IVR Integration Guide: Implement ScrambleID Voice (Twilio + NICE Patterns)

Voice & Contact Center / Last updated 2026-06-11 / https://www.scrambleid.com/learn/ivr-integration-guide

In one sentence: You integrate ScrambleID Voice into an IVR by creating a per-call session, speaking a short **Dynamic Identifier (DID)**, waiting in a controlled loop, and letting ScrambleID **intercept/redirect** the live call as soon as the DID is confirmed in-app.

> **Scope:** the integration surface documented here is the spoken-code webhook flow. When a caller's number matches a registered device, the ScrambleID app offers an in-app push confirmation as the default fast path; that's part of the caller experience, not a separate integration. Everything your IVR builds (sessions, prompts, wait loops, intercepts) is the spoken-code flow described below.

TL;DR (canonical)

- The IVR never collects secrets; it only speaks a **short-lived, single-use DID**.
- The authenticator is the ScrambleID app, which confirms the DID with device-bound keys.
- Your IVR must be ready for **retries** (idempotency), **timeouts**, and **late confirmations**.
- The key engineering requirement is **fast call redirection** on confirmation, fast enough that the caller never notices a gap.

What does the IVR integration architecture look like?

```
Caller → IVR: reaches verification node
IVR → ScrambleID: create auth session (bound to call id)
ScrambleID → IVR: DID + TTL + suggested prompts
IVR → Caller: speaks DID
Caller → ScrambleID App: enters DID
App → ScrambleID: signed confirmation
ScrambleID → IVR Provider: redirect/intercept live call
IVR → Caller: success handler / routing
```

What identifiers and bindings are required?

> **Binding to the live call:** the verifier records the IVR-supplied call session id (CallSid for Twilio, interaction id for Genesys, contact id for NICE) in the session record at session-start. A confirmation that does not reference the same call session id is rejected. The full cryptographic specification (signed payload structure, atomic verification, race-condition handling) is in Session binding cryptography.

At minimum, bind the ScrambleID session to your IVR's stable call identifier:

- **Twilio:** CallSid
- **NICE / Genesys / other:** provider call id / contact id

This binding is what prevents "confirmation replay" from affecting the wrong call.

What are the implementation steps?

Step 1, Create a per-call session

When the caller enters the verification node, your IVR hits your backend (or directly ScrambleID) to create a session.

Request (illustrative)

```
POST /scramble/caller/v1/sessions
Content-Type: application/json

{
  "provider": "twilio",
  "callId": "CA123...",
  "dnis": "+18005551234",
  "ani": "+14155550000",
  "locale": "en-US",
  "metadata": {"crmCaseId": "CASE-9911"}
}
```

Response (illustrative)

```
{
  "sessionId": "csn_abc",
  "did": "123456",
  "ttlSeconds": 60,
  "maxRetries": 3,
  "prompts": {
    "speak": "Your verification code is 1 2 3 4 5 6.",
    "loop": "Open the ScrambleID app and enter the code. Press 1 when done, or 0 to cancel."
  }
}
```

Step 2, Speak DID and enter a wait loop

The wait loop pattern is critical for UX and abuse control.

Recommended behavior:

- Speak the DID clearly (space-separated digits).
- Offer a repeat option.
- Use a bounded loop: **N repeats** or **T seconds**.
- Give deterministic "success / timeout / canceled" outcomes.

Step 3, Confirm in-app (out-of-band)

The ScrambleID app confirms the DID and the session binding. You don't need to handle this in IVR.

Step 4, Intercept / redirect the live call on confirmation

When ScrambleID receives confirmation, it calls back into your provider integration to redirect the live call.

The mechanism varies by provider:

- **Twilio:** Update the call's TwiML URL or redirect to a TwiML bin / webhook.
- **NICE / Genesys:** Use the platform's routing API or flow control primitives.

What patterns work for Genesys and NICE CXone?

If you're not on Twilio, the core idea is the same: **Architect/Studio hosts the wait loop**, while ScrambleID confirmation changes the call state so the flow can route to a success handler.

