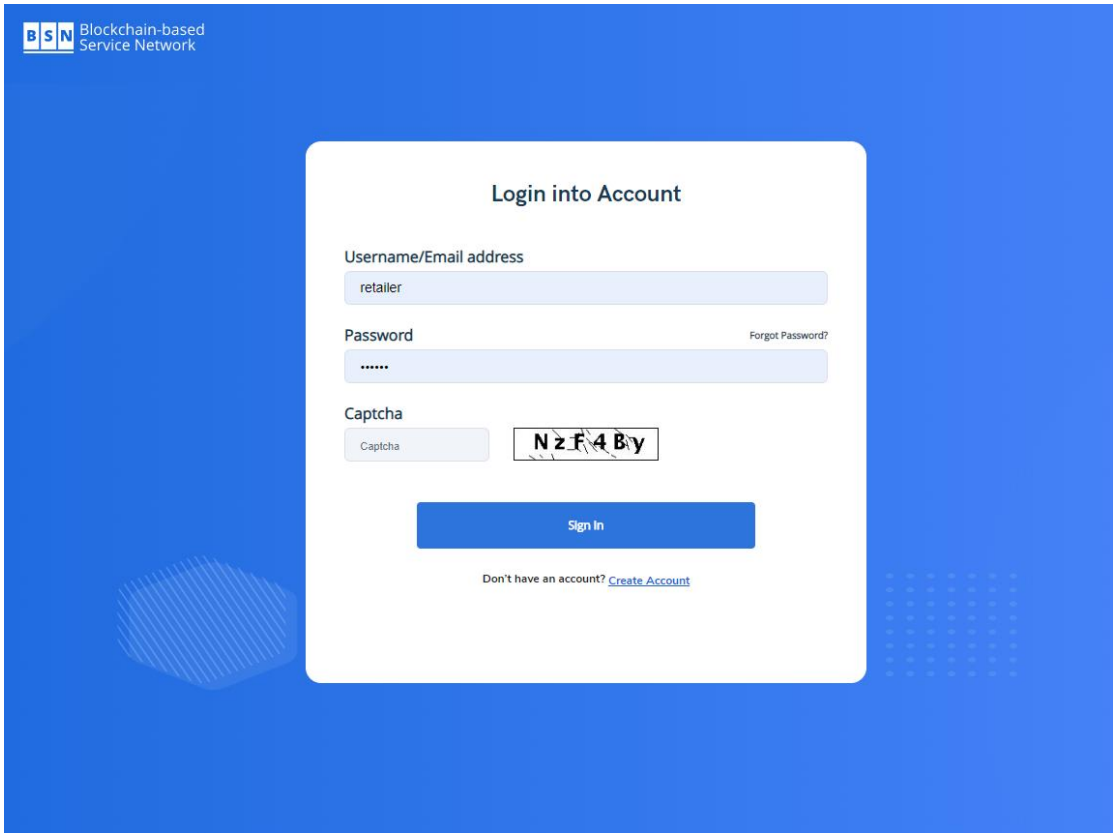


6. You will receive an email from BSN requesting that you confirm your registration.
7. Click on the link in the email to confirm your registration and enter your **Password** and **Password Confirmation**.
8. Click **Confirm** when done to return you to the login page.

4.2 Login

After you have successfully registered your account on BSN, you can login by following these steps:

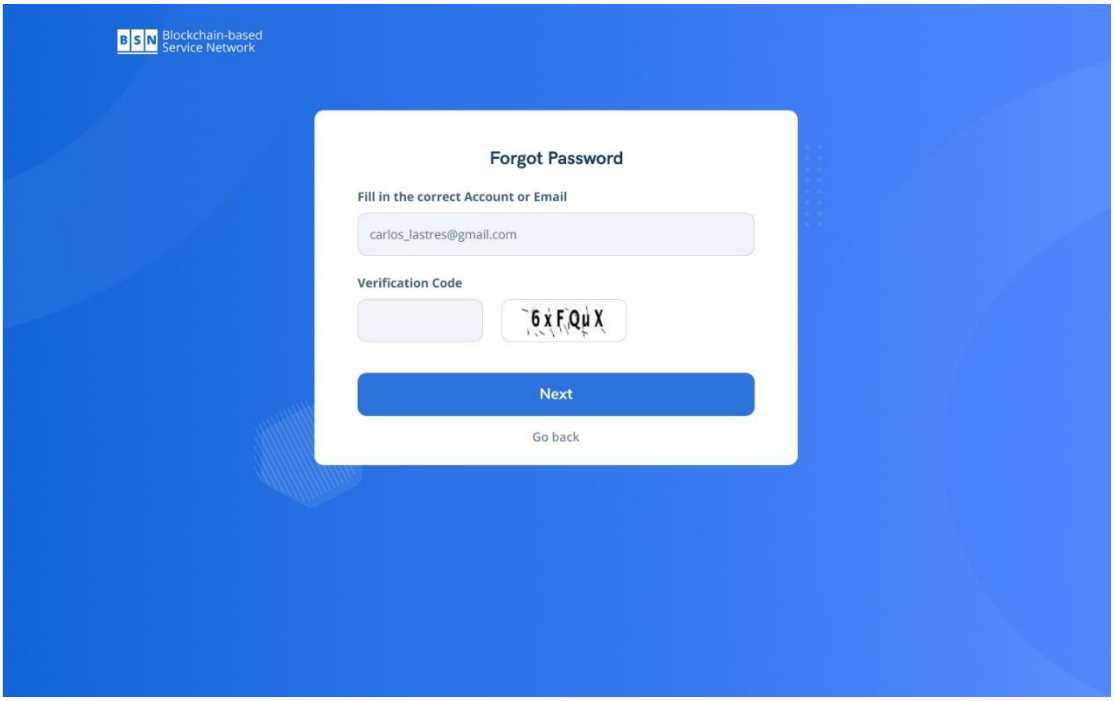
1. Click on the Login link to access the login page.
2. On the login page, enter the **Username/Email**, **Password, and** the Verification **Code**.
3. Click **Sign In**to access the **Home** page.



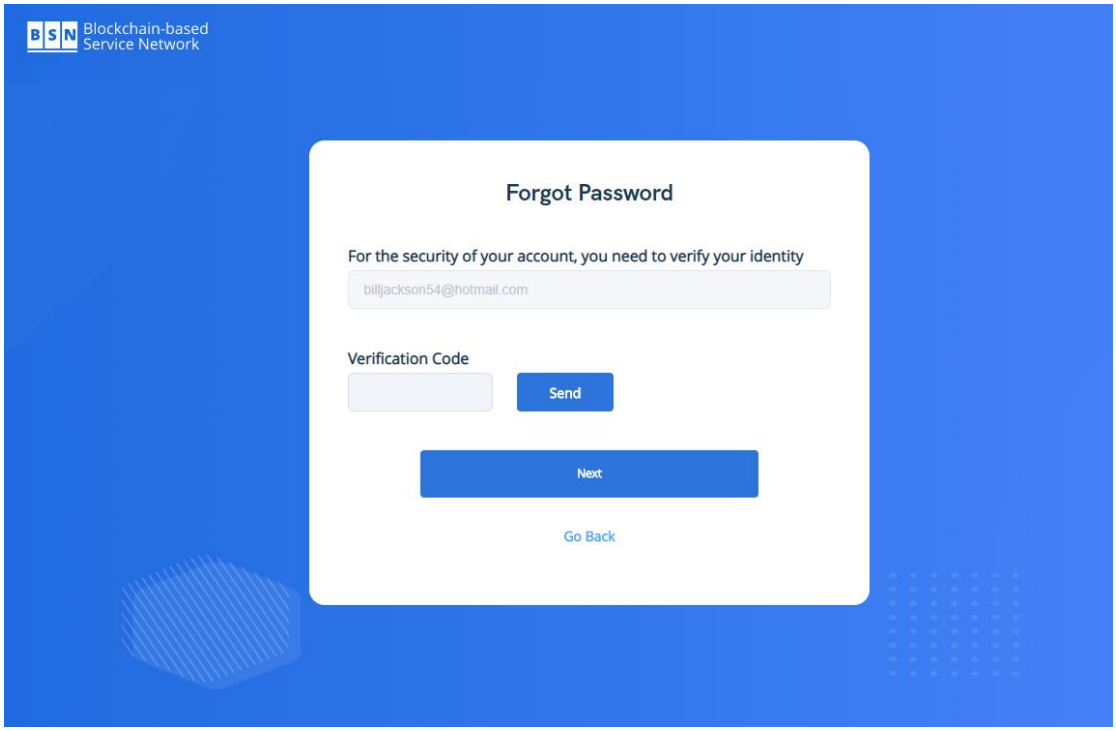
4.3 Forgot Password

If at any time you have forgotten your password, you can follow these steps to retrieve it:

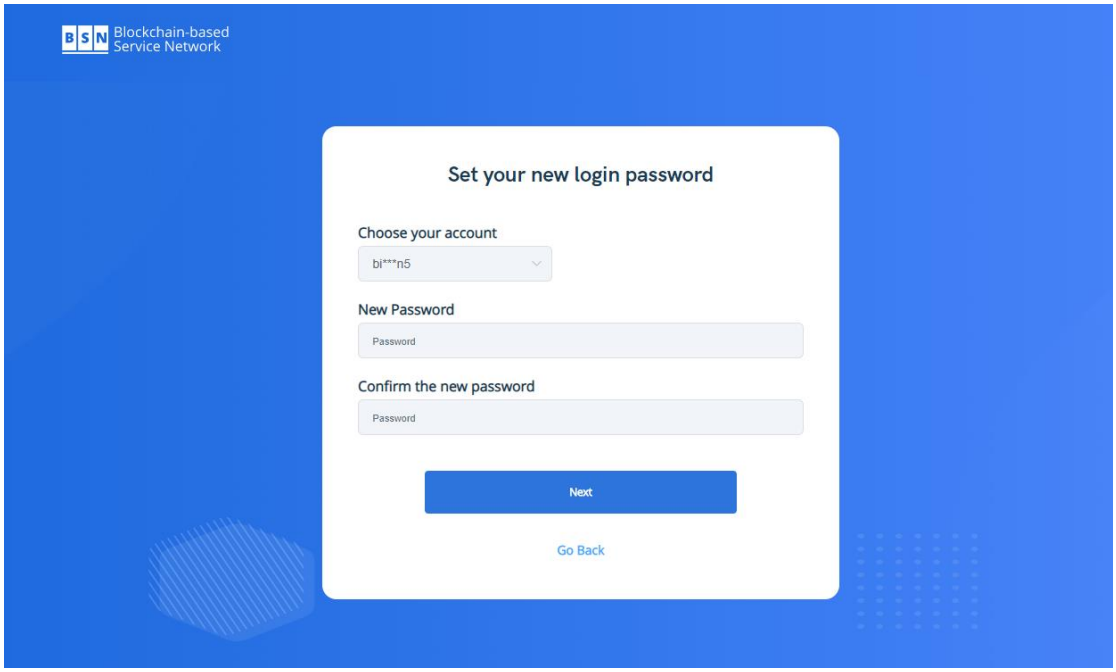
1. On the **Login** page, click the **Forgot Password** to open the forgot password page.
2. On the page, enter the **correct account or email**.
3. In the **verification code**, enter the displayed code. If you wish to generate another code, click on the code to generate another.



4. Click **Next** to view the **Authentication** page.
5. On the **Authentication** page, click the **Send** button to get verification code. This will generate a code that will be sent to your registered email address.



6. Enter the code that was received in your mailbox and click **Next**.
7. On the reset login password page, enter your **New password** and **Confirm password**.
8. Click **Confirm** to change your password.



5 Permissioned Services

5.1 Overview

The Permissioned service is one of the core services provided by BSN. Its goal is to make it easy for developers to publish decentralized applications (DApps) based on the framework of the permissioned blockchain on their selected public city nodes. Compared with the permissionless blockchain DApp, the permissioned blockchain DApp is more flexible in terms of architecture design, operation efficiency, and smart contract programming. It also has a larger space for innovation. However, from the perspective of development, because the developers need to build their underlying environments, and the environment for the public chain is readily available, the development, operation and maintenance of the permissioned chain DApp are relatively difficult. The developer's off-BSN system can access to DApp for data processing through the BSN public city node gateway.

Although BSN has greatly reduced the difficulty of permissioned blockchain DApp development, developers still need to have an in-depth understanding of the following three aspects which will be explained in detail in the following chapters.

1. Keys and Certificates Mechanism: the blockchain application itself is based on encryption algorithm technology, so the requirements of the keys and certificates are very high.
2. DApp services publication and participation: To build a permissioned blockchain DApp, the developer should firstly set up the chain and deploy the smart contracts. This part is entirely carried out on the BSN global portal (www.bsnbase.io), including the operations of smart contract upload, certificate mode selection, role's permissions setting, peer node configuration, public city node location, etc. Finally,

developers need to upload or download keys to facilitate the access from off-BSN system.

3. Off-BSN system access: This part contains a detailed description of the access parameter configuration, SDK usage, and the description of public city node gateway APIs to which the off-BSN systems connect. The API section includes all APIs of the currently permissioned blockchain frameworks that BSN has adapted.

5.2 BSN Keys and Certificates Mechanism

5.2.1 BSN Keys and Certificates Mechanism

Once a publisher deploys a permissioned DApp on BSN, the off-BSN systems of all participants (including the publisher) connects to the DApp via the PCN gateways to execute and record transactions based on the DApp's smart contracts. During this process, the participants need two key pairs to complete all steps: the *DApp Access Key Pair* and *User Transaction Key Pair*. When publishing and deploying a DApp on BSN, its publisher can choose from two modes to manage the DApp's keys and certificates: *Key Trust Mode* and *Public Key Upload Mode*. The key trust mode means that the two key pairs and related certificates will be generated and hosted by BSN when a participant joins the DApp. The participant can then download the private keys from the BSN portal, and use them to access BSN and sign transactions sent to the DApp from the off-BSN systems. The public key upload mode means that the two key pairs will be generated and stored locally on the participant's off-BSN system, and the public key is uploaded via the BSN portal or PCN gateway API, to BSN to generate the certificates. Once a mode is selected for the DApp, it cannot be changed. We strongly suggest all developers use the public key upload mode which is much more flexible and secure.

1. DApp Access Key Pair based on Key Trust Mode: DApp access key pair is used to generate the certificate to access the PCN gateway. If the DApp is on Key Trust Mode, the key pair can be generated on the BSN portal, and the private key can be downloaded. Please refer to the BSN Help Manual's service participation section.
2. User Transaction Key Pair based on Key Trust Mode: User transaction key pair is used to verify the requests and transactions sent to the DApp. If the DApp is on Key Trust mode, the key pair can be generated via the PCN gateway APIs by executing requests from the off-BSN systems. If the off-BSN systems have sub-users, it can even generate different key pairs for different sub-users. Refer to the API sections in this document for Hyperledger Fabric and FISCO BCOS frameworks to see how to generate the key pairs and use them to verify the transactions.
3. DApp Access Key Pair based on Public Key Upload Mode: In this mode, the DApp access key pair is generated and stored locally. The participant must upload the public key to BSN via the BSN portal to generate the access certificate to the PCN gateway. Please refer to section 5.2.2 below to see how to generate the key pair locally. Please refer to the "Public Key Upload" section of this document to learn how to upload the public key to BSN via the portal.
4. User Transaction Key Pair based on Public Key Upload Mode: In this mode, the

user transaction key pair is also generated and stored locally. Instead of using the BSN portal, the user transaction public key (one of the pair) is sent and registered on BSN via the PCN gateway certificate registration API. If the off-BSN systems have sub-users, they can also upload different public keys to generate different transaction certificates for different sub-users by using the API. Please refer to section 5.2.2 or the instructions inside the gateway SDK package about generating the key pair locally. Refer to the API sections in this document for registering the certificate via gateway APIs.

Please click the link to download the PCN Gateway SDK Package:

https://github.com/BSNDA/PCNGateway-Go-SDK

https://github.com/BSNDA/PCNGateway-Java-SDK

https://github.com/BSNDA/PCNGateway-PY-SDK

https://github.com/BSNDA/PCNGateway-CSharp-SDK

Currently, both permissioned frameworks Hyperledger Fabric and FISCO BCOS DApps support both Key Trust Mode and Public Key Upload Mode.

5.2.2 Locally generate the DApp access key pair

If the DApp service you participate in adopts Public Key Upload Mode for its application access key, you will need to generate the pair of public and private keys on the local client then save the private key locally and upload the public key to BSN via the portal.

It is recommended to use the latest version of OpenSSL to generate the keys. Please use the **prime256v1** cryptographic algorithm for Hyperledger Fabric and**secp256k1** for FISCO BCOS. The steps are as follows:

1. Preparation: Download the latest version of OpenSSL from https://www.openssl.org/source/ and create a data.txt file in which some test phrases are entered, such as - Hello world.
2. Input "OpenSSL" in the terminal to show the open SSL command line.

   OpenSSL>
3. Input the command - "ecparam -name **prime256v1** -genkey -out key.pem" to generate a private key file key.pem.

   OpenSSL> ecparam -name **prime256v1** -genkey -out key.pem
4. Input the command - "ec -in key.pem -pubout -out pub.pem" to generate a public key file pub.pem with the private key in the key.pem file.

   OpenSSL> ec -in key.pem -pubout -out pub.pem

   read EC key

   writing EC key

5. Input the command - "dgst -sha256 -sign key.pem -out signature.bin data.txt" to sign the data.txt file with the private key in the key.pem file to generate the signature file: signature.bin.

   OpenSSL> dgst -sha256 -sign key.pem -out signature.bin data.txt

6. Input the command - "dgst -verify pub.pem -sha256 -signature signature.bin data.txt". Use the public key in the pub.pem file to sign and verify the data.txt and signature.bin files.

   OpenSSL> dgst -verify pub.pem -sha256 -signature signature.bin data.txt

   Verified OK

7. If "Verified OK" is displayed, input the command - "base64 -in signature.bin -out signature64.txt" to convert the signature file signature.bin to base64 encoded signature64.txt.

   OpenSSL> base64 -in signature.bin -out signature64.txt

8. Input the command - "pkcs8 -topk8 -inform PEM -in key.pem -outform PEM -nocrypt -out keypkcs8.pem" to convert the private key in the key.pem file to pkcs8 format.

   OpenSSL> pkcs8 -topk8 -inform PEM -in key.pem -outform PEM -nocrypt -out keypkcs8.pem

9. Save the keypkcs8.pem file locally and copy all the contents of pub.pem, data.txt, and signature64.txt to the public key, test data, and signature data text boxes respectively on the Public Key Upload Mode page to verify the public key and submit it to BSN.

5.3 DApp Services Publication and Participation

5.3.1 Overview

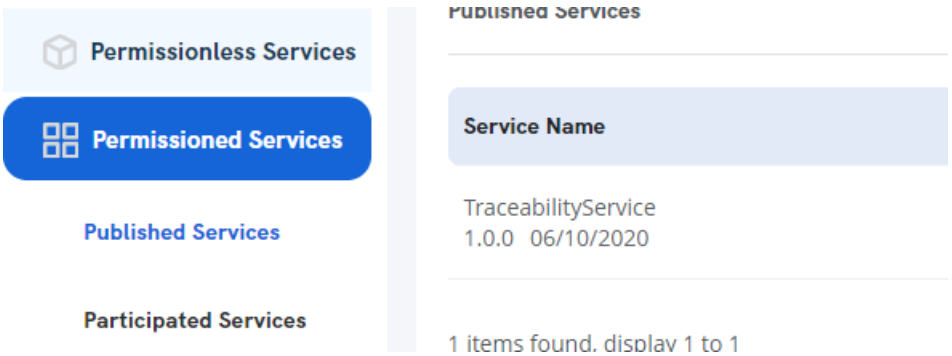
Permissioned DApp services refer to blockchain and DLT applications that are already deployed and operational on BSN. Users can use a BSN portal or the publisher's business system to apply to and join the service. Published services are private and cannot be browsed or searched by users through the BSN portal. DApp service participation must be initialized by the publishers' invitation links.

5.3.2 DApp Services Publication

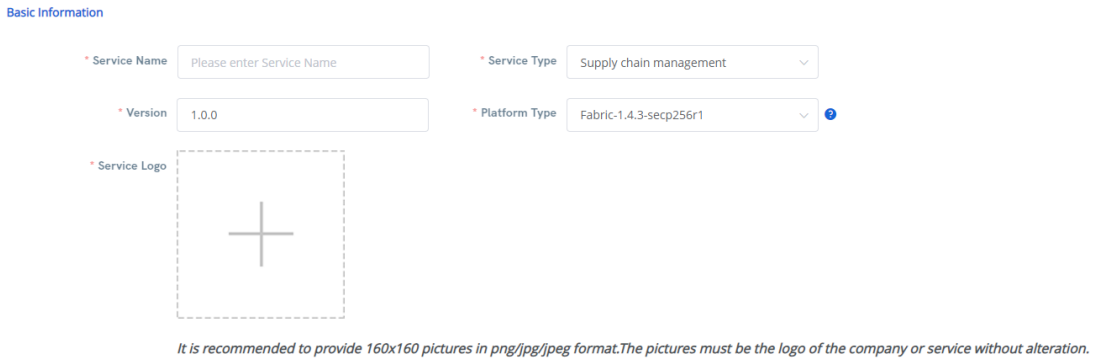
5.3.2.1 Create a New DApp Service

To create a new DApp service, follow these steps

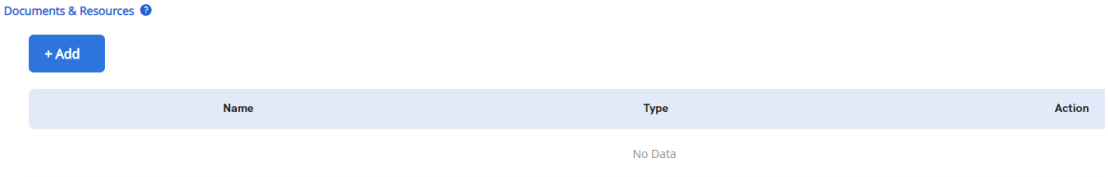
1. In the **BSN** menu, click the **Permissioned Service** dropdown, in the list, click **Published Services** to open the **Published Services** page.



2. On the published services page, click the **Create a New Service** button.
3. In the **Basic Information** section enter or select the following:
   - **Service Name** – Enter an applicable name for the service to be provided
   - **Service Type** – In the dropdown select from the various available service types
   - **Version** – The default version 1.0.0 is entered automatically. Unless necessary, leave it as is.
   - **Platform Type** – Select from either **Fabric-1.4.3-secp256r1** and **FISCO-2.4-secp256k1**
   - **Service Logo** – Click on the icon to locate the image on your PC. Note that the image must be in png/jpg/jpeg format and should be exactly 160 x 160 pixels.



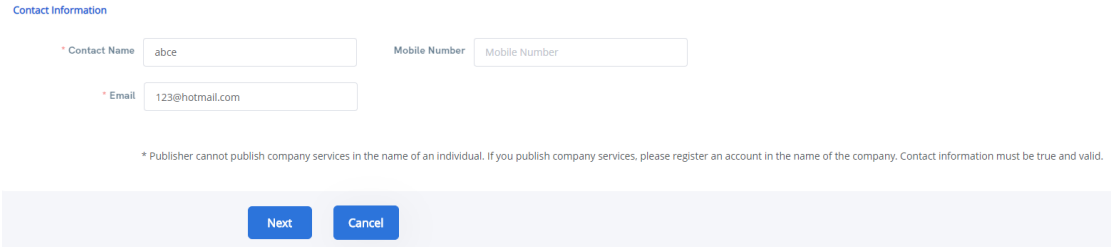
   - **Service Introduction** – Enter a brief description of the service
   - **Service Description** – Enter a detailed description of the service including text and pictures where applicable
4. **Documents** – Documents can be added so that other users can easily understand your product. Click **Add** to display the **Add Document** dialog box. Click **Select** to locate the document on your PC.



Enter a **Name**, and choose a **Type** for the document. Click **Confirm** to add the

document.

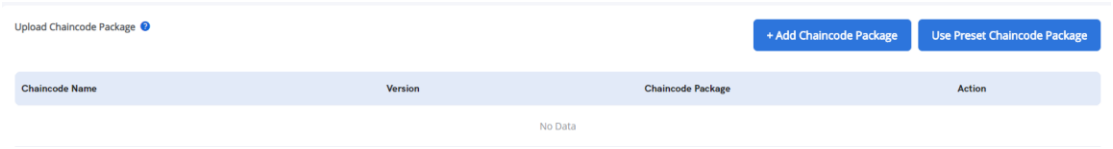
5. In the **Contact Information section**, the login details of the user are automatically populated, including the **Contacts** and **Email**. If necessary, you can add a telephone number.



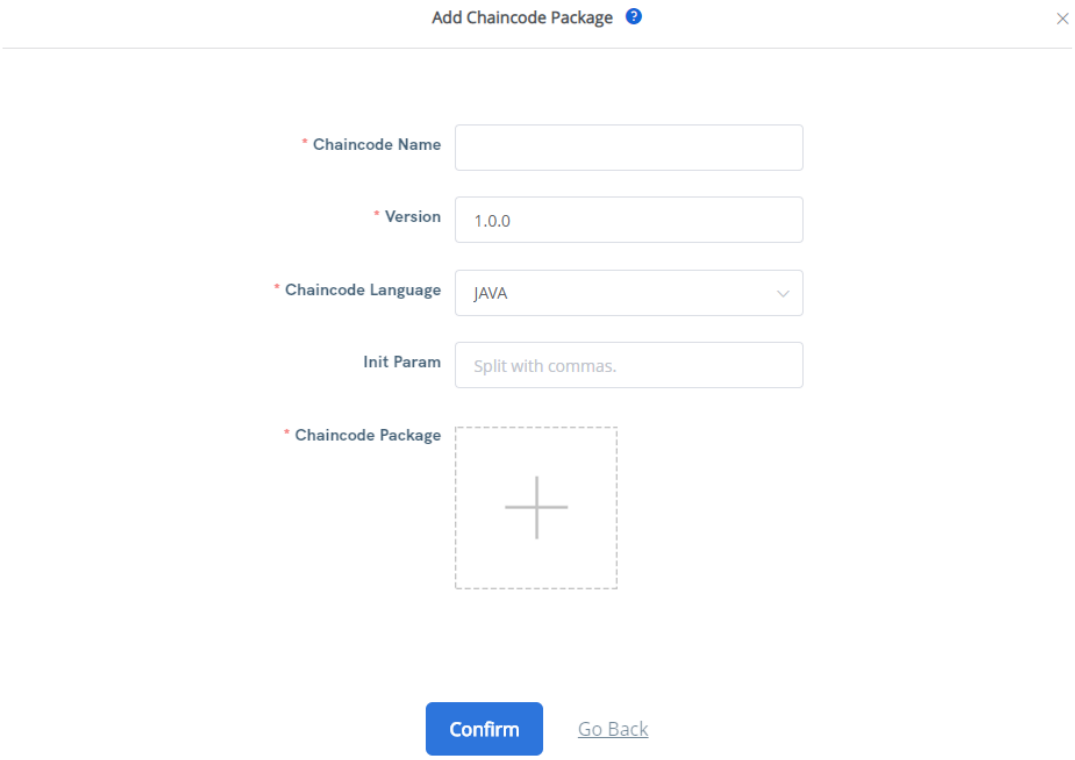
6. Click **Next** to continue.

5.3.2.2 Upload chaincode package

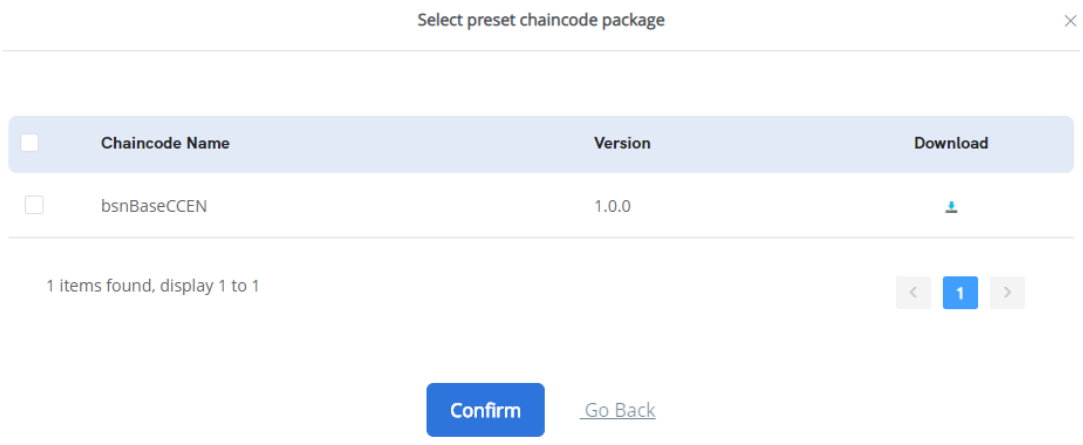
In the **Upload chaincode package** section, you can add your chaincode/smart contract package or use the preset chaincodes available in the system.



1. To **Add chaincode package,** click on the button to display the **Add chaincode Package** where you enter or select the following:
   - **Chaincode Name** – Enter a name for the chaincode
   - **Version** – Enter the chaincode version
   - **Chaincode Language** – Select from one the languages (Java, Golang or NodeJS)
   - **Initparam** – enter the initialization parameters and if multiple, separate it with commas
   - **Chaincode Package** – Click on the icon to select the package file from the PC. Package files are to be in the .zip file format and the file name should only contain letters and numbers or underscores

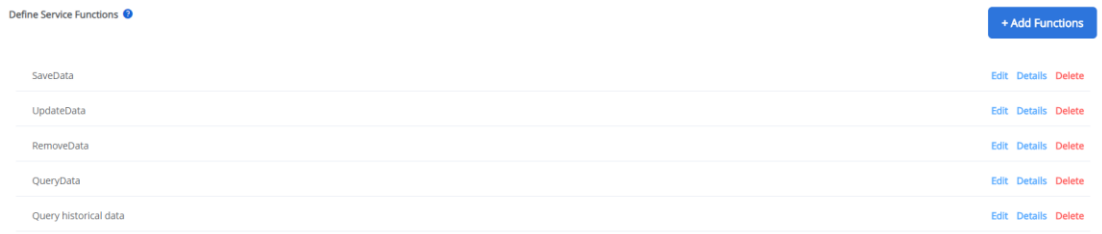


2. To **Use Preset Chaincode Package**, click on the button to display the **Select preset chaincode package** option. In the list of packages, select one of the listed packages and click **Confirm** to add it.



