



Order Lookup — Integration Guide

Connect your store to Krisp so customers can collect their order credentials in the chat.

Version 1.2

For developers

Updated 2026-10-19

CONTENTS

1. How it works
2. Set it up in five steps
3. The request Krisp sends
4. Verify the signature — always
5. Your response
6. Matching rules
7. What the customer sees when something goes wrong
8. Recognising order IDs
9. Security checklist
10. Testing
11. Turning it off, and changing settings
12. Frequently asked questions

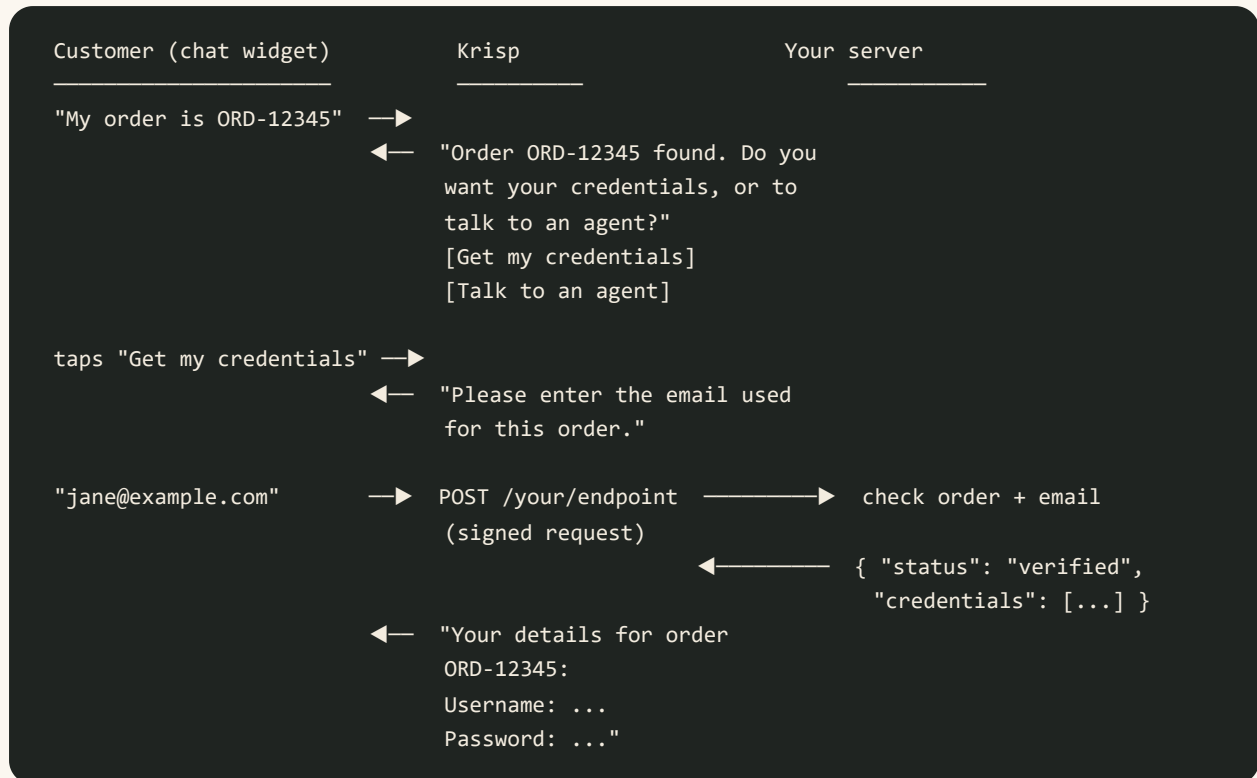
Version 1.2 · Updated 2026-10-19 · For the developer connecting your store or app to Krisp

New in version 1.2: order lookup answers whether the AI chatbot is on or off, and stays out of a conversation your team is handling, including one a teammate has replied in (sections 1 and 12); requests go only to the addresses that were checked, within one 8-second limit for the whole exchange (sections 5 and 7); your own order-ID pattern is timed when you save it and runs under a time limit (section 8); the signing secret is stored encrypted, and emailed copies of delivered credentials are explained (section 9).

Order lookup lets your customers collect what they bought — account logins, licence keys, download codes — straight from the chat. A customer pastes their order ID (or, if they only mention an order, is asked for it), chooses **Get my credentials**, confirms the email they ordered with, and Krisp asks *your* server whether to release the order's details. If your server says yes, the details are posted in the chat. If not, or if the customer prefers, a person from your team takes over.