Genesys Cloud (Architect), recommended pattern

Use Architect to:

1. Call a backend "create session" endpoint (return DID + TTL)
2. Speak DID
3. Loop on a "poll status" endpoint (bounded retries / bounded time)
4. On `confirmed`, route to a "verified" handler flow (or queue)

Implementation primitives to look for in Architect:

- a "call a web service" node (often branded as **Data Actions**)
- a **loop / decision** node that can repeat prompts
- a routing primitive like **Transfer to Flow** to branch into a verified handler

Official reference (Architect routing primitive): https://help.mypurecloud.com/articles/transfer-to-flow-action/

Architect flow, pattern-level pseudocode (your backend hosts the endpoints):

```
Data Action:  POST https://your-backend.example.com/ivr/sessions  → { did, ttlSeconds }
Play Audio:   "Your verification code is <did>"
Loop (bounded, for example 6 polls at 5s):
  Data Action: GET https://your-backend.example.com/ivr/sessions/{id}  → { status }
  Decision:    status == confirmed  → Transfer to Flow: verified handler
On timeout:   route to the fallback queue tagged unverified
```

NICE CXone (Studio), recommended pattern

Use Studio to:

1. Call your backend "create session" endpoint (return DID + TTL)
2. Speak DID
3. Loop on status checks (bounded retries / bounded time)
4. On confirmation, branch to a success script/queue

Practical build blocks to look for in Studio:

- the **REST API** Studio action (preferred for JSON/REST integrations)
- a loop/decision pattern for retries and timeouts

Official reference (CXone Studio API calls):
https://help.nicecxone.com/content/studio/advanced/apis/apisandscripts.htm

Studio REST action, pattern-level example (your backend hosts the endpoints):

```
REST API action:
  POST https://your-backend.example.com/ivr/sessions
  body: { "ani": "{ANI}", "contactId": "{ContactId}" }
  response: { "did": "482913", "ttlSeconds": 60 }
PLAY: speak the code
Loop (bounded retries):
  REST API action: GET https://your-backend.example.com/ivr/sessions/{sessionId}
  CASE status:
    confirmed → branch to the success script
    expired   → reissue once, then route to fallback
```

Five9, compatible via standard webhook patterns

Five9 IVR scripts follow the same shape: call your session endpoint from the script, speak the code, poll on a bounded loop, and branch on confirmation. There is no named ScrambleID Five9 integration today; treat Five9 as compatible via standard webhook patterns, the same contract the Twilio reference implements.

What does a Twilio reference implementation look like?

Below is a safe, production-friendly pattern (simplified).

A) TwiML: speak DID and gather DTMF

```
<Response>
  <Say>Your verification code is 1 2 3 4 5 6.</Say>
  <Gather numDigits="1" action="/ivr/verify/wait" method="POST" timeout="5">
    <Say>Open the ScrambleID app and enter the code. Press 1 when done, or 0 to cancel.</Say>
  </Gather>
  <Redirect method="POST">/ivr/verify/wait</Redirect>
</Response>
```

B) Backend: wait endpoint (idempotent)

Pseudocode:

```
// POST /ivr/verify/wait
// idempotency key: CallSid + loopIndex

const callSid = req.body.CallSid
const digit = req.body.Digits // 1, 0, or undefined

if (digit === '0') {
  await scramble.cancelSession(callSid)
  return twiml.say("Verification canceled.")
}

const status = await scramble.getSessionStatus(callSid)

if (status === 'confirmed') {
  return twiml.redirect('/ivr/verify/success')
}

if (status === 'timeout') {
  return twiml.redirect('/ivr/verify/failure')
}

// otherwise keep waiting
return twiml.redirect('/ivr/verify/prompt')
```

C) Intercept: redirect in-progress call

When ScrambleID confirms, it can call a backend endpoint you expose (recommended), which then updates Twilio:

```
// POST /ivr/intercept
// Verify Scramble signature/HMAC, then:
await twilio.calls(callSid).update({ url: 'https://yourdomain.com/ivr/verify/success', method:
'POST' })
```

Why are idempotency and retries non-negotiable?