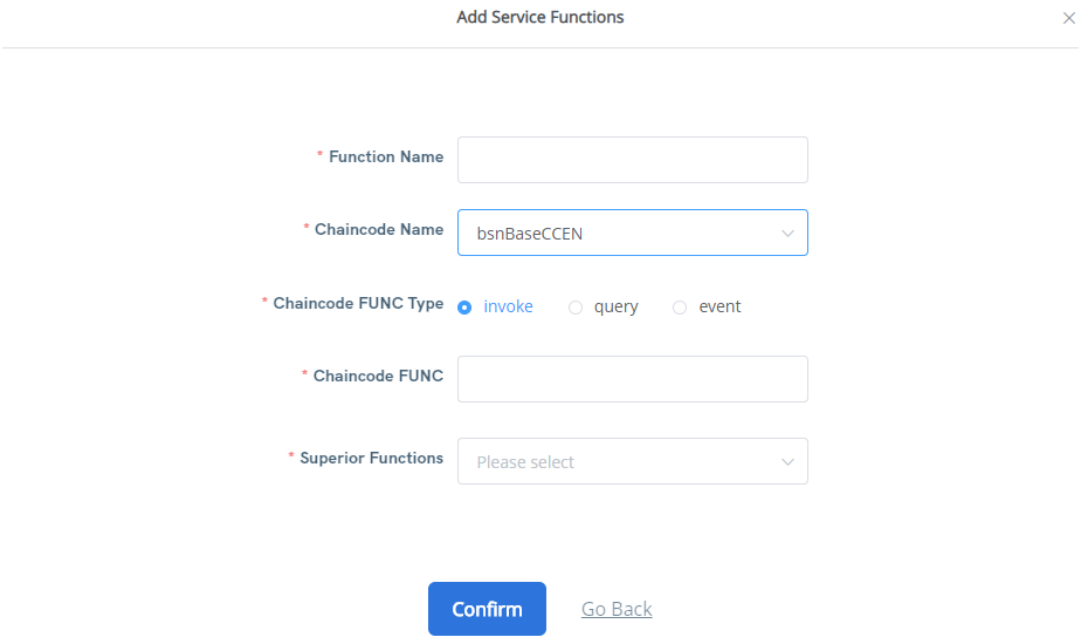
5.3.2.3 Define Service Functions and Roles

1. By selecting a **Preset chaincode package**, a set of automatic service functions are added to the service and each of the functions can be **Edited, Viewed**, or **Deleted.**



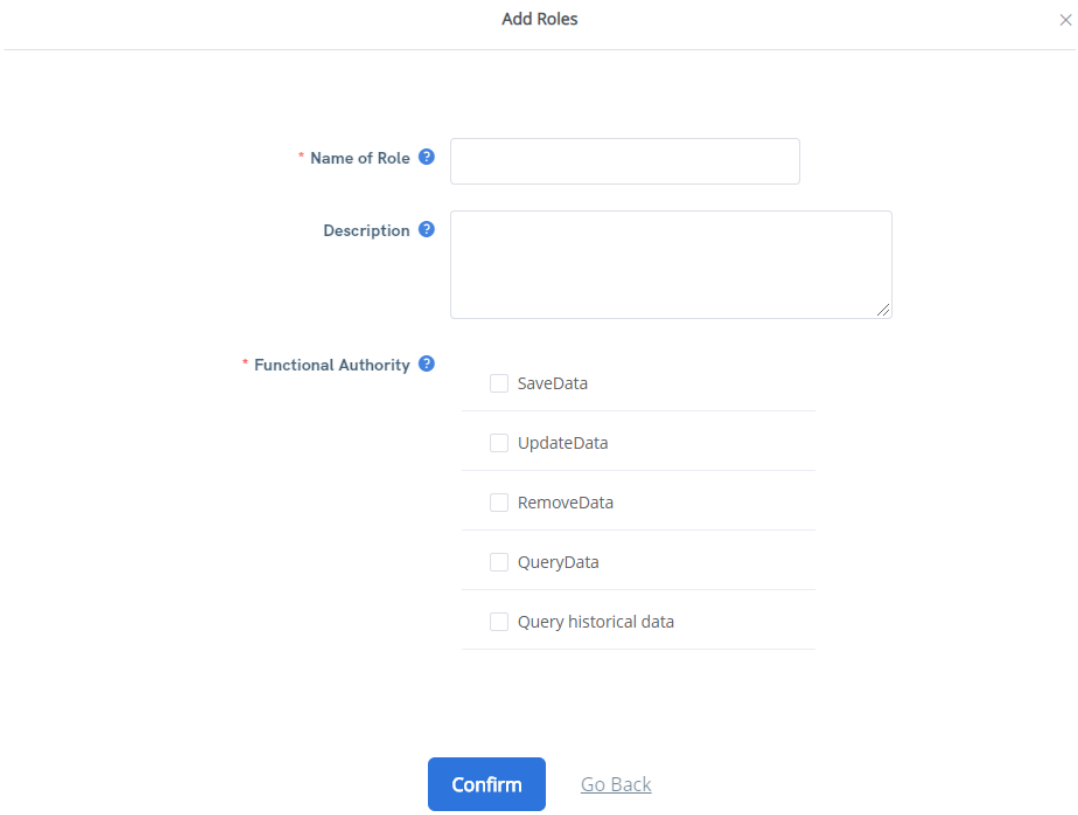
2. If you wish to add more functions, click the **Add Functions** button to display the dialog box. In it, enter or select the following:

   - **Function Name** – Enter a name for the function
   - **Chaincode Name** – Select from the list of chain codes
   - **Chaincode FUNC type** – Choose from **invoke**, **query** or **event**
   - **Chaincode FUNC** – Enter a description of the function
   - **Superior Functions** – Select a function from the list of functions in the system



3. Click **Confirm** to add it to the functions.
4. When the **Use Preset Chaincode package** is selected, a system administrator role is automatically created with full access to the published service. To create another role, Click **Add Roles** to display the **Add Roles** function and enter or select the following:

   - **Name of Role –** Enter a role name
   - **Description –** Enter a description for the role
   - **Functional Authority –**Choose one or more from the DApp's existing functions, for example: **SaveData, UpdateData, RemoveData, QueryData,** and **Query historical data from the preset chaincode package.**

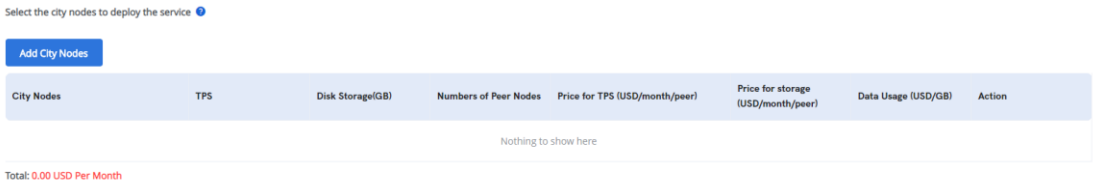


5. When done, click **Confirm** to add the role.

5.3.2.4 Select the Public City Nodes to deploy the service

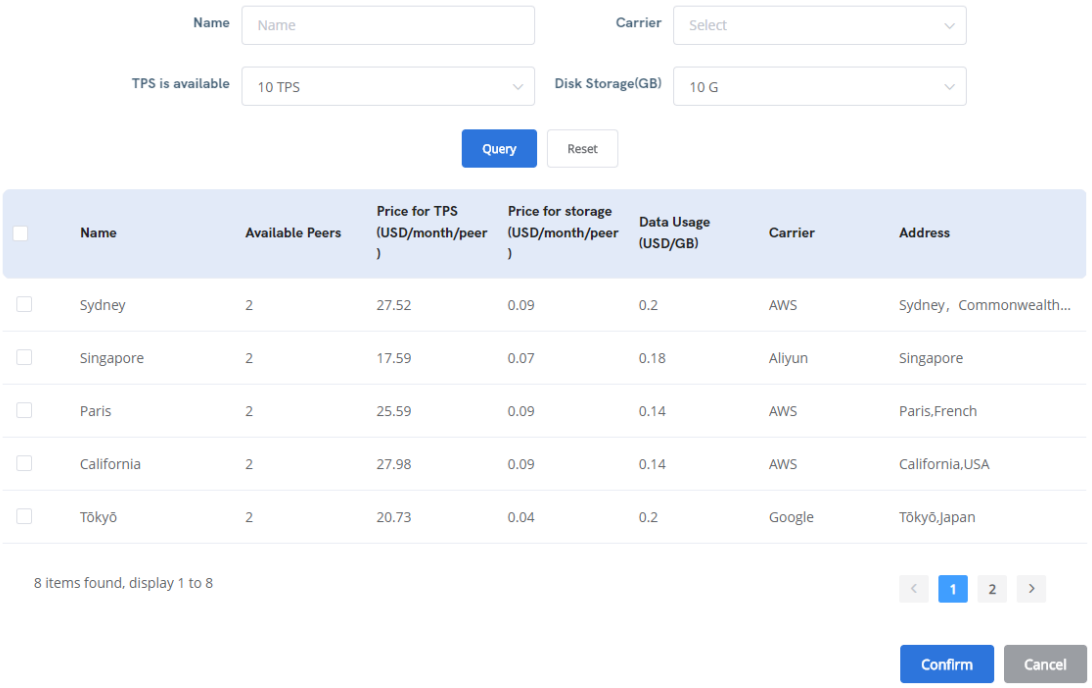
Public city nodes are used by permissioned DApp publishers to deploy DApp's peers and smart contracts. Publishers can deploy all peers into one or more PCNs, so that all peers connect together to form the DApp. We strongly suggest not to deploy all peers onto one single PCN for data safety reasons. To add a public city node, follow these steps

1. In the **Select the City Nodes to deploy the service** section, click **Add City Nodes.**



2. In the **Add City Nodes** window, enter or select:
   - **Name –** Enter a name for the city code
   - **Disk Storage (GB) –** 10 GB is allocated by default
   - **TPS is available –** 10 is allocated by default
   - **Carrier –** All carriers are listed, however, if you prefer a particular carrier, click the dropdown and select that carrier

3. Click **Search** to list carriers.
4. In the list of carriers, select more than one carrier for redundancy purposes. When done, click **Confirm**.



The city nodes that have enough resources according to the TPS and storage configuration are displayed alongside their costs. The resource costs are different for each public city node.

5.3.2.5 Select Certificate Mode

There are two certificate modes, **Key Trust Mode** and **Public Key Upload Mode.** The key trust mode certificates are generated and hosted by BSN while the public key upload mode certificates are generated by developers, and the private key is stored locally and the public key is uploaded to BSN. It is recommended that all developers use the **Public Key Upload Mode.**

1. To use the certificate mode, in the **Certificate Mode** section, click either **Key Trust Mode** or **Public Key Upload Mode**.



2. Click **Next** to continue.

5.3.2.6 Pay bills and submit for approval

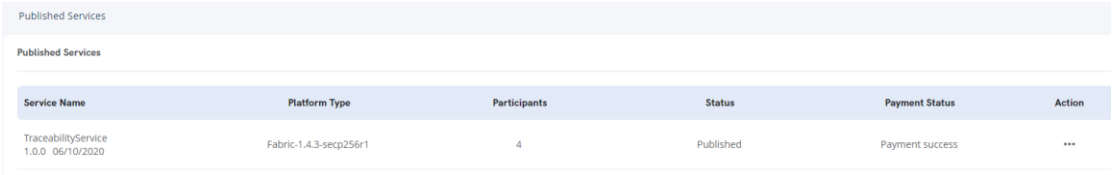
In the bill detail section, the resource usage fees from the added city nodes are displayed alongside a monthly total payment fee. If the bill is satisfactory, click the **Confirm** button to proceed and make the payment. However, if you need to make changes to the bill, click **Back** and make changes in the **Add City Nodes** section.

City Nodes	TPS	Disk Storage(GB)	Numbers of Peer Nodes	Data Usage (USD/GB)	Price for TPS (USD/month/peer)	Price for Storage (USD/month/peer)
Paris	10	10	1	0.14	25.59	0.09
Singapore			1	0.18	17.59	0.07
Sydney			1	0.20	27.52	0.09

Payment Amount: 73.20 USD Per Month 803.20 USD Per Year (Discounts 75.2 USD)

Once the payment is successfully made, you will receive an email in your mailbox informing you that your BSN service has been submitted successfully and will be reviewed. You will be informed via email when the reviewed has finished.

Once the service has been approved, the service will be seen in the **Published Services** section.



5.3.3 DApp Services Management

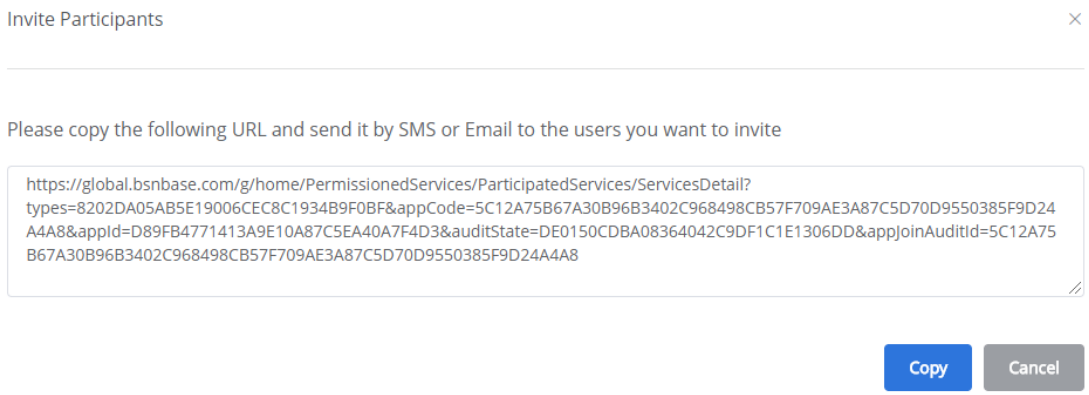
After the request for a service approval has been given, it will be listed in the Permissioned Services - **Published Services** section. For each listed service some Actions can be carried out. This includes **Invite participants, Basic Information Editing, Service Upgrade** and **Detail**.

5.3.3.1 Invite participants

After the service has been approved and the service is in use, you can invite other users of the blockchain network to participate in your service. To invite participants, follow these steps:

1. In the **BSN** menu, click the **Permissioned Services** dropdown and click **Published Services** to display the list of published services.
2. In the **Action** column, select the **Invite Participants** link to display the details to send to participants who intend to join the service.

Click **Copy** to copy the link details. This can be emailed to the participants who login with their BSN credentials to join or register with BSN first to use the service.



5.3.3.2 Basic Information Editing

After the service has been running and participants have joined, the publisher can edit basic information regarding the service including **service name, type, platform type, version, service logo, documents**, and **contact information.** To edit the basic information, follow these steps:

1. In the list of published services, locate the service to be edited. In the **Action** column of the service, select **Basic Information Editing** to display the editing page.
2. Add, edit or remove the basic detail of the service and click **Save** to store changes. If no changes were made click **Back** to return to the **Published Services** page.

5.3.3.3 Service Upgrade

After a service has been published, the publisher can use the **Service Upgrade** option to update the smart contracts and other functions. It will be reviewed again before it can be used. To edit the **Service Upgrade**, follow these steps:

1. In the list of published services, locate the service to be edited. In the **Action** column of the service, choose **Service Upgrade.**
2. In the **Basic Information** page, change the **Version Number**, which is mandatory and/or any other details in the **Basic Information** page. Click **Next** to upload the new smart contracts and set functions and roles as described before.

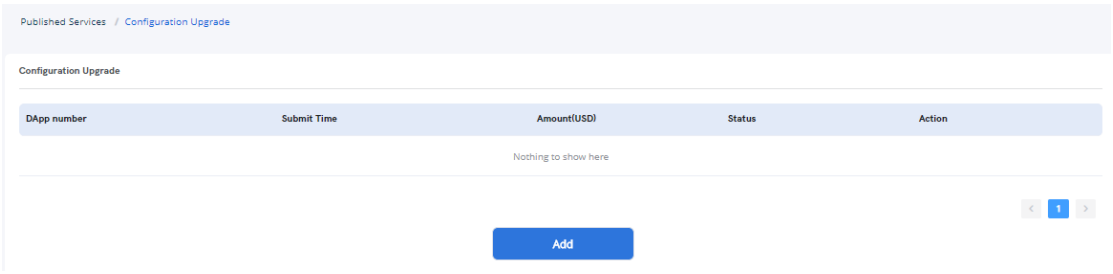
When done, click **Confirm**

5.3.3.4 Configuration Upgrade

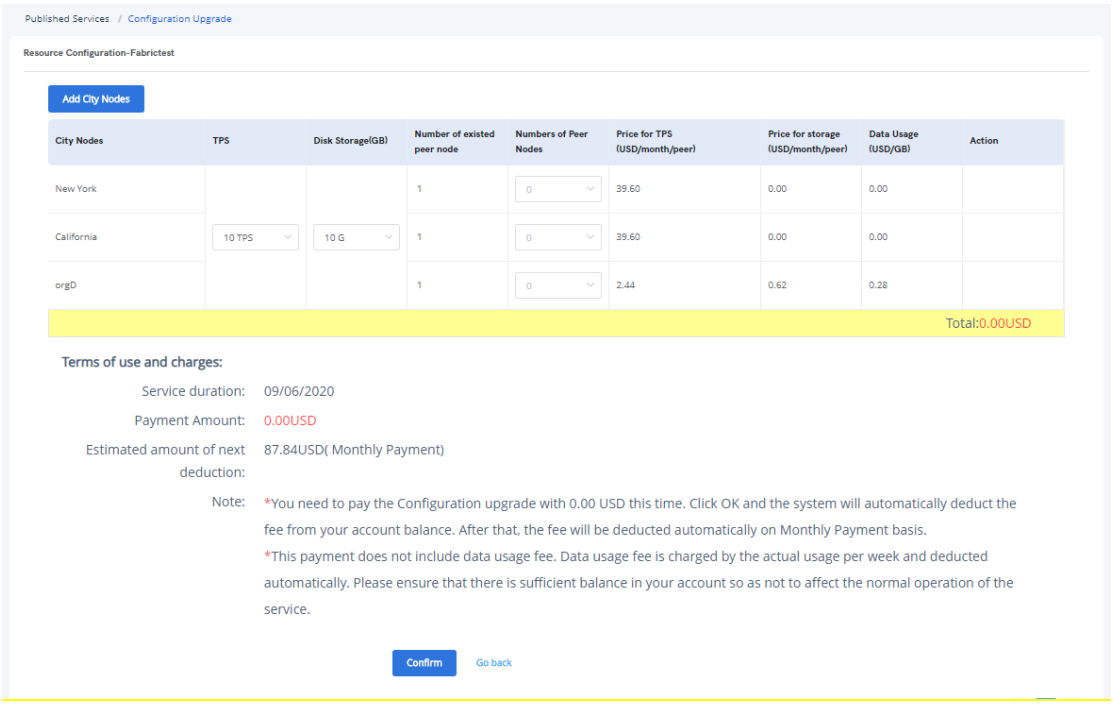
In order to join the DApp services, the publisher should send out invitation links to the potential participant. The potential participant can then click on the link to the services' main page and apply for the service.

To upgrade the configuration, follow these steps:

1. Go to **Published Services** and select the enabled service on the list. Click **configuration upgrade** to enter the configuration upgrade list page as below:



2. Click **Add** to create a configuration upgrade application form, and then click **Add city nodes** to add new city nodes:



3. Click **Submit** to submit the configuration upgrade application. When submitting, the system will prompt the publisher to pay the corresponding configuration upgrade fee. After the publisher confirms, the system generates the configuration upgrade bill and deducts money from the user's credit card. Whether the payment is successfully charged, or not, the configuration upgrade application will go through the review process. If the payment is successfully charged and the application is approved, the system will conduct a configuration upgrade process and complete the upgrade; if the charge fails, the bill will be kept for 72 hours and then expires. If the publisher still wants to upgrade the configuration, he/she needs to apply again.

Note: The fee paid when configuring the upgrade is the upgrade fee, which makes up the difference in the remaining payment period between the pre-upgrade configuration and the post-upgrade configuration of the billing cycle. After the upgrade is successful, future charges will be made according to the new configuration from the next period.

5.3.3.5 Detail

The **View** option allows the publisher to view all the details of the published service including **Basic Info**, **Deployment**, **Roles, Approval records**, **Ledger Info**,

Comments, and **Historical Version.** To view these options, follow these steps:

1. In the list of published services, locate the service to be edited. In the **Action** column of the service, click **Detail** to display the view page tabs.
2. In the **Basic Info** tab, you can see all the details of the service that has been deployed including the **Service name**, **Service type**, **Version**, **Platform type**, **Service logo**, **Service Introduction, Service description, Documents,** and **Contact Information.**
3. In the **Deployment** tab, the information that can be viewed includes the **Chaincode package**, **Service function**, and **City Nodes**.
4. In the **Roles** tab, the roles and their related functions are listed. To **View** a role, click on the **view** link for that role name.
5. In the **Approval records** tab**,** you will see all the requested approval and their status as well as time logs.
6. The **Ledger Info** tab shows more information about the published service than any other tab. It shows the **number of users accessed**, **number of transactions, peer nodes, chaincodes, blocks**, and **logs** of how the activities took place.
7. The **Comments** tab shows the comments made on the published service that can be viewed by the publisher.
8. The **Historical Version** tab shows the history of the service including the **Service Name, Version, Service Type, Service Introduction,** and **Action.**

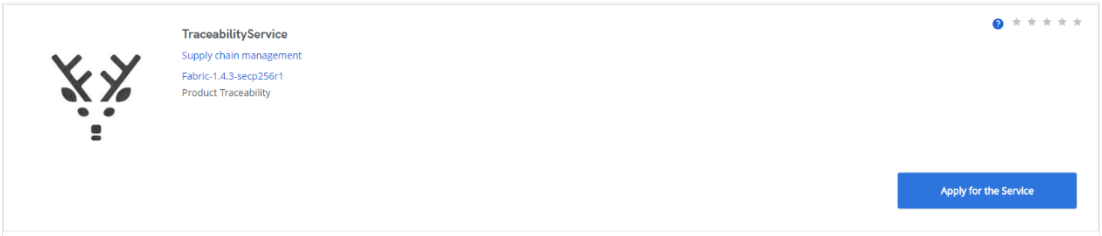
5.3.4 DApp Services Participation

In order to join the DApp service, the publisher should send out invitation links to the potential participant. The potential participant can then click on the link to the services' main page and apply for the service.

5.3.4.1 Apply for a Service

To apply for a service, follow these steps:

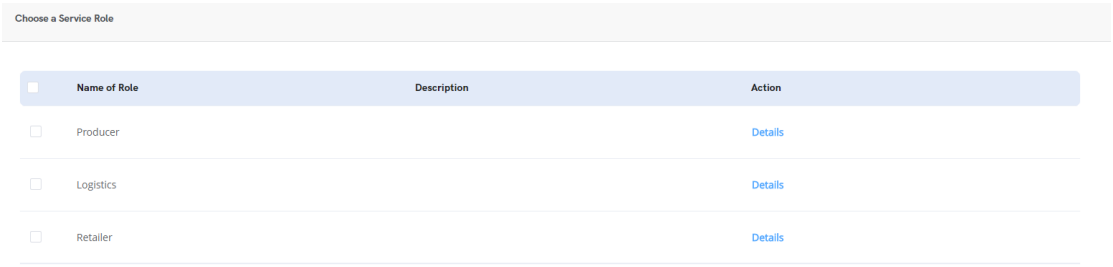
1. Click the link that was shared. This will take you to the service information page.
2. In the service header, click **Apply for the Service**.



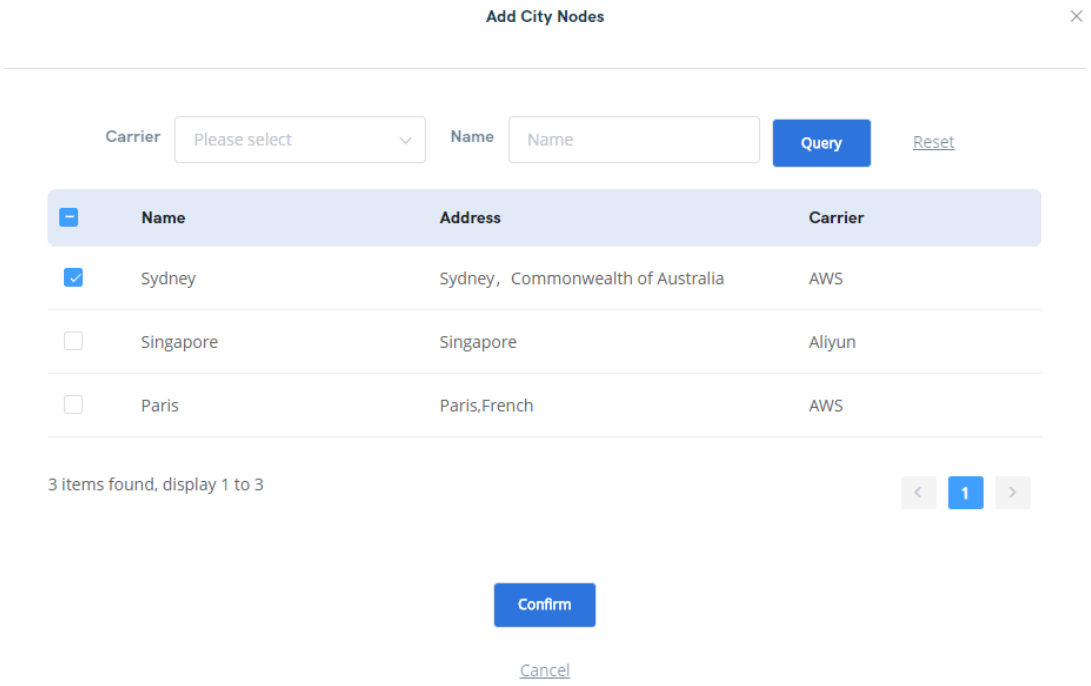
5.3.4.2 Select Roles and City Nodes

1. In the list of roles, select a role you want to use. You can click the **View** link in each of the roles to see more details about the role. You can select more than one

role.



2. In the Public **City nodes**, click **Add city nodes** to display the Public City Nodes the DApp is deployed on. You can select more than one node. The selected nodes' gateways are where the off-BSN systems connect to. Please select the public city node that is closest to you.

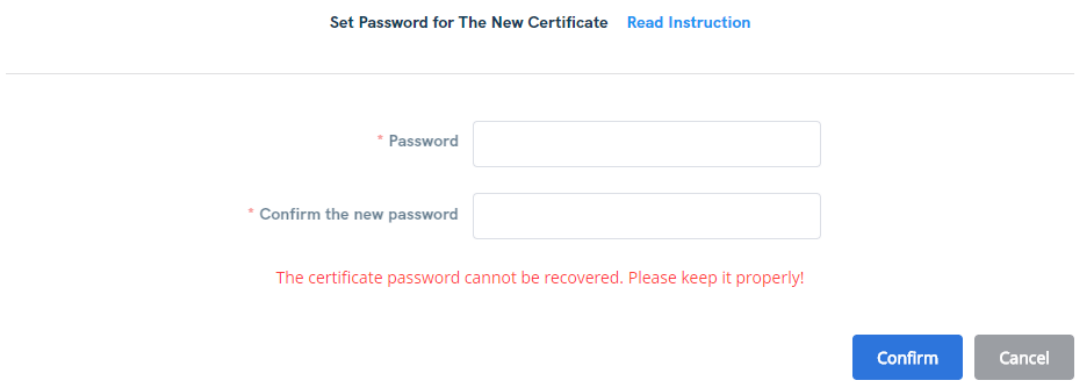


Click **Confirm** to view the nodes that were selected.

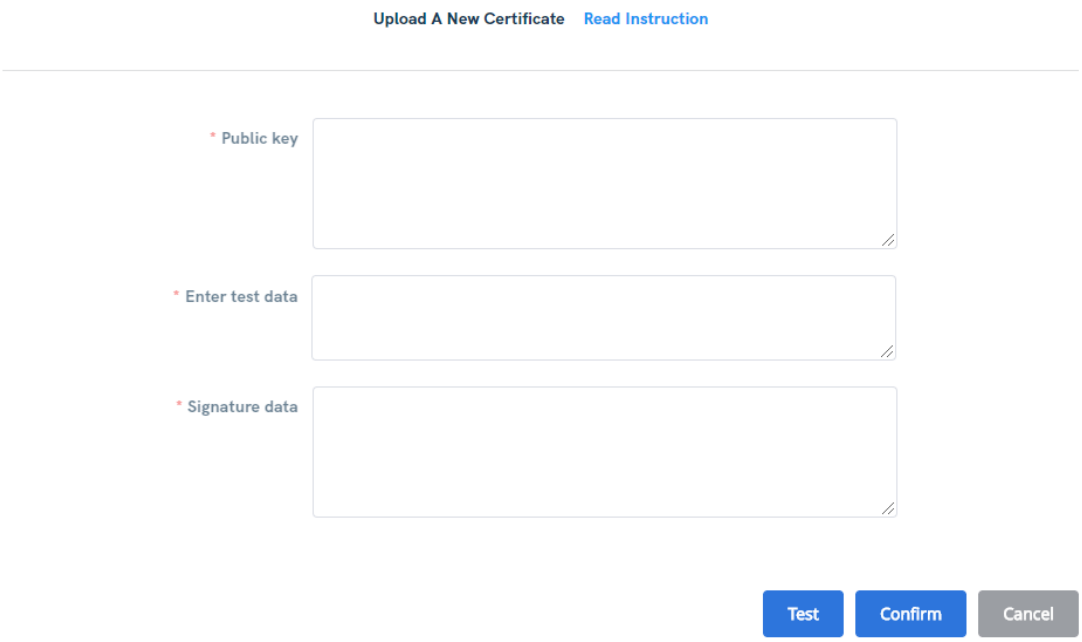
5.3.4.3 Apply Certificate Mode

Depending on the settings of the service publisher, there are two certificate modes for service participation: Key Trust Mode and Public Key Upload Mode.

Key Trust Mode: Participants can select existing certificates on the city node or apply for a new certificate.

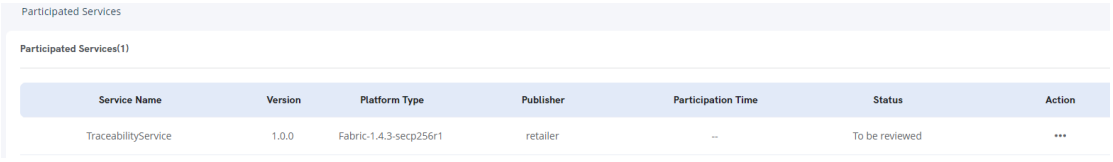


Public Key Upload Mode: Participants should upload the public key, test data and signature data. The generation of public and private keys can be viewed by clicking **Read Instruction**.



5.3.4.4 Submit for approval

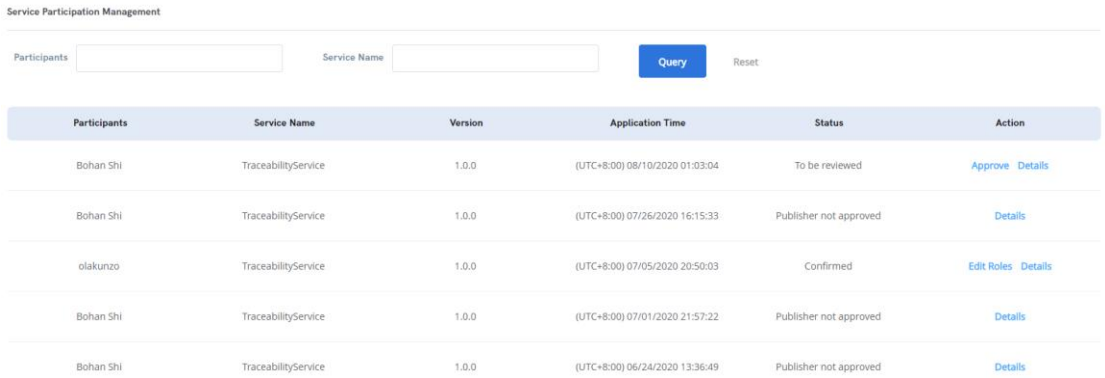
Click **Confirm** to join the service pending the **publisher's approval**.



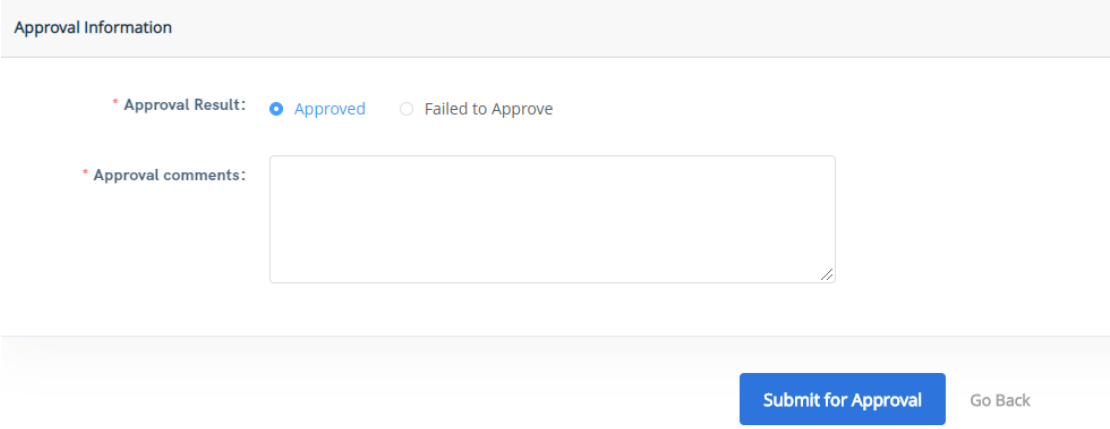
5.3.4.5 Approve a service

As the publisher of a service, in the **service participation list**, the publisher has can approve, deny or disable a participant from using the service. To perform any of these actions, follow these steps:

1. In the **Service Participation List** section, locate the participant to review.



2. For the participant to be reviewed, click the **Review** link in the **Action** column to view the participant details. In the **Approval Information** section select either **Approved** or **Failed to Approve** and write a comment in the **Approval Comments** box to give details.
3. Click **Submit for Approval** or **Back** to return to the participant's list.



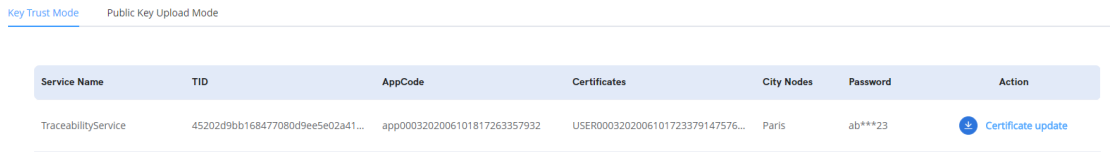
4. If the participant is approved, a message will prompt showing that the service participation approval was successful.
5. After the approval has been given, the participant can view the service from their **Participated Services** page as well as add more **city nodes**.

5.3.4.6 Download and renew a certificate

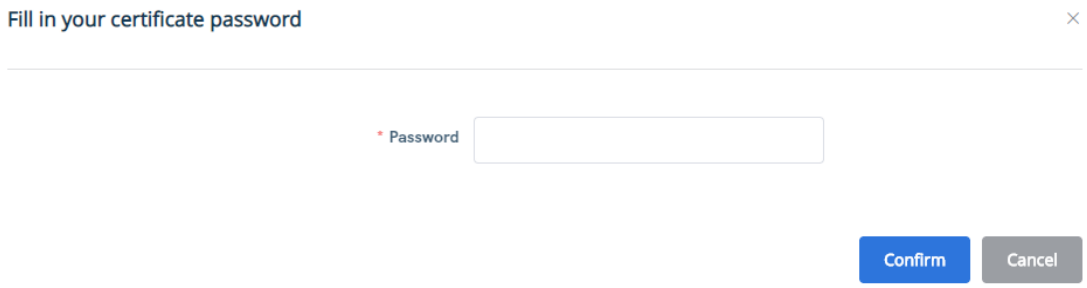
The BSN development team intends to build BSN into a most secure blockchain infrastructure network. The certificate and key mechanisms of BSN are complex. There are two kinds of key pairs used in generating certificates: DApp Access Key Pair and User Transaction Key Pair. For each, there are two modes, the Key Trust Mode and the Public Key Upload Mode. To work with certificates, follow these steps:

Key Trust Mode:

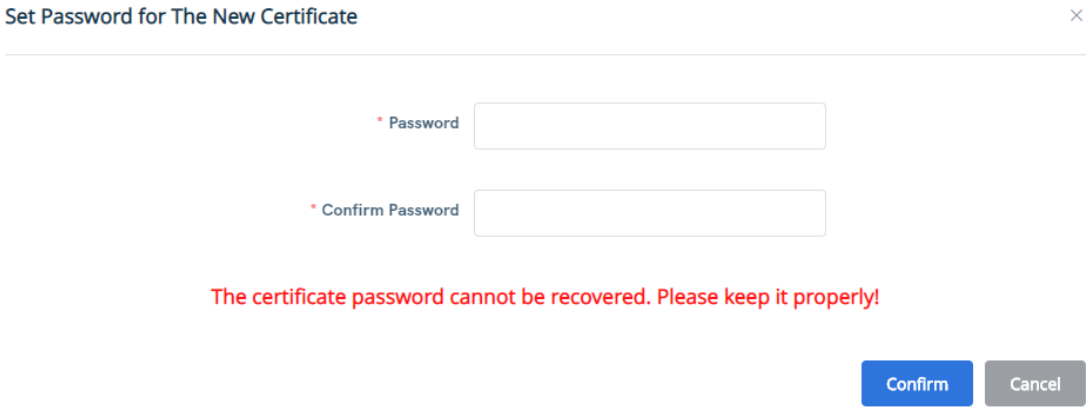
1. In the **My Certificates** menu, click **Key Trust Mode**. The certificate page will be displayed.



2. To download the certificate, click the icon. You will be required to enter the certificate password.



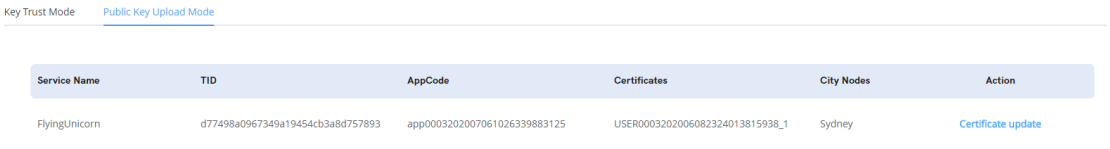
3. To update the certificate, click the **Certificate update** link. You will be requested to set a password for the certificate and confirm the password.