You build **one HTTPS endpoint**. Krisp never receives a copy of your orders: it asks about one order at a time, and only after the customer has given the email to check.

1. How it works



- A customer who mentions an order without its ID ("I purchased yesterday", "where is my order?") is asked **"What is your order ID?"**. Once they send it, the bot goes straight to the email question. It is not asked before buying ("can I order ten?", "how do I purchase?"), and at most once every 30 minutes in one chat; a reply without an ID lets the question lapse and the message is answered as usual.
- A long ID, such as a UUID, is shortened in what the bot says ("Order 7d1f3c2a... found"). Your server always receives it in full.
- **Talk to an agent** hands the conversation to your team at any point. The bot then stays out of it.
- A wrong email can be tried again (3 tries in all, by default). After that, the conversation goes to your team.
- If your server is down, slow or answers something unexpected, the customer is told so and the conversation goes to your team. Nobody is left waiting on a bot.
- The whole feature has an **on/off switch** in the dashboard. While it is off, order IDs are treated like any other message. The switch is its own: order lookup answers whether the AI chatbot is on or off.
- Order lookup stays out of a conversation your team is handling: one assigned to a teammate or waiting for your team (Pending), and one a teammate has replied in - that one until it is resolved. There an order ID is a message for your team like any other.

2. Set it up in five steps

1. **Build the endpoint** described in sections 3–5. It receives a `POST` with a JSON body and answers with JSON.
2. In Krisp, open **Settings** → **Order lookup**.
3. Enter your **endpoint address** (for example `https://shop.example.com/krisp/order-lookup`) and click **Save connection**. It must be an `https://` address on the public internet.
4. Click **Reveal** next to the signing secret, copy it, and store it on your server as an environment variable (for example `KRISP_ORDER_LOOKUP_SECRET`). Your endpoint uses it to prove each request really comes from Krisp.
5. Use **Send a test request** with a real order ID and email. When the result says *Connected · order verified*, switch **Order lookup** on.

Only workspace owners and admins can change these settings, reveal the secret, send a test request or switch the feature on and off.

3. The request Krisp sends

`POST` to your endpoint, with a JSON body.

Headers

Header	Example	Meaning
<code>Content-Type</code>	<code>application/json</code>	Always JSON
<code>Accept</code>	<code>application/json</code>	Your answer must be JSON too
<code>User-Agent</code>	<code>Krisp-OrderLookup/1.0</code>	Identifies the caller
<code>X-Krisp-Event</code>	<code>order.lookup</code>	The kind of request
<code>X-Krisp-Delivery</code>	<code>5b0c6f0e-...</code>	Unique ID of this request (also in the body as <code>deliveryId</code>)
<code>X-Krisp-Timestamp</code>	<code>1791072000</code>	When it was sent, in Unix seconds
<code>X-Krisp-Signature</code>	<code>v1=8f3c...</code>	HMAC-SHA256 signature — see section 4

Body

JSON

```
{
  "type": "order.lookup",
  "test": false,
  "deliveryId": "5b0c6f0e-6a64-4c3e-9a0e-2f0d3f6f4b1a",
  "lookupId": "c1a2b3d4-0000-4000-8000-000000000001",
  "orderId": "ORD-12345",
  "email": "jane@example.com",
  "workspace": { "id": "ws_...", "slug": "your-shop" },
  "conversationId": "e7f8a9b0-...",
  "channel": "website",
  "visitor": { "id": "...", "externalId": "cust_42", "email": "jane@example.com" },
  "sentAt": "2026-10-04T10:00:00.000Z"
}
```

Field	Type	Meaning
type	string	Always <code>order.lookup</code>
test	boolean	<code>true</code> when sent from the dashboard's Send a test request button. Answer normally, but do not change anything on your side (for example, do not mark the order as delivered)
deliveryId	string	Unique per request. Useful for logging and for ignoring a duplicate
lookupId	string or null	Groups the attempts one customer makes for one order. <code>null</code> for a test
orderId	string	The order ID as the customer typed it — see section 6
email	string	The email the customer typed, lower-cased
workspace	object	Your Krisp workspace
conversationId	string or null	The chat this came from. <code>null</code> for a test
channel	string or null	Where the customer is chatting, and where the credentials will be posted: <code>website</code> (your chat widget or hosted chat page), <code>telegram</code> , <code>whatsapp</code> , <code>instagram</code> , <code>facebook</code> (Messenger) or <code>email</code> . <code>null</code> for a test. Answer <code>not_ready</code> with a <code>message</code> if you do not want credentials released on a channel

Field	Type	Meaning
<code>visit</code> or	object or null	On the website, what your page told the chat widget about the customer (<code>externalId</code> , <code>email</code>), if anything. On another channel, how Krisp knows the customer there: an <code>externalId</code> such as <code>telegram:...</code> , or for email the address the message came from. Treat it as a hint, not as proof — it is whatever the page or the sender said. <code>null</code> for a test
<code>sentAt</code>	string	ISO 8601 time the request was made

New fields may be added in future versions. Ignore fields you do not recognise.

