HAI Gaming | Casino integration

Tequity-compatible seamless wallet
English technical guide | 14 November 2026 | Contract: hai-wallet-v1

1. Connect and launch

You keep the player's wallet. We host the game, generate results and call your wallet for each stake and win. Implement the four callbacks in this guide; the game handles its own RGS API calls.

Scope: paid-game launch and wallet settlement. Open the returned game URL directly or in an iframe. Tequity's separate browser `postMessage` control API is not provided by the current 232 game launch; do not depend on its mute, freeze, exit or lifecycle messages.

Integration sandbox: `https://sandbox.hai-gaming.com` - test funds only; no scheduled data resets. **Game IDs:** `4-pots-of-egypt`, `4-lucky-diamonds`, `book-of-sparta`, `candy-swipe`. Use only games/currencies enabled for your assigned operator.

Exchange during onboarding

- You provide: a public HTTPS wallet base URL, casino/brand identifiers, technical contact, requested currencies, and a funded test-player key.
- We provide: your assigned `wallet` and `operator` identifiers and the confirmed RGS base URL.
- Exchange a shared HMAC secret of at least 32 characters through an agreed private channel. Keep it on your server.
- Provide a fresh one-time player key for our wallet conformance check. Launches remain disabled until it passes.

Launch from your server

Create a short-lived, single-use key tied to the player and their wallet currency. Call this endpoint with `Content-Type: application/json`; replace the sample identifiers with your assigned values:

```
POST /tequity/wallet/your-wallet/launch/real
```

```
{
  "operator": "your-operator",
  "game": "4-pots-of-egypt",
  "key": "ONE_TIME_PLAYER_KEY",
  "currency": "try",
  "language": "en"
}
```

Success: HTTP 200 with `{"url":"<player launch URL>"}`. Open the returned URL for that player without modifying it. The RGS first authenticates the key against your wallet. No HMAC or operator bearer header is required on this launch endpoint: the one-time key authenticates the player through your wallet. Never put the shared secret in a launch request.

The game URL is private to the player. Do not log keys, tokens or full launch URLs. If a launch response is lost, create a new one-time key; do not reuse a consumed key. The path says `real` for wallet-backed play, including development test funds.

2. Implement authentication and balance

All four callbacks go to your wallet base URL. Requests have JSON bodies and `X-Server-Authorization`, a lowercase hexadecimal HMAC-SHA256 of the **exact UTF-8 request-body bytes**, using the shared secret. Verify it before processing, with a constant-time comparison. Do not parse and reserialize JSON before verifying.

`/balance`, `/transaction` and `/cancel` also carry `Authorization: Bearer <token>` using the player token you returned from `/authenticate`. Bind that token to the player and currency. `/authenticate` has no player bearer token.

POST /authenticate

Example request from the RGS:

```
{
  "operator": "your-operator",
  "key": "ONE_TIME_PLAYER_KEY",
  "wallet": "your-wallet",
  "provider": "232rgs",
  "game": "4-pots-of-egypt"
}
```

Validate and consume the key once. Return HTTP 200:

```
{
  "nativeId": "player-123",
  "token": "PLAYER_SESSION_TOKEN",
  "balance": 1000,
  "currency": "try",
  "brand": "your-brand"
}
```

All five response fields are required. `nativeId` must be stable and unique across your operator's players and brands. `currency` is lowercase and must match the requested currency. `brand` is your non-empty brand identifier. The RGS may also send `ip` and `channel`; accept these optional fields. Optional wallet session limits are documented in the full contract.

POST /balance

```
{
  "nativeId": "player-123",
  "game": "4-pots-of-egypt",
  "provider": "232rgs"
}
```

Return HTTP 200 with the current balance, for example `{"balance":1000}`. A balance query never moves funds. Use JSON numbers in **major currency units**: `50` means TRY 50.00, and `12.5` means TRY 12.50. Do not send strings, integer kurus, negative values or more than two decimal places for TRY. Use decimal or integer-minor-unit accounting internally.