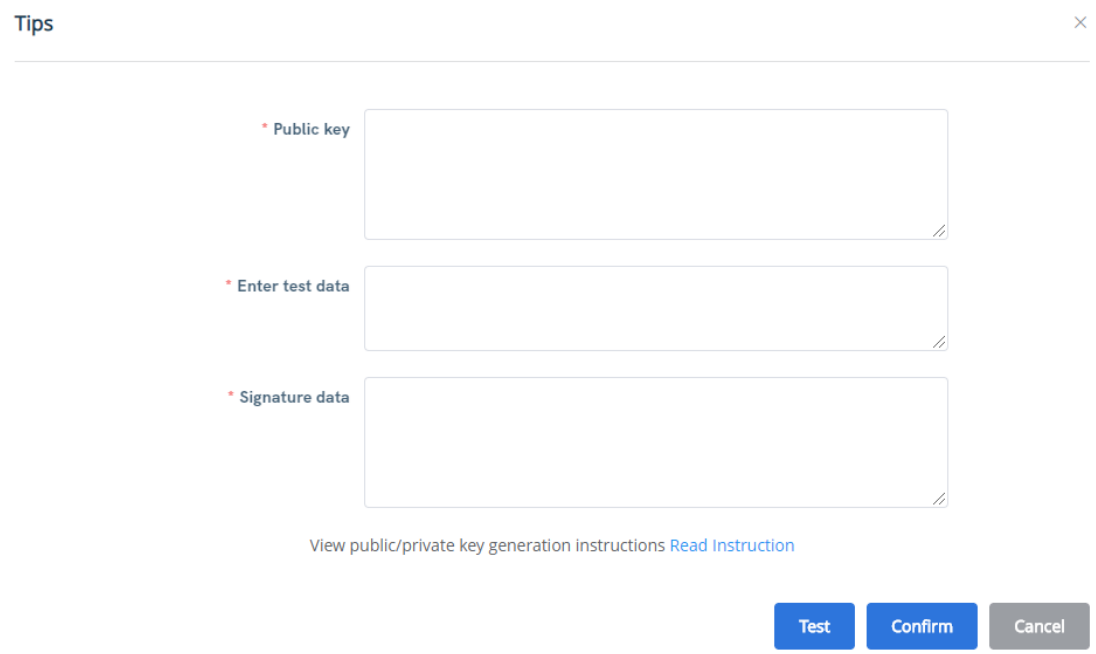
4. Click **Confirm** to update the certificate.

Public Key Upload Mode:

1. In the **My Certificates** menu, click **Public Key Upload Mode**. The certificate page will be displayed.



2. To update the certificate, the public key, test data and signature data need to be re-uploaded, and the update can only be completed after the test passes.

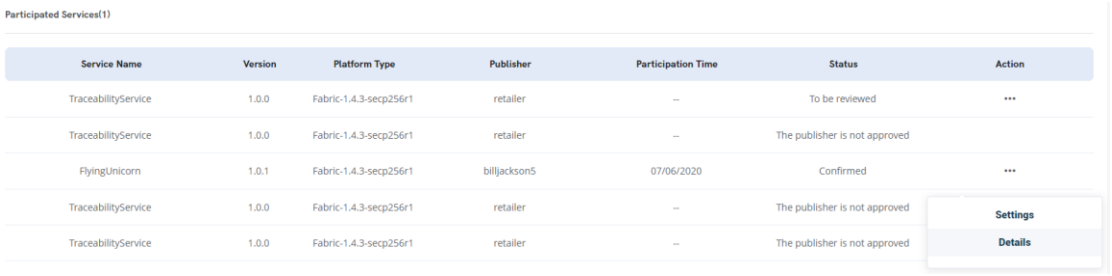


3. The user only needs to upload the public key in the Public Key Upload Mode. The private key is kept locally by the user, so there is no need to download the certificate.

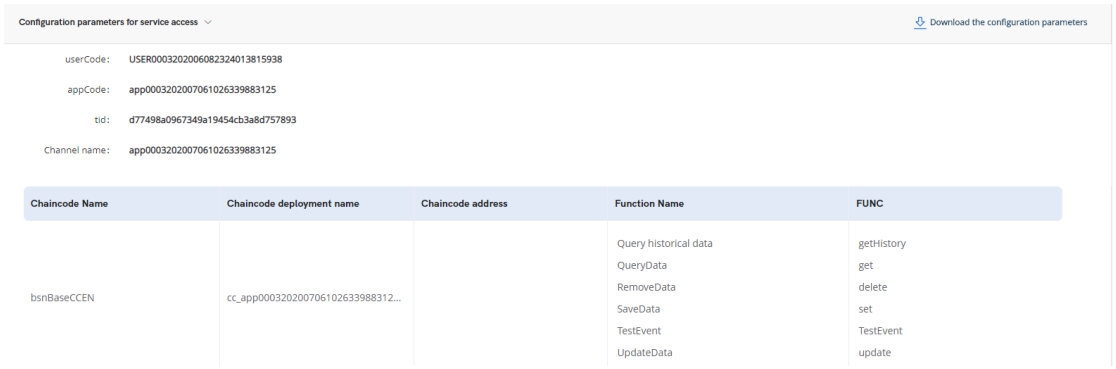
5.3.4.7 Configuration parameters for service access

To view and download the configuration parameters, follow these steps:

1. In the **Permissioned Services** menu, click **Participated Services**.
2. In the list of services, click the **Detail** option in the **Action** column for the service.



3. Click the dropdown beside the **configuration parameters for service access** to view its configuration.



4. To download the **parameters for service access**, click **Download the configuration parameters** to begin the download.

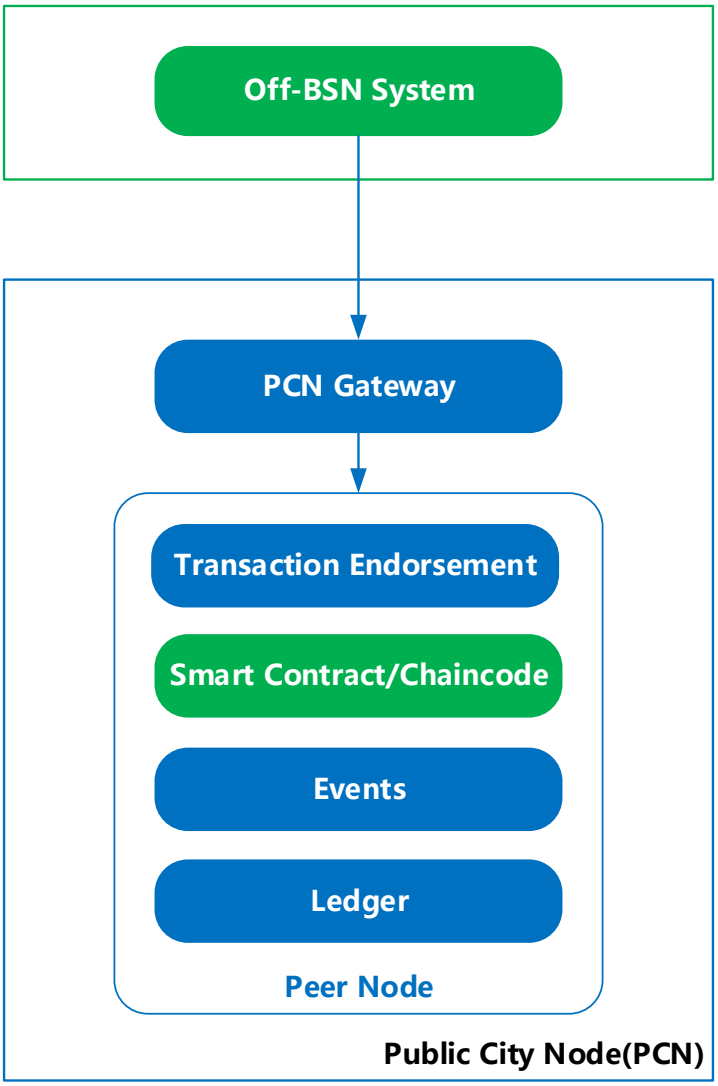
5.4 Off-BSN system Access Guide

5.4.1 Overview

Blockchain-based Service Network (hereinafter “Service Network” or “BSN”) is a cross-cloud, cross-portal, cross-framework global infrastructure network to deploy and operate all types of blockchain and distributed ledger technology (DLT) applications (DApp).

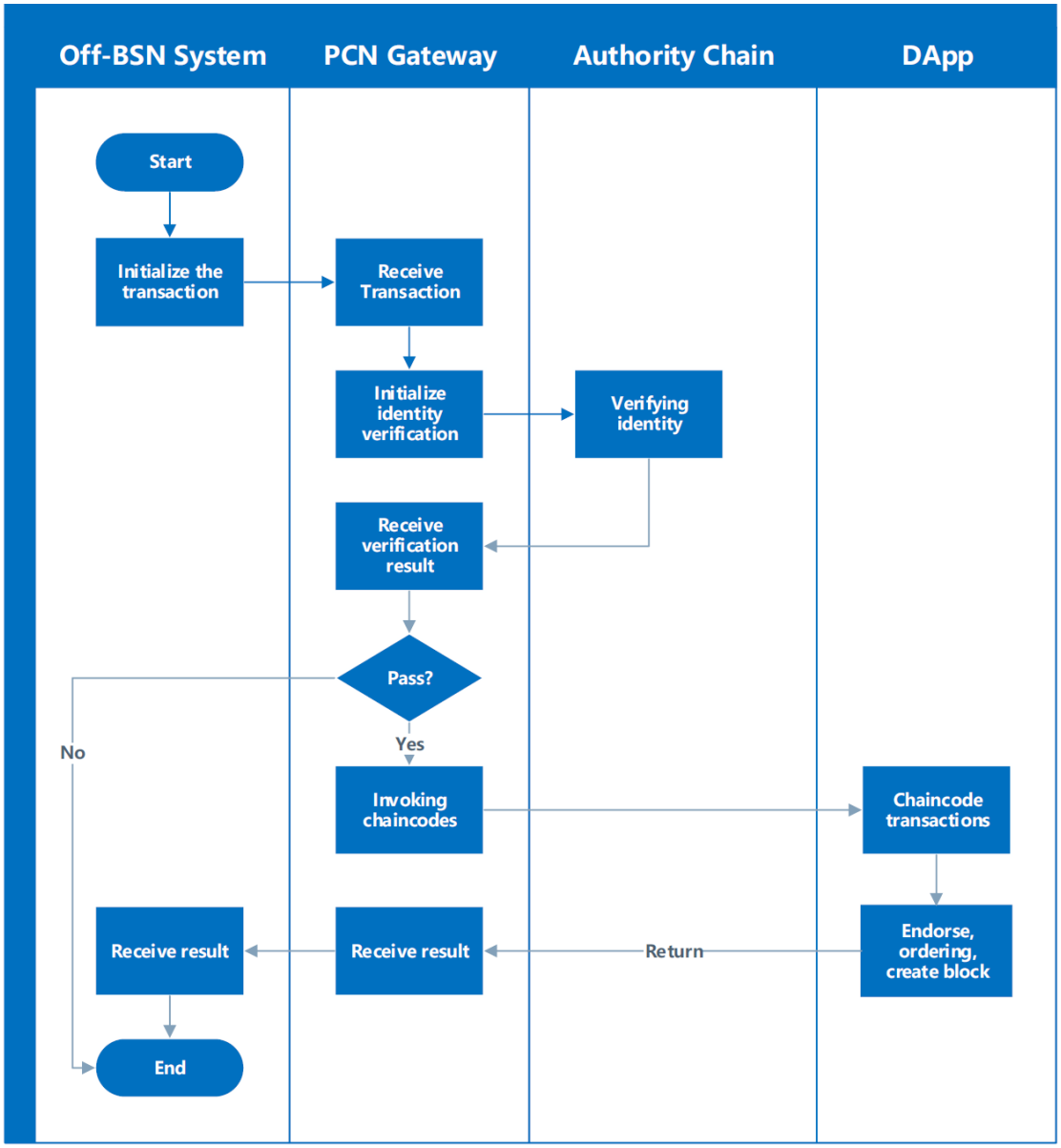
BSN aims to lower the cost of developing and deploying DApps by providing public blockchain resources and environment to developers, just like the internet. It can further reduce the costs associated with the development, deployment, operations, maintenance, and regulation of DApps and, thereby, accelerate the development and universal adaptation of blockchain and DLT technologies.

A complete DApp system based on BSN generally consists of two parts: the on-BSN DApp smart contracts and the off-BSN systems. The off-BSN systems use the BSN Public City Note (PCN) gateways to invoke the DApp smart contracts deployed on the PCN to carry out on-chain operations such as executing transactions, writing data chain, data queries, etc. The DApp service publishers and participants can deploy their off-BSN systems on any cloud services they choose and then connect to the BSN PCN gateways through the internet access DApp smart contracts and data.

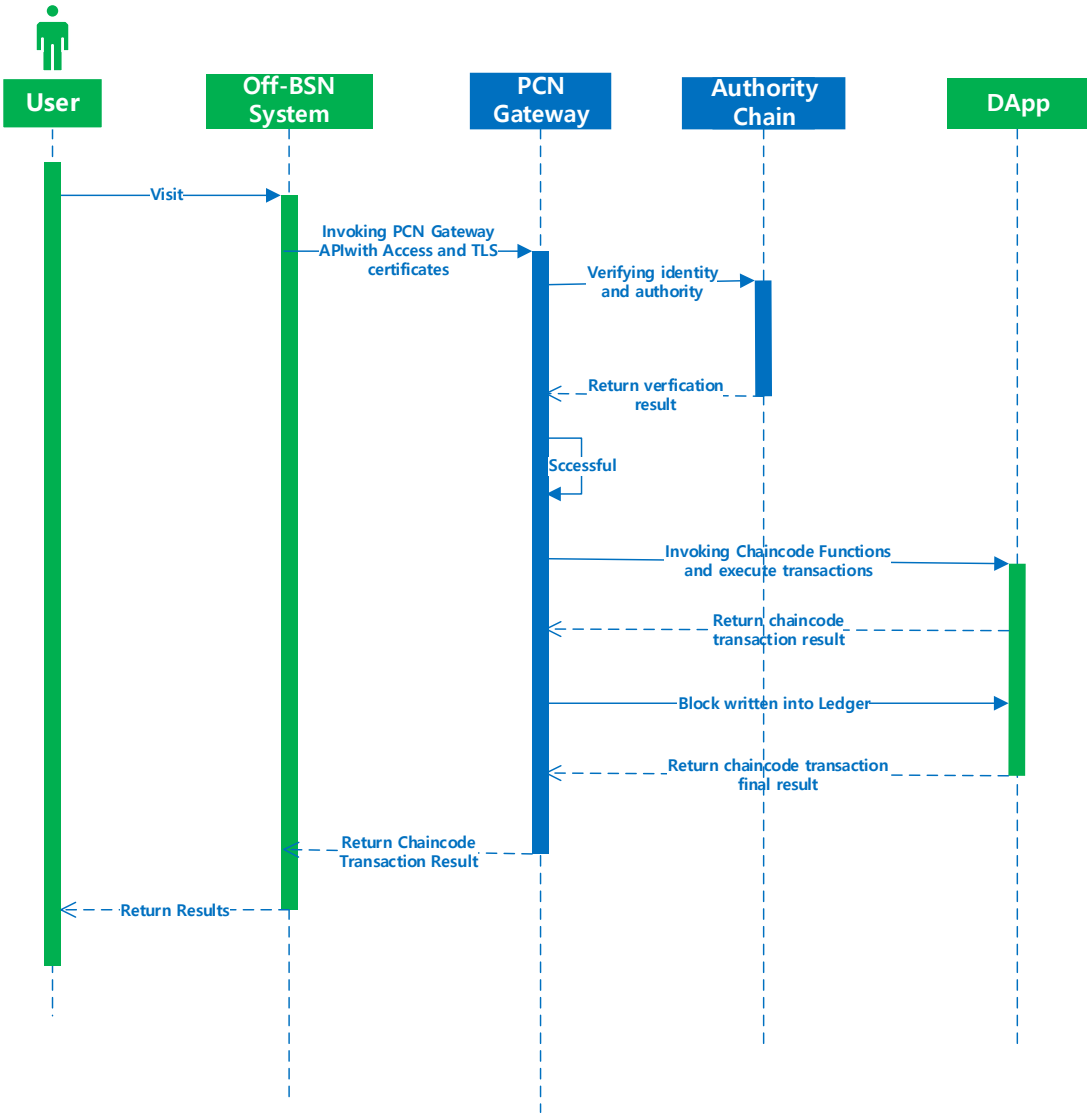


The BSN DApp service publishers and participants should have their off-BSN systems so that they can access the DApp smart contracts to execute transaction and query data via the PCN gateway APIs. The following are the charts to show the connecting flow and transaction sequences.

Off-BSN System Connection Flow:


Off-BSN System
PCN Gateway
Authority Chain
DApp
Start
Initialize the transaction
Receive Transaction
Initialize identity verification
Verifying identity
Receive verification result
Pass?
No
Yes
Invoking chaincodes
Chaincode transactions
Receive result
Receive result
Return
Endorse, ordering, create block
End

Off-BSN System calling sequence:



5.4.2 BSN Smart Contract Package Requirements

A smart contract, also known as chaincode in Hyperledger Fabric, is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts allow the performance of credible transactions without a third party. These transactions are trackable and irreversible. A smart contract is invoked to automatically execute a transaction and operate ledger data. A DApp service on BSN can deploy multiple smart contracts. Each smart contract can contain multiple functions.

5.4.2.1 Hyperledger Fabric smart contract package requirements

Hyperledger Fabric ("Fabric") chaincode can be compiled by multiple programming languages, including Golang, java, and node.js. Each chaincode program must implement a chaincode interface which usually consists of three basic functions: Init, Invoke, and Query.

- **Init:** This function is called during the chaincode instantiation and its purpose is to prepare the ledger for future requests. This function must be implemented in all chaincodes.
- **Invoke:** The Invoke function is called for all future requests from the off-BSN systems towards the DApps. Here all DApp custom functions or what the DApps can do (for example, to read data from the ledger, to write data in the ledger, to update data, to delete data) are defined. Simply put, Invoke can be understood as an entry point to the chaincode functions. The Invoke function also must be implemented in all chaincodes.
- **Query:** The Query function provides a method of querying ledger data. This function can only be used for query purposes and does not offer any operations of ledger data. The Query function is not required to be implemented in all chaincodes.

To realize the automatic deployment of DApp services and to improve deployment efficiency, the following Fabric chaincode packaging requirements have been issued with different programming languages.

1. Golang

The main function must be at the same or higher level as all chaincodes in the project. The zipping path must be the same level folder where the main function is located, and the main function path is the src-based path.

Example: BsnBaseCC Package (the preset chaincode package)

```
BsnBaseCC
├─main.go
├─ChainCode/
├─models/
└─utils/
```

The package should be zipped under BsnBaseCC/ (package name is not required), and the main function path (reference path) is BsnBaseCC.

Example: FabricBaseChaincode chaincode package on github (preset chaincode package)

github.com

└──BSNDA

└──FabricBaseChaincode

└──chaincode

└──go

└──bsnBaseCC

└──main.go

└──ChainCode/

└──models/

└──utils/

It should be zipped under github.com/BSNDA/FabricBaseChaincode/chaincode/go/bsnBaseCC/ (package name is not required), and the main function path (reference path) is github.com/BSNDA/FabricBaseChaincode/ chaincode/go/bsnBaseCC.

Description: main.go: the entry; ChainCode: chaincode; models: entities; utils: utilities.

2. Java

gradle or maven-built projects, the projects must contain build.gradle or pom.xml files.

Example: BsnBaseCC package

BsnBaseCC

└─build.gradle

└─src

└─main

└─java

└─com.example.javacc

└─javacc.java

Package needs to be zipped under BsnBaseCC/. Zip package name is not required.

Description: src/main/java: project directory; com.example.javacc: package name; javacc.java:

chaincode information

3. Node.Js

   package.json must be built into the project's root directory. Package needs to be zipped under BsnBaseCC/. Zip package name is not required.

   Example: BsnBaseCC package

   BsnBaseCC

   └marbles_chaincode.js

   └package.json

Description: marbles_chaincode.js: chaincodes

Note: when publishing DApp services in the BSN portal, chaincode packages should be created in the project's root directory using .zip format.

5.4.2.2 Hyperledger Fabric prebuilt smart contract package

A prebuilt chaincode package (Golang) is provided to BSN developers which contains basic functions such as add, delete, edit, and query. New DApp developers can learn from this package about Fabric chaincode programming and further extend the functions if needed. The chaincode in this package supports data types such as string, integer, float point, and sets (map, list), etc.

Please click this link to download:

https://github.com/BSNDA/FabricBaseChaincode

DApp publishers can also select the prebuilt chaincode package directly from the DApp publishing page on the BSN portal.

The Prebuilt Chaincode package functions are as follows:

1. Add data (set)

   Input parameter description:

   baseKey: a unique primary key identifier of data

   baseValue: stored data information

   Example: {"baseKey": "str","baseValue": "this is string"}

   Of which, the baseKey cannot be a blank string and the baseValue can be any type of data. If the baseKey already exists, then directly return that it already exists and cannot be added; if it does not exist, then add data.

2. Edit data (update)

   Input parameter description:

baseKey: a unique primary key identifier of data

baseValue: stored data information

Example: {"baseKey": "str","baseValue": "this is string"}

Of which, the baseKey cannot be a blank string and the baseValue can be any type of data. If the baseKey does not exist, then it cannot be updated; if it already exists, then update the data.

3. Delete data (delete)

Input parameter description

baseKey: a unique primary key identifier of data

Example: "str"

Of which, the baseKey value cannot be blank and must exist, else it cannot be deleted.

4. Get data (get)

Input parameter description

baseKey: a unique primary key identifier of data

Example: "str"

Of which, the baseKey value cannot be blank and must exist, else it cannot be retrieved.

5. Get history ledger data (getHistory)

Input parameter description

baseKey: a unique primary key identifier of data

Example: "str"

Of which, the baseKey value cannot be blank. Response results: transaction Id (txId), transaction time (txTime), whether to delete (isDelete) and transaction information (dataInfo).

5.4.2.3 FISCO BCOS smart contract package requirements

To realize automatic audit and deployment of FISCO BCOS (FISCO) DApp services and to improve efficiency, the following FISCO smart contract packaging requirements have been issued:

1. Package Structure of the Solidity smart contract

All smart contracts must be stored in a single-level folder including smart contracts, libraries, and external contract interfaces. Import method of all contracts is import "./XXXX.sol".

2. Smart Contract deployment instruction document (deploy.md)

deploy.md is used to explain clearly how the smart contract is initialized and deployed. It consists of three main parts:

- Contract Description: to briefly describe the basic information of each contract.
- User Description: to describe the basic information of each transaction signing users during initialization and deployment.
- Contract initialization description: to describe the steps of smart contract initialization and deployment, so that BSN tech personnel can follow to complete the process.

3. Contract uploading specifications

When uploading a chaincode package (smart contract package), fill in the chaincode name (contract name) that is consistent with the main contract class name and the main contract file name.

Example: BsnBaseGlobalContract chaincode package (preset chaincode package)

BsnBaseGlobalContract

└──BsnBaseGlobalContract.sol

└──Table.sol

Package must be zipped under BsnBaseGlobalContract/. The zipped package name is not required. If the main contract class name is BsnBaseGlobalContract, the main contract file name should be BsnBaseGlobalContract.sol, and the chaincode name (contract name) must be filled in as BsnBaseGlobalContract.

4. BSN Adaptation for FISCO Solidity Version Descriptions

Currently, FISCO BCOS in the BSN only supports Solidity 0.4.25 and older versions.

5.4.2.4 FISCO BCOS prebuilt smart contract package

The FISCO Prebuilt Smart Contract package is chosen from the Table.sol provided by the FISCO BCOS development team, and can provide developers with basic functions such as insert, remove, update, or query (using Solidity). New DApp developers can learn from this package about FISCO smart contract programming and further extend the functions, if needed. The stored data types supported by this smart contract include int256(int), address, and string, of which string cannot exceed 16MB. To ensure on-chain performance, there is no analysis of duplicate base_id and base_key. This should be handled by the off-BSN system. It is recommended that each base_id has only one corresponding base_key and base_value.

Please click this link to download:

https://github.com/BSNDA/FISCOBaseContract

The prebuilt smart contract functions are as follows:

1. Insert data (insert)

Input parameter description

base_id: the primary key identifier that requires inserting

base_key: the key of the data to be inserted

base_value: the value of the data to be inserted

Example: {"base_id": "1","base_key":1,"base_value":"this is string"}

Of which, base_id and base_value cannot be blank strings and the base_key is in int256 data type.

2. Update data (update)

   Input parameter description

   base_id: the primary key identifier that requires updating

   base_key: the key of the data to be updated

   base_value: the value of the data to be updated

   Example: {"base_id":"1","base_key":"1","base_value":"this is string"}

   Of which, base_id and base_value cannot be blank strings and the base_key is in int256 data type. If the base_id and base_key do not exist, then they cannot be updated; if they already exist, then the data will be updated.

3. Remove data (remove)

   Input parameter description

   base_id: the primary key identifier that requires removing

   base_key: the key of the data to be removed

   Example: {"base_id":"1","base_key":"1"}

   Of which, the base_id and base_value cannot be blank and must exist, otherwise they cannot be removed.

4. Select data (select)

   Input parameter description

   base_id: the value of the primary key identifier that requires being selected

   Example: {"base_id":"1"}

Of which, the base_id cannot be blank and must exist, otherwise, it is not possible to select the corresponding data.

5.4.3 PCN Gateway Fabric API

A PCN gateway is deployed on each public city node (PCN) to receive off-BSN system requests signed and verified by DApp access keys. Then requests are routed to the corresponding Fabric-based DApp chaincodes. Invoking the PCN gateway is realized by sending HTTP requests to each PCN gateway service. The gateway is responsible for verifying user and application identities and then uses these identities and chaincode functions to process chaincode parameters and to send the chaincode transaction results back to the off-BSN systems.

5.4.3.1 DApp Access Signature Algorithm

Whenever an off-BSN system sends requests to the PCN gateway, the HTTP request message should be signed with the participant's DApp access private key. When the PCN gateway receives the message with the digital signature, it will verify the authentication and message integrity with the corresponding hosted or uploaded DApp access public key. The gateway will only process the request message further after the verification is passed.

1. Assemble signature string

Convert the request parameters into a joined string according to the order of the parameter table,

of which the request parameter prioritizes joining UserCode and AppCode of the Header and the response parameter prioritizes joining code and msg. Then join the parameters in the Body according to the order of the parameter tables in the definition of APIs.

2. Different type conversion formats

Type	Rule	Example	Result
String	No conversion	abc	Abc
Int/int64/long	Decimal conversion	-12	-12
Float	Decimal conversion; see notes for values after decimal point	1.23	1.23
Bool	Convert to “true” or “false”	true	True
Array	Join according to parameter sequence and type	{“abc”, “xyz”}	Abcxyz
Map[key]value	Join key and value according to parameter sequence	{“a”:1, “b”:2}	a1b2
Object	Convert the attributes in the object one by one according to the document in the above-described format	{“name”: “abc”, “secret”: “123456”}	abc123456

3. Signature rules

- Getting the Hash value - The converted string to be signed is required to be computed with the SHA256 algorithm with UTF-8 encoding.
- Sign the Hash value - The hash value and private key should be encrypted with the ECDSA (secp256r1) algorithm. If signed with SHA256WithECDSA, which includes hash value computation, the first step is not necessary.
- Encoding the signature result to Base64.

4. Example

Parameters:

{"header":{"userCode":"user01","appCode":"app01"},"mac":"","body":{“userId”:”abc”,”list”:[“abc”,”xyz”]}}

Result: user01app01abcabcxyz

5.4.3.2 Key and Certificate Modes

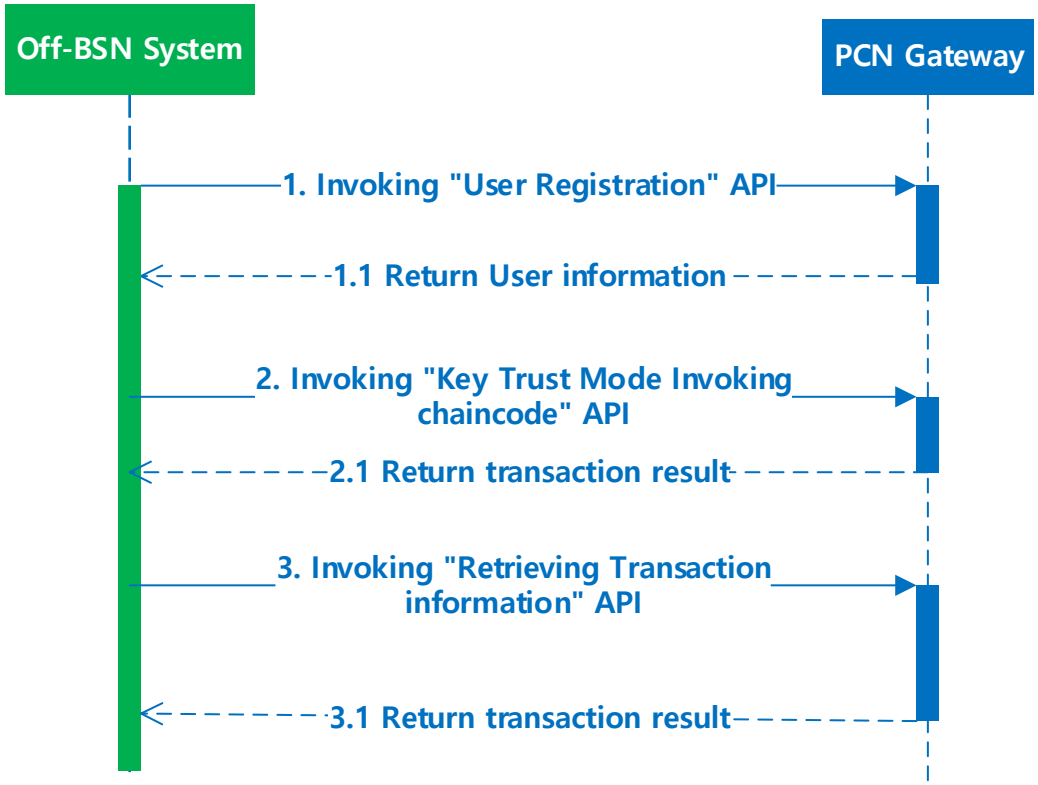
1. Key Trust Mode

As described in chapter 5, DApp participants require two sets of key pairs to access the DApp: DApp access key pair and user transaction key pair. With key trust mode, the pairs are generated and hosted by BSN. The participants only need to download the private key (DApp access key) from the BSN portal.

- DApp Access Key Pair: After the participant has successfully joined the DApp, BSN will generate one key pair (private and public keys) that corresponds to the DApp’s framework algorithms under the Key Trust Mode. The participant can download the private key from the “My Certificates” section of the BSN global portal and use it to sign the request message sent to the PCN gateway. The gateway will use the hosted public key from the generated key pair to validate the signature.
- User Transaction Key Pair: This is the identity of a participant used to invoke the

chaincodes. Under the Key Trust Mode, after successfully joining a DApp, a participant's user transaction key pair will be created automatically by BSN by default. The participant's off-BSN system can use the participant's UserCode to invoke the certificate generated by the key pair. If the participant's off-BSN system has multiple sub-users, the off-BSN system can invoke the gateway's "User Registration API" to register the sub-users and generate separate user transaction key pair for each sub-user. The sub-users can use their UserCode to connect to the DApp to execute transactions.

Transaction process:



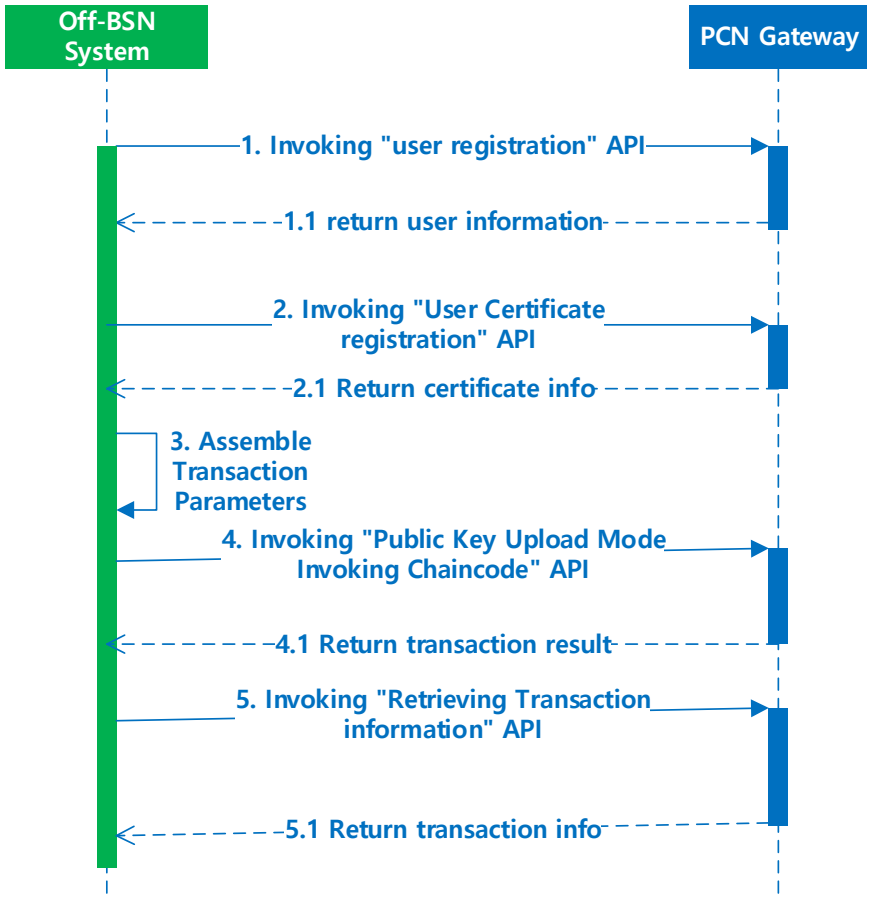
2. Public Key Upload Mode

As described in chapter 5, DApp participants require two sets of key pairs to fully access the DApp: DApp access key pair and user transaction key pair. With public-key upload mode, the key pairs are generated and stored locally by the participants. The participants only need to upload the public keys to BSN via the BSN portal or gateway APIs.

- DApp Access Key Pair: The DApp participant must generate the DApp access key pair locally according to the DApp framework algorithm after successfully joining the DApp. The participant stores the private key locally and uploads the public key to BSN via the BSN global portal. The participant's off-BSN system uses the private key to sign the transaction messages when invoking the PCN gateway. The PCN gateway will use the public key uploaded by the participant to verify the signature and validate the legality of the transaction.
- User Transaction Key Pair: This is the identity of a participant to invoke the chaincodes. Under the Key Trust Mode, the participant must generate the user transaction key pair locally and use the public key to generate the "public key registration application.", then from the participant's off-BSN system to submit the registration application to BSN by invoking the "Public Key Upload Mode user certification registration" API on the PCN gateway to receive the public key certificate. If the off-BSN system has sub-users, it should

first invoke the “User Registration” API to register the sub-users before sending their public key registration applications.

Transaction process:



5.4.3.3 Retrieving DApp information API

Invoke this interface to get certain basic DApp information; this interface can be used with Public Key Upload Mode transactions.