4. Verify the signature — always

Every request is signed with your workspace's **signing secret** (it starts with `olsec_`). Check the signature before you look at anything else in the request. An unsigned or wrongly signed request must be refused with `401`.

The algorithm

1. Read the raw request body exactly as received — before any JSON parsing.
2. Build the string `<X-Krisp-Timestamp>.<raw body>` (the timestamp, a full stop, the body).
3. Compute HMAC-SHA256 of that string with your signing secret, as lowercase hex.
4. Prefix it with `v1=` and compare it with `X-Krisp-Signature` using a constant-time comparison.
5. Reject the request if the timestamp is more than **5 minutes** away from your server's clock. This stops a captured request from being replayed later.

Node.js (Express)

```
import express from 'express';
import crypto from 'node:crypto';

const SECRET = process.env.KRISP_ORDER_LOOKUP_SECRET; // olsec_...
const app = express();

// express.raw keeps the exact bytes, which the signature is computed over.
app.post('/krisp/order-lookup', express.raw({ type: 'application/json', limit: '64kb' })),
async (req, res) => {
  const timestamp = req.get('X-Krisp-Timestamp') ?? '';
  const signature = req.get('X-Krisp-Signature') ?? '';
  const raw = req.body.toString('utf8');

  if (!/^\\d+$/.test(timestamp) || Math.abs(Date.now() / 1000 - Number(timestamp)) > 300) {
    return res.status(401).json({ error: 'stale request' });
  }

  const expected = 'v1=' + crypto.createHmac('sha256',
SECRET).update(`${timestamp}.${raw}`).digest('hex');
```

JS

```

const a = Buffer.from(signature);
const b = Buffer.from(expected);
if (a.length !== b.length || !crypto.timingSafeEqual(a, b)) {
  return res.status(401).json({ error: 'bad signature' });
}

const { orderId, email, test } = JSON.parse(raw);
const order = await findOrder(normaliseOrderId(orderId)); // your own lookup

// Wrong order and wrong email get the same answer - see section 6.
if (!order || order.customerEmail.toLowerCase() !== email.toLowerCase()) {
  return res.json({ status: 'not_found' });
}

if (!order.paid || !order.credentials) {
  return res.json({ status: 'not_ready', message: 'Your order is still being prepared. Please check back in about 10 minutes.' });
}

if (!test) await markCredentialsSent(order.id); // optional bookkeeping

return res.json({
  status: 'verified',
  order: { id: order.id, product: order.productName },
  credentials: [
    { label: 'Username', value: order.credentials.username },
    { label: 'Password', value: order.credentials.password },
  ],
  note: 'Please do not change the password.',
});
});

const normaliseOrderId = (id) => String(id).trim().replace(/^#/, '').toUpperCase();

```

PHP

```

<?php
$secret = getenv('KRISP_ORDER_LOOKUP_SECRET');
$raw = file_get_contents('php://input');
$timestamp = $_SERVER['HTTP_X_KRISP_TIMESTAMP'] ?? '';
$signature = $_SERVER['HTTP_X_KRISP_SIGNATURE'] ?? '';

header('Content-Type: application/json');

if (!ctype_digit($timestamp) || abs(time() - (int) $timestamp) > 300) {
  http_response_code(401);
  exit(json_encode(['error' => 'stale request']));
}

$expected = 'v1=' . hash_hmac('sha256', $timestamp . '.' . $raw, $secret);
if (!hash_equals($expected, $signature)) {
  http_response_code(401);
  exit(json_encode(['error' => 'bad signature']));
}

$data = json_decode($raw, true);
$orderId = strtoupper(ltrim(trim($data['orderId']), '#'));
$order = find_order($orderId); // your own lookup

