

# Player Accounts Overview

## Introduction

Player accounts represent a collection of data spread out over MongoDB collections. This document will walk you through how these accounts are accessed, secured, and managed. Account features at a glance:

- Players can log in either as an Anonymous Account tied to a specific device or log in with Single Sign On (SSO).
- SSO accounts can be accessed from multiple devices and are never lost so long as the SSO link exists.
- Players can upgrade Anonymous accounts to SSO through their account screen in the game client by logging into their chosen provider.
  - If logging into an existing SSO account, the player is presented with a choice to orphan an account of their choosing; they can keep the current Anonymous account, the existing SSO account, or cancel the login.
- Rumble provides its own internal SSO, and passwords for this are *never sent across a network*, but rather a calculated hash.
- Accounts are secured by JSON Web Tokens (tokens), a message format containing an account ID, permissions, screen name, and encrypted email address (only if using SSO).
- Services use tokens to identify players.
- Token permissions limit which services they can interact with.
- Tokens are publicly inspectable, but contain a signature that can be used to verify authenticity.
- Permissions can be revoked for specific services or features for abuse, allowing players to still access the game in a limited capacity, or blocked completely.
- Platform is a collection of microservices (services), each responsible for specific functionality. Player data is spread out across multiple MongoDB collections, tied together by Account IDs.
- Data is typically server-authoritative.

## Login

### Anonymous Login Flow (game client only)

1. A player downloads the Towers & Titans application to their device and opens it. The application creates and stores a unique *install ID* on the local device.
2. The player taps "Play as Guest" on the initial menu screen. On future launches, the initial menu screen will be skipped.
3. The application sends this ID and basic device information to **player-service**, which returns a JSON Web Token (token) that represents a unique account tied to the device information.
  - a. If the *install ID* has not been seen before, a new account is created.
  - b. If the *install ID* exists on the database, the matching account is accessed.
4. If the player uninstalls the application, access to the account is lost. The only way to retrieve the account is to confirm identity with Customer Service (CS) via purchase confirmations.
  - a. Players can protect against this by signing in with SSO, which ties their account to a secondary identity provider.

## SSO Login Flow (web & game clients)

1. A player downloads the Towers & Titans application to their device and opens it. The application creates and stores a unique *install ID* on the local device.
2. The player taps “Sign In” or “Create Account” on the initial menu screen and selects one of our identity providers (Apple, Google, Plarium, or Rumble).
  - a. “Sign In” will fail if the account doesn’t exist; “Create Account” will fail if the SSO is already tied to an existing account.
  - b. On a successful login, these providers provide a unique external account ID that’s tied to player accounts.
  - c. The client sends encrypted SSO data containing this information to **player-service** which uses the external ID to locate the internal account.
  - d. Security methods vary between providers; see the Security section for more details.
3. **player-service** returns the token representing the player account.
4. If the player uninstalls the application, they can use their SSO to access their account later.

### Child Accounts

When a player uses SSO to log into an existing account from a new device, the following happens on the backend:

1. A new player account is created in the **player-service** database.
2. This new account is *linked* to the account the SSO represents. This is a separate field that references the Platform Account ID ( `parent` ) .
3. The parent account ID is used for token generation instead.

Child accounts can be used to manually link accounts together in situations where Customer Service has verified a player’s identity and has otherwise lost access to their SSO account (or doesn’t have one).