1. Interface address:

   https://PCNgatewayAddress/api/app/getAppInfo

2. Call Method: POST
3. Signature Algorithm: Not Required
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Yes	
2	Body	body	Map	No	
3	Signature value	mac	String	Yes	
Header					
1	User unique ID	userCode	String	Yes	
2	DApp unique ID	appCode	String	Yes	
Body					

Example:					
{"header":{"userCode":"USER00012020041519580108712 92","appCode":"app000120200416102015291845 1","tId":""},"mac": "","body":{}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response Message	msg	String	Y	
Body					
1	DApp name	appName	String	Y	
2	DApp type	appType	String	Y	
3	DApp encryption key type	caType	Int	Y	1: Key Trust Mode 2: Public Key Upload Mode
4	DApp algorithm Type	algorithmType	Int	Y	1: SM2 2: ECDSA (secp256r1)
5	City MSPID	mspId	String	Y	
6	DApp chain name	channelId	String	Y	Fabric corresponding channelId, fisco corresponding groupId
Example:					


```
{
  "header": {
    "code": 0,
    "msg": "Transaction Successful"
  },
  "mac":
"MEUCIQDE9zv0E/w4V/ILG6wUCFP08a7NDCAtX/IoZOcCyY4gIQIgUTYWsFTA1KE8
8gE6452jKnnVBrhznGVOV2HPMCbNh8A=",
  "body": {
    "appName": "sdktest",
    "appType": "fabric",
    "caType": 2,
    "algorithmType": 2,
    "mspId": "OrgbNodeMSP",
    "channelId": "app00012020041610201529184 51"
  }
}
```


5.4.3.4 User Registration API

Under the both Key Trust Mode and Public Key Upload Mode, when a participant joins, a Fabric DApp needs to create user transaction key certificates for the sub-users of his/her off-BSN system. The off-BSN system should invoke the User Registration API to register the sub-users on the PCN first. A sub-user's username is name@appCode in the call parameters

1. Interface address:

https://PCNGatewayAddress/api/fabric/v1/user/register

2. Call Method: POST
3. Signature Algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	signature value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	user name	name	String	Y	Length<20
2	user password	secret	String	N	For Key Trust Mode DApps, this can be blank; for public key upload mode DApp, if this is blank then return with a random password
Example:					
{"header":{"userCode":"USER000120200415195801087 1292","appCode":"app000120200416102015291 8451","tId":""},"mac":"MEUCIQDCa3T1c8Fim3LFVfgvelllC/wKWtFnyOl5FK7FXgddFwIgGHXApypixu9RpkHl13z80ZYdVeyRObX7icU3XWk2+VI=","body":{"name":"user01","secret":"123456"}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response Message	msg	String	Y	
Body					
1	user name	name	String	Y	Length<20
2	user password	secret	String	Y	For public key upload mode DApps, if the call parameter password is blank then return with a random password
Example					
{ "header": { "code": 0, "msg": "Transaction Successful" }, "mac": "MEUCIQClfufMU8kRI1gMHIGqfWOh1iv2KIhS+H0dlUUdEuUrLQIgYJz98xp5w/KdVP6bJjHhV2pZPTe9Cn4xcOrPV4E7ZsA=", "body": { "name": "user01",					

```
    "secret": "123456"
  }
}
```

5.4.3.5 Key Trust Mode invoking chaincode API

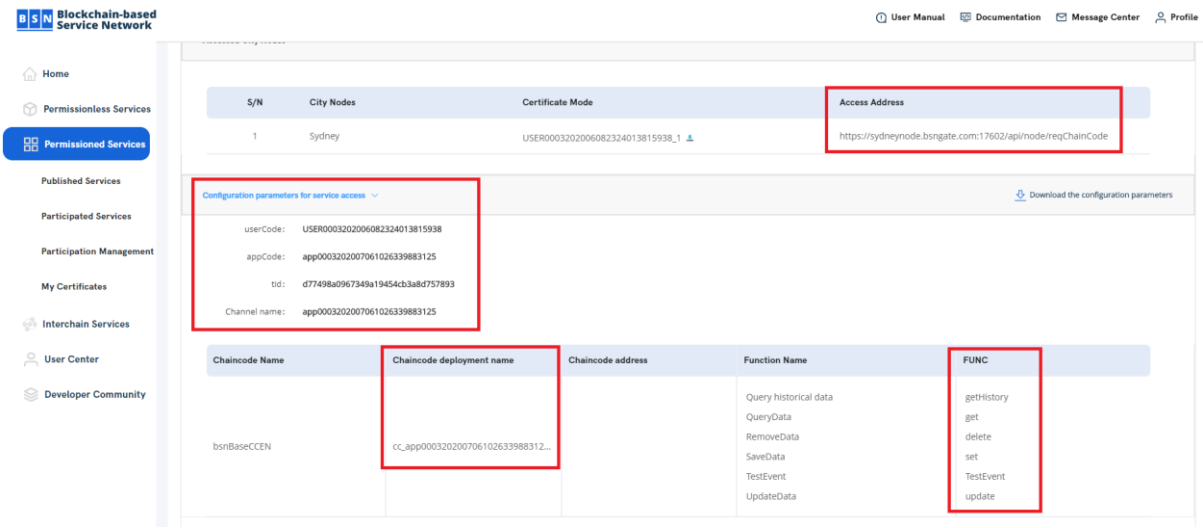
For key trust mode DApp, when the off-BSN system invokes the chaincode functions via PCN gateway, it is required to include the call parameters in the request. The gateway will return the execution result from the chaincode.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/node/reqChainCode

This interface will directly respond the result without waiting for the generation of blocks. Please use the interface "Retrieving transaction information" described in section 5.4.3.8 to check the status of a block generated based on transaction ID.

Note: After a participant has successfully joined in a DApp service, the participant can view and download the DApp's configuration parameters which are used for off-BSN systems to connect to this DApp's chaincodes, including the PCN gateway address and Dapp access keys, as shown below:



2. Call Method: POST
3. Signature Algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
3	User and DApp	tId	String	N	

	mapping ID				
Body					
1	user name	userName	String	N	
	random string	nonce	String	Y	Use 24 random byte array of the base64 encoding
1	chainCode	chainCode	String	Y	
2	function name	funcName	String	Y	
3	Call parameters	args	String[]	N	
4	Transient data	transientData	Map<string,string>	N	
Example:					
{"header":{"userCode":"USER000120200416100930940741 3","appCode":"app0001202004161017141233920","tId":""},"mac":"MEQCICJpE1jfeJKtw/ZboVuKSLy2RmmSdkhrEVPGFJhm9IaIAiA/Qqs6RNz0ndSS4/AFSwBj7vC76Py1hXnqO5zMD9pNtA==","body":{"userName":"","nonce":"lgH7Ozfv6npqg9D3pSbq9c6o+rAcpa5D","chainCode":"cc_app0001202004161017141233920_00","funcName":"set","args":["{\"baseKey\":\"test2020048\",\"baseValue\":\"this is string \"}"],"transientData":{}}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
1	block information	blockInfo	blockInfo	N	If code is not 0, then leave blank
2	chaincode response result	ccRes	ccRes	N	If code is not 0, then leave blank
blockInfo					
1	Transaction ID	txId	String	Y	
2	Block HASH	blockHash	String	N	On synchronous mode returns Block HASH
3	status value	status	Int	Y	Refer to the detailed transaction status description in 6.3.13
ccRes					
1	chaincode response status	ccCode	Int	Y	200: Successful 500: Failed
2	chaincode response result	ccData	Str	N	Actual chaincode response result
Example					
{ "header": {					

```
    "code": 0,
    "msg": "Transaction Successful"
  },
  "mac":
"MEUCIQCBtfO1AfYkoJ2hIlp8CfKK1iuhVEAYkPY8YFRAdvPJlAIgDjSqYgwlORJRyF6
KZPU/uC5Fx/DxXxu9VgKwU9+JhjU=",
  "body": {
    "blockInfo": {
      "txId": "a144149150ee615a9d11c68485600f43dc2c3eb2a98d7b36de53a6b99e03c495",
      "blockHash": "",
      "status": 0
    },
    "ccRes": {
      "ccCode": 200,
      "ccData": "SUCCESS"
    }
  }
}
```


5.4.3.6 Public Key Upload Mode user certification registration

For public-key upload mode DApp, after the participant registered its sub-users on the PCN by using the “User Registration” interface (section 5.4.3.4), use this interface to upload public key registration applications and receive the certificates (DApp access key pair certificates) for the sub-users. Invoking this interface from Key Trust Mode DApp will return an error.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/user/enroll

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	user name	name	String	Y	user name used at registration
2	user password	secret	String	Y	Password created at registration
3	Certificate Application file content	csrPem	string	Y	Use the ECDSA (secp256r1) algorithm to generate the certificate application file; the certificate CN

					is name@appCode

Example:
{"header":{"userCode":"USER000120200415195801087 1292","appCode":"app00012020041610201521 8451","tId":""},"mac":"MEQCICQaYMzs+edIQkfpt5hoaSO5dWqcrY7Q75FYwyJo/B4rAiAQ10aEpdNATsZYHVcJJ4TxVCgY8XdQBBIyTAOqUmSjkw==","body":{"name":"user01","secret":"123456","csrPem":"-----BEGIN CERTIFICATE REQUEST-----\nMIHoMIGQAgEAMC4xLDAqBgNVBAMMI3VzZXIwMUBhcHAwMDAxMjAyMDA0MTYxMDIw\nMTUyOTE4NDUxMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEnguk1xunmuU1bnKB\nam8QmeK6Geg/O6kL2D2ig85UMQTpG/sb9iYkduz8iC9SRnF9TvLiHuvJX2FGAOAQ\nK1Vz8aAAMAoGCCqGSM49BAMCA0cAMEQCIE19Iin91KlfEvfFIbxhF14enFHhtvOU\n5rK86huFiMMQAiBYXO4fJBq6eLGjaavR71O9fOvVZ5W7X+GQjIlQDuDgPQ==\n-----END CERTIFICATE REQUEST-----\n"}}

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response Message	msg	String	Y	
Body					
1	Certificate content	cert	String	Y	

Example

{
"header": {
"code": 0,
"msg": "Transaction Successful"
},
"mac":
"MEUCIQCE0gg5VHWsZluNKAV2+xOJANGnCkw6f9J4+mFT1TWz/gIgfu93jqzTzk0DU2lfMKnExcwVbgelWMLvLmwKplCXNBA=",
"body": {
"cert": "-----BEGIN CERTIFICATE-----\nMIICvTCCAmSgAwIBAgIUcqn2HmCYmq/V2yKbnxuvc49KU00wCgYIKoZIzj0EAwIw\nTjELMAkGA1UEBhMCQ04xEDAOBgNVBAgTB0JlaWppbmcxDDAKBgNVBAoTA0JTTjEP\nMA0GA1UECxMGY2xpZW50MQ4wDAYDVQQDEwVic25jYTAgFw0yMDA0MjEwNTAzMDBa\nGA8yMTAwMDMyMTExMDQwMFowbDE8MA0GA1UECxMGY2xpZW50MA8GA1UECxMIb3Jn\nYm5vZGUwDgYDVQQLEwdic25iYXNlMAoGA1UECxMDY29tMSwwKgYDVQQDDCN1c2Vy\nMDFAYXBwMDAwMTIwMjAwNDE2MTAyMDE1MjkxODQ1MTBZMBMGByqGSM49AgEGCCqG\nSM49AwEHA0IABJ4LpNcbp5rlNW5ygWpvEJniuhnoPzupC9g9ooPOVDEE6Rv7G/Ym\nJHbs/IgvUkZxfU7y4h7ryV9hRgDgECtVc/Gjgf8wgfwwDgYDVR0PAQH/BAQDAgeA\nMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFG28toKRbzJTFa6v/xlIYr6S9EvaMB8G\nA1UdIwQYMBaAFAcI4H+kIs8vn94ZYYpkrd+5ldMKMIGbBggqAwQFBgcIAQSBjnsi\nYXR0cnMiOnsiaGYuQWZmaWxpYXRpb24iOiJvcmdibm9kZS5ic25iYXNlLmNvbSIs\nImhmLkVucm9sbG1lbnRJRCI6InVzZXIwMUBhcHAwMDAxMjAyMDA0MTYxMDIwMTUy\nOTE4NDUxIiwiaGYuVHlwZSI6ImNsaWVudCIsInJvbGUiOiJjbGllbnQifX0wCgYI\nKoZIzj0EAwIDRwAwRAIgLtlTps/DOHK8S3La7bnlChB+88b1Fko9bOAL36oAFPIC\nIHQPCC30MoTHIId/X3fC5IxNukssmlMnEuDX73zRL55/\n-----END CERTIFICATE-----\n"

```
  }
}
```

5.4.3.7 Public Key Upload Mode invoking chaincode API

For Public Key Upload Mode DApp, the participant needs to assemble the transaction message locally, and invoke this interface to initiate the transaction from the off-BSN system to the DApp's chaincode.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/node/trans

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remark
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Transaction data	transData	String	Y	the transaction data should be encoded with base64

Example:

{"header":{"userCode":"USER000120200415195801087129 2","appCode":"app0001202004161020152918451","tId":""},"mac":"MEUCIQCv8EZ2OqbSbI9xGGKX06Mquh+g+NhhbUoAJBbnemXdagIgNMF7W7ecu5uej9BpVx04qwJuVijbgcp3VYIcjDK0Z38=","body":{"transData":"Cq0KCrsJCpcBCAMaCwi9gPr0BRD0o+Z2IhxhcHAwMDAxMjAyMDA0MTYxMDIwMTUyOTE4NDUxKkBjM2M2NTIzOTU4YzM4MTExOTJiOGQzNThkZDI2MTdmMWIxNGNiNjYxZGU2YjAyMmMxYTgyMjI2OWU4YThjNDhkOiYSJBIiY2NfYXBwMDAwMTIwMjAwNDE2MTAyMDE1MjkxODQ1MV8wMBKeCAqBCAoLT3JnYk5vZGVNU1AS8QctLS0tLUJFR0lOIENFUlRJRklDQVRFLS0tLS0KTUlJQ3ZUQ0NBbVNnQXdJQkFnSVVWanBGZTJFaERFaHJlOHBBVTh4bkd3dXhPbU13Q2dZSUtvWkl6ajBFQXdJdwpUakVMTUFrR0ExVUVCaE1DUTA0eEVEQU9CZ05WQkFnVEIwSmxhV3BwYm1jeEREQUtCZ05WQkFvVEEwSlRUakVQCk1BMEdBMVVFQ3hNR1kyeHBaVzUwTVE0d0RBWURWUVFERXdWaWMyNWpZVEFnRncweU1EQTBNVGt3TkRNek1EQmEKR0E4eU1UQXdNRE15TVRFeE1UUXdNRm93YkRFOE1BMEdBMVVFQ3hNR1kyeHBaVzUwTUE4R0ExVUVDeE1JYjNKbgpZbTV2WkdVd0RnWURWUVFMRXdkaWMyNWlZWE5sTUFvR0ExVUVDeE1EWTI5dE1Td3dLZ1lEVlFRRERDTjBaWE4wCk1ESkFZWEJ3TURBd01USXdNakF3TkRBd01USXdNakF3TkRBd01USXdNakF3TkR

taG1Ma1Z1Y205c2JHMWxiblJKUkNJNkluUmxjM1F3TWtCaGNIQXdNREF4TWpBe
U1EQTBNVFl4TURJd01UVXkKT1RFNE5EVXhJaXdpYUdZdVZIbHdaU0k2SW1Oc
2FXVnVkQ0lzSW5KdmJHVWlPaUpqYkdsbGJuUWlmWDB3Q2dZSQpLb1pJemowR
UF3SURSd0F3UkFJZ1ZZNi9jZ1NDTmpENkxwTXVaZEQzVWYvWko5c3FSUVVT
R3hSQU9SeGZONThDCklFN0JHTDljOHRCcHJiVmpYTldtQmpObWhqeUE3N0l3S
W8rbUg1ZXp4R1B1Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0KEhiQKmgB1Ibwb
gLAyoHXUNnjZSGOqBDheQMSbQprCmkIARIkEiJjY19hcHAwMDAxMjAyMDA0
MTYxMDIwMTUyOTE4NDUxXzAwGj8KA3NldAo4eyJiYXNlS2V5IjoidGVzdDIw
MjAwNDA0IiwiYmFzZVZhbHVlIjoidGhpcyBpcyBzdHJpbmcgIn0SRjBEAiB+mOUK
Y7fRjcZ1/qc96YP9GGod3UK56jJaWaE4o3J90QIgeirrjyzL6zQLN89tv3jDpI7vxKChk
GM9u8IEFiFEGYo="}}

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	If code=0, can be null
Body					
1	block information	blockInfo	blockInfo	N	If code is not 0, then leave blank
2	chaincode response result	ccRes	ccRes	N	If code is not 0, then leave blank
blockInfo					
1	Transaction Id	txId	String	Y	
2	Block HASH	blockHash	String	N	On synchronous mode, returns Block HASH
3	status value	status	Int	Y	refer to detailed transaction status description in 6.3.13
ccRes					
1	chaincode response status	ccCode	Int	Y	200: successful 500: failed
2	chaincode response result	ccData	Str	N	actual chaincode response result

Example

{
"header": {
"code": 0,
"msg": "Transaction Successful"
},
"mac":
"MEQCICXNk40O+Gkqqe2XgoaxdOoIvDQe4RfLtwXkxjC7ce8TAiBLVu6PjOqWueV
B3t4h7REpNdcVf6L0qVzfdA1yovuc7g==",
"body": {
"blockInfo": {
"txId":

```
"c3c6523958c3811192b8d358dd2617f1b14cb661de6b022c1a822269e8a8c48d",
     "blockHash": "",
     "status": 0
    },
    "ccRes": {
     "ccCode": 200,
     "ccData": "SUCCESS"
    }
  }
}
```


5.4.3.8 Retrieving transaction information API

The off-BSN system can use this interface to get the transaction information based on transaction ID.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/node/getTransaction

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	transactionId	txId	String	Y	
Example:					
{"header":{"userCode":"USER000120200415195801087 1292","appCode":"app000120200416102015291 8451","tId":""},"mac":"MEUCIQDIbcNl+C1iBbXWGW3qjhf80IRgCgvJuyxx0WXU2vn2TAIgZgA020L2aXBtrdLsYEkYPyiOJ9+AFrXOEwfuzy8B4bE=","body":{"txId":"c3c6523958c3811192b8d358dd2617f1b14cb661de6b022c1a822269e8a8c48d"}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response message	msg	String	N	if code=0 then can be null

Body					
1	Block Hash	blockHash	String	Y	
2	Block Number	blockNumber	Long	Y	
3	Transaction status	status	Int	Y	refer to detailed transaction status description in 6.3.13
4	on-chain user name	createName	String	Y	
5	Timestamp Second	timeSpanSec	Int64	Y	“second” in the timestamp
6	Timestamp Nanosecond	timeSpanNsec	Int64	Y	“nanosecond” in the timestamp

Example


```
{
  "header": {
    "code": 0,
    "msg": "Transaction Successful"
  },
  "mac":
"MEUCIQDUFw5pa4QJcEiQjYeLTl2L94HbsZbz7DArF+djgzWoTQIgU8u+dG6CcHw
BZjuf9PvhYdEFAa/ujwo8UAPbAmKxRq0=",
  "body": {
    "blockHash":
"ab9366cf63881228863c884527fceefabc9ad2e375aa0bcbf71f17f75c7d3ff5",
    "blockNumber": 7,
    "status": 0,
    "createName": "test02@app000120200416102015291845 1",
    "timeSpanSec": 1587445821,
    "timeSpanNsec": 249139700
  }
}
```


5.4.3.9 Retrieving block information API

After the data is stored on-chain, the off-BSN system can use this interface on the PCN gateway to retrieve the block information of the current transaction (body.blockInfo), the status (body.blockInfo.status), and transaction ID (body.blockInfo.txId). If the status value is 0, it signifies that the transaction has been successful and a block has been created. The block information can be queried according to the transaction ID.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/node/getBlockInfo

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	

3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Block number	blockNumber	Int64	N	Can't be null at the same time
2	Block HASH	blockHash	String	N	Can't be null at the same time
3	Transaction Id	txId	String	N	Can't be null at the same time
Example:					
{"header": {"userCode": "USER000120200415195801087 1292","appCode": "app000120200416102015291 8451","tId": ""},"mac": "MEUCIQCrGthrAvNalUsWEdnDxZkNXF4nCpXOxIFQdp1YYhGvugIgKvYql9Ex6RCcOhqt6coufNPH/QhtKYNeThWJ2rEL+4g=","body": {"blockNumber": 6,"blockHash": "","txId": ""}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	header	header	Map	Y	
2	body	body	Map	Y	
3	signature value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
1	Block Hash	blockHash	String	Y	
2	Block Number	blockNumber	Long	Y	
3	Previous Block Hash	preBlockHash	String	Y	
4	Block Size	blockSize	Long	Y	byte
5	The number of transactions on current block	blockTxCount	Int	Y	
6	Transaction detail	transactions	[]Transaction Data	Y	Transaction Detail
TransactionData					
1	Transaction Id	txId	String		
2	Transaction Status	Status	Int		refer to detailed transaction status description in 6.3.13
3	Transaction Provider	createName	String		
4	Transaction timestamp second	timeSpanSec	Int64		
5	Transaction	timeSpanNsec	Int64		

	timestamp nonasecond				

Example


```
{
  "header": {
    "code": 0,
    "msg": "Transaction Successful"
  },
  "mac":
"MEUCIQC8nfFnHw4sEYJmaSTT1xepxMGgomxyJtt0ysyGgPB0AwIgfuuiegdGEbBi/2wmF
Cco39wmik3isLWtvnN9ZmJDTdk=",
  "body": {
    "blockHash": "fc83c306677925efee540b4d7b7ca73e06f144cae34c706f1101d6b395ada2da",
    "blockNumber": 6,
    "preBlockHash":
"93c86551d812229274e144093cd4bd17dacb35bc6a01779930e11f43f886bf34",
    "blockSize": 7020,
    "blockTxCount": 1,
    "transactions": [
      {
        "txId": "a8639f3a796267e048d475b00fe7646a4524f1c20d71880e19708821177b7bdb",
        "status": 0,
        "createName": "test02@app000120200416102015291845 1",
        "timeSpanSec": 1587271285,
        "timeSpanNsec": 26436800
      }
    ]
  }
}
```


5.4.3.10 Retrieving the newest ledger information API

Use this interface to retrieve the newest ledger information, including block HASH, previous block HASH, and the height of the current block, etc.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/node/getLedgerInfo

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Example:					
{"header":{"userCode":"USER000120200415195801087 1292","appCode":"app00012020 04161020152918451","tId":""},"mac":"MEQCID7Z3J2PiRDOx7JasRamBZRTAHXj1X					

AG1K/DUkzJEwuiAiBIY5p3H2kArE7OuYLOgEqMHl15Xgj5Voi5zVPGhyU/+w==","body":{}}

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
1	Block Hash	blockHash	String	Y	
2	Block Height	height	Long	Y	
3	Previous Block Hash	preBlockHash	String	Y	

Example


```
{
  "header": {
    "code": 0,
    "msg": "Transaction Successful"
  },
  "mac":
"MEUCIQC4PhYTBNyt1rSeBeZTdOly42CxILVgK1b/RlieA33G1gIgeodoEa5Ou0X4uWc/VGp0n6NKByhXlBbo22FME4xQ8aw=",
  "body": {
    "blockHash":
"ab9366cf63881228863c884527fceefabc9ad2e375aa0bcbf71f17f75c7d3ff5",
    "height": 8,
    "preBlockHash":
"fc83c306677925efee540b4d7b7ca73e06f144cae34c706f1101d6b395ada2da"
  }
}
```


5.4.3.11 Registering chaincode event API

Chaincode event in a DApp can trigger the off-BSN system to process further transactions. This interface is used to register the chaincode event to be monitored.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/chainCode/event/register

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	

2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	ChainCode	chainCode	String	Y	
2	Chaincode event key	eventKey	String	Y	
3	Chaincode event notification URL	notifyUrl	String	Y	URL to receive the monitored chaincode event
4	Attached additional parameters	attachArgs	String	N	
Example:					
{"header":{"appCode":"CL20191107112252","userCode":"lessing"},"body":{"attachArgs":"name=TOM&age=20","chainCode":"cc_bsn_test_00","eventKey":"test01","notifyUrl":"http://192.168.6.128:8080/api/event/notifyUrl"},"mac":"MEUCIQCjzPr4KZVild2Vm5YgcunOXTh9mQK2QfWcRnYCk+jOzgIgDW6oHca7/249M43p2ElwiMNbuejdwAnyW5OwiMqiWCQ="}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response Message	msg	String	Y	
Body					
1	Event ID	eventId	String	Y	
Example					
{ "header": { "code": 0, "msg": "Event Registration Successful" }, "body": { "eventId": "bd3391deedbe44a7ad5b7f80ce59abfa" }, "mac": "MEQCIENLpj2R9mRL100vcMXs0X5rwfSjB/U7kMg+76GjEPNJAiBlUo/Eyj49uXTPrzRW0m4rJ0NQIkZnDMPbyalxojXwrA=="}					

5.4.3.12 Registering block event API

Block event in a DApp can trigger the off-BSN system to process further transactions. This interface is used to register the block event to be monitored.

6. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/chainCode/event/blockRegister

7. Call method: POST
8. Signature algorithm: required and refer to Section 5.4.3.1
9. Call parameters

No.	Field name	Field	Type	Required	Remarks

1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Chaincode event notification URL	notifyUrl	String	Y	URL to receive the monitored block event
2	Attached additional parameters	attachArgs	String	N	
Example:					
{"header":{"userCode":"USER000120200710164124351616 3","appCode":"app00120210119141123842626 6","tId":""},"mac":"MEUCIQClsjKy/ee1qaYrItzCO1bMfjs0g0kPu8+YOCjbk3rPRAIgSfeyYvfeoh8QciZPG4fZQepaiyh7PmmWjYzFSqylT/c=","body":{"chainCode":"","eventKey":"","notifyUrl":"http://192.168.6.78:58011/v1/fabric/test","attachArgs":"a=1"}}					

10. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response message	msg	String	Y	
Body					
1	Event ID	eventId	String	Y	
Example					


```
{
  "header": {
    "code": 0,
    "msg": "success"
  },
  "mac":
"MEUCIQC6PKsSqfkQGLrqi2vMpZzBP5beLhyP+fXVr8S5aqhaagIgaEtAnsuiubibYoYZzQ/8aGYErzm5rtU8Oj952OuHgCo=",
  "body": {
    "eventId": "002f0e1f0b0f4331ab541461547a38d6"
  }
}
```

5.4.3.13 Chaincode and block event query API

Use this API to query the list of monitored chaincode and block events that have been registered.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/chainCode/event/query

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Example:					
{"header":{"appCode":"CL20191107112252","userCode":"lessing"},"body":{},"mac":"MEQCIAnJxvuKVe0u/bG0VYCjM3g3ctxTYIWkejYp462okNlcAiBcOTGvAkF7xErL2w1PiwgfFjIu3Sszgyfzym/pEwRGxA=="}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	[]body	Y	Event List
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: Query successful -1: Query failed
2	Response Message	msg	String	Y	
body					
1	Event ID	eventId	String	Y	
2	Chaincode Event key	eventKey	String	N	Null if it's a block event
3	Chaincode Event Notification URL	notifyUrl	String	Y	
4	Attached additional parameters	attachArgs	String	N	
5	Creation Time	createTime	String	Y	
6	PCN ID	orgCode	String	Y	
7	user unique ID	userCode	String	Y	
8	DApp unique code	appCode	String	Y	
9	Chaincode ID	chainCode	String	N	Null if it's a block event
10	Event type	eventType	String	N	Returns "block" if it's a block

					event; Null if it's chaincode event

Example

{ "header": { "code": 0, "msg": "Query Event Successful" }, "body": [{ "eventKey": "test001", "notifyUrl": "http://192.168.6.128:8080/api/event/notifyUrl", "attachArgs": "a=123\u0026b=456", "eventId": "945ee631d26140118963ad3104c81713", "createTime": "2019-11-18 14:22:59", "orgCode": "ORG1571365934172", "userCode": "lessing", "appCode": "CL20191107112252", "chainCode": "cc_bsn_test_00" }, { "eventKey": "test002", "notifyUrl": "http://192.168.6.128:8080/api/event/notifyUrl", "attachArgs": "hahahhahhahahahah", "eventId": "346617a493d84c6d8512b8dddad87811", "createTime": "2019-11-18 14:29:28", "orgCode": "ORG1571365934172", "userCode": "lessing", "appCode": "CL20191107112252", "chainCode": "cc_bsn_test_00" }, { "eventKey": "test01", "notifyUrl": "http://192.168.6.128:8080/api/event/notifyUrl", "attachArgs": "name=Zhangsan\u0026age=20", "eventId": "bd3391deedbe44a7ad5b7f80ce59abfa", "createTime": "2019-11-19 10:52:15", "orgCode": "ORG1571365934172", "userCode": "lessing", "appCode": "CL20191107112252", "chainCode": "cc_bsn_test_00" }], "mac": "MEQCIEYXFMa8dfBrjy/s9H5JAoFIrjROJBiw+7/daELUbF5eAiA7a6HvqqbOpv6vlkunHGxCB1o5DoeuJFD0FM6kLoU34Q=="}

5.4.3.14 Remove chaincode and block event API

This interface is used to remove a chaincode event's registration from the event list.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/chainCode/event/remove

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.3.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Event ID	eventId	String	Y	
Example:					
{"header":{"appCode":"CL20191107112252","userCode":"lessing"},"body":{"eventId":"bd3391deedbe44a7ad5b7f80ce59abfa"},"mac":"MEQCIE3/CLG5LxZZN7En7LZvzthajwxHzpvDduXSsw4Tb1JFAiAXGJ4WVtyCKbtCasQGofCkge8NOgZDNPgJIdTCtCi2SQ==="}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: remove successful -1: remove failed
2	Response Message	msg	String	Y	
Example					
{ "header": { "code": 0, "msg": "Remove Event Successful" }, "body": null, "mac": "MEUCIQCaTFLliY7pPjkwcmSsLXOth7k9bQj9Sblq+1nMVjkFAAIgUsizFO+f1+dxU3/hPxjf/+na4qG6aQFftJIWGtMhlVI="}					

5.4.3.15 Chaincode and block event notification message API

This interface is implemented on the off-BSN system side. When the PCN gateway receives the notification of a triggered event, it uses this interface to notify the off-BSN system about the execution result.

After receiving the notification successfully, the off-BSN system returns a string containing “success”, otherwise, the gateway will send the notification again at 3, 12, 27, and 48 seconds respectively, for a total of five times.

1. Call method: POST
2. Signature algorithm: required and refer to Section 5.4.3.1
3. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
1	Chaincode ID	chainCode	String	N	Null when the block event notification
2	PCN ID	orgCode	String	Y	
3	Registered Event key	eventKey	String	N	
4	Registered Event ID	eventId	String	Y	
5	Registered Event parameters	attachArgs	String	N	Additional parameters entered during registration
6	Monitored event key	eventName	String	N	The event name in the chaincode, null when the block event notification

7	Current Chaincode transaction Id	txId	String	N	Null when the block event notification
8	Monitored event value	payload	String	N	
9	Current Block Height	blockNumber	Long	Y	
10	Response random string	nonceStr	String	Y	Off-BSN system uses this value to judge if the notification is already received. This string remains the same at the repeated notifications.
11	Previous hash	previousHash	String	N	Null when chaincode event notification

Example:

Chaincode event nofitication

{"header":{"userCode":"lessing","appCode":"CL20191107112252"},"body":{"chainCode":"cc_bsn_test_00","orgCode":"ORG1571365934172","eventKey":"test:\\S{32}","eventId":"2964a0f60b3e460f834618b3664af2da","attachArgs":"abc=123211","eventName":"test:12345678123456781234567812345678","txId":"32fc105681820fa556b8a460efc1e43a47daa864b959ea1753abb4640f2dce49","payload":"","blockNumber":74,"nonceStr":"522c8061b5e84837bad72ca08c6a353f"},"mac":"MEQCIDU4tROyjLtvD1b8TTbWWAlCPuUbmdPAEUXwRRgVn7kIAiA58je5u/7xDuRPcgeUWL3nB9mouUGQ6dGKJMmD7Jm08g=="}

Block event notification

{"header":{"userCode":"USER000120200710164124351616 3","appCode":"app00012021011914112384262 66"},"body":{"orgCode":"ORG20200411141716923 60","eventId":"8746bb9a1e854c9f8b3710f5a63f7c59","attachArgs":"a=1","previousHash":"022281f6089e3684501251775166b6b0afd18a176ec98a835cb5d09aff0d4950","blockNumber":12,"nonceStr":"79a7baa26c854caeb2e2e7abc0b7f07e"},"mac":"MEUCIQDiZrwf8fKG/3fuaVrsfTN3BKmLx+qnnEuuSaHfvIBbMQIgS+1qHKXeVR24WXwOGu3Nze/tLLziQ0LkjXaueYu0ctM="}

5.4.3.16 Transaction status description

Under both Key Trust Mode and Public Key Upload Mode, the description of the returned transaction status when the off-BSN system invokes the DApp chaincodes via PCN gateway APIs are shown as follows:

No.	Status Code	Remarks
	0	Successful
	-1	Block creation time out
	1	Submitted data empty
	2	Unusual response

	3	Error in the submitted information
	4	Error in the creator's signature
	5	Invalid "endorser" transaction
	6	Invalid transaction settings
	7	Unsupported transaction response
	8	Error in the transaction ID
	9	Duplicate transaction ID
	10	Failed endorsement
	13	Unknown transaction type
	14	Cannot locate target chaincode
	17	Expired chaincode
	18	Conflict in chaincode version
	254	Invalid transaction
	255	Invalid transaction for other reasons

5.4.4 PCN gateway FISCO API

A PCN gateway is deployed on each public city node (PCN) to receive off-BSN system requests signed and verified by DApp access keys, then used to route the requests to the corresponding FISCO BCOS-based DApp smart contracts. Invoking the PCN gateway is realized by sending HTTP requests to each PCN gateway service. The gateway is responsible for verifying user and application identities, and then uses these identities and smart contract functions to process smart contract parameters then sends the smart contract transaction results back to the off-BSN systems.

5.4.4.1 DApp Access Signature Algorithm

Whenever an off-BSN system sends requests to the PCN gateway, the HTTP request message should be signed with the DApp participant's DApp access private key. When the PCN gateway receives the message with the digital signature, it will verify the authentication and message integrity with the corresponding hosted or uploaded DApp access public key. The gateway will only process the request message further after the verification is passed.

1. Assemble signature string

Convert the request parameters into a joined string according to the order of the parameter table, of which, the call parameter prioritises joining UserCode and AppCode of the Header and the response parameter prioritises joining code and msg. Then join the parameters in the Body according to the order of the parameter tables in the definition of APIs.