```

```

if (!$order || strcasecmp($order['customer_email'], $data['email']) != 0) {
    exit(json_encode(['status' => 'not_found']));
}
if (!$order['paid'] || empty($order['credentials'])) {
    exit(json_encode(['status' => 'not_ready', 'message' => 'Your order is still being
prepared.']));
}

echo json_encode([
    'status'      => 'verified',
    'order'       => ['id' => $order['id'], 'product' => $order['product_name']],
    'credentials' => [
        ['label' => 'Username', 'value' => $order['credentials']['username']],
        ['label' => 'Password', 'value' => $order['credentials']['password']],
    ],
    'note'        => 'Please do not change the password.',
]);

```

Python (Flask)

```

import hashlib, hmac, os, time
from flask import Flask, request, jsonify

SECRET = os.environ["KRISP_ORDER_LOOKUP_SECRET"].encode()
app = Flask(__name__)

@app.post("/krisp/order-lookup")
def order_lookup():
    raw = request.get_data()                # exact bytes, before parsing
    ts = request.headers.get("X-Krisp-Timestamp", "")
    sig = request.headers.get("X-Krisp-Signature", "")

    if not ts.isdigit() or abs(time.time() - int(ts)) > 300:
        return jsonify(error="stale request"), 401
    expected = "v1=" + hmac.new(SECRET, ts.encode() + b"." + raw, hashlib.sha256).hexdigest()
    if not hmac.compare_digest(expected, sig):
        return jsonify(error="bad signature"), 401

    data = request.get_json()
    order = find_order(data["orderId"].strip().lstrip("#").upper()) # your own lookup

    if not order or order.customer_email.lower() != data["email"].lower():
        return jsonify(status="not_found")
    if not order.paid or not order.credentials:
        return jsonify(status="not_ready", message="Your order is still being prepared.")

    return jsonify(
        status="verified",
        order={"id": order.id, "product": order.product_name},
        credentials=[
            {"label": "Username", "value": order.credentials.username},
            {"label": "Password", "value": order.credentials.password},
        ],
    )

```

PYTHON

```
note="Please do not change the password.",  
)
```

5. Your response

Answer with HTTP **200** and a JSON body whose `status` is one of three values.

status	Use it when	Other fields
<code>verified</code>	The order exists, the email matches it, and there is something to deliver	<code>credentials</code> (required), <code>order.product</code> , <code>note</code> (optional)
<code>not_found</code>	The order does not exist or the email does not match it	none
<code>not_ready</code>	The order and email match, but there is nothing to deliver yet (payment pending, still being provisioned...)	<code>message</code> (optional) — shown to the customer as written

`verified`

```
{  
  "status": "verified",  
  "order": { "id": "ORD-12345", "product": "Streaming Premium – 1 month" },  
  "credentials": [  
    { "label": "Username", "value": "jane.stream" },  
    { "label": "Password", "value": "s3cret-Pa55" },  
    { "label": "Login page", "value": "https://example.com/login" }  
  ],  
  "note": "Please do not change the password."  
}
```

The customer sees:

Your details for order ORD-12345 (Streaming Premium – 1 month):

Username: jane.stream
Password: s3cret-Pa55
Login page: https://example.com/login

Please do not change the password.

Keep these details private.

not_found

```
{ "status": "not_found" }
```

JSON

not_ready

```
{ "status": "not_ready", "message": "Your order is still being prepared. Please check back in about 10 minutes." }
```

JSON

Limits

Item	Limit
Time to answer	8 seconds for the whole exchange: finding your server, connecting and reading your full answer (aim for under 2)
Response size	64 KB
credentials entries	20 (each needs a non-empty label and value)
label	60 characters
value	1,000 characters (numbers are accepted and shown as text)
order.product	120 characters
note , message	1,000 characters each
The whole message the customer receives	8,000 characters

Text longer than a limit is shortened. Credentials without both a label and a value are skipped.

6. Matching rules

- **Order IDs arrive as the customer typed them.** ord-12345 , ORD-12345 and #ORD-12345 may all mean the same order. Normalise before you look it up — trim spaces, remove a leading # , compare case-insensitively.
- **Emails arrive lower-cased.** Compare them case-insensitively with the email on the order.
- **Never say which part was wrong.** A wrong order ID and a wrong email must both get not_found . Telling them apart would let someone discover which order IDs exist.
- **Your server makes the decision.** Krisp only asks after the customer has typed an email, and caps how many emails one chat may try (3 per order by default, 10 per hour across all orders). The rest is up to you: you can, for example, refuse orders older than 30 days, or deliver each order's credentials only a limited number of times.