Every webhook and status-check endpoint must be idempotent, IVR platforms will retry on timeout, network hiccup, or failover. If your endpoints are not idempotent, retries will create duplicate sessions, double-redirects, or phantom verifications.

Requirements:

- Treat `callId` (CallSid) as a stable session key.
- Make "create session" idempotent (same callId → same pending session).
- Ensure intercept calls are idempotent (multiple intercepts should be safe).
- Terminal states (`confirmed`, `timeout`, `canceled`) must remain terminal.

What events should you instrument?

Emit structured events for:

- session_created
- did_issued
- loop_iteration
- canceled
- timeout
- confirmed
- intercept_success / intercept_failed

These become high-signal metrics and security indicators:

- wrong-DID bursts (vishing pressure)
- elevated timeout rates (UX friction)
- intercept failures (platform integration issues)

What failure modes must you handle safely?

Wrong or expired DID

- The app rejects it; the session remains pending.
- The IVR keeps waiting until timeout.
- Wrong-code bursts should raise an alert.

Late confirmation (after hangup)

Policy decision:

- Either ignore late confirmations
- Or accept as "recently verified" only if correlated to a specific case id and within a tight TTL

ScrambleID outage / degraded mode

Avoid dangerous fallback.

- For sensitive flows, fail closed (route to agent fraud script or deny), consistent with CISA phishing-resistant MFA guidance.
- For low-risk flows, fail open but log degraded.

What security checks are required for IVR integration?

- [ ] Verify signatures/HMAC for inbound callbacks (provider → you, Scramble → you).
- [ ] Use TLS for all webhooks.
- [ ] Rate-limit session creation and status polls.
- [ ] Do not treat ANI/caller ID as identity (see FCC STIR/SHAKEN for network-layer caller ID authentication).
- [ ] Audit every verify attempt with correlation IDs.

Key Takeaway

IVR integration works by creating a per-call verification session, speaking a short-lived Dynamic Identifier (DID), waiting while the caller confirms it via the ScrambleID app, and intercepting (redirecting) the live call once confirmation arrives. Use webhooks with idempotency keys, keep DID TTLs short (60-90s), and always provide graceful fallback for unrecoverable errors, but log fallback usage as a security signal.

FAQ

What does "intercept" mean here?

It means redirecting the in-progress call immediately after the DID is confirmed in-app, so the caller routes to a success handler without waiting for the next loop.

Why include a DTMF "press 1" loop?

It keeps the caller engaged and allows repeat prompts without aggressive polling. It also gives users control to cancel.

Can we keep our existing call tree?

Yes. Caller Auth is designed as a drop-in node in your existing IVR.

Does this work with agents (not just IVR)?

Yes. You can trigger DID issuance from an agent console; confirmation still updates call state.

How fast should redirects be?

Fast enough to feel instantaneous. The caller shouldn't perceive a gap between confirming in the app and the call moving forward.

How do we measure success?

Containment %, time-to-verify, wrong-DID rate, and AHT deltas on the target queues.

References (public)

- CISA guidance on social engineering and phishing: https://www.cisa.gov/news-events/news/avoiding-social-engineering-and-phishing-attacks
- FCC overview of caller ID authentication and STIR/SHAKEN: https://www.fcc.gov/call-authentication
- Twilio Voice, Call resource (redirect / control calls in progress): https://www.twilio.com/docs/voice/api/call-resource
- Twilio TwiML, Gather: https://www.twilio.com/docs/voice/twiml/gather
- Twilio TwiML, Redirect: https://www.twilio.com/docs/voice/twiml/redirect
- Genesys Cloud, Transfer to Flow action: https://help.mypurecloud.com/articles/transfer-to-flow-action/
- NICE CXone Studio, APIs and Scripts (REST API Studio action): https://help.nicecxone.com/content/studio/advanced/apis/apisandscripts.htm

---

Related reading

- Caller Authentication: Replace KBA and Stop Vishing
- KBA Is Dead: Contact Center Playbook
- Metrics + ROI Playbook