2. Different type conversion formats

Type	Rule	Example	Result
String	No conversion	abc	abc
Int/int64/long	Decimal conversion	-12	-12
Float	Decimal conversion; see notes for values after decimal point	1.23	1.23
Bool	Convert to "true" or "false"	true	true
Array	Join according to parameter sequence and type	{"abc","xyz"}	abcxyz
Map[key]value	Join key and value according to parameter sequence	{"a":1,"b":2}	a1b2
Object	Convert the attributes in the	{"name":"abc","sec	abc123456

	object one by one according to the document in the above-described format	ret”:”123456”}	

3. Signature rules

1. FISCO BCOS framework DApp using ECDSA (secp256k1) secret key algorithm

- Getting the Hash value: The converted string to be signed is required to be computed with SHA256 algorithm with UTF-8 encoding.
- Sign the Hash value: The hash value and private key should be encrypted with ECDSA (secp256k1) algorithm. In the processing of some programming languages (C#, Java), if signed with SHA256WithECDSA, which includes hash value computation, therefore, the first step is not necessary.
- Encoding the signature result to Base64.

2. FISCO BCOS framework DApp using SM secret key algorithm

- Getting the Hash value: The converted string to be signed is required to be computed with SM3 algorithm with UTF-8 encoding.
- Sign the Hash value: The hash value and private key should be encrypted with SM2 algorithm.
- Encoding the signature result to Base64.

4. Example

Parameters:

{"header":{"userCode":"user01","appCode":"app01"},"mac":"","body":{“userId”:”abc”,”list”:[“abc”,”xyz”]}}

Result: user01app01abcabcxyz

5.4.4.2 Key and Certificate Modes

1. Key Trust Mode

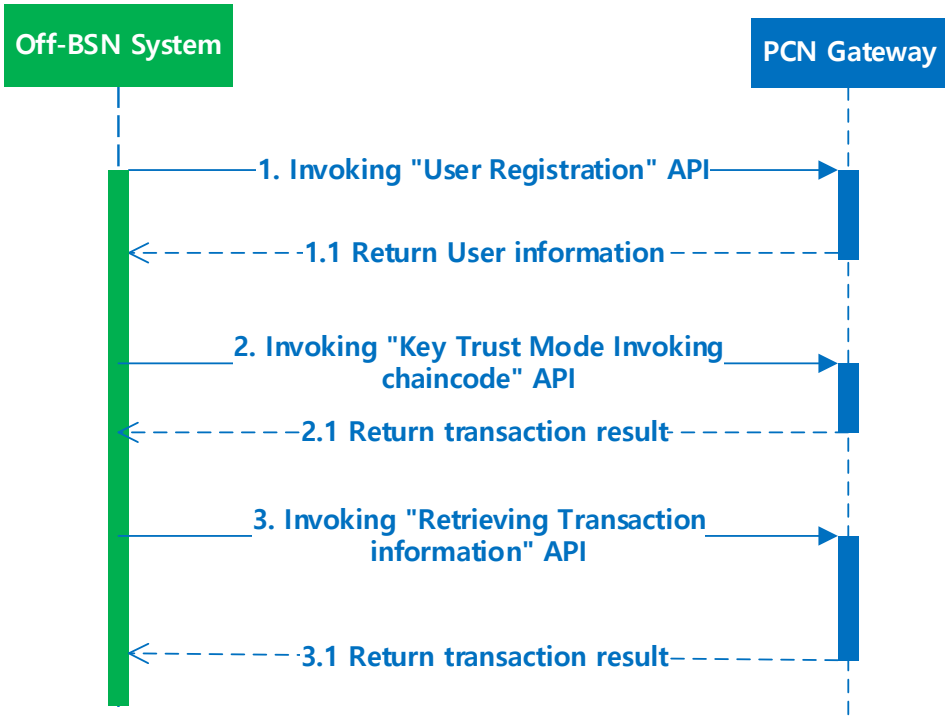
As described in the chapter 5, DApp participants require two sets of key pairs to access the DApp: DApp access key pair and user transaction key pair. Under the key trust mode, the pairs are generated and hosted by BSN. The participants only need to download the private key (DApp access key) from the BSN portal.

DApp Access Key Pair: After the participant has successfully joined the DApp, BSN will generate one key pair (private and public keys) that corresponds to the DApp’s framework algorithms under the Key Trust Mode. The participant can download the private key from “My Certificates” section of the BSN global portal and use it to sign the request message sent to the PCN gateway. The gateway will use the hosted public key from the generated key pair to validate the signature.

User Transaction Key Pair: This is the identity of a participant to invoke the chaincodes. Under the Key Trust Mode, after successfully joining the DApp, a participant’s user transaction key pair will be created automatically by BSN by default. The participant’s off-BSN system can use the participant’s UserCode to invoke the certificate generated by the key pair. If the participant’s off-BSN system has multiple sub-users, the off-BSN system can invoke the

gateway's "User Registration API" to register the sub-users and generate a separated user transaction key pair for each sub-user. The sub-users can use their own UserCode to connect to the DApp to execute smart contract transactions.

Transaction process:

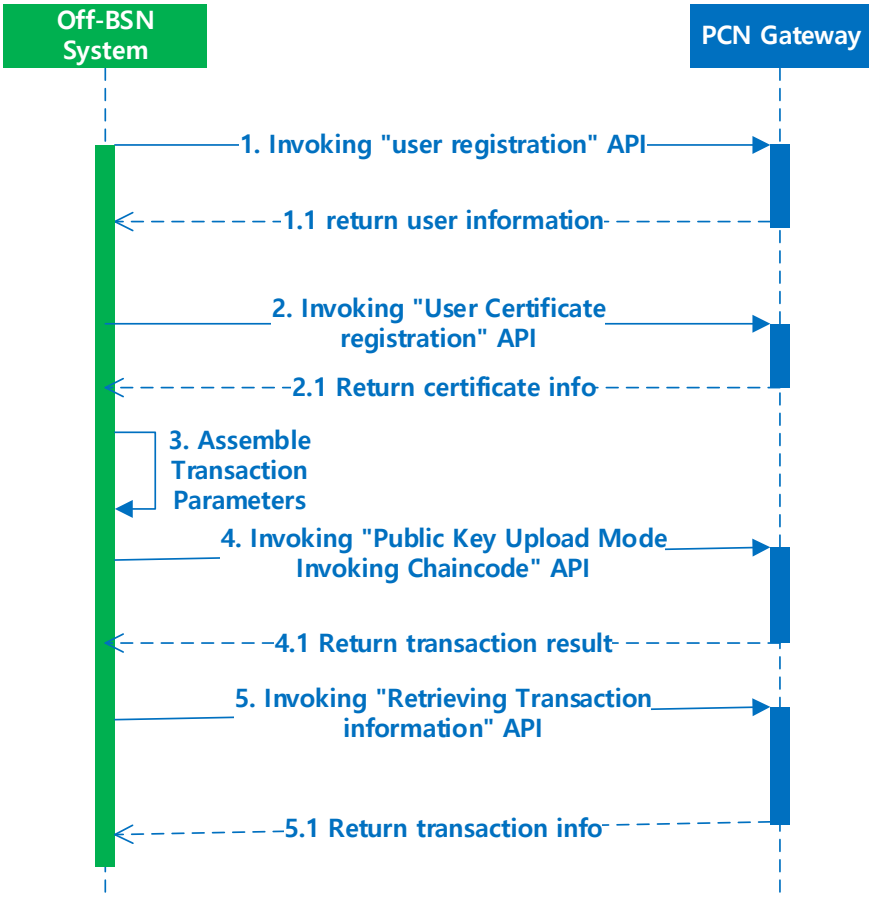


2. Public Key Upload Mode

As described in chapter 5, DApp participants require two sets of key pairs to fully access the DApp: DApp access key pair and user transaction key pair. With public-key upload mode, the key pairs are generated and stored locally by the participants. The participants only need to upload the public keys to BSN via the BSN portal or gateway APIs.

- DApp Access Key Pair: The DApp participant must generate the DApp access key pair locally according to the DApp framework algorithm after successfully joining the DApp. The participant stores the private key locally and uploads the public key to BSN via the BSN global portal. The participant's off-BSN system uses the private key to sign the transaction messages when invoking the PCN gateway. The PCN gateway will use the public key uploaded by the participant to verify the signature and validate the legality of the transaction.
- User Transaction Key Pair: This is the identity of a participant to invoke the chaincodes. Under the Key Trust Mode, the participant must generate the user transaction key pair locally and use the public key to generate the "public key registration application", then from the participant's off-BSN system to submit the registration application to BSN by invoking the "Public Key Upload Mode user certification registration" API on the PCN gateway to receive the public key certificate. If the off-BSN system has sub-users, it should first invoke the "User Registration" API to register the sub-users before sending their public key registration applications.

Transaction process:



5.4.4.3 Retrieving DApp information API

Invoke this interface to get certain basic DApp information; this interface can be used with Public Key Upload Mode transactions.

1. Interface address:

   https://PCNgatewayAddress/api/app/getAppInfo

2. Call Method: POST
3. Signature Algorithm: Not Required
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Yes	
2	Body	body	Map	No	
3	Signature value	mac	String	Yes	
Header					
1	User unique ID	userCode	String	Yes	
2	DApp unique ID	appCode	String	Yes	
Body					
Example:					
{"header":{"userCode":"USER000120200415195801087 1292","appCode":"app000120200416102015291 8451","tId":""},"mac": "","body":{}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response Message	msg	String	Y	
Body					
1	DApp name	appName	String	Y	
2	DApp type	appType	String	Y	
3	DApp encryption key type	caType	Int	Y	1: Key Trust Mode 2: Public Key Upload Mode
4	DApp algorithm Type	algorithmType	Int	Y	1: SM2 2: ECDSA(secp256r1)
5	City MSPID	mspId	String	Y	
6	DApp chain name	channelId	String	Y	Fabric corresponding channelId, fisco corresponding groupId
Example:					


```
{
  "header": {
    "code": 0,
    "msg": "Transaction Successful"
  },
  "mac":
"MEUCIQDE9zv0E/w4V/ILG6wUCFP08a7NDCAtX/IoZOcCyY4gIQIgUTYWsFTA1
KE88gE6452jKnnVBrhznGVOV2HPMCbNh8A=",
  "body": {
    "appName": "sdktest",
    "appType": "fabric",
    "caType": 2,
    "algorithmType": 2,
    "mspId": "OrgbNodeMSP",
    "channelId": "app000120200416102015291845 1"
  }
}
```


5.4.4.4 User Registration API

After a participant has successfully joined in a FISCO BCOS (FISCO) DApp, his/her off-BSN system can invoke this interface to generate the user account and user address to execute smart contract transactions.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/user/register

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1

4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
1	user name	userId	String	Y	Registered user name
Example:					
{ "header":{"appCode":"CL188103887322019090211431 4","userCode":"newuser"}, "body": { “userId”:”abc” }, "mac":"MEQCIBRhaM2szckWl9N9qcqnaYXOXGQw7SfII9DlRvxcI3YVAiBt4XeNs+EUjhBNSr3IjLRPZucsuGHxfjt9RiaNIQS8cA==" }					
signature value:					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
body					
1	User information	data	[]string	N	If code is not 0, then leave blank
data					
1	User ID	userId	String	Y	
2	User Address	userAddress	String	Y	
Example					
{ "header": { "code": 0, "msg": "Transaction Successful" }, "mac": "MEQCIEI5VKMyJUXls2Hf8TLoPXjZLT4/L2wyXoddgTnZdqRsAiBxEBMeCOZ8M97OCRUAMZNMcL974vhzjOS/tk8/wbgbsA==", "body": { "userId": "100003", "userAddress": "0x14647a48303b5e1c77934583883ebc327ba3b297" }					

```
}
```

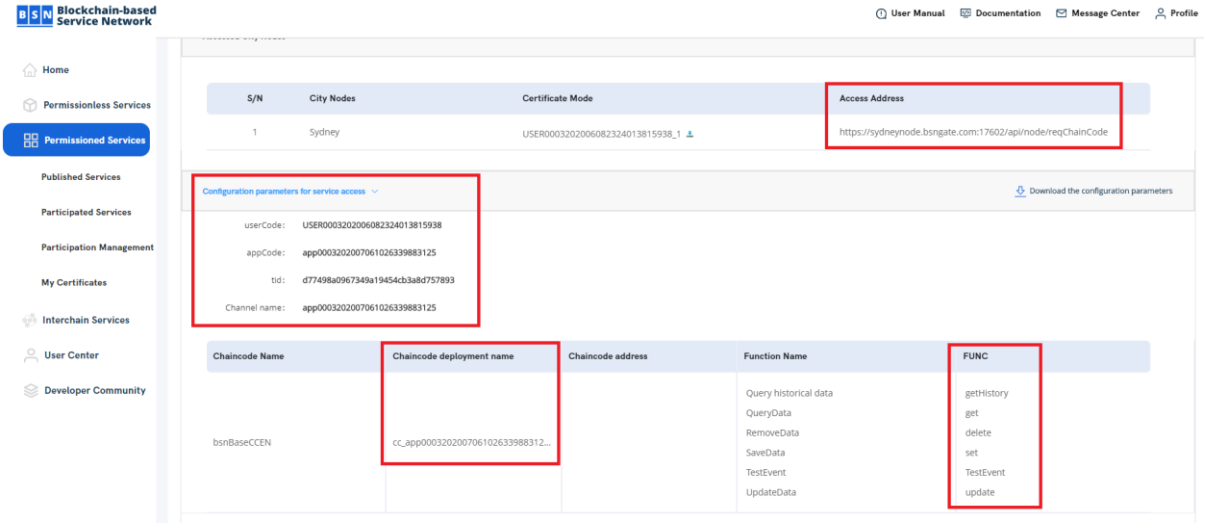
5.4.4.5 Key Trust Mode Invoking Smart Contract API

For the key trust mode FISCO DApps, when the off-BSN system invokes the smart contract functions via PCN gateway, it is required to include the call parameters in the request. The gateway will return the transaction result from the smart contract.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/reqChainCode

Note: After a participant has successfully joined in a FISCO DApp service, the participant can view and download the DApp's configuration parameters which are used for off-BSN systems to connect to this DApp's smart contracts, including the PCN gateway address and Dapp access keys, as shown below:



2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	User ID	userId	String	Y	Registered user ID via 7.3.1 API
2	Smart Contract Name	contractName	String	Y	
3	Function Name	funcName	String	Y	
4	Function Parameters	funcParam	string	N	convert array type to json string format

Example:
{"header":{"appCode":"cl000620200318192657367 7572","userCode":"USER00062020 03181951281835816"},"body":{"contractName":"HelloWorld","userId":"100003","func Name":"set","funcParam":[\"abc\"]},"mac":"MEUCIQDTFe2Gerdf7YJrG1a1Yt99M0Z Q3T1lGpsXdNmFV7WuTgIgSkZ19abUhAJbMrJMBoD8N7f26xhpQRuR4vNAfY7EE bs="}

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
1	Invoke Type	constant	Bool	N	
2	Query information	queryInfo	String	N	If Constant is true, this field has value.
3	Transaction hash	txId	string	N	If Constant is false, this field has value and is valid.
4	Block HASH	blockHash	String	N	If Constant is false, this field has value and is valid.
5	Block Number	blockNumber	Int	N	If Constant is false, this field has value and is valid.
6	Gas Used	gasUsed	Int	N	If Constant is false, this field has value and is valid.
7	Transaction Status	status	String	N	If Constant is false, this field has value and is valid. 0x0 means transaction successful, status value refer to transaction receipt status in 7.3.9
8	From account	from	String	N	If Constant is false, this field has value and is valid.
9	To account	To	String	N	If Constant is false, this field has value and is valid.
10	Input	Input	String	N	If Constant is false, this field has value and is valid.
11	Ouput	output	String	N	If Constant is false, this field has value and is valid.

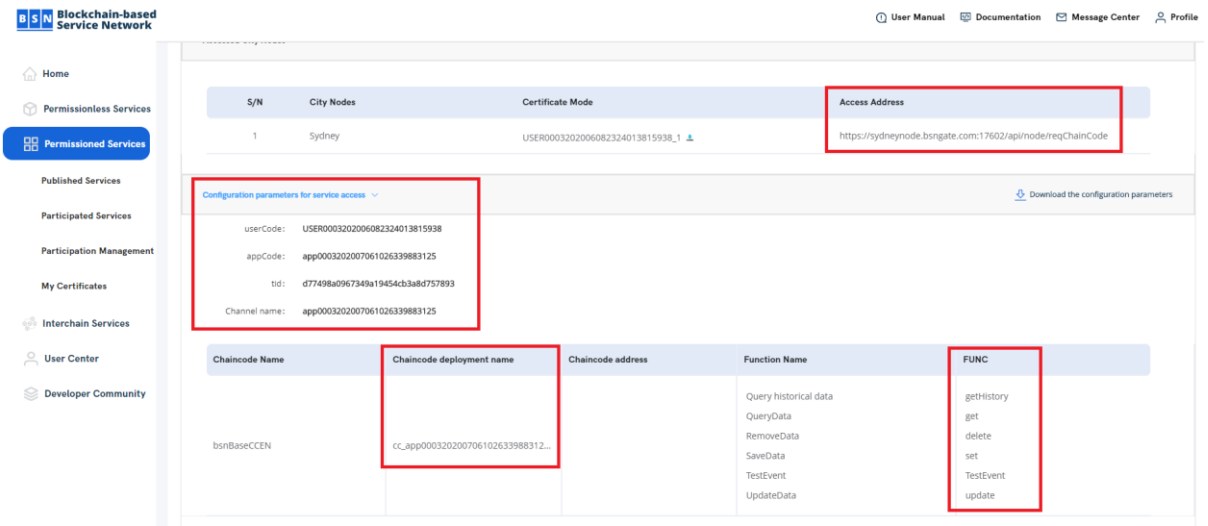
Example

5.4.4.6 Public Key Upload Mode Invoking Smart Contract API

When the off-BSN system invokes the node gateway, it should follow the API descriptions to add the corresponding parameters. After invoking the node gateway, the node gateway returns the execution result of the smart contract. In the transaction of Public Key Upload mode, the private key of the transaction on the chain is generated and saved by the user. Then the client performs the assembly and signature of the data locally. The signed data is uploaded to the node gateway, which forwards the data to the corresponding blockchain node to initiate the transaction request. Data assembly in this pattern requires information such as the contract ABI, which is compiled when developing the contract, and the contract address, which is available on the application details page. In the SDK of the gateway, the assembly method of the data on the link has been implemented, which can be directly called.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/trans



Note: After a participant has successfully joined in a FISCO DApp service, the participant can view and download the DApp's configuration parameters which are used for off-BSN systems to connect to this DApp's smart contracts, including the PCN gateway address and Dapp access keys, as shown below:

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Smart Contract	contractName	String	Y	

	Name				
2	Transaction Data	transData	String	Y	
3	Contract address	contractAddress	String	N	
4	Contract ABI	contractAbi	String	N	
Example:					
{"header":{"userCode":"USER000120200604232157969244 0","appCode":"app0001202006042323057101002","tId":""},"mac":"MEUCIQCrjIeRVSt1uwFFGkr37bVM8pF0JgAWb40mKEBc5HbpjgIgEzXRIgG+Q7obwuD2MY4EHo9sIsI1W71M+aQKOfAN3wU=","body":{"contractName":"BsnBaseContractk1","transData":"0xf9016fa008d8ebcb4b1f8205fd7883aa3ce9b9c844424070e55a3af6a5da5d7ee97d287385051f4d5c0083419ce07794866aefc204b8f8fdc3e45b908fd43d76667d7f7680b8e4ebf3b24f0000000000

					and is valid. 0x0 means transaction successful, status value refers to transaction receipt status in 7.3.9
8	From account	from	String	N	If Constant is false, this field has value and is valid.
9	To account	To	String	N	If Constant is false, this field has value and is valid.
10	Input	Input	String	N	If Constant is false, this field has value and is valid.
11	Output	output	String	N	If Constant is false, this field has value and is valid.
Example					

5.4.4.7 Retrieving Transaction Receipt API

After the smart contract executes one transaction, this interface can be used to retrieve the transaction receipt information according to the transaction HASH value.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/getTxReceiptByTxHash

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
2	Transaction Hash	txHash	string	Y	
Example:					
{"header":{"appCode":"cl000620200318192657367 7572","userCode":"USER0006202003181951281835816"},"body":{"txHash":"0x755f3e7833778f674e1b025f513f05722ba7248be43a3c9168b880847814021a"},"mac":"MEYCIQCe6sI9zqspsy1bS6Ka9Q8O+pE7TEDWdsWj4UBSg6FM7AIhAJrud/EoxnURQcDc47iwTdh7OdxJEJPE+raK9UaHjNaJ"}					
signature value:					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	

2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
1	Transaction Receipt Info	txId	string	N	If code is not 0, then leave blank
	Block HASH	blockHash			
	Block Number	blockNumber			
	Gas Used	gasUsed			
	From account	from			
	To account	to			
	Smart Contract Address	contractAddress			

Example


```
{
   "header": {
      "code": 0,
      "msg": "Transaction successful"
   },
   "mac":
"MEUCIQCUlhnvH9a4HN/YITf4OWgTuHmmz6qMEO89I4effHdcIwIgStdeb/dVplhn3/FoCjeSc
VRyiEUhpkbze9bVm1gaXqs=",
   "body": {
      "blockHash":
"0x199eca276b60473dd65f8b36641684456694b419d89ef41b4953a9cdac848305",
      "gasUsed": 2154887,
      "blockNumber": 1,
      "txId": "0x8ee0c68e222742b5b70878265d3fdbd3a8e0d549da42a298a4ae872ca4fbfd89",
      "contractAddress": "0x20453db36c492fa49da9fab1b80db7fa5f46b01e",
      "from": "0x08ac3132a6c7e6ca5a7fbaf0521bb8b6f370ed35",
      "to": "0x0000000000000000000000000000000000000000"
   }
}
```


5.4.4.8 Retrieving Transaction information API

After the smart contract executes one transaction, this interface can be used to retrieve the transaction detailed information according to the transaction HASH value.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/getTxinfoByTxHash

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Transaction HASH	txHash	string	Y	
Example:					
{"header":{"appCode":"cl000620200318192657367 7572","userCode":"USER0006202003181951281835816"},"body":{"txHash":"0x755f3e7833778f674e1b025f513f05722ba7248be43a3c9168b880847814021a"},"mac":"MEUCIQDDQudQBvHkI5tIpeTDGkQA+LPRMTA2k9u7hCZAYVobvQIgNseUfaVw8d/LxooPPWyQSo2O4EUt6wmEISgtnTcUO7k="}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
	Transaction HASH	txId	String		
	Block HASH	blockHash	String		
	Block Number	blockNumber	Int		
	Gas Used	gasUserd	Int		
	From account	from	String		
	To account	to	String		
		value	Int		
		input	String		
Example					


```
{
   "header": {
      "code": 0,
      "msg": "Transaction Successful"
   },
   "mac":
"MEQCIBMqntmqQqZXkBbrLhmXEcuOqTG4YWvlfGJmebzEDbzcAiAKKHut9MBShqpSAEo8ts2MEQCIBMqntmqQqZXkBbrLhmXEcuOqTG4YWvlfGJmebzEDbzcAiAKKHut9MBShqpSAEo8ts2+OBIRmEEbedjihix5FZZvrw==",
   "body": {
      "blockHash":
"0x199eca276b60473dd65f8b36641684456694b419d89ef41b4953a9cdac848305",
```

```
        "input":
"0x6080604052348015620000115760008fd5b5061100160008061010000a81548173ffffffffffffffff
ffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550600080905490610
01000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1663c92a780
160408051908101604052806006815260200017f745f626173650000000000000000000000000000
000000000000000000000000815250604051826 3ffffffff167c0100000000000000000000000000000
000000000000000000000000000028152600401620001019190620002 4a565b60206040518083038
1600087803b1580156200011c57600080fd5b505af11580156200013157 3d6000803e3d6000fd5b50
5050506040513d601f19601f820116820180604052506200015791908101906200017 4565b506200
02f4565b60006200016c8251620002a3565b905092915050565b60006020828403121562000187 57
600080fd5b60006200019784828501620001 5e565b91505092915050565b6000620001ad826200029
8565b808452620001c381602086016020860162000 2ad565b620001ce81620002e3565b6020850101
91505092915050565b6000601382527f626173655f6b65792c626173655f76616c7565000000000000
0000000000000000602083015260408201905091905056 5b6000600782527f626173655f6964000
0000000000000000000000000000000000000000000000000602083015260408201905091905056 5b6
000606082019050818103600083015262000266818462000 1a0565b9050818103",
        "gasUsed": 100000000,
        "blockNumber": 1,
        "txId": "0x8ee0c68e222742b5b70878265d3fdbd3a8e0d549da42a298a4ae872ca4fbfd89",
        "from": "0x08ac3132a6c7e6ca5a7fbaf0521bb8b6f370ed35",
        "to": "0x0000000000000000000000000000000000000000",
        "value": 0
    }
}
```


5.4.4.9 Retrieving Block Information API

Corresponding block information can be queried according to block number or the block HASH. The block number and block HASH cannot simultaneously be blank. When neither is blank, query the block number first to get the block information.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/getBlockInfo

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Block Height	blockNumber	string	N	When null, blockHash cannot be null
2	Block Hash	blockHash	String	N	When null, blockNumber

					cannot be null
Example: { "header":{"appCode":"CL188103887322019090211431 4","userCode":"newuser"}, "body": { "blockNumber":22, "blockHash":"0xf27ff42d4be65329a1e7b11365e190086d92f9836168d0379e92642786db7ade" }, "mac":"MEQCIBRhaM2szckWl9N9qcqnaYXOXGQw7SfII9DlRvxcI3YVAiBt4XeNs+EUjhBNSr3IjLRPZucsuGHxfjt9RiaNIQS8cA=="}					
signature value:					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
	Block HASH	blockHash	String	Y	
	Block Number	blockNumber	Int	Y	
	Parent Block HASH	parentBlockHash	String	Y	
	Block Size	blockSize	Int	Y	
	Block Time	blockTime	Int	Y	Timestamp in millisecond format
		author	String	Y	
	Transaction Information	transactions	[]Transaction Data	Y	
TransactionData					
	Transaction Id	txId	String	Y	
	Block HASH	blockHash	String	Y	
	Block Number	blockNumber	Int	Y	
	Gas Used	gasUsed	Int	Y	
		from	String	Y	
		to	String	Y	
		value	Int	Y	
		input	String	Y	

Example
{
"header": {
"code": 0,
"msg": "Transaction successful"
},
"mac":					

```
"MEQCIHX8SuEn/sDiPscd5li3X1GdseyggAyC2o9L92FjhzrfAiBLyFW/rguLkqz/Lz62Vt
X3m7Y1nHqcFqcNdM7Wq0wGLQ==",
   "body": {
      "blockHash":
"0x199eca276b60473dd65f8b36641684456694b419d89ef41b4953a9cdac848305",
      "blockNumber": 1,
      "parentBlockHash":
"0xa6886f12ee91470e35546432413ed372615f8d4c23fa82e8381b3e5b31219d4c",
      "blockSize": 0,
      "blockTime": 1587125168039,
      "transactions": [
         {
            "txId":
"0x8ee0c68e222742b5b70878265d3fdbd3a8e0d549da42a298a4ae872ca4fbfd89",
            "blockHash":
"0x199eca276b60473dd65f8b36641684456694b419d89ef41b4953a9cdac848305",
            "blockNumber": 1,
            "gasUsed": 100000000,
            "from": "0x08ac3132a6c7e6ca5a7fbaf0521bb8b6f370ed35",
            "to": "",
            "value": 0,
            "input":
"0x6080604052348015620000115760008fd5b5061100160008061010000a81548173ffffffff
ffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021790555060
0080909054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffff
ffffffffff1663c92a7801604080519081016040528060068152602001 7f745f6261736500000
0000000000000000000000000000000000000000000000000815250604051826 3ffffffff167c0
1000000000000000000000000000000000000000000000000000000000000000002815260040162000
10191906200024a565b60206040518083038160008780 3b1580156200011c57600080fd5b5
05af115801562000131573d6000803e3d6000fd5b505050506040513d601f19601f82011682
0180604052506200015791908101906200017456 5b50620002f4565b60006200016c825162
0002a3565b905092915050565b60006020828403121562000187576000 80fd5b6000620001
97848285016200015e565b9150509291505056 5b6000620001ad8262000298565b80845262
0001c381602086016020860162000 2ad565b620001ce81620002e3565b6020850101915050
92915050565b6000601382527f626173655f6b65792c626173655f76616c756500000000000
0000000000000006020830152604082019050919050565b6000600782527f626173655f696
4000000000000000000000000000000000000000000000000000000602083015260408201905091
9050565b60006060820190508181036000830152620002668184620001a0565b905081810
360208301526200027b8162000213565b905081810360408301520000 00"
         }
      ]
   }
}
```


5.4.4.10 Retrieving DApp Block Height API

This interface is used to retrieve block height in a DApp.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/getBlockHeight

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1

4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
Example:					
{"header":{"appCode":"cl000620200318192657367 7572","userCode":"USER000620200318195 1281835816"},"body":{},"mac":"MEQCIHb2o7hb0apDukOQBXkZftETsizDBaftnHxO9A9ux5 EtAiABuiFrVYPWT5FiU+Wd9HpXF/AJh0Yh2SXtL6h98m4eZw=="}					
signature value:					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response message	msg	String	N	if code=0 then can be null
Body					
1	Block Height	data	string	N	If code not 0, then leave blank
Example					
{ "header": { "code": 0, "msg": "Transaction Successful" }, "mac": "MEQCICtCOdv4ZL72M3WoA9nAei2P0/PpKjlgI0Y5qeuzg61uAiA9D3TcB/+b2RMu NwVq+X0vgiglHfM5NBhoTJPR0gCPMA==", "body": { "data": "4" } }					

5.4.4.11 Retrieving Total Count of DApp Transactions API

This interface is used to retrieve the total count of transactions in a DApp.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/getTxCount

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
Example:					
{"header":{"appCode":"cl000620200318192657367757 2","userCode":"USER000620200 3181951281835816"},"body":{},"mac":"MEQCIBRhaM2szckWl9N9qcqnaYXOXGQw7 SfII9DlRvxcI3YVAiBt4XeNs+EUjhBNSr3IjLRPZucsuGHxfjt9RiaNIQS8cA=="}					
signature value:					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response Message	msg	String	N	if code=0 then can be null
Body					
1	Transaction Information	data	string	N	If code not 0, then leave blank
Example					
{ "header": { "code": 0, "msg": "Transaction Successful" }, "mac": "MEQCIGgXINn3B9d/hC/ow0IJvi5eKDj59QbZRFdrCqcUeNCgAiApI4jkwhTY33qev1 RwsJ3veDBKXokvIiSe3ck7SKlxmg==", "body": { "data": "{\"txSum\":5,\"blockNumber\":5,\"txSumRaw\":\"0x5\",\"blockNumberRaw\":\"0x5\"}" } }					

5.4.4.12 Retrieving Total Count of Block Transactions API

This interface is used to retrieve the total count of transactions inside a block according the block number in a FISCO DApp. The block number cannot be empty.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/node/getTxCountByBlockNumber

2. Call Method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
2	Block number	blockNumber	string	Y	
Example:					
{ "header":{"appCode":"CL188103887322019090211431 4","userCode":"newuser"}, "body": { "grouId":1, "blockNumber":22, }, "mac":"MEQCIBRhaM2szckWl9N9qcqnaYXOXGQw7SfII9DlRvxcI3YVAiBt4XeNs+EUjhBNSr3IjLRPZucsuGHxfjt9RiaNIQS8cA=="}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: authentication successful -1: authentication failed
2	Response message	msg	String	N	if code=0 then can be null
Body					
1	Block total count of transactions info	data	string	N	If code not 0, then leave blank
data					
Example					
{ "header": { "code": 0, "msg": "Transaction Successful" }, "mac": "MEUCIQCMFbVhfH9X8pJ1mNI3YpzKIBcXCpfmf2AniF/42ak9EwIgTWDEF+xW5l39ZDUnDSSSc8Zv8J1glEf9izp16eW/Rn4=",					

```
  "body": {
    "data": "1"
  }
}
```

5.4.4.13 Registering Smart Contract Event API

Smart contract event in a DApp can trigger the off-BSN system to process further transactions. This interface is used to register the smart contract event to be monitored.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobcos/v1/event/register

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Event Type	eventType	String	Y	1.Block generation event 2.Contract event
2	Contract address	contractAddress	String	N	EventType is 1 then can be null; EventType is 2 then EventType and contract Name cannot be null at the same time
3	Contract name	contractName	String	N	EventType is 1 then can be null; EventType is 2 then EventType and contractName cannot be null at the same time
4	Notification URL	notifyUrl	String	Y	
5	Attached parameters	attachArgs	String	N	
Example:					
{"header":{"userCode":"USER000120200604232157969244 0","appCode":"app000120200604232 3057101002","tId":""},"mac":"MEUCIQCMP1ToZS5e8S94kYZ/8y5XfeyjRyUrPFpeIQMES3SGp QIgO8b6O8Kk/qpNTo1vbNTwyAYNaw6HBi9OkAH8Rp23j8s=","body":{"eventType":1,"contra ctAddress":"0x866aefc204b8f8fdc3e45b908fd43d76667d7f76","contractName":"BsnBaseContract k1","notifyUrl":"http://127.0.0.1:18080","attachArgs":"abc=123"}}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: successful -1: failed
2	Response Message	msg	String	Y	
Body					
1	Event ID	eventId	String	Y	Null when the code is not 0
Example					
{ "header": { "code": 0, "msg": "Transaction successful" }, "mac": "MEUCIQDYSTwYhh6EDHT5Z7ukcqXW9LMjZW6WPnrv8Xt14RuH2AIgIwa5K7NK4/TThzs8z6VfkpNNJU+dzAXeypFmfjkru88=", "body": { "eventId": "xxxxxxxxxxxxxxxxxxxxxxxxxxx" } }					

5.4.4.14 Smart Contract Event Query API

Use this API to query the list of monitored smart contract events that have been registered.