7. What the customer sees when something goes wrong

What happens	What the customer sees	Conversation
<code>not_found</code> , attempts left	"That email doesn't match this order. Please try again (2 tries left)."	Stays with the bot
<code>not_found</code> , no attempts left	"That email doesn't match this order. Okay, a teammate will reply here shortly."	Handed to your team
<code>not_ready</code>	Your <code>message</code> (or "This order isn't ready yet. Please check back later, or talk to an agent."), with a Talk to an agent button on the website	Stays open
No answer within 8 s, a connection error, a non-2xx status, a redirect, an answer over 64 KB, invalid JSON, JSON that is not an object, an unknown <code>status</code> , <code>verified</code> with no usable credentials, or an endpoint address that leads to a private network	"I can't check orders right now. Okay, a teammate will reply here shortly."	Handed to your team
More than 10 email checks in one chat within an hour	"Too many tries for now. Okay, a teammate will reply here shortly."	Handed to your team
The customer taps Talk to an agent (or types "agent", "human" ...)	"Okay, a teammate will reply here shortly."	Handed to your team

Redirects are not followed: if your endpoint moves, update the address in the dashboard.

The endpoint must be on the public internet. A private address (`10....` , `192.168....` and the like) cannot be saved. In production the endpoint's name is also looked up at every request, and the request goes only to the addresses that lookup gave, every one of which must be public; your certificate is still checked against the name, and the `Host` header is the name. If any of those addresses is private, no request is sent, the customer is handed to your team as above, and **Send a test request** reports an address that leads to a private network. A name that cannot be found is treated like a server that cannot be reached.

8. Recognising order IDs

By default Krisp recognises IDs like `ORD-12345` , `ORDER#A1B2C3` or `ord_778899` (the letters ORD or ORDER, an optional `-` , `_` or `#` , then 4–20 letters and digits including at least one digit).

If your IDs look different, set your own pattern in **Settings** → **Order lookup** as a regular expression (matched case-insensitively). If the pattern has a capture group, the first group that matched is sent as `orderId`. Only the first 1,000 characters of a message are searched, and an ID longer than 64 characters is cut to 64.

Your IDs look like	Pattern	Sent as <code>orderId</code>
#100234 (Shopify style)	#(\d{4,10})	100234
WC-55821	\bWC-\d{3,8}\b	WC-55821
A1B2-C3D4-E5F6	\b[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}\b	A1B2-C3D4-E5F6
Order 778899	order\s*(?:no\.? number #)?\s*(\d{5,9})	778899
3f2b8c1e-9a4d-4b7e-8c2f-1a2b3c4d5e6f (UUID)	(?<![\w-])([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})(?![\w-])	3f2b8c1e-9a4d-4b7e-8c2f-1a2b3c4d5e6f

The dashboard lets you try a pattern against sample messages before saving it. A pattern is limited to 200 characters. Patterns that could stall the server (a repeated group whose contents also repeat or branch, such as `(\d+)+`, `(a|b)*` or `(?:-[0-9a-f]{4}){3}`) are refused, so write repeated parts out in full, as the UUID pattern above does. A pattern that matches an empty message is refused too, since it would match every message. The ID is sent as the customer typed it, so compare it case-insensitively on your side.

A pattern is also timed when you save it, on long runs of digits, letters and separators, and refused if it is slow on them. Quantifiers in a row that can match the same characters, such as `\d*\d*` or `\w+\w+`, are the usual cause. Pattern changes are limited to 20 a minute per workspace.

Every message is then searched with your pattern under a time limit of 50 ms. A message the pattern runs out of time on is read as having no order ID. A pattern that is slow on a message (over 20 ms, and again on a second try) is set aside for five minutes, even if you change it meanwhile, and until then messages are read as having no order ID. Krisp's default pattern runs in linear time and needs no such limit.

9. Security checklist

- ☐ The endpoint is served over **HTTPS** (required in production).
- ☐ Every request's **signature is checked** with a constant-time comparison, over the raw body, before anything else.
- ☐ Requests with a timestamp more than **5 minutes** old are refused.
- ☐ The signing secret is kept in an environment variable or secret manager — **never in front-end code** or a public repository.
- ☐ A wrong order ID and a wrong email **both** answer `not_found`.

- ❑ Only the details the customer needs are returned. Do not include payment data or anything you would not show them on their account page.
- ❑ Requests are logged with their `deliveryId` — but the credentials you return are **not** logged.
- ❑ If the secret may have leaked, **rotate** it (Settings → Order lookup → Rotate). The old secret stops working at once, so update your server straight away.