## Account Example

Below is an anonymized sample “Player Record” from the `player-service` database, from the `players` collection. This is solely for reference, but this is the record that’s responsible for logging in a player and contains a complete set of data that would be used for any login.

```
1 {
2   "_id": "deadbeefdeadbeefdeadbeef",
3   "apple": {
4     "iss": "https://appleid.apple.com",
5     "aud": "com.rumbleentertainment.towersandtitans",
6     "_id": "12345.deadbeefdeadbeefdeadbeef.1234",
7     "email": "abcdefghij@privaterelay.appleid.com",
8     "verified": "true",
9     "isPvt": "true",
10    "authTime": 1682438558
11  },
12  "created": 1675197238,
13  "device": {
14    "vClient": "2.7.1.3689",
15    "vData": null,
16    "install": "badf00dbadf00dbadf00dbadf00d",
17    "lang": "English",
18    "vOS": "Windows 10",
19    "t": "G125VY"
```

```

20 },
21 "login": 1699040074,
22 "plarium": {
23   "_id": 97392301,
24   "email": "cloud@example.domain",
25   "login": "cloud@example.domain"
26 },
27 "rumble": {
28   "verified": [ ... ],
29   "code": null,
30   "exp": 0,
31   "email": "cloud@example.domain",
32   "hash": "$2b$06$qEBvLrdCvjcGXqGRsa5aqEBvLrdCvjcGXqGRsa5aqEBvLrdCvjcGX",
33   "status": 2,
34   "username": "cloud@example.domain",
35   "clientLogins": 5,
36   "ip": "71.198.50.59",
37   "logins": 5
38 },
39 "sn": "Cloud Strife",
40 "geo": {
41   "nat": "United States",
42   "cc": "US",
43   "ip": "47.146.120.92"
44 },
45 "google": {
46   "email": "cloud@example.domain",
47   "verified": true,
48   "host": "rumbleentertainment.com",
49   "_id": 123456789012345678901
50   "name": null,
51   "pic": "https://lh3.googleusercontent.com/a-/..."
52 },
53 "linkCode": null,
54 "disc": 7777,
55 "parent": null
56 }

```

Notable Field	Description
_id	Platform Account ID. This is what links all services to a given player, and what issued tokens represent.
apple._id	External Account ID from Apple SSO.
google._id	External Account ID from Google SSO.
plarium._id	External Account ID from Plarium SSO.
rumble.email	The email a player signed up with for Rumble's in-house SSO.
rumble.hash	The password hash calculated by the client. <i>This is not the password, nor player passwords stored anywhere in our database, encrypted or otherwise.</i>
parent	A link to another player account. This is only used for <i>Child Accounts</i> .

## Data Storage

Platform is a collection of microservices (services). Each service has its own specific data stored in a separate MongoDB collection that cannot be accessed directly by other services. MongoDB collections have the name format of `{service}-{environment number}` unless otherwise noted.

Service	Player data stored (not including Account IDs)
alert-service	None by default; developers add any relevant data they need for logs.
calendar-service	None
chat-service	User-generated messages
dmz-service (tower-admin-portal)	Email addresses; whenever we send email or an email bounces, entries are added.
dynamic-config	None
guild-service	User-generated guild names, guild rank
leaderboard-service	Scores earned from in-game activities
mailbox-service	Rewards, notifications, and CS grants
match-making-service	
multiplayer-service	
nft-service	
player-service	Device & SSO information, screen names, discriminators, and game progression data such as campaign, replays, team compositions, currencies, and quests. This is the primary data storage for accounts.  Also stores the player salt (see Security) if used.
pvp-service	
receipt-service	Purchase histories
telemetry-service	
token-service	Screennames, email addresses, and the last 10 tokens used


MongoDB has regularly-scheduled backups throughout each day. Backups are currently stored up to 13 months.

## Personal Identifiable Information (PII)

PII refers to any data that can be used to uniquely identify an end-user. In order to be compliant with GDPR, COPPA, and similar guidelines, this data must be deleted on request. PII can be but is not limited to: any user-generated content like chat messages or screennames, device information, location data, or email addresses.

Services only store player data that is relevant to their operation. **player-service** is the largest repository for PII, as it contains device information, IP addresses, email addresses, and everything else necessary to identify a player. Other services, unless they have a pressing need for specific PII,

should only use the Account ID from tokens. The Account ID by itself does not constitute PII, as it's just a 24-digit hex value assigned by MongoDB to uniquely identify a player record.

 In addition to MongoDB used by Platform services, a large amount of PII is piped to Amazon Redshift for telemetry purposes, some to logging solutions like Loggly, and a very small amount to Slack if a developer uses Slack interoperability.

When purging PII, all of these sources will also need to be purged for compliance.