1. Interface address:

   https://PCNGatewayAddress/api/fiscobocs/v1 /event/query

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Example:					
{"header":{"userCode":"USER000120200604232157969244 0","appCode":"app000120200604232 3057101002","tId":""},"mac":"MEUCIQC2NTuUlsxQSWPpZwwhJK9zXEMaeYZC04Ar0P5Twy p5AQIgFvZrskasuLiYfOGxd1F9TCetWHIfENg8BCiYfNS1xGk=" }					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
Header					
1	Response ID	code	int	Y	0: Query successful -1: Query failed
2	Response Message	msg	String	Y	
body					
1	Block generation event	blockEvent	[]blockEvent	Y	Null when the code is not 0
2	Contract event	contractEvent	[]contractEvent	Y	
blockEvent					
1	block generation event	eventId	string	Y	Null when the code is not 0
2	App code	appcode	String	Y	
3	User code	userCode	String	Y	
4	Notification URL	notifyUrl	String	Y	
5	Attachment parameters	attachArgs	String	N	
6	Create time	createTime	String	Y	UTCtime
contractEvent					
1	block generation event	eventId	string	Y	
2	App code	appcode	String	Y	
3	User code	userCode	String	Y	
4	Notification URL	notifyUrl	String	Y	
5	Attachment parameters	attachArgs	String	N	
6	Create time	createTime	String	Y	UTCtime
7	Contract address	contractAddress	String	Y	

Example


```
{
  "header": {
    "code": 0,
    "msg": "Transaction succssful"
  },
  "mac":
"MEUCIQCQ/RjmlVkLKZw6jcLKBPh1BwK4EIQE001vUAKPVq1HTgIgXUQ7Bn+y8
D8xQxYUwtZOoh/bpteAPCUtKXZeAiN7cMU=",
  "body": {
    "blockEvent": [
      {
        "eventId": "ba537419953e4e219ceb0fe26ad5e125",
        "appCode": "app000120200604232305710100 2",
        "userCode": "USER000120200604232157969244 0",
        "notifyUrl": "http://127.0.0.1:18080",
        "attachArgs": "abc=123",
        "createTime": "0001-01-01 00:00:00.000 +0000 UTC"
```

```
    }
   ],
     "contractEvent": [
 {
     "eventId": "ba537419953e4e219ceb0fe26ad5e126",
     "appCode": "app000120200604232305710100 2",
     "userCode": "USER000120200604232157969244 0",
     "notifyUrl": "http://127.0.0.1:18080",
     "attachArgs": "abc=123",
     "createTime": "0001-01-01 00:00:00.000 +0000 UTC",
    “contractAddress”:” 0x866aefc204b8f8fdc3e45b908fd43d76667d7f76”
    }
     ]
 }
}
```


5.4.4.15 Remove Smart Contract Event API

This interface is used to remove a smart contract event’s registration from the event list.

1. Interface address:

   https://PCNGatewayAddress/api/fabric/v1/chainCode/event/remove

2. Call method: POST
3. Signature algorithm: required and refer to Section 5.4.4.1
4. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
Body					
1	Event ID	eventId	String	Y	
Example:					
{"header":{"appCode":"CL20191107112252","userCode":"lessing"},"body":{"eventId":"bd3391dedbe44a7ad5b7f80ce59abfa"},"mac":"MEQCIE3/CLG5LxZZN7En7LZvzthajwxHzpvDduXSsw4Tb1JFAiAXGJ4WVtyCKbtCasQGofCkge8NOgZDNPgJIdTCtCi2SQ=="}					

5. Response parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	Y	
3	Signature Value	mac	String	Y	
header					
1	Response ID	code	int	Y	0: remove successful -1: remove failed

2	Response Message	msg	String	Y	
Example					
{"header": {"code": 0, "msg": "Remove Event Successful"}, "body": null, "mac": "MEUCIQCaTFLliY7pPjkwcmSsLXOth7k9bQj9Sblq+1nMVjkFAAIgUsizFO+f1+dxU3/hPxjf/+na4qG6aQFftJIWGtMhlVI="}					

5.4.4.16 Smart Contract Event Notification Message API

This interface is implemented on the off-BSN system side. When the PCN gateway receives the notification of a triggered event, it uses this interface to notify the off-BSN system about the execution result.

After receiving the notification successfully, the off-BSN system returns a string containing "success", otherwise, the gateway will send the notification again at 3, 12, 27, and 48 seconds respectively, for a total of five times.

1. Call method: POST
2. Signature algorithm: required and refer to Section 5.4.4.1
3. Call parameters

No.	Field name	Field	Type	Required	Remarks
1	Header	header	Map	Y	
2	Body	body	Map	N	
3	Signature Value	mac	String	Y	
header					
1	user unique ID	userCode	String	Y	
2	DApp unique ID	appCode	String	Y	
body					
1	Registered Event ID	eventId	String	Y	
2	PCN ID	orgCode	String	Y	
3	Registered Event parameters	attachArgs	String	N	Additional parameters entered during registration
4	Response random string	nonceStr	String	Y	Off-BSN system uses this value to judge if the notification is already received. This string remains the same at the repeated notifications.
5	Event type	eventType	String	Y	
6	Event data	eventData	String	Y	
Example:					
{"header":{"userCode":"USER000120200604232157969244 0","appCode":"app000120200604232 3057101002"},"body":{"eventId":"5b5b865f8dc94ae59d215cf26aa81d69","orgCode":"ORG20200 41114171692360","appCode":"app000120200604232305710100 2","attachArgs":"abc=123","nonc eStr":"52f080f27ff045eb87e21812d12cee40","eventType":1,"eventData":"{\"appId\":\"app000120 2006042323057101002\",\"blockNumber\":17,\"eventType\":1,\"groupId\":135}"},"mac":"MEUCI					

QD3Sp6xuI4DHy/GOb9z3nH6kQisEzfXvZ/Hn/mfZXIAOgIgYsISRfBKSJGt4FrmxETflfR4A8VenCZHvxthMFUWRkc="}

5.4.4.17 Transaction Receipt Status

Under Key Trust Mode, the description of the returned transaction status when the off-BSN system invokes the FISCO DApp smart contracts via PCN gateway APIs are shown as follows:

status(Decimal/ Hexadecimal)	message	Explanation
0(0x0)	None	No Error
1(0x1)	Unknown	Unknown Error
2(0x2)	BadRLP	Invalid RLP Error
3(0x3)	InvalidFormat	Invalid Format Error
4(0x4)	OutOfGasIntrinsic	The length of smart contract exceeds gas limit/smart contract invoking parameters exceed gas limit
5(0x5)	InvalidSignature	Invalid Signature Error
6(0x6)	InvalidNonce	Invalid nonce Error
7(0x7)	NotEnoughCash	Not enough cash Error
8(0x8)	OutOfGasBase	Parameters too long (RC version)
9(0x9)	BlockGasLimitReached	Gas limit reached Error
10(0xa)	BadInstruction	Bad Instruction Error
11(0xb)	BadJumpDestination	Bad Jump Destination Error
12(0xc)	OutOfGas	Out of gas to execute the smart contract/the length of smart contract exceeds the limit.
13(0xd)	OutOfStack	Out of Stack Error
14(0xe)	StackUnderflow	Stack Under Flow Error
15(0xf)	NonceCheckFail	Nonce check failed Error
16(0x10)	BlockLimitCheckFail	Block limit check failed Error
17(0x11)	FilterCheckFail	Filter check failed Error
18(0x12)	NoDeployPermission	No Deployment Permission Error
19(0x13)	NoCallPermission	Illegal call Error
20(0x14)	NoTxPermission	Illegal transaction Error
21(0x15)	PrecompiledError	Precompiled Error
22(0x16)	RevertInstruction	Revert Instruction Error
23(0x17)	InvalidZeroSignatureFormat	Invalid Signature Format
24(0x18)	AddressAlreadyUsed	Address Already Used Error
25(0x19)	PermissionDenied	Permission Denied
26(0x1a)	CallAddressError	Call Address does not exist Error

5.5 Development SDK and Examples

5.5.1 BSN Gateway SDK Example

Normally, if an off-BSN system wants to communicate with a permissioned DApp service on BSN, it has to call the public city nodes (PCN) gateway APIs. We provide a BSN Gateway SDK (Software Development Kit) which can help developers quickly implement an off-BSN system to call the PCN Gateway. Inside the SDK, we provide PCN gateway API encapsulation which you can use to implement the transaction querying, transaction interface calling, generate public key and private key locally, register user certificate, generate certificate signature, encrypt and decrypt data, etc.

Download links:

https://github.com/BSNDA/PCNGateway-Go-SDK

https://github.com/BSNDA/PCNGateway-Java-SDK

https://github.com/BSNDA/PCNGateway-PY-SDK

https://github.com/BSNDA/PCNGateway-CSharp-SDK

5.5.2 Off-BSN System Examples

For your reference, the following examples are sample source code of chaincode/smart contract invocation through gateway API by the off-BSN systems developed based on prefabricated chain code/smart contract package, including Golang, Java, C#, and python language examples.

- **Fabric example**

Download links:

https://github.com/BSNDA/FabricBaseChaincode

- **FISCO BCOS example**

Download links:

https://github.com/BSNDA/FISCOBaseContract

We invite experienced developers who are interested in BSN to work together to optimize the SDK and sample packages. If you'd like to participate, please contact us on GitHub.

5.6 BSN Testnet Services

5.6.1 Overview

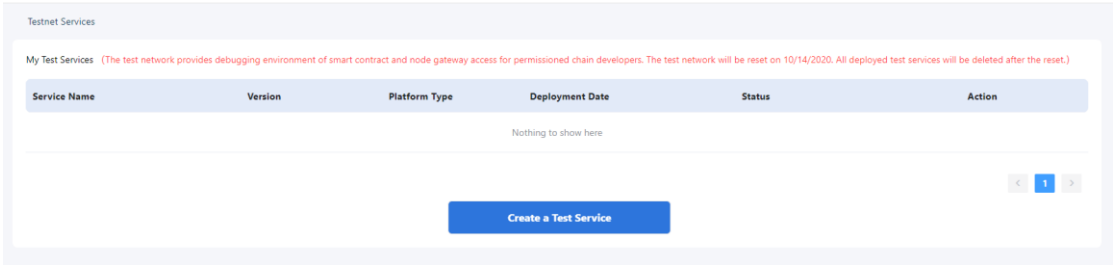
BSN Testnet is a free test environment for developers to test their permissioned DApp services. Developers can publish an unlimited number of permissioned DApp services on the testnet. Unlike the BSN production environment, it is not necessary to choose the public city nodes and configure the invocation authorities of smart contracts when publishing DApp services on the testnet. The Testnet supports Hyperledger Fabric and FISCO BCOS frameworks, and will continue to integrate all BSN-adapted permissioned frameworks. Like all testnets do, we will occasionally reset the Testnet and delete all smart contracts and ledger data. Therefore, please do not use the Testnet as a commercial or production environment. We welcome developers to try the service and provide us with feedback and suggestions as we continue to make improvements.

5.6.2 Permissioned DApp Service Publication

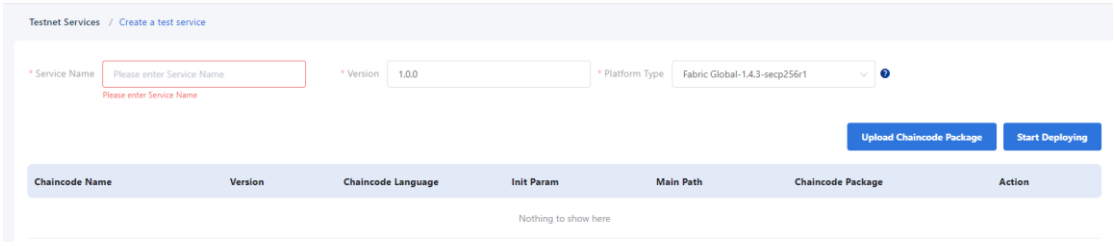
The steps to publish a permissioned DApp service for testing are as follows:

1. Create a new test service

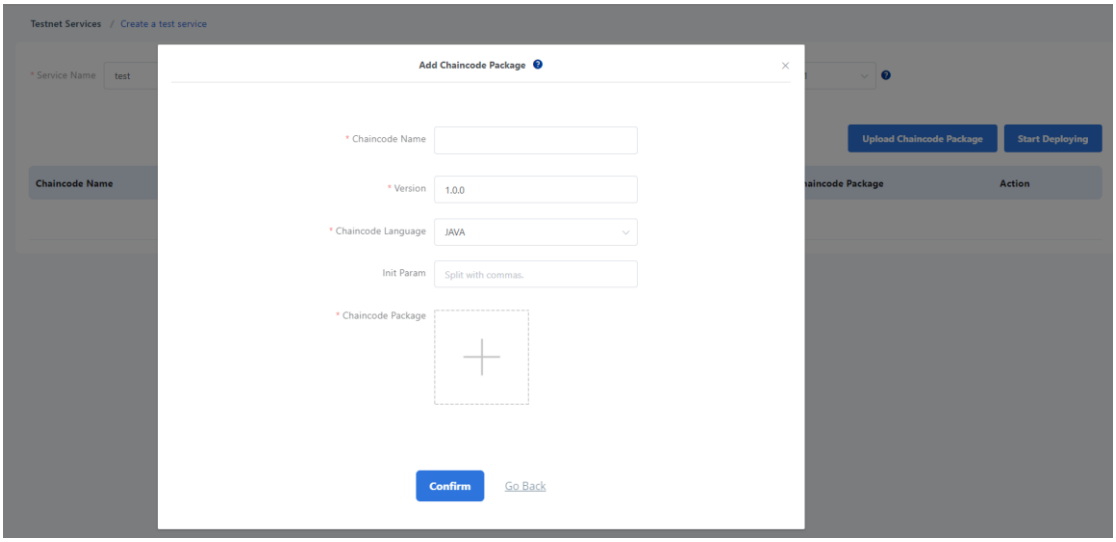
   Go to the **Permissioned Services** > **Testnet Services** page to publish the service.



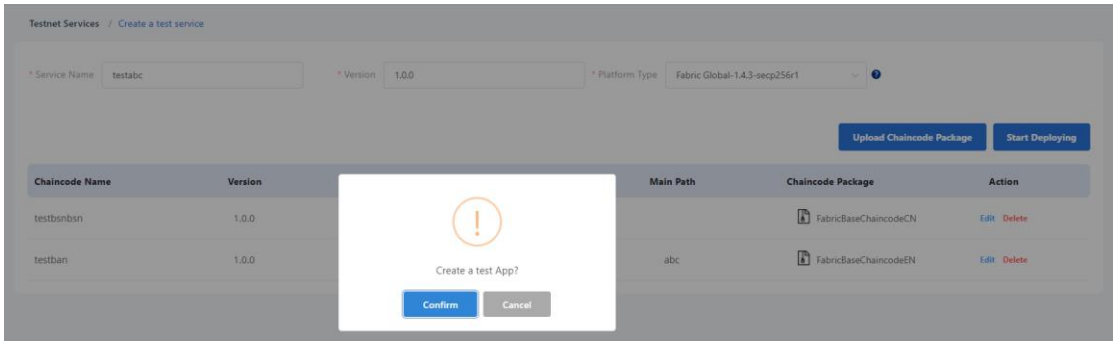
Click **Create a Test Service** and input the service name, version, and select a platform type.

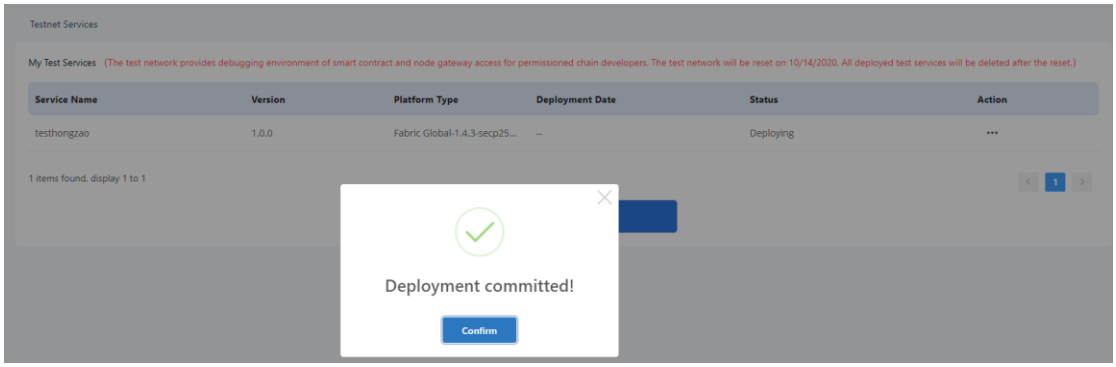


Click **Upload Chaincode Package** to upload the chaincode or smart contract package. You can upload multiple chaincode/smart contract packages in a permissioned DApp service. Input the information and click **Confirm** to upload the package.



2. Deploy the permissioned DApp service:
   Click **Start Deploying** to deploy the service.





After successfully deploying the chaincode/smart contract, developers can call it from their off-BSN systems so that they can configure and debug the functions easily.

Note: To keep the resources stable, DevOps will periodically clean up the chaincode/smart contract packages and ledger data on the Testnet.

5.6.3 Interchain Services on BSN Testnet

A demo version of Interchain Communications Hub (ICH) is now live on the Testnet, integrating two interchain solutions based on the relay chain mechanism: (1) Poly Enterprise developed by Onchain Tech and (2) IRITA developed by Bianjie AI. We welcome developers to try out and provide feedback and suggestions, and we will continue to improve the functionality and expect to release a commercial version in 2021.

For detailed descriptions and examples of ICH services, please refer to chapter 7, "Interchain Services"

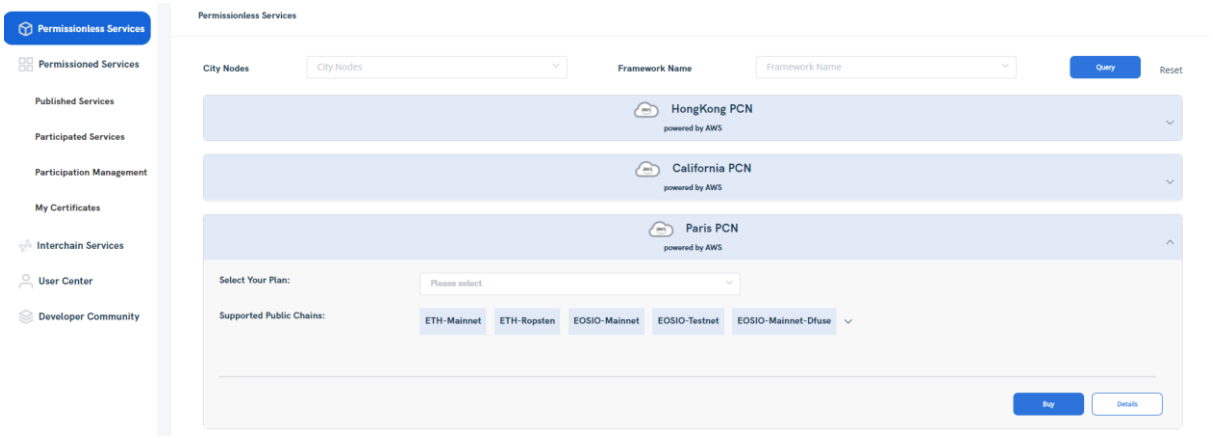
6 Permissionless Services

6.1 Overview

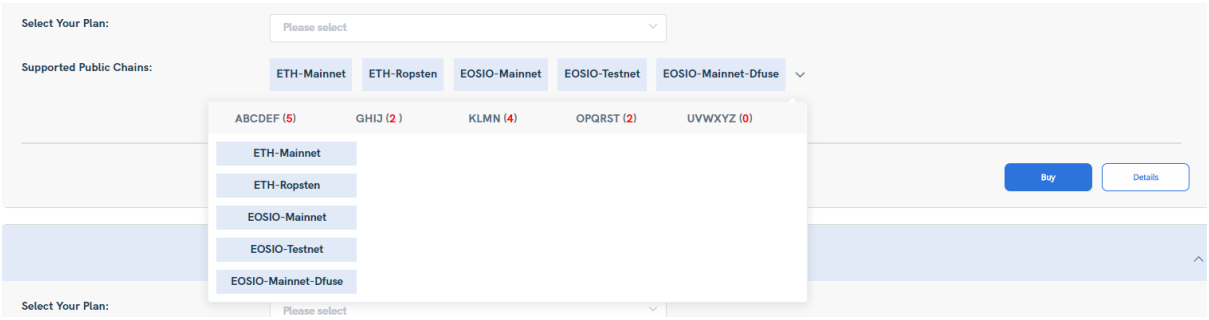
The Permissionless service allows the participant to select a public city node to access a plan that can be a free plan or a premium plan. When this is done, the participant can create a project, obtain the project ID, key and access parameters which can be used to access selected public chain node gateway. With the Permissionless service, the default plan is free for participants, however, it has limited daily requests and projects. BSN has created several other plans that can be upgraded to, for a certain fee, paid on a monthly basis.

6.2 Select Plans

On the page of Permissionless services, users can select different city nodes to participate in Permissionless services. The nodes in blue at the top of the list represent those activated for the free plan or premium plans on that city node. The nodes in grey at the bottom represent no plans are purchased or used on that city node.

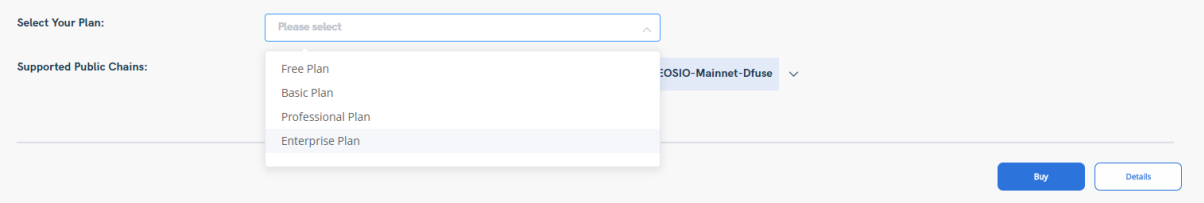


When you click and expand the public city node, you can see all public chain frameworks supported by the city node. Users can decide whether to choose this city node as the access entrance according to their needs. The public chain frameworks supported by different city nodes may be different. In general, we recommend that developers choose a city node that is close to them, so that the access speed will be relatively fast.

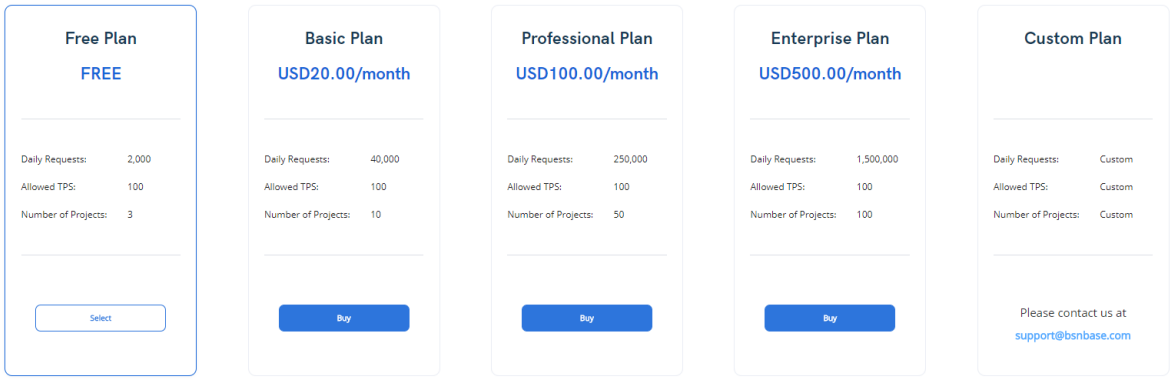


By default, participants on the Permissionless service have a **free** plan that is free to use up to 2000 daily requests, allowed TPS of 100 and maximum of 3 projects. However, a participant can upgrade to a higher plan available on the platform. To select plans, follow these steps

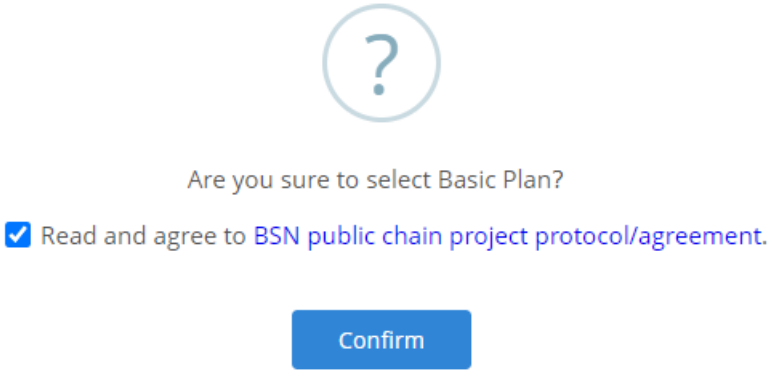
1. On the Permissionless page, click **Buy** in the **Select Your Plan** section.



2. In the **Details** page, locate the **Select or Update your plan** section and click **Buy** on the appropriate plan.

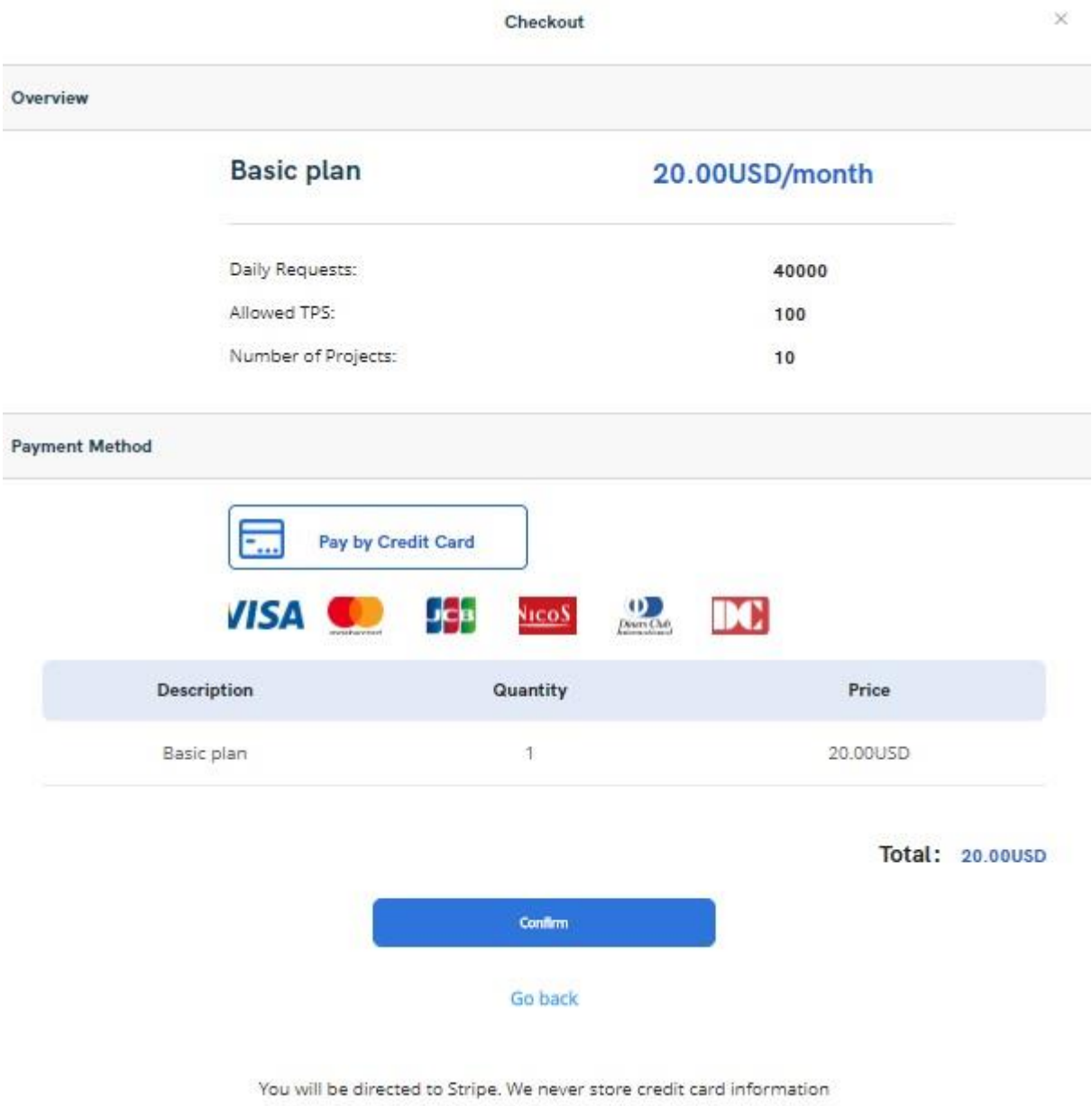


3. In the **Are you sure you want to buy package** window, click the project agreement and click **Confirm**.

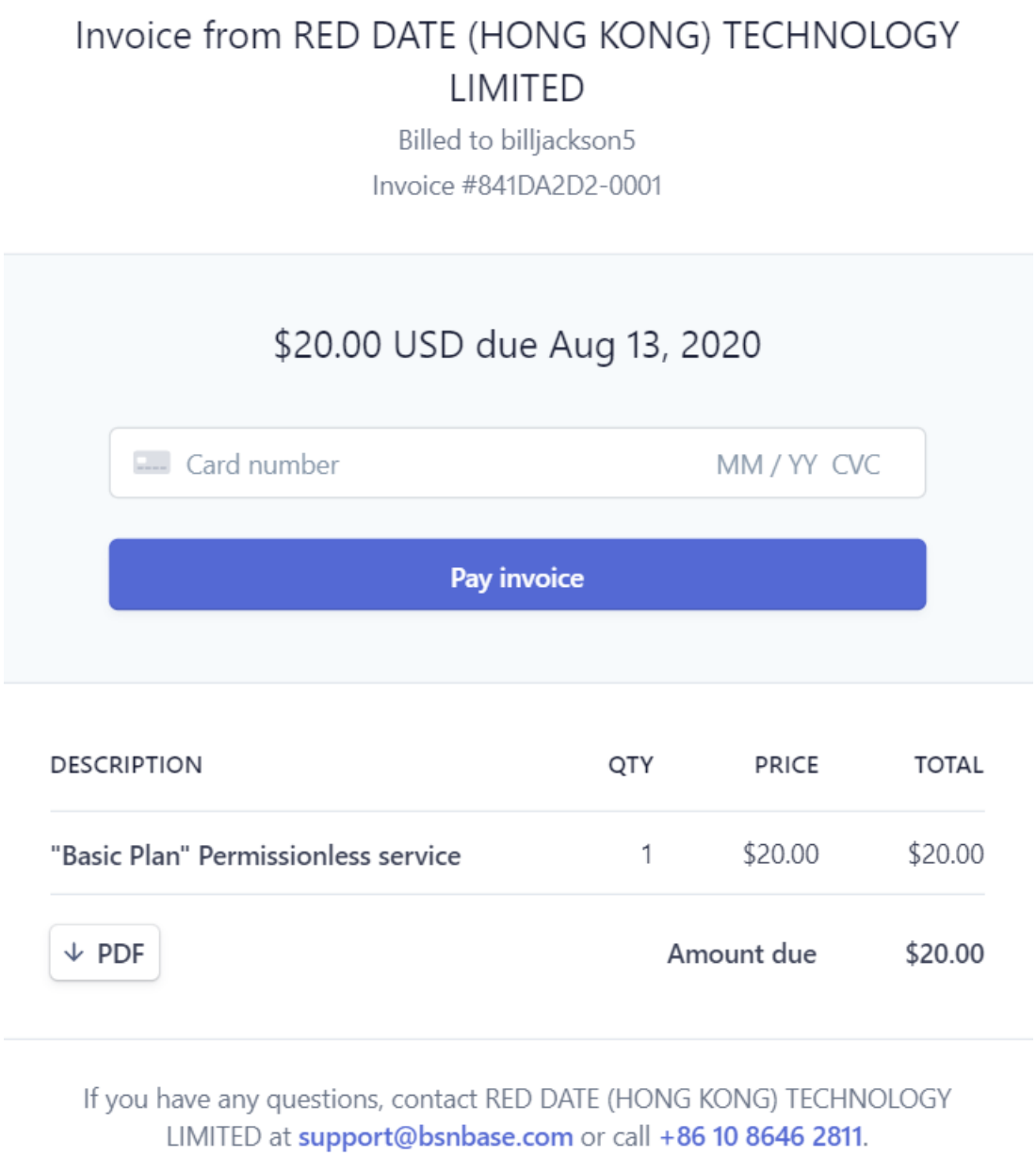


4. In the **Select Payment Method** page, select the appropriate payment method and click **Next Step to be redirected to Stripe**.

The BSN portal never records and stores any credit card information.



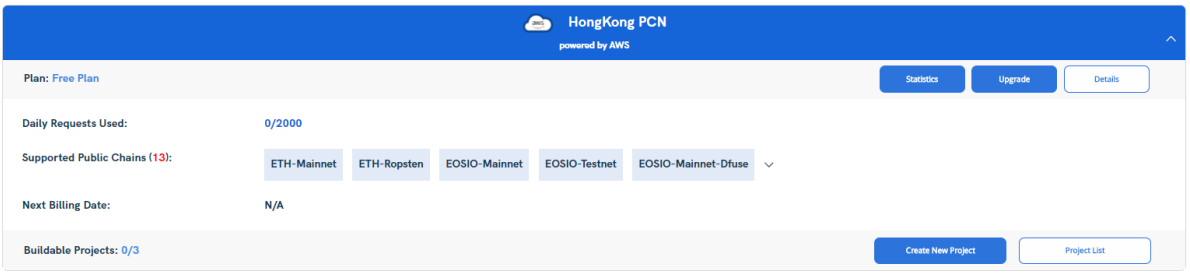
5. On the **Stripe Payment** page, click **Pay** to display the **Receipt** and **Invoice**.



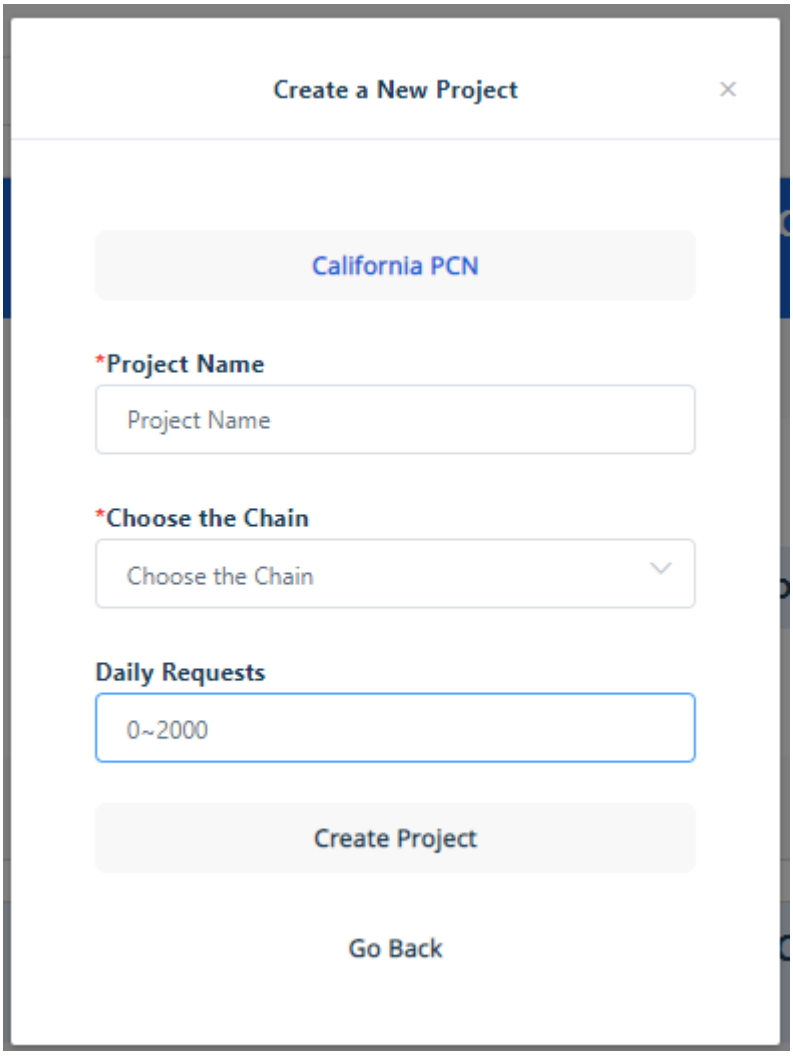
6.3 Create and Manage Projects

With the Permissionless service, projects can be created in a much simpler way when compared with Permissioned service as plans are embedded into the project, making it easier for participants to manage. To create and manage projects follow these steps:

1. In the Permissionless Service page, click **Create new project** in the development plan section.



2. In the **Create a new project** window, enter the **Project Name**, select the Public **Chain to access** from the dropdown list, input the **Daily Requests** number if needed. Then click **Create Project**. The Daily Requests number is optional, and it is used to control the TPD (transactions per day) for this project.