About the signing secret. Krisp keeps it encrypted (AES-256-GCM). Only workspace owners and admins can reveal it, and every reveal and rotation is recorded in the workspace's audit log.

About the chat transcript. Delivered credentials are a chat message: the customer can scroll back to them, and your team can see them in the Krisp inbox, like any other message in the conversation. On plans with the email channel (Starter and up) they also go out by email as any other message does: in a conversation on the email channel, to a website customer who gave an email address and left the website without seeing them in the chat, and in the transcript a customer with an email address is sent when your team resolves the conversation (unless you turned transcripts off). Krisp's delivery records keep a copy of each email. Krisp does not store your order data or your server's answer anywhere else.

10. Testing

From the dashboard. Settings → Order lookup → **Send a test request** sends a real, signed request with `"test": true` to your saved endpoint, whether order lookup is switched on or not, and shows what came back: the status, how long it took, and the labels of the credentials — with their values masked after the first two characters.

From the command line. To check your endpoint by hand, sign a request yourself:

```
SECRET='olsec_your_secret_here'
BODY='{ "type": "order.lookup", "test": true, "deliveryId": "manual-1", "lookupId": null, "orderId": "ORD-12345", "email": "jane@example.com", "workspace": { "id": "ws", "slug": "your-shop" }, "conversationId": null, "channel": null, "visitor": null, "sentAt": "2026-10-04T10:00:00Z" }'
TS=$(date +%s)
SIG="v1=$(printf '%s.%s' "$TS" "$BODY" | openssl dgst -sha256 -hmac "$SECRET" -hex | sed 's/^.* //')

curl -i https://shop.example.com/krisp/order-lookup \
  -H 'Content-Type: application/json' \
  -H "X-Krisp-Timestamp: $TS" \
  -H "X-Krisp-Signature: $SIG" \
  --data "$BODY"
```

Local development. Outside production, Krisp also accepts a plain `http://` address on the same machine (for example `http://localhost:4000/krisp/order-lookup`), so you can try the whole flow before deploying. Addresses on your local network are refused there too.

In the chat. Open your chat widget, type a real order ID, tap **Get my credentials** and enter the order's email.

11. Turning it off, and changing settings

- **Switch off:** Settings → Order lookup → toggle off. It takes effect for the next message. Your endpoint address and secret are kept, so switching back on needs nothing else.
 - **Email attempts:** how many wrong emails one customer may try per order before the chat goes to your team (1–10, default 3).
 - **Assistant name:** the name shown on the bot's messages (default "Order assistant").
 - **Rotate secret:** issues a new signing secret. The old one stops verifying immediately.
-

12. Frequently asked questions

Does Krisp store our orders or credentials?

No. It asks your server about one order at a time and stores only the outcome (delivered, handed over, and so on). The credentials appear only in the chat message sent to the customer, and in any email that carries that message (section 9).

What if a customer pastes an order ID while a teammate is already talking to them?

Nothing automatic happens. Order lookup stays out of a conversation your team is handling: one assigned to a teammate or waiting for your team (Pending), and one a teammate has replied in - that one until it is resolved. The same applies if a teammate takes the conversation while your server is answering: the answer is not posted.

Does it work while the AI chatbot is switched off?

Yes. Order lookup has a switch of its own and answers whether the AI chatbot is on or off. With the AI chatbot set to *Off* (Settings → Automated replies → AI chatbot mode, or Settings → Widget → AI chatbot replies), a message that is not for order lookup goes straight to your team as a Pending conversation, and from then on order lookup stays out of that conversation while it is with your team, as above. In every other mode, order lookup answers before the automations and the AI do.

Can we deliver something other than a username and password?

Yes. `credentials` is a list of label/value pairs: licence keys, gift-card codes, download links, PINs — whatever the order includes.

Can the same order be looked up twice?

Yes, unless your server decides otherwise. If you want to limit how often an order's credentials are released, keep a count on your side (use `test` to leave test requests out of it) and answer `not_ready` with a suitable `message` when the limit is reached.

Which channels support it?

The Krisp website chat widget and hosted chat page, where the choices are buttons, and Telegram, WhatsApp, Instagram, Facebook Messenger and email, where the same choices are put in words: the customer replies "credentials" or "agent". The request's `channel` field says which one, so you can refuse a channel by answering `not_ready`. Discord is not supported.

Which IP addresses does the request come from?

The address of your Krisp server. Rely on the signature rather than on an IP allow-list.

Krisp Order Lookup — Integration Guide, version 1.2.