## Security

When a player logs in to **player-service**, the service authenticates a player based on the following criteria:

- The install ID matches an existing player record, OR
- An external SSO ID matches one stored on an existing player record OR
- The player is using a Rumble SSO login, which requires an email address and password hash to match

SSO login will always be more secure, and as such is preferred. While install IDs are unique to every device, this is an unchanging text value. SSO logins instead are only valid for a brief time window, determined by the service provider, so they cannot be reused indefinitely. Platform has to validate each type differently:

Login Method	Validation method
Device Install ID	<ul style="list-style-type: none"><li>• Clients send the application's unique generated install ID to <b>player-service</b>, which finds the account with a matching install ID.</li></ul> <p><i>This is a naive plaintext match on an unchanging value and is the least secure method.</i></p>
Apple	<ul style="list-style-type: none"><li>• Clients log in to Apple services which returns a token.</li><li>• Clients send the token to <b>player-service</b>, which then downloads keys from an Apple API, and decrypts the token to obtain an external ID.</li></ul>
Google	<ul style="list-style-type: none"><li>• Clients log in to Google services which returns a token.</li><li>• Clients send the token to <b>player-service</b>, which uses a Google API to obtain an external ID.</li></ul>
Plarium	<ul style="list-style-type: none"><li>• Clients log in to Plarium services which returns either a Plarium token (app) or a code (web) which can be used to get a Plarium token.</li><li>• Clients send the response to <b>player-service</b>, which uses Plarium-specific private keys to get an external ID from Plarium.</li></ul>
Rumble	<ul style="list-style-type: none"><li>• Clients send <b>player-service</b> an email address, which returns a randomly generated string (salt).</li><li>• Clients use the returned salt to convert a plaintext password into a hash. This uses BCrypt to protect against brute force attacks. This can be configured to force logins take longer if desired.</li><li>• Clients send the email and hash to <b>player-service</b> which must match values on a player account.<ul style="list-style-type: none"><li>◦ For account creation, players must confirm their account by clicking on a verification links sent to their email before they can log in.</li></ul></li></ul>

- The password is never sent over the network.

**!** Platform trusts that these SSO providers are securely authenticating the players themselves; once these logins provide an external account ID, **player-service** uses it to locate a matching player record and trusts the player is who they say they are. Consequently, it is theoretically possible that a security flaw in a third-party SSO provider could compromise our player accounts. In such a situation and if it's severe enough to warrant immediate action, Platform can disable logins from providers by redeploying an existing build of **player-service** without the necessary variables to authenticate with the affected SSO provider.

Once an account is found using one of the above methods, **player-service** compiles the following information and sends it off to **token-service** to obtain a token:

- Platform Account ID
- Expiration (4 days from issue)
- Time Issued
- Permission Set
- Screen Name
- Discriminator, a 4-digit identifier allowing players to use non-unique screennames
- Last known IP Address
- Last known Country (based on IP)

**token-service** encodes this data into a token. **player-service** returns the token to the client, which can then use it in all API requests to identify itself.

## Tokens

An example token looks like this:

```
eyJhbGciOiJIJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhbnQ0IjkiZmRlYWRiZWVhZGVhZGJlZWIjLCJleHAiOiJlE3MTAxMDk2NzAsImIzcyI6IiJ1bWJsZSBub2t1biBTZXJ2aWNlIiwiaWF0IjoxNzA5Njc3NjcwLCJwZXJtIjo2ODM4NzgsInNuIjojoiQ2xvdWQgU3RyZWZlIiwiaWZCI6Nzc3NywiQCI6InN4bEF5a2dtTHk1OGdWMGtvSGFTRGdrR0Y2dFRlYitSbkJ0aFBUMUp0SEU9IiwicmVxIjoicGxheWVyLXNlcnZpY2UgKGRldikiLCJna2V5IjojNTc5MDFjNmRmODJhNDU3MDgwMThtYTYwMDQifQ.McMha6pvKZxXzKbZUz481rDwIw0gONmefDbX2sxijwNgBtf84qKRi7w58o2u_H9lR22ioDNHS7P5GByge080g
```

Tokens are *publicly inspectable*, meaning that the information contained within it is visible to anyone who wants to see what's in it. You can test this out yourself by going to <https://jwt.io/> and pasting in a token, which yields the following output:


```
1 {
2   "aid": "deadbeefdeadbeefdeadbeef",
3   "exp": 1710109454,
4   "iss": "Rumble Token Service",
5   "iat": 1709677454,
6   "perm": 683878,
7   "sn": "Cloud Strife",
8   "d": 7777,
9   "@": "sx1AykgmLy58gV0koHaSDgkGF6tTHb+RnBNhPT1JtHE=",
10  "req": "player-service (dev)",
11  "gkey": "57901c6df82a45708018ba73b8d16004"
12 }
```