3. Implement transactions and cancellation

PUT /transaction

A normal paid round has one `withdraw` for the total stake and one `deposit` for the win. Example stake:

```
{
  "nativeId": "player-123",
  "transactionId": "11111111-1111-4111-8111-111111111111",
  "type": "withdraw",
  "provider": "232rgs",
  "amount": 50,
  "game": "4-pots-of-egypt",
  "roundId": "33333333-3333-4333-8333-333333333333",
  "category": "normal",
  "currency": "try"
}
```

Debit TRY 50.00 atomically and return HTTP 200 `{"balance":950}`. The win uses the same fields with **a different transaction ID**, `"type":"deposit"`, the win amount, and `"roundFinished":true`. It keeps the same `roundId`. For a TRY 75.00 win, return `{"balance":1025}`. A loss still sends a zero-amount deposit to finish the round.

- Store each `transactionId` durably with its player, currency, operation and amount. Apply it at most once, atomically with the balance change. A matching repeat returns the current balance without another movement; reject conflicting reuse.
- Accept both zero withdrawals and zero deposits. A repeated successful request must not become an insufficient-funds error.
- Validate the player/token/currency binding. Accept optional `ip`. Do not calculate the stake from game mode: debit the exact `amount` sent by the RGS, including any ante or bonus-buy cost.

DELETE /cancel

```
{
  "nativeId": "player-123",
  "transactionId": "11111111-1111-4111-8111-111111111111",
  "roundId": "33333333-3333-4333-8333-333333333333"
}
```

Refund the referenced withdrawal once and return HTTP 200 with the current balance. Repeating the cancel does nothing further. **Cancel an unseen ID too:** record a durable cancellation and return the balance. A cancelled transaction ID must never debit later, even if the original request arrives after the cancel. Serialize competing transaction/cancel requests in your database. Deposits are never cancelled by this flow.

4. Errors, testing and production handover

Wallet errors and recovery

Return a non-2xx status with this JSON shape. For insufficient funds, use HTTP 400:

```
{
  "error": {
    "code": "INSUFFICIENT_FUNDS",
    "message": "Insufficient funds"
  }
}
```

Use `PLAYER_UNAUTHORIZED` or `SESSION_EXPIRED` for an invalid player session; `SERVER_UNAUTHORIZED` for failed signature verification. Other recognized final errors are `LOSS_LIMIT`, `TRANSACTION_FAILED` and `BLOCKED_TERRITORY`. A final rejection must mean the operation was not applied. Never return a final failure after committing a balance movement.

Timeouts, unreadable responses, unknown codes, `UNKNOWN` and `TRANSACTION_NOT_FOUND` are ambiguous. The RGS resends stored transaction bytes with the same ID. An unresolved withdrawal enters cancellation; an unresolved deposit stays owed and enters retry/manual review. Default recovery has a finite retry budget (10 attempts) before manual review. Never assume a timed-out call failed, or create replacement credits/debits outside reconciliation. Return promptly; the default callback timeout is 5 seconds.

Currency behaviour

Wagers, wins, refunds and balances stay in the player's wallet currency. The game receives its permitted bet levels automatically; limits are set per casino/brand, game and currency. Game math scales the selected bet by an outcome multiplier. No player money is converted to EUR or universal tokens. The casino handles deposits and any deposit-to-wallet conversion.

Reference exchange rates are used for historical EUR reporting, not player-wallet conversion or daily bet repricing. Accepted rounds retain their original reporting rate. Missing, stale or quarantined rates stop new wagers in that currency while accepted-round recovery remains available. Enable each additional currency with us before launching it.

Acceptance checklist

1. Pass our wallet conformance check using a funded test player: balance, duplicate withdrawal, duplicate cancel, unseen-ID cancel, cancelled-ID rejection, zero transactions and insufficient funds. The player must finish at its starting balance.
2. Test game launch, a win, a loss, insufficient funds, duplicate callbacks, concurrent cancel/withdraw, and a dropped response after your wallet commits a debit or credit.
3. Refresh/reopen during an unfinished round and verify recovery, one debit/credit per ID, and exact TRY balance reconciliation. Check expired keys/tokens and invalid signatures.
4. Configure RGS base URL, wallet callback URL, HMAC secret and assigned wallet/operator/brand identifiers per environment; update network allowlists if used. Production has fresh credentials/data and must pass conformance again. The agreed callback contract and game IDs stay the same on AWS; no integration rewrite is required.

The hosted demo player-key helper is evaluation-only. For production, integrate your own wallet above. Paid play is the initial scope; free rounds need separate enablement. See the full Tequity wallet contract and free-rounds contract. Share transaction/round IDs and redacted errors with us for support, never player tokens or the HMAC secret.