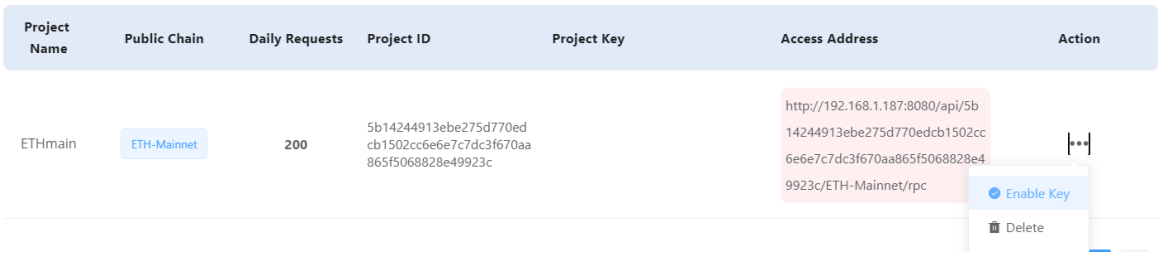


This will automatically create the project and list it in the **Project Information** tab.

After a project has been created it can be managed using available tools for the project. To manage a project, follow these steps:

1. Locate the project to be managed, click **Upgrade** to display the **Plans** page.
2. Select the appropriate plan to **Upgrade** to and click **confirm** to display the payment page.

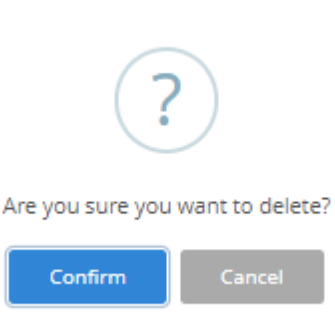
3. To enable the project key, in the Permissionless Service page, click **Project list** to display the list of projects. In **Action**, click **Enable Key** to enable the project key. Then the information page on enabling the key will be displayed. Click **Confirm.**



4. To update a project key, click **Update Key**. Then the information page on updating the key will be displayed. Click **Confirm.**



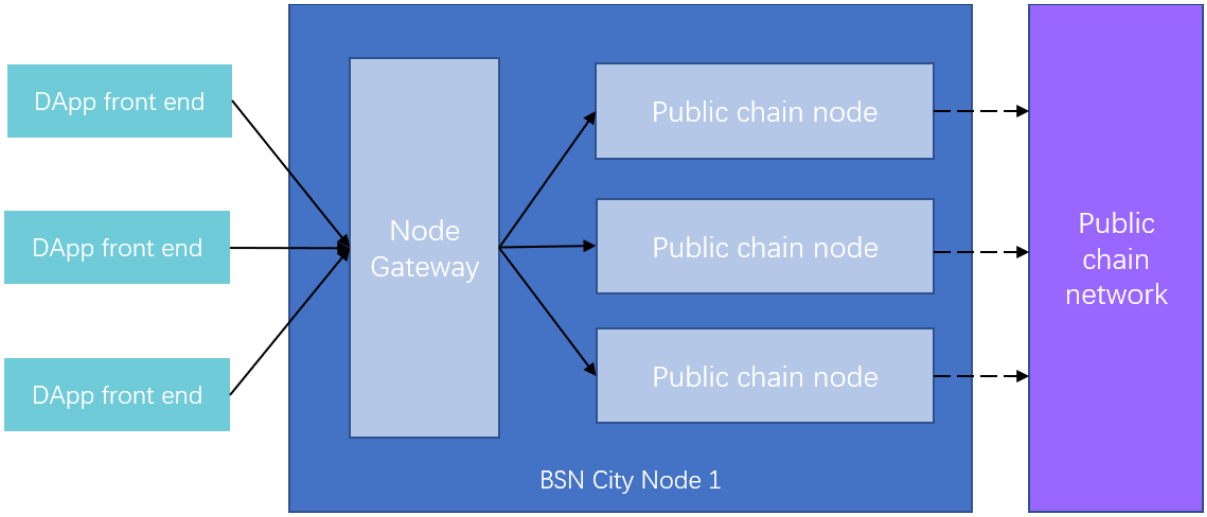
5. To delete a project, click **Delete**. A confirmation message will be displayed asking if you wanted to delete the project. Click **Confirm** to delete it.



6.4 Off-BSN system Access Guide

6.4.1 Overview

BSN provides shared or dedicated public chain nodes for public chain application developers. Developers can quickly access all public chain networks by accessing the gateway of the public city node.



After developers select the public chain framework (netcode) in the BSN portal to create the public chain project, they will get the gateway's domain name address (url), project number (id), project key (key), public chain supportive protocol {protocol} and public chain gateway API address.

The developer accessing the PCN gateway via HTTP should concatenate the request address in "https://{url}/api/{id}/{netcode}/{protocol}/{subUrl}" format. If project key is enabled, "x-APi-key:{key}"should be added to the request header. If the public chain nodes provide multiple components, they should add {subUrl}; If the Nervos CKB has an Indexer component service in addition to the RPC service, "{subUrl}" should fill the indexer value, {subUrl} is optional.

The developer accessing the node gateway via WebSocket, should concatenate the Key and SubUrl to the path address of the target machine and concatenate to the format of {url}/api/{id}/{key}/{netcode}/{subUrl}. If the project key is not enabled, then the {key} filed should be null. If there is no subUrl, this field can be null. That is, developers can think of the content after/API as the method name of a target machine.

6.4.2 Ethereum

Ethereum is a global, open-source platform for decentralized applications. On Ethereum, you can write code that controls digital value, runs exactly as programmed, and is accessible anywhere in the world.

For more resources, please visit: https://ethereum.org/en/developers/

The BSN public city node gateway is adapted to the Ethereum JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JS-RPC. For detailed docking instructions please visit: https://eth.wiki/json-rpc/API

The following table shows additional error code definitions for public city node gateways:

Error code	Transaction error code	Error code description
500	-32099	Service internal exception
503		
429	-32098	TPS, TPD current limit

401	-32097	Authentication permission failed

6.4.3 EOS

EOSIO is a blockchain platform designed for the real world. Built for both public and private use cases, EOSIO is customizable to suit a wide range of business needs across industries with rich role-based security permissions, industry-leading speeds and secure application processing.

For more resources, please visit: https://developers.eos.io

The BSN city node gateway is adapted to EOSIO's JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit:

https://developers.eos.io/manuals/eos/latest/nodeos/plugins/chain_api_plugin/api-reference/index#operation/get_block

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

6.4.4 Nervos

The Nervos Network is an open source public blockchain ecosystem and collection of protocols solving the biggest challenges facing blockchains like Bitcoin and Ethereum today.

For more resources, please visit: https://docs.nervos.org/

The BSN city node gateway is adapted to the Nervos JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC.

For detailed docking instructions please visit:

https://docs.nervos.org/docs/reference/rpc

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
500	-32099	Service internal exception
503		
429	-32098	TPS, TPD current limit
401	-32097	Authentication permission failed

6.4.5 NEO

NEO is an open-source, community driven platform that is leveraging the intrinsic advantages of blockchain technology to realize the optimized digital world of the future.

For more resources, please visit: https://neo.org/dev

The BSN city node gateway is adapted to the NEO JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit:

https://docs.neo.org/docs/zh-cn/reference/rpc/latest-version/api.html

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
500	-32099	Service internal exception
503		
429	-32098	TPS, TPD current limit
401	-32097	Authentication permission failed

6.4.6 Tezos

Tezos is an open-source platform for assets and applications backed by a global community of validators, researchers, and builders. Tezos is designed to provide the safety and code correctness required for assets and other high value use cases. Its native smart contract language, Michelson, facilitates formal verification, a methodology commonly used in mission-critical environments such as the aerospace, nuclear, and semiconductor industries.

For more resources, please visit: https://developers.tezos.com

The BSN city node gateway is adapted to the Tezos JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit:

https://tezos.gitlab.io/api/rpc.html

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

6.4.7 IRISnet

Within the ecosystem of IRISnet, the core innovation is embodied in three aspects: integrate the service-oriented infrastructure into the Cosmos network; integrate business services provided by heterogeneous systems, including public chains, consortium chains, and existing systems; the connectivity of services is realized through the blockchain Internet.

For more resources, please visit: https://www.irisnet.org/docs

The BSN city node gateway is adapted to the IRISnet JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit:

https://www.irisnet.org/docs/light-client/intro.html#rest-apis

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
500	-32099	Service internal exception
503		
429	-32098	TPS, TPD current limit
401	-32097	Authentication permission failed

6.4.8 dfuse-eos

dfuse is a massively scalable open-source platform for searching and processing blockchain data. It provides real-time, historical and fork-aware search engine (dfuse Search), transaction push guarantees (dfuse Push Guarantee), transaction lifecycle (dfuse Lifecycle), historical state services (dfuse State), and many more blockchain building blocks. dfuse empowers developers with capabilities to build modern blockchain applications with fast, fluid interfaces that deliver exceptional user experiences.

For more resources, please visit: https://docs.dfuse.io

The current dfuse EOS mainnet access on BSN is available through dfuse Community Edition hosted by EOS Nation. Try out the dfuse API features on the [GraphiQL playground] (https://eos.dfuse.eosnation.io/graphiql).

If your needs exceed the Community Edition limits, please contact dfuse to set up an [Enterprise plan] (https://dfuse.io/zh/pricing/?utm_source=BSN).

The BSN PCN gateway is equipped with sfuse's JSON RPC API and GraphQL, so developers can issue EOSIO transaction requests to the node gateway via HTTP JSONrpc or GraphQL.

For detailed instructions, please visit the link: https://docs.dfuse.io/reference/eosio

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

6.4.9 Solana

The Solana Program Library (SPL) is a collection of on-chain programs targeting the Sealevel parallel runtime. These programs are tested against Solana's implementation of Sealevel, solana-runtime, and deployed to its mainnet. As others implement Sealevel, we will graciously accept patches to ensure the programs here are portable across all implementations.

For more resources, please visit below websites:

Solana Documentation Homepage: https://docs.solana.com/

Solana Program Library (SPL) Documentation: https://spl.solana.com/

JavaScript API Reference: https://solana-labs.github.io/solana-web3.js/

Developing apps on Solana: https://docs.solana.com/apps

The BSN city node gateway is adapted to the Solana JSON RPC API and WSS, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit: https://docs.solana.com/apps/jsonrpc-api

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

6.4.10 ShareRing

ShareRing is built on distributed ledger technology, allowing for a transparent, decentralized ecosystem.

For more resources, please visit: https://sharering.network/media-kit.html

The BSN city node gateway is adapted to the ShareRing JSON RPC API and WSS, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit: https://sharering.network/resources/ShareRing+API+Overview.pdf

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

6.4.11 Algorand

Algorand built and developed the world's first open, permissionless, pure proof-of-stake blockchain protocol that, without forking, provides the necessary security, scalability, and decentralization needed for today's economy. With an award-winning team, Algorand enables traditional finance and decentralized financial businesses to embrace blockchain for decentralized applications.

For more developer resources, please visit: https://developer.algorand.org/

The BSN city node gateway is adapted to the Algorand Rest API, so developers can initiate transaction requests to the node gateway via Rest. Developers can also use Algorand SDK to connect to BSN nodes for developing and deploying applications.

For detailed docking instructions please visit:

https://developer.algorand.org/docs/reference/sdks/

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	401	Authentication permission failed
429	429	TPS, TPD current limit
500	500	Service internal exception
503	503	Service Unavailable

6.4.12 BTY

BTY (Bityuan) is a simple, stable and scalable public chain network. It is developed using the technology of Chain33 and is the world's first public chain project with a multi-chain (parachain) architecture.

Multiple parachains can be developed on the BTY blockchain. The BTY's main chain is responsible for transaction settlement, and smart contracts and virtual machines are deprived from the main chain and put on the parachain for independent executions, and multiple parachains co-exist to improve computing efficiency. In addition, parachains can be interconnected by the main chain.

At present, there are a number of application cases of parachain based on BTY public chain, such as DeFi, C2C trading, royalty points, prepaid cards, games, real estate, commodities, smart clearing, etc.

For more developer resources, please visit: https://chain.33.cn/

The BSN City Node Gateway is adapted to the BTY JSON RPC API, so developers can initiate BTY transaction requests to the node gateway by means of JSON-RPC.

For detailed docking instructions, please visit: https://chain.33.cn/document/142

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

6.4.13 Oasis Network

The Oasis Network, as a next generation blockchain, is the first scalable, privacy-enabled blockchain network for open finance and a responsible data economy. Combined with its high throughput and secure architecture, the Oasis Network is able to power private, scalable DeFi - expanding it beyond traders and early adopters to a mass market, unlocking new use cases like under-collateralized loans, private AMMs and more. The Oasis Network's privacy features can also create a new type of digital asset called CryptoData, which allows users to control their data and earn rewards. It is achieved through an innovative technical architecture that separates computation and consensus layers into ParaTime and consensus layers.

Oasis Network's powerful, privacy-preserving design has been used in the following projects:

1) CryptoSafe.

2) Nebula Genomics, the first consumer genomics application that gives users complete control over their genomes.

(3) Fortune 500 Healthcare Company, for confidential data sharing.

Investors of Oasis Network include a16z, Accel, Polychain, Pantera, IOSG, etc.

To learn more information, please join the Oasis Community at t.me/oasisprotocolcommunity and follow us on Twitter @oasisprotocol.

For more resources, please visit: https://docs.oasis.dev/general/

The BSN city node gateway is adapted to the Oasis Network REST API, so developers can initiate transaction requests to the node gateway via HTTP REST. For detailed docking instructions please visit: https://www.rosetta-api.org

Error code	Error code description
401	Authentication permission failed
429	TPS, TPD current limit
500	Service internal exception
503	Service Unavailable

6.4.14 Polkadot

Polkadot is a next-generation blockchain protocol that unites an entire network of purpose-built blockchains, allowing them to operate seamlessly together at scale. Because Polkadot allows any type of data to be sent between any type of blockchain, it unlocks a wide range of real-world use cases. By bringing together the best features from multiple specialized blockchains,

Polkadot paves the way for new decentralized marketplaces to emerge, offering fairer ways to access services through a variety of apps and providers. Polkadot's design offers several distinct advantages over existing and legacy networks, including heterogeneous sharding, scalability, upgradeability, transparent governance and cross-chain composability.

For more resources, please visit: https://substrate.dev/

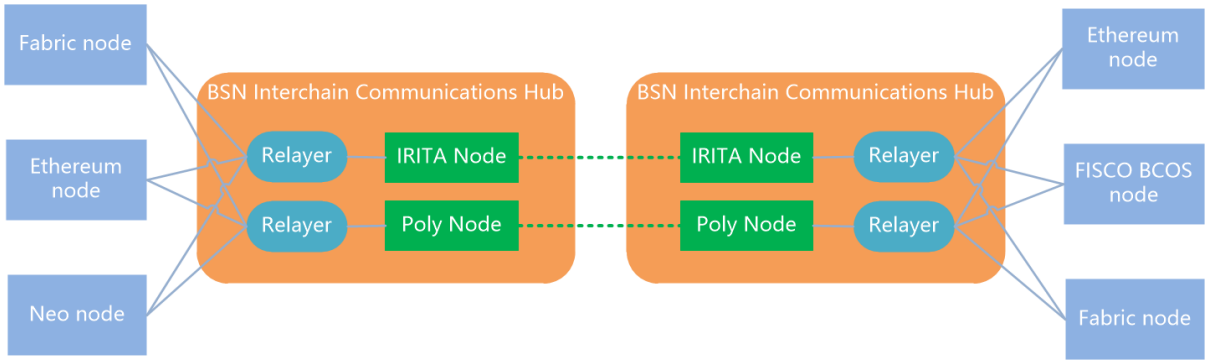
The BSN city node gateway is adapted to the Polkadot JSON RPC API, so developers can initiate transaction requests to the node gateway via HTTP JSON-RPC. For detailed docking instructions please visit: https://polkadot.js.org/docs/substrate/rpc/#chain

The following table shows additional error code definitions for public city node gateway:

Error code	Transaction error code	Error code description
401	3090000	Authentication permission failed
429	3210000	TPS, TPD current limit
500	3100000	Service internal exception
503		Service Unavailable

7 Interchain Services

A cross-chain mechanism is the interoperability between two or more relatively independent blockchains, and it enables the swap and transfer of data, asset and information. On the BSN, every blockchain maintains its own transactions, consensus, and ledgers, carrying business data and information of different DApps. The cross-chain mechanism realizes data sharing and business collaboration among blockchains, and to break the silos between chains, allows data to flow securely and reliably across multiple chains. The main functions of the cross-chain system include: cross-chain registration management mechanism, cross-chain contract functions, cross-chain transaction verification, cross-chain message routing protocol, cross-chain transaction atomicity guarantee, etc.



The BSN Interchain Communications Hub (ICH) adopts the cross-chain protocol of heterogeneous chains and the design of double-layer structure, using relay chains as cross-chain coordinators, multiple heterogeneous chains as cross-chain transaction executors, and acts as a relayer of cross-chain data. By solving validity, security, and transactional issues of cross-chain data, a secure, easy-to-use and efficient cross-chain system is implemented：

- Supports both isomorphic and heterogeneous chains.
- Supports any information to cross the chains.
- Very easy to access. Application chains do not need to do custom development adaptation, just deploy one smart contract per chain.
- Transactional support, supporting not only scenarios with the need for ultimate consistency of transactions, but also scenarios with the need for strong consistency of transactions, with support for any transaction, and scalable to any number of chains.
- Cross-chain protocols are secure and reliable, based on cryptography and consensus algorithms, and each application chain can verify the legitimacy of cross-chain transactions on its own, thus ensuring the security of cross-chain interactions.

The BSN's "Interchain Communications Hub" (ICH) is now commercially available and integrates with Onchain's Poly Enterprise cross-chain solution. It supports cross-chaining between permissioned chains and cross-chaining between permissioned chains and ETH Ropsten testnet and NEO testnet. The IRITA-based cross-chain solution is also being adapted and is expected to be commercially available in the next iteration.

A demo version of ICH is also live on the BSN Testnet, integrating two interchain solutions based on the relay chain mechanism: Poly Enterprise developed by Onchain and IRITA

developed by Bianjie AI. We welcome all developers to try out and provide feedback and suggestions to us and we will continue to improve the interchain functionality.

7.1 Interchain Service Management

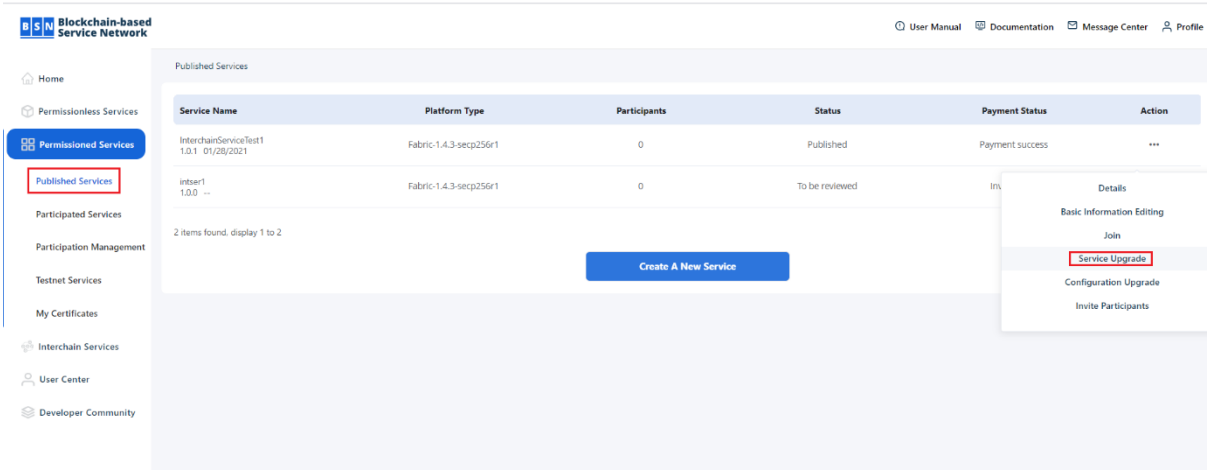
7.1.1 Open Interchain Services

There are two ways to open Interchain Services: permissioned DApp service publishers can either open it when upgrading their services, or they can open Interchain Services separately.

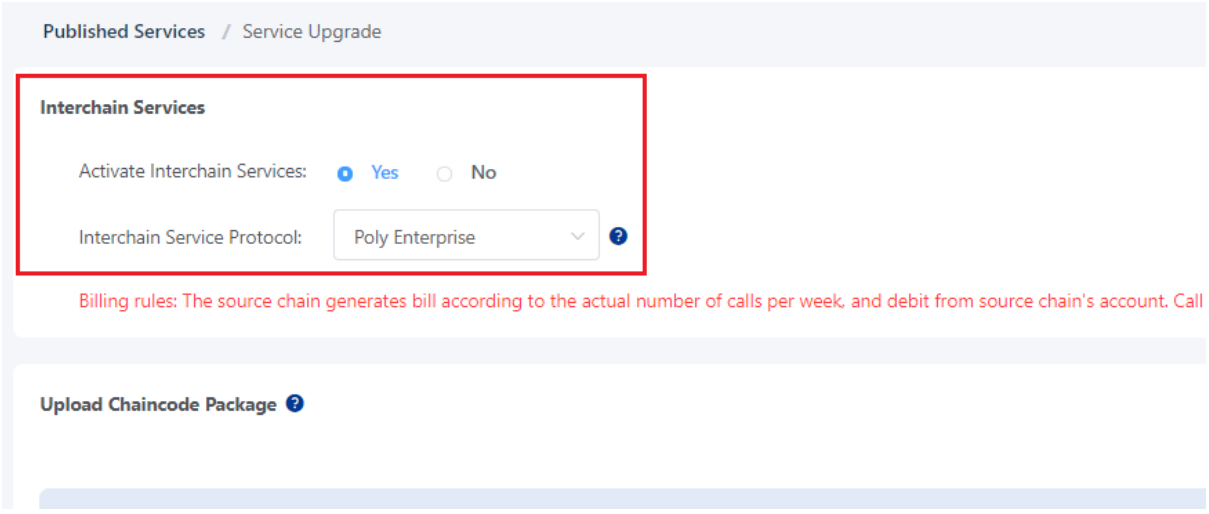
1) Open the Interchain Service when upgrading the permissioned service.

For published permissioned services, publishers can open Interchain Services through the **Service Upgrade** function:

On the home page, click **Permissioned Services** -> **Published Services**, click **Service Upgrade** in the **Action** column to enter the service upgrade page.



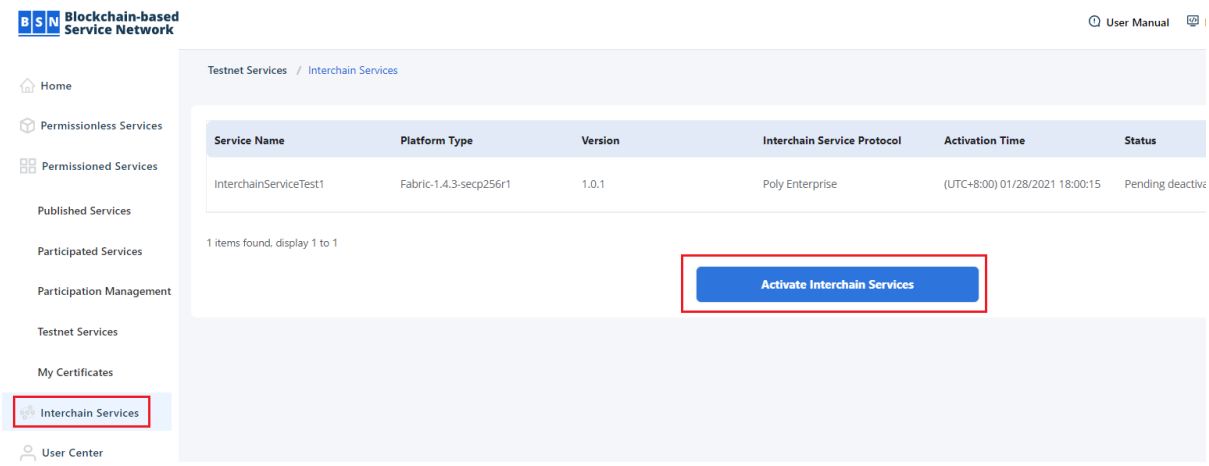
In the Interchain Services section, select **Yes** to activate Interchain Services, and choose the Interchain Service Protocol. Then, click **Confirm** to submit the service upgrade. After the system review and approval, the Interchain Service is successfully opened.



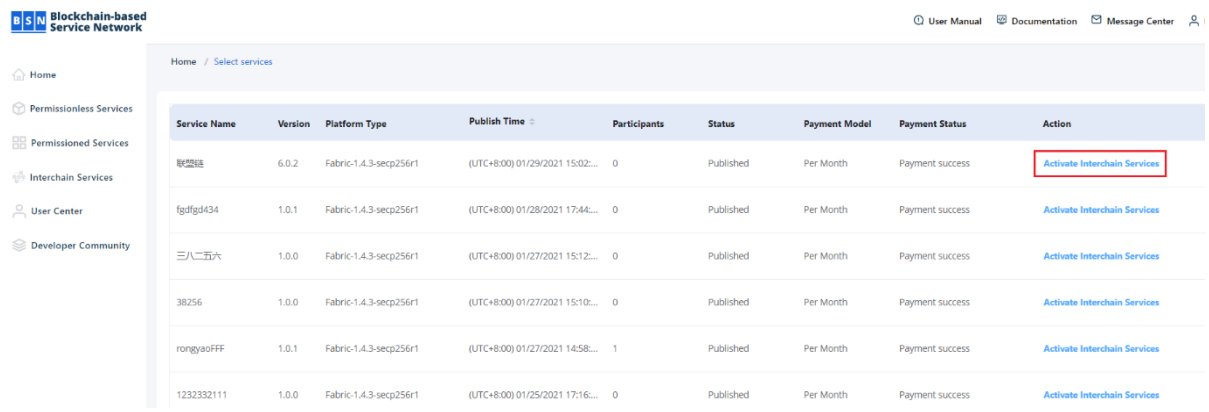
Note: If you open Interchain Services only, you don't need to upload new chaincode package; after opening the service, when calling across the chain, both source chain and target chain need to communicate off the BSN about cross-chain parameters, methods and specifications.

2) Directly open the service in Interchain Services

On the home page, click **Interchain Services**



Click **Activate Interchain Services** button to enter **Select services** page, click **Activate Interchain Services** in the **Action** column.

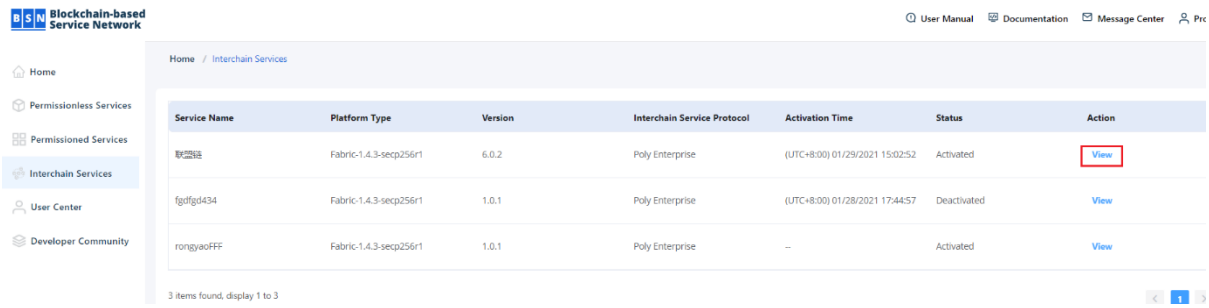


The following steps can refer to **Open the Interchain Service when upgrading the permissioned service**.

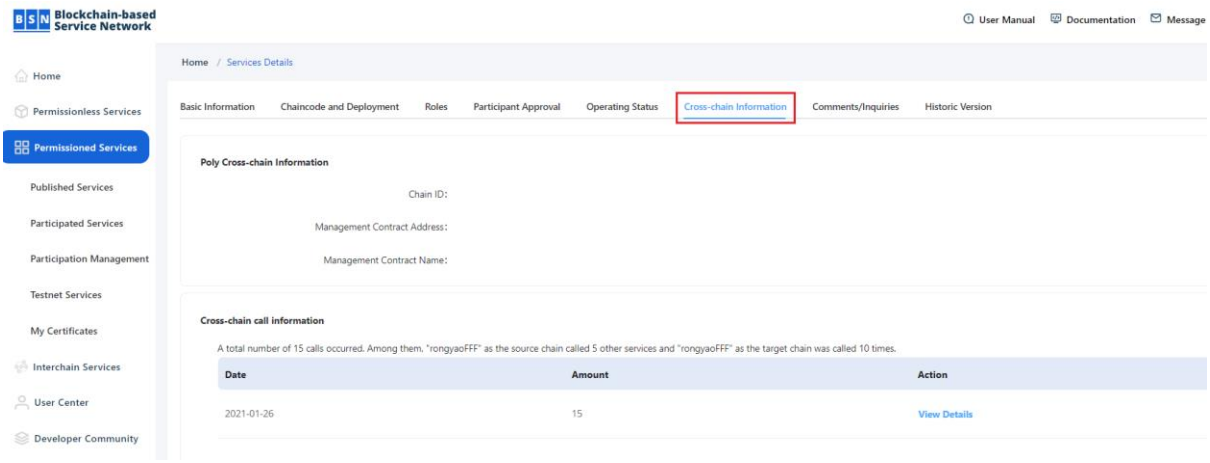
Note: For activated interchain services, users cannot change the interchain service protocols. The protocol can only be changed by re-opening the interchain services.

7.1.2 View Interchain Services

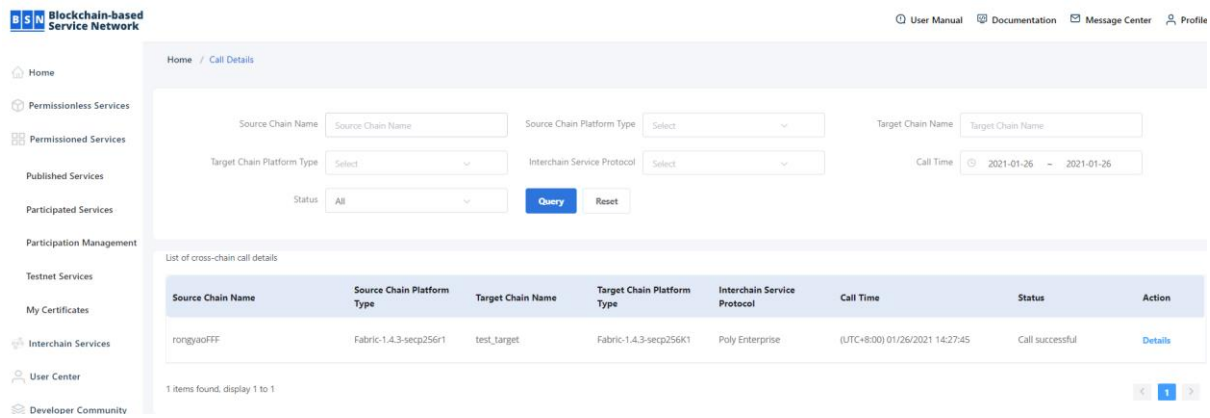
On the home page, click **Interchain Services**, users can find the service list of their activated interchain services.



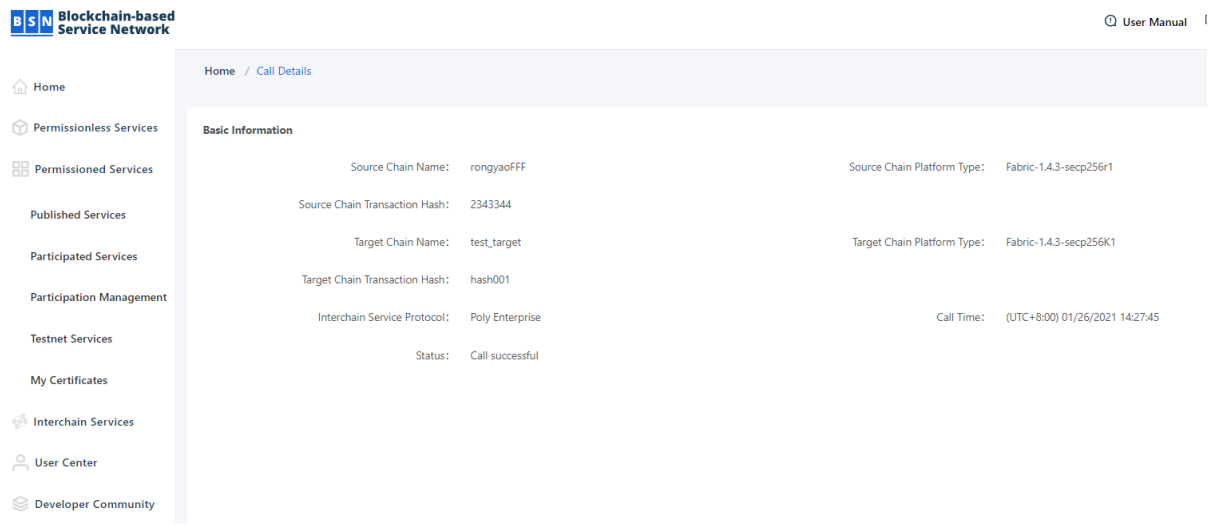
Select the service to be checked, click **View** in the **Action** column, select **Cross-chain Information**, users can check the chain ID, management contract address, management contract name and cross-chain information.



On the Cross-chain Information page, click **Details** button to jump to **Call Details** page. Select the parameter and click **Query** to retrieve the detailed cross-chain call information.