To break down these fields:

Field	Description
aid	The Platform Account ID.
exp	The time the token expires, as a Unix timestamp.
iss	The name of the service that generated the token. This is only for internal auditing and has no bearing on functionality.
iat	The time the token was Issued <b>At</b> .
perm	The player's permission set.
sn	The player's screenname.
d	The player's discriminator.
@	The encrypted email address for the player. Only <b>token-service</b> has the keys to decode this.
req	The requesting service, also known internally in the codebase as "origin".
gkey	The game key, a unique value that can identify a token with a specific game or deployment.

## Token Security Features

- **token-service** is the single point of entry for token generation and validation.
- Tokens come in two flavors: player tokens and admin tokens.
  - Player tokens represent a player, and have limited access and capabilities. For example, a player token cannot be used in a request that alters a player's currencies.
  - Admin tokens, however, can be a requirement for API endpoints that impact players. These can represent services or company representatives who have the authority to make these changes.
- All tokens have a permission set defined. The permission set determines which services or features they can be used with. See *Permissions* for more information.
- By default, tokens are valid for 4 days, but can be configured when generating the token.
  - For player tokens, 4 days is the maximum value. This allows a player to be in-game without interruption for this time period. A shorter time period can be used in **player-service**, but this will force clients to log in again more frequently.
  - For admin tokens, 10 years is the maximum value. After this, tokens will need to be re-generated.
  - When a token expires, traffic will be denied until a new token is obtained.
- The last 10 tokens generated for an account are stored, and valid if not expired.
- Tokens have a signature that is environment-specific. This is how **token-service** validates that it's the one that issued the token and that the token doesn't come from a different deployment, e.g. using a dev token in production.
- Existing tokens can be invalidated, for example, as the result of a permissions change.
  - If admin tokens are compromised in any way, they can be deleted and automatically regenerated, and all admin tokens can be invalidated in a single call to **token-service**.

 To reduce traffic load and services performant, token authorizations are cached in memory for 15 minutes. This means that if a token is invalidated, it can take up to a maximum of 15 minutes before a service will re-evaluate the token. There are ways to force this refresh manually if necessary.

## Permissions

Every service and a few specific features are identified as Permissions (or internally in the codebase, “Audience”). These are managed as a single binary integer, and passed into token generation to identify which services or features a token is valid in. As a short example, say that a permission set is defined as:

Digit Position (from right)	Corresponding Service / Feature
1	chat-service
2	dynamic-config
3	leaderboard-service
4	mail-service
5	player-service

A player token with the permission set to a binary number `11101` (decimal: 29) would represent a token capable of accessing all the services listed except **dynamic-config**. Since dynamic config is used to configure services themselves, there's no situation in which we should allow a player's token to be used to access it; after all, it contains sensitive information. **player-service**, being the entry point for all clients, is responsible for determining player token permissions.

However, this also comes into play when a player is banned, either temporarily or permanently, from a service or feature. Should a player post messages in violation of our community guidelines, they might be banned from **chat-service**, in which case their tokens would be invalidated and future token permissions would be the binary number `11100` (decimal: 28). Any and all requests to chat-service using a token without the requisite permission will be denied.

Permissions provide a highly granular method of access control, and can also be used on admin tokens. Doing so on admin tokens significantly limits the potential for collateral damage if one ever leaked.

**!!** Leaked admin tokens are extremely dangerous. Admin tokens are used to secure almost all of our API endpoints that can impact players directly, change service configurations, compensation, et cetera. While malicious use of the token would still require knowing API endpoints and structure, these are the keys to the kingdom.

Admin tokens are stored in Dynamic Config. Access to this feature is therefore necessarily limited in production environments.

If any admin token is known to have been leaked or otherwise exposed in a place it shouldn't have been, such as posted into Slack, even by accident, that token should be invalidated and replaced as soon as possible.

## Further Reading

Technical documentation for these features can be found here:

- [Implementing Login for Clients](#)
- [token-service Readme](#)
- [Revoking Player Permissions with Bans](#)