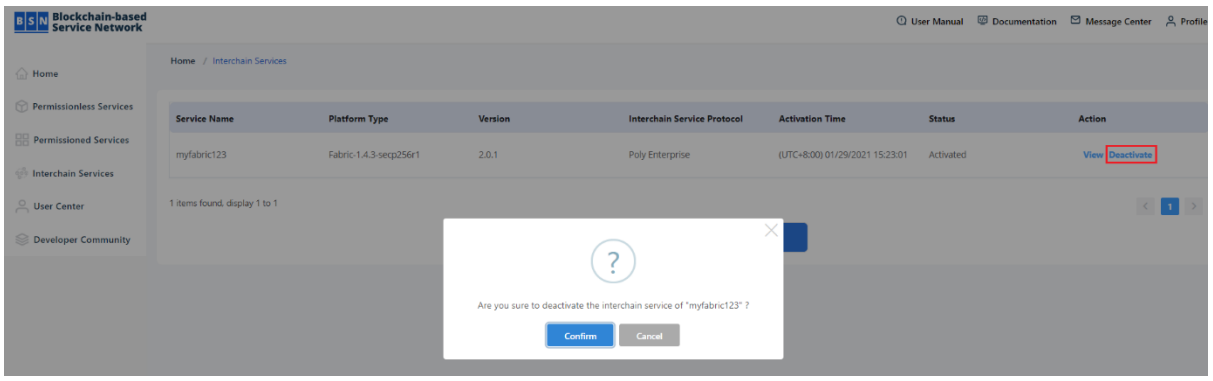
Go to **List of cross-chain call details** section, click **Details** button in **Action** column to enter the Basic Information page, you can view the basic information of the cross-chain call details, as shown in the figure:



7.1.3 Deactivation and Activation of Interchain Services

1) Deactivation of Interchain Services

On the home page, click **Interchain Services**, users can see a list of their activated interchain services. Select the service which needs to be deactivated and click **Deactivate** button in **Action** column.

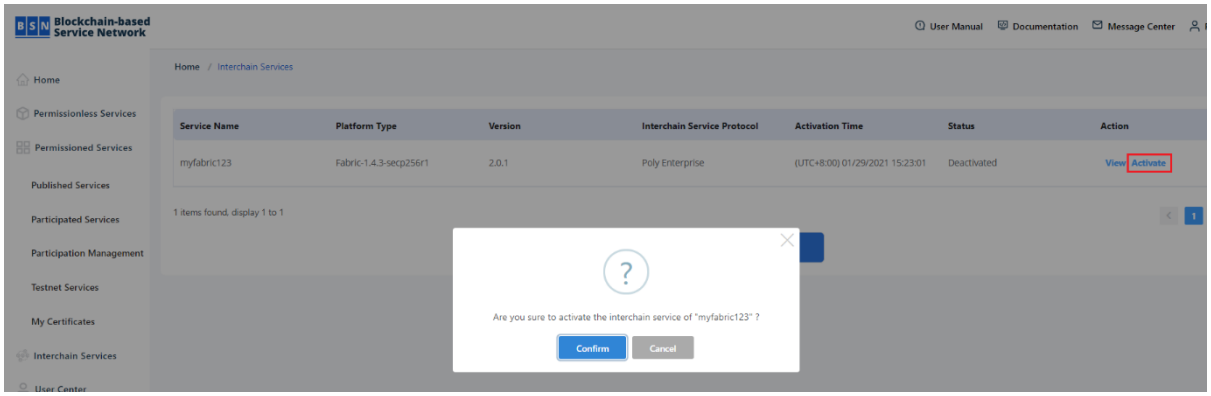


Click **Confirm** in the pop-up message to deactivate the interchain service.

Note: It takes a few minutes to deactivate the interchain service, please be patient.

2) Activation of Interchain Services

On the home page, click **Interchain Services**, users can see a list of their activated interchain services. Select the service which needs to be activated and click **Activate** button in **Action** column.



Click **Confirm** in the pop-up message to activate the interchain service.

7.2 Interchain Services based on Poly Enterprise

7.2.1 Overview

A complete cross-chain transaction requires application contracts for multiple chains. For example, there is an application contract on the Ethereum Ropsten and a FISCO BCOS application contract on the BSN. These two contracts can interact across chains through the cross-chain protocol to ensure the correctness of the information. The cross-chain contract includes a management contract and an application contract. The management contract implements the core logic of the cross-chain protocol, developed by the BSN development team and is deployed in each chain; the application contract needs to be implemented by blockchain application publishers according to the cross-chain protocol and deployed in the blockchain network.

Management contracts include the following implementations.

1. ETH and FISCO BCOS

- EthCrossChainManager: contains logic of management.
- EthCrossChainData: used to save and manipulate data.
- EthCrossChainManagerProxy: used to implement logical contract upgrades.

2. Neo

- CCMC: contains the logic of management.

3. Fabric

- CCMC: contains the logic of management.

4. BSN Testnet Cross-chain management contract address

The following table shows the framework names, chain IDs, and cross-chain contract names or addresses for Poly Enterprise-based cross-chain services.

Testnet	Framework	Chain ID	Cross-chain contract names/addresses	Application Example Contract
China	Fabric	8	ccm	myhellopoly

	FISCO BCOS	6	0x315bF6da1f040355d283CCc41f04e8933Ff98f01	0x69d0ba0866ee3d9abd19b06ad8ac6f49023e19b8
International	Fabric	9	ccm	myhellopoly
	FISCO BCOS	7	0xaF92fAe702C24CF5B214645AdFE25821b5664667	0xbc393345ab22eea8f16ac98dd430657903e19e73
Ropsten	Ethereum	2	0xF6993b7d73B2827420689Dbc0b3068D24E6e467F	0x0b89e4f2103c4700de5ae96f370f3708c5572211
Testnet	Neo	4	0x10b6edbb6e44188d0ff39065442081b13bbd109b	0x0ea9e760ca350d950d01b32c35127b3f7c0c18b5

The application cross-chain contains the following functions:

1. Outbound: The source chain's application contract initiates a cross-chain transaction request and transfers this request from the source chain to the target chain. The user can call a self-defined method in the source chain's application contract which calls the ‘crossChain’ method of the management contract. This will send the cross-chain data through events.
2. Inbound: The target chain application contract receives the cross-chain transaction request. This request information sent from the source chain is passed to the target chain application contract. The cross-chain management contract receives and verifies the cross-chain information. The cross-chain protocol requires the target chain application contract and function name to be included in the cross-chain information. Then the management contract invokes the specified method for the specified contract address and passes the information to the target chain application contract.

7.2.2 Interchain Services based on Hyperledger Fabric

7.2.2.1 Application Contract Development Guide in BSN production environment

The development of Fabric application contract is based on its own business scenario. The main implementation includes two parts: if the source chain initiates a cross-chain transaction, its application contract needs to get outbound to access the target chain; if the target chain receives a cross-chain transaction, its application contract needs to get inbound. Fabric's chain IDand cross-chain management contract’s name are automatically assigned and generated through the BSN operations and maintenance system when users open interchain services, and can be viewed in the BSN portal..

Below is an example of a source chain initiating a cross-chain transaction call:

```
/**
 * @Author AndyCao
 * @Date 2020-11-4 18:27
 * @Description: This method is used for calling other target chains across the chain. (This method can be self-defined)
 * @Param
 * @Return
```

```
 **/
func (t*HelloPoly) say(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
   setLogger("The say method is called.....")
   defer setLogger("End called say method.....")
   // Checking configuration
   if len(args) != 4 {
      return shim.Error("Parameter error！ ！ ！ ")
   }
   // Save data
   if err:=stub.PutState(args[1],[]byte(args[2]));err!=nil{
      return shim.Error(fmt.Sprintf("Failed to save data: %v", err))
   }
   //Build cross-chain management contract calling parameters
  // args[0] is the target chain ID
  // args[1] is the target chain application contract name (or contract address)
  // args[2] is the calling method name of the  target chain
  // args[3] is the calling parameter

    ccArgs := []string{"crossChain",
   args[0],
   hex.EncodeToString([]byte(args[1])),
   args[2],
   hex.EncodeToString([]byte(args[3]))}   // Calling cross-chain management contract
   var resp = stub.InvokeChaincode(string("ccm"), invokeArgs, "")
   if resp.Status != shim.OK {
      return shim.Error(fmt.Sprintf("Failed to call the cross-chain management
contract %s: %s", "ccm", resp.Message))
   }
   // Set event notification
   if err := stub.SetEvent("from_ccm", resp.Payload); err != nil {
      return shim.Error(fmt.Sprintf("Event setting failed: %v", err))
   }
   setLogger("Successfully call the cross-chain management contract for cross-chain: (target
chain ID: %d, target chain contract address: %x, cross-chain message: %s)", args[1], args[1],
args[2])
   return shim.Success(nil)
}
func packArgs(args []string) [][]byte {
   r := [][]byte{}
   for _, s := range args {
      temp := []byte(s)
      r = append(r, temp)
   }
   return r
}
```

Below is an example of a target chain call when receiving a cross-chain transaction:

```
/**
 * @Author AndyCao
```

```
 * @Date 2020-11-4 18:28
 * @Description: This method is used for calling other target chains across the chain. (This
method can be self-defined)
 * @Param
 * @Return
 **/
func (t *HelloPoly) hear(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
   setLogger("The hear method is called.....")
   defer setLogger("End called hear method.....")
   // Checking parameters
   if len(args) != 1 {
      return shim.Error("Parameter error！ ！ ！ ")
   }

   // Save the cross-chain information committed by the source chain
   if err:=stub.PutState("FABRIC_CROSS_CHAIN",[]byte(args[0]));err!=nil {
      return shim.Error(fmt.Sprintf("Failed to save data: %v", err))
   }
   return shim.Success([]byte("SUCCESS"))
}
```

7.2.2.2 Application Contract Development Guide in BSN Testnet

Fabric's chain ID in the BSN China Testnet is 8 and in the BSN International Testnet is 9. This chain ID is registered in Poly Enterprise, not the channel ID corresponding to Fabric itself. The name of Fabric cross-chain contract is ccm.

Below is an example of a source chain initiating a cross-chain transaction call:

```
/**
 * @Author AndyCao
 * @Date 2020-11-4 18:27
 * @Description: This method is used for calling other target chains across the chain. (This
method can be self-defined)
 * @Param
 * @Return
 **/
func (t*HelloPoly) say(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
   setLogger("The say method is called.....")
   defer setLogger("End called say method.....")
   // Checking configuration
   if len(args) != 4 {
      return shim.Error("Parameter error！ ！ ！ ")
   }
   // Save data
   if err:=stub.PutState(args[1],[]byte(args[2]));err!=nil{
      return shim.Error(fmt.Sprintf("Failed to save data: %v", err))
   }
   //Build cross-chain management contract calling parameters
   invokeArgs := make([][]byte, 6)
```

```
  // Set the calling method of the cross-chain management contract
  invokeArgs[0] = []byte("crossChain")
  // Set the chain corresponding to the target chain in Poly network
  invokeArgs[1] = []byte(args[0])
   // Set the target chain’s smart contract address, note：
      // 1. If the target chain is fabric, then this is the name of the application contract, for example：mycc; if the target chain is fisco/eth/neo, then this is the name of the application contract, for example: 11..., don't add "0x" prefix.
      // 2. The parameters passed to the cross-chain management contract must be EncodeToString converts bytes to hex strings and then transform to byte arrays.
if args[0]=="8"||args[0]=="9"{
   invokeArgs[2] = []byte(hex.EncodeToString([]byte(args[1])))
 }else{
   invokeArgs[2] = []byte(args[1])
        }
  // Target chain’s application contract method
  invokeArgs[3] = []byte("hear")
  // Cross-chain information that target chain’s application contract needs to pass（Note: The parameters passed to the cross-chain management contract must be EncodeToString converts bytes to hex strings and then transform to byte arrays.）
  invokeArgs[4] = []byte(args[2])
  // Name of the application contract
  invokeArgs[5] = []byte(args[3])
  // Calling cross-chain management contract
  var resp = stub.InvokeChaincode(string("ccm"), invokeArgs, "")
  if resp.Status != shim.OK {
    return shim.Error(fmt.Sprintf("Failed to call the cross-chain management contract %s: %s", "ccm", resp.Message))
  }
  // Set event notification
  if err := stub.SetEvent("from_ccm", resp.Payload); err != nil {
    return shim.Error(fmt.Sprintf("Event setting failed: %v", err))
  }
  setLogger("Successfully call the cross-chain management contract for cross-chain: (target chain ID: %d, target chain contract address: %x, cross-chain message: %s)", args[1], args[1], args[2])
  return shim.Success(nil)
}
```

Below is an example of a target chain call when receiving a cross-chain transaction:

```
/**
 * @Author AndyCao
 * @Date 2020-11-4 18:28
 * @Description: This method is used for calling other target chains across the chain. (This method can be self-defined)
 * @Param
 * @Return
 **/
func (t *HelloPoly) hear(stub shim.ChaincodeStubInterface, args []string) peer.Response {
  setLogger("The hear method is called.....")
```

```
  defer setLogger("End called hear method.....")
  // Checking parameters
  if len(args) != 1 {
    return shim.Error("Parameter error！ ！ ！ ")
  }

  // Save the cross-chain information committed by the source chain
  if err:=stub.PutState("FABRIC_CROSS_CHAIN",[]byte(args[0]));err!=nil {
    return shim.Error(fmt.Sprintf("Failed to save data: %v", err))
  }
  return shim.Success([]byte("SUCCESS"))
}
```

7.2.2.3 Demo Contract Example

BSN production environment:

https://github.com/BSNDA/ICH/tree/main/sample/polychain/fabric-contract/online/hellopoly

BSN Testnet:

https://github.com/BSNDA/ICH/tree/main/sample/polychain/fabric-contract/testnet/hellopoly

7.2.3 Interchain Services based on FISCO BCOS

7.2.3.1 Application Contract Development Guide in BSN production environment

The development of FISCO BCOS application contract is based on its own business scenario. The main implementation includes two parts: if the source chain initiates a cross-chain transaction, its application contract needs to get outbound to access the target chain; if the target chain receives a cross-chain transaction, its application contract needs to get inbound.

Below is an example of a source chain initiating a cross-chain transaction call:

```
/**

 * @dev: Implements a cross-chain call by invoking the “say” method

 * @param _toChainId: The chain ID corresponding to the target chain being called in the
Poly network

 * @param _somethingWoW: Parameters passed across the chain

 * @return bool

 **/

function say(uint64 _toChainId, bytes _somethingWoW) public returns (bool){

   //  Get the cross-chain management contract interface

   IEthCrossChainManagerProxy eccmp =
IEthCrossChainManagerProxy(managerProxyContract);
```

```
    // Get the cross-chain management contract address

    address eccmAddr = eccmp.getEthCrossChainManager();

    //  Get the cross-chain management contract object

    IEthCrossChainManager eccm = IEthCrossChainManager(eccmAddr);

    //  Get the target chain application contract address

    bytes memory toProxyHash = proxyHashMap[_toChainId];

    //  Call across the chain

    require(eccm.crossChain(_toChainId, toProxyHash, "hear", _somethingWoW),
"CrossChainManager crossChain executed error!");

    emit Say(_toChainId, toProxyHash, _somethingWoW);

    return true;

}
```

Below is an example of a target chain call when receiving a cross-chain transaction:

```
/**

 * @param _somethingWoW: Parameters passed across the chain

 * @param _fromContractAddr: The address of the application contract being invoked

 * @param _toChainId: Contract framework chainId being called

* @return bool

 **/

function hear(bytes _somethingWoW, bytes _fromContractAddr, uint64 _toChainId) public
returns (bool){

    hearSomeThing = _somethingWoW;

    emit Hear(_somethingWoW, _fromContractAddr);

    return true;

}
```

7.2.3.2 Application Contract Development Guide in BSN Testnet

FISCO's chain ID in the BSN China Testnet is 6 and in the BSN International Testnet is 7. This chain ID is registered in Poly Enterprise, not the group ID corresponding to FISCO itself.

The application contract example in BSN test network is the same as the production environment, please visit 7.2.3.1 Application Contract Development Guide in BSN Production Environment for details.

7.2.3.3 Demo Contract Example

BSN Production Environment and Testnet:
https://github.com/BSNDA/ICH/tree/main/sample/polychain/fisco_contracts/contracts

7.2.4 Interchain Services based on Ethereum Ropsten

Application Contract Development Guide

The development of the ETH application contract is based on its own business scenario. The main implementation includes two parts: if the source chain initiates a cross-chain transaction, its application contract needs to get outbound to access the target chain; if the target chain receives a cross-chain transaction, its application contract needs to get inbound. ETH's chain ID in the BSN Testnet is 2. This chain ID is registered in Poly Enterprise and the configuration is applicable to both BSN Production Environment and Testnet.

Below is an example of a source chain initiating a cross-chain transaction call:

```
/**

* @dev: Implements a cross-chain call by invoking the “say” method

 * @param _toChainId: The chain ID corresponding to the target chain being called in the Poly network

 * @param _somethingWoW: Parameters passed across the chain

* @return bool

 **/

function say(uint64 _toChainId, bytes _somethingWoW) public returns (bool){

  // Get the cross-chain management contract interface

   IEthCrossChainManagerProxy eccmp = IEthCrossChainManagerProxy(managerProxyContract);

   // Get the cross-chain management contract address

   address eccmAddr = eccmp.getEthCrossChainManager();

   // Get the cross-chain management contract object

   IEthCrossChainManager eccm = IEthCrossChainManager(eccmAddr);
```

```
    // Get the target chain application contract address
    bytes memory toProxyHash = proxyHashMap[_toChainId];
    // Call across the chain
    require(eccm.crossChain(_toChainId, toProxyHash, "hear", _somethingWoW),
"CrossChainManager crossChain executed error!");
    emit Say(_toChainId, toProxyHash, _somethingWoW);
    return true;
}
```

Below is an example of a target chain call when receiving a cross-chain transaction:

```
/**
* @param _somethingWoW: Parameters passed across the chain
 * @param _fromContractAddr: The address of the application contract being invoked
 * @param _toChainId: Contract framework chainId being called
 * @return bool
 **/
function hear(bytes _somethingWoW, bytes _fromContractAddr, uint64 _toChainId) public
returns (bool){
    hearSomeThing = _somethingWoW;
    emit Hear(_somethingWoW, _fromContractAddr);
    return true;
}
```

Demo Contract Example

GitHub: https://github.com/BSNDA/ICH/tree/main/sample/polychain/eth_contracts/contracts

7.2.5 Interchain Services based on Neo Testnet

Application Contract Development Guide

The development of Neo application contract is based on its own business scenario. The main implementation includes two parts: if the source chain initiates a cross-chain transaction, its application contract needs to get outbound to access the target chain; if the target chain receives a cross-chain transaction, its application contract needs to get inbound. Neo's chain ID in the BSN Testnet is 4. This chain ID is registered in Poly Enterprise the configuration is applicable to both BSN Production Environment and Testnet.

Below is an example of a source chain initiating a cross-chain transaction call:

```
/// <summary>

    /// This method is used to make cross-chain calls to other target chains (this method is self-defining)

    /// </summary>

    /// <param name="toChainId">The chain ID of the target chain in the Poly network</param>

    /// <param name="msg">The cross-chain information that the target chain needs to pass to apply the contract</param>

    /// <returns></returns>

    [DisplayName("say")]

    public static bool Say(BigInteger toChainId, byte[] msg)

    {

        // Get the application contract on the target chain

        var toProxyHash = HelloPoly.GetProxyHash(toChainId);

        // Get the CCMC contract address

        var ccmcScriptHash = HelloPoly.GetProxyHash(neoChainID);

        // Call across the chains

        bool success = (bool)((DynCall)ccmcScriptHash.ToDelegate())("CrossChain", new object[] { toChainId, toProxyHash, "hear", msg });

        HelloPoly.Notify(success, "[HelloPoly]-Say: Failed to call CCMC.");

        // Event notification

        HelloPoly.SayEvent(toChainId, toProxyHash);

        return true;

    }
```

Below is an example of a target chain call when receiving a cross-chain transaction:

```
/// <summary>

    /// This method is used to make cross-chain calls to other target chains (this method is self-defining)

    /// </summary>

    /// <param name="fromChainId">The chain ID of the source chain in a Poly network</param>

    /// <param name="toChainId">The chain ID of the target chain in the Poly network</param>

    /// <param name="msg">Receive a cross-chain message sent by the source chain</param>

    /// <param name="callingScriptHash">Callback script hash</param>

    /// <returns></returns>

    [DisplayName("hear")]

    public static bool Hear(byte[] inputBytes, byte[] fromProxyContract, BigInteger fromChainId, byte[] callingScriptHash)

        //commit into ledger

        Storage.Put(fromProxyContract, inputBytes);

        // Event notification

        HearEvent(fromChainId, fromProxyContract, inputBytes);

        return true;

    }
```

Demo Contract Example

GitHub: https://github.com/BSNDA/ICH/tree/main/sample/polychain/neo_contracts

7.3 Interchain Services based on IRITA

7.3.1 Overview

BSN interchain services based on IRITA is a cross-chain network and is a part of the ICH. In the interchain communication process, developers initiate an interchain service call by their application contracts, and after receiving this call, the relayer of that application chain will initiate a cross-chain request to the ICH. After receiving the request, the service provider transfers the request to the target chain, acquires the transaction result and returns it to the ICH. Finally, the relayer returns the transaction result to the application chain by calling the cross-chain contract.

The application contract, cross-chain contract, relayer, ICH, service provider and services are

working together to complete the entire process of interchain services.

Testnet	Framework	Cross-chain contract names/addresses
China	Fabric	cc_cross
	FISCO BCOS	0xdaa3b22adeef09c9416f01db654035ba8c729522
International	Fabric	cc_cross
	FISCO BCOS	0xcdad7e31edb24fc5f7c1aaf9edb8b8640d2fe3ca

7.3.2 Interchain Architecture based on IRITA

1. Cross-chain Contract

a. Service Market

Service Market is a module responsible for managing interchain service information. It contains the following functions:

- Add Service Binding: Add interchain service information to the application chain, including: service name, service description, service provider, etc.
- Update Service Binding: Function to modify the service binding information.
- Get Service Bindings: Query the available interchain services.

b. Service Core

As the core of the entire cross-chain contract, Service Core is responsible for receiving a cross-chain request from the application contract, the request result from the relayer, and return the cross-chain call result, including the following functions:

- Call Service: Function to make a cross-chain request. It is invoked by the application contract by passing the interchain service name, input parameters, callback contract info, etc. A successful invocation will return a unique request ID.
- Set Response: Return the cross-chain invocation result called by relayer and write the result to the application chain. If an application contract method needs to be called back during service invocation, it needs to be called in this function.
- Get Response: Query the cross-chain invocation result by request ID.

2. Application Contract

Application contract is developed by the developer for the interchain services. In an application contract, the developer can invoke the call service function of the cross-chain contract to make a cross chain call. A callback function in the application contract also needs to be provided to get the call result, otherwise developers will have to call the Get Response function in an application contract to query the service invocation results.

3. Relayer

Relayer is a service that listens for the cross-chain request submitted to ICH by a source chain. It is responsible for listening the call service function event, and submitting to the ICH.

4. Service Provider

Service provider, connects the HUB to the destination chain. It is a service that listens for the cross-chain request from the ICH, invokes the destination chain, and returns the transaction results to the ICH.

5. Interchain Service

The interchain service is developed by the service provider. It could be the smart contract deployed on one blockchain for an application on another blockchain to call across the chain. Support from the service provider is required to turn a service into an interchain service.

6. Interchain Communications Hub

As the core component of BSN's interchain service, the ICH is responsible for receiving cross chain requests submitted by the source chain Relayer, verifying the transaction and initiating the cross-chain transaction with the Service Provider of the destination chain. After the transaction is complete, it is also responsible for obtaining the transaction result from the Service Provider, verifying and returning the result to the source chain Relayer.

7.3.3 Interchain Services in BSN Testnet

In the BSN Testnet, the MintBase contract deployed on ETH Ropsten and Store contract deployed on FISCO BCOS are available for interchain services. To experience the interchain services, developers can publish Fabric or FISCO BCOS application contracts, and call application smart contracts between them.

Developers need to develop application contracts or use sample contracts provided by BSN to make interchain service invocations.

7.3.3.1 Interchain Application Contract based on Fabric

Interchain Application Contracts Development Guide

1. Preparation:

   Run the code below to obtain the BSN interchain consuming contract help package (ICH.git). Currently, only the GO language version is supported but more versions will be added later.

   ```
   cd $GOPATH
       mkdir -p src/github.com/BSNDA && cd src/github.com/BSNDA
       git clone https://github.com/BSNDA/ICH.git
   ```

2. Initiate the interchain service request

   After creating the Fabric chain code struct and invoke function, import below package:

   ```
   import (
       "github.com/BSNDA/ICH/sample/irita/consumers/fabric/crosschaincode"
       )
   ```

3. Call the crosschaincode.CallService method in the invoke function using the parameters of the method as follows:
   - stub: shim.ChaincodeStubInterface
   - serviceName: the interchain service name to invoke, nft for ETH, and bcos-store for FISCO BCOS
   - input: the input object for interchain service
   - callbackCC: callback chaincode name
   - callbackFcn: callback chaincode function name
   - timeout: timeout

The input parameter varies according to the interchain service and type passed in. In the ETH service, the input structure is as follows:

```
type Input struct {
                ABIEncoded   string `json:"abi_encoded,omitempty"`
                To           string `json:"to"`
                AmountToMint string `json:"amount_to_mint"`
                MetaID       string `json:"meta_id"`
                SetPrice     string `json:"set_price"`
                IsForSale    bool   `json:"is_for_sale"`
               }
```

In the FISCO BCOS service, the input structure is as follows:

```
type BcosInput struct {
                    Value string `json:"value"`
                   }
```

A unique request ID will be returned after successful invocation. Keep this value and use it to determine the cross-chain results in the callback function.

4. Implement the callback interface：

After the cross chaincode receives the service response from Fabric Relayer, the callback method name and callback chaincode name passed in will be called to return the service response. The first parameter of the call returns a JSON-formatted string. Below is the service response structure:

```
type ServiceResponse struct {
    RequestId   string `json:"requestID,omitempty"`
    ErrMsg      string `json:"errMsg,omitempty"`
    Output      string `json:"output,omitempty"`
    IcRequestId string `json:"icRequestID,omitempty"`
    }
```

Method crosschaincode.GetCallBackInfo() could be called to Serialize the value. The requestID is unique and can be used to conduct the business processing. Output a JSON-formatted string which is the return value of the crosschain response. Below is the input data structure:

```
type InputData struct {
    Header interface{} `json:"header"`
    Body interface{}        `json:"body"`
    }
```

Parameter “Body” is the output object for the service. In ETH service, the input structure is as follows:

```
type Output struct {
    NftID string `json:"nft_id"`
    }
```

In FISCO BCOS service, the output structure is as follows:

```
type BcosOutput struct {
    Key string `json:"key"`
    }
```

5. Package the chaincode

   ICH.git, which is imported by the chaincode, needs to be packaged with the chaincode together by govendor. If you haven't installed govendor, you can install it as below:

   Install govendor:

   ```
   go get -u -v github.com/kardianos/govendor
   ```

   Execute in the main method directory:

   ```
   govendor init
   govendor add -tree github.com/BSNDA/ICH/sample/irita/consumers/fabric/crosschaincode
   ```

   After execution, the vendor folder will be generated. For the last step, compress the project and vendor folder together, then upload and deploy it in BSN portal.

Application Contract Example

GitHub: https://github.com/BSNDA/ICH/tree/main/sample/irita/consumers/fabric/chaincode

7.3.3.2 Interchain Application Contract based on FISCO BCOS

The interchain consuming contracts based on iService with Solidity are applicable for the EVM compatible application blockchain platform like Ethereum and FISCO BCOS.

Application Contracts Development Guide

IService Client: For convenience, the contract named iService Client is built to encapsulate the interaction with the iService Consumer Proxy and handle logistics including event triggering, request validation and status maintaining, which helps improve development efficiency.

The iService Client source code can be found in the Example Contract.

1. **Import iService Client**

   To use the iService Client, import the corresponding contract path. For example:

   ```
   import ServiceClient.sol
   ```

Note: You can directly use the iService Client code as a part of the consuming contract.

2. **Inherit iService Client**

   ```
   contract <consuming-contract-name> is iServiceClient {

   }
   ```

3. **Initiate the interchain service request**

   - **Set the iService Consumer Proxy (i.e. iService Core Extension) contract address.**

     The contract address in BSN Testnet is **0xcdad7e31edb24fc5f7c1aaf9edb8b8640d2fe3ca**

     This can be performed by invoking the setIServiceConsumerProxy method inherited from the iService Client. A constructor taking the iService Consumer Proxy address can be used. E.g.

     ```
     constructor(
         address _iServiceConsumerProxy
     ```

```
    )
public
    {
    setIServiceConsumerProxy(_iServiceConsumer);
    }
```

- Implement the callback interface

  When the iService Consumer Proxy receives the service response, the method implementing the callback interface will be called to paginate the response to the corresponding consuming contract.

  The below is the callback interface signature.

```
function callback(
    bytes32 _requestID,
    string calldata _output
    )
```

- **Initiate iService invocation**

  The iService request can be sent by the sendIServiceRequest function in the iService Client.

```
bytes32 memory requestID = sendIServiceRequest(
  serviceName,
  requestInput,
  timeout,
  address(this),
  this.callback.selector
  );
```

  Note: Developers need to retrieve information related to the service from the iService Market Ex deployed on the application chain for the interchain service invocation, such as the service name and schemas of the input and output.

4. NFT Service Consuming Contract Example

The NFT service is provided by the NFT contract on the Ethereum Ropsten to create NFT assets.

The definition of the NFT service is as follows:

- Service name：nft
- Service Input JSON Schema：

```
{
"type": "object",
"properties": {
```

```
"to": {
"description": "address to which the NFT will be minted",
"type": "string"
},
"amount_to_mint": {
"description": "amount of the NFT to be minted",
"type": "string"
},
"meta_id": {
"description": "meta id",
"type": "string"
},
"set_price": {
"description": "price in Ethereum Wei",
"type": "string"
},
"is_for_sale": {
"description": "whether or not the minted NFT is for sale",
"type": "boolean"
}
}
}
```


- Service Output JSON Schema：


```
{
"type": "object",
"properties": {
        "nft_id": {
        "description": "id of the minted NFT",
        "type": "string"
        }
}
}
```


Developers can develop contracts on application chains to implement to mint NFT assets across chains.

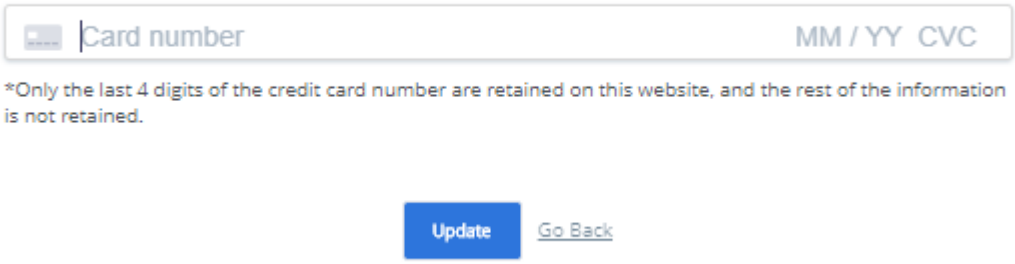
Application Contract Example

GitHub: https://github.com/BSNDA/ICH/tree/main/sample/irita/consumers/fiscobcos/NFTServiceConsumer

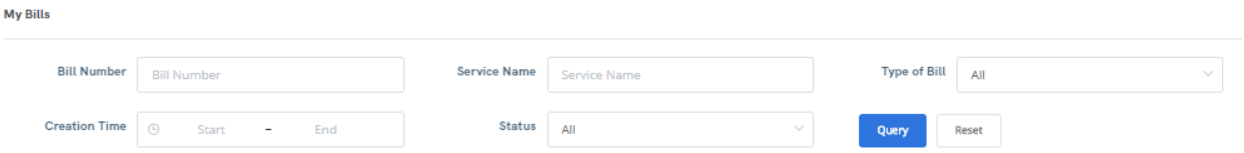
8 Account Management

In the **My Account** page, the user can view details of their card and transactions they performed on the network. To work with **My Account**, follow these steps:

1. In the **User Center** menu, click the dropdown to reveal the list, in the menu list, click **My Account** to display the page.
2. To update the user **Card Information**, click the **Update card information** to display the **My Credit Card** page. The user will be redirected to the Stripe website. The BSN portal can never see and does not store credit card information.
3. Update the card details as needed and click **Update**.



4. To search a bill in the **My Bills** section, enter or select the following:
   - **Bill Number** - Enter the bill number if known
   - **Creation time** - Select a start and end date
   - **Service Name** - Enter a service name if known
   - **Status** - Select from the options available in the dropdown
   - **Type of Bills** - Select from the options available in the dropdown
   - Click **Query** to display the bill information.



5. In the **Bill list**, under the **Status** and **Action** columns, the user can perform certain actions including **Pay** and **Details** on each bill. To **pay** a bill, click **Pay** and to **View** a bill, click **Details**.



9 Online Documentation

White Papers			
Name	**Version**	**Update**	**Details**
BSN Introduction White paper	V1.05	February 5th,2020	PDF
BSN Technical White Paper	V1.0.0	April 25th,2020	PDF

Site Documents			
Name	**Version**	**Update**	**Details**
User Manual	1.4.0	January 31st,2021	Online PDF
Fabric Examples	1.0.1	April 24th,2020	Github
FISCO BCOS Examples	1.0.1	April 24th,2020	Github
SDK Examples	1.0.1	April 24th,2020	Github

Permissioned Frameworks		
Name	**Official Website**	**Details**
Hyperledger Fabric	https://www.hyperledger.org/	Github Documentation
FISCO BCOS	http://fisco-bcos.org/	Github Documentation

Public Chains		
Name	**Official Website**	**Details**
Nervos	https://www.nervos.org/	Github Documentation
NEO	https://neo.org/	Github Documentation
ETH	https://ethereum.org/	Github Documentation
Tezos	https://tezos.com/	Github Documentation
EOS	https://eos.io/	Github Documentation
IRISNET	https://www.irisnet.org/	Github Documentation
dfuse-eos	https://www.dfuse.io/en/home/?utm_source=BSN	Github Documentation
Algorand	https://algorand.foundation/	Github Documentation
Solana	https://solana.com/	Github Documentation

ShareRing	https://sharering.network/	Github Documentation
BTY	https://www.bityuan.com/index	Github Documentation
Oasis Network	http://www.oasisprotocol.org	Github Documentation
Polkadot	https://polkadot.network/	Github Documentation

10 Contact Us

If you have any questions or find any errors in this manual, please contact us:

Customer service hotline: +86-400-071-8215 (workday: 08:00 - 17:30)

Email: support@bsnbase.com

Telegram BSN Support Group: https://t.me/bsnsupport

International Social Media:

