How Link Works

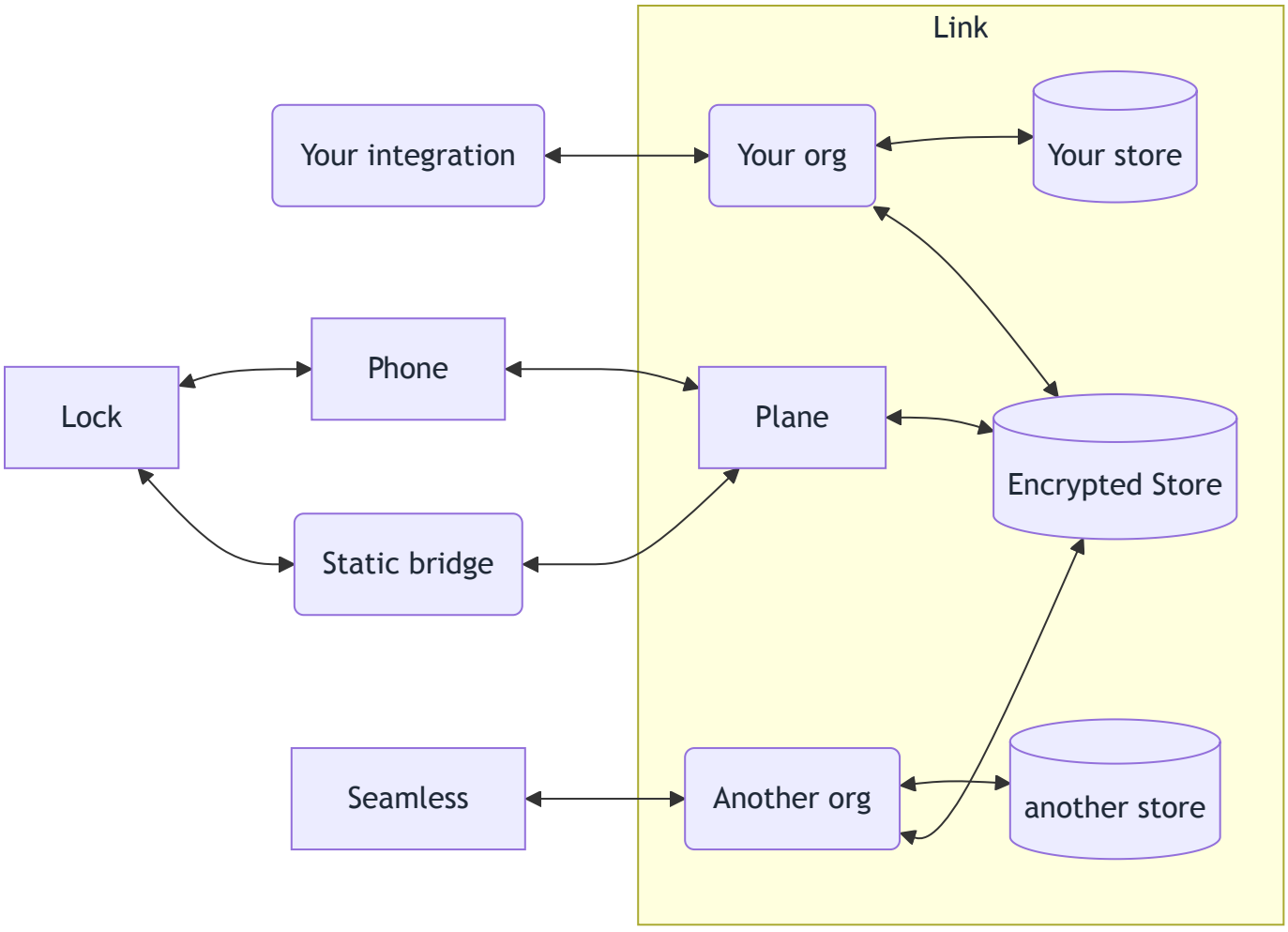
How Link works

Link allows you to control and monitor your Last Lock devices. This tutorial will walk through: finding your device, listening to audits on your device, assigning credentials to your device, streaming audits.

All messages Link sends, both assigning credentials and access audits, are sent encrypted through a network of bridge devices. Bridges can never decrypt messages. But, be sure you have a bridge nearby by to ferry messages between devices and Link.

If you want to make your app a bridge all you will need to do is import the Proxy Kotlin Multiplatform library.

Below is a diagram of Link. Integrations only need to talk directly to their org server. Forwarding of information to and from your device is done by bridges over the Plane server. You do not need to directly integrate with Plane to forward information. You just have to import the Proxy Library.



Talking to your Org server

First let's create an Org client to your Org server:

```
import grpc
import link_pb2
import link_pb2_grpc

host = 'localhost'
channel = grpc.insecure_channel(host+':3001')
org = link_pb2_grpc.OrgStub(channel)
```

Checking that we can find your device

We can do this by asking your Org server for that device's Device Summary. In the below example we use 14, but you should be able to find your device's ID on its packaging.

```
response = org.ListDeviceSummaries(link_pb2.ListDevicesRequest(
        device_ids = [14]
))
print(response)
```

With that you should get a response like this one.

```
device_summaries {
  device_id: 14
  time_created: 1745174255
}
```

You may notice that we passed ids above. Not `id`. This allows you to get many devices in one request. Additionally you can filter by device creation time. We find this useful within access control systems to get all devices that were created after a certain time to easily sync up with new devices.

```
response = org.ListDeviceSummaries(link_pb2.ListDevicesRequest(
        created_at_or_after = 1
))
print(response)
```

Adding a credential to your device

Have at your ready a support credential in Link- in this example we'll use a HID Prox Corporate 1000. and try scanning it with your device. You should see it flash red: no access. We can verify this with.

Note: we have this input here. We'll see how to remove it shortly, but you may have to loop a bit here waiting for the audit to come in.

```
input("I've waited a couple of seconds for an audit")
auditResponse = org.ListAudits(link_pb2.ListAuditsRequest(
        device_ids = [14],
))
print(auditResponse)
```

We will see your credential come up like so. Hopefully your credential looks different!

Let's break this down into parts. You'll see your device ID, an audit message, and a timestamp 'when server'. When server is when this audit reached the server via a bridge. It is a UTC UNIX timestamp. Inside the audit, you'll see an audit type, a credential, and a timestamp 'when device'. Here we see `UnauthorizedUser` as an audit type: makes sense. You'll see a credential with a credential type and the data associated with your credential. Here the PROX credential is the raw data on the prox fob. For other credentials this maybe be a public key or even an id. Not all audits will have credentials: some locks, such as a padlock, have no user associated with locking the device, so the credential field will be missing in this case. Finally you'll see when device: this is when the audit occurred according to the device's clock. The device's clock is not guaranteed to be monotonic: secure time syncs may move the device's clock back.

```
audits {
  device_id: 14
  audit {
    audit_type: UnauthorizedUser
    credential {
      credential_type: HID_PROX_CORP_1000
      data: "010203040506"
    }
    when_device: 1745179900
  }
  when_server: 1745179901
}
```

Let's now set the credentials on the device.

We'll grab the credential from the previous audit and assign your device to just allow access to that credential. This is a *set*: your Org server will internally determine the required messages to send to add and remove credentials from your local. If you do not pass a device ID in the below assignments map it will be unaffected.

While a little chatty, we have found that a *set* is a reliable way to get correct credentials on the device in a efficient way.

```
onlyAudit = auditResponse.audits[-1]
credential = onlyAudit.audit.credential

assignment = link_pb2.AssignmentsRequest(
        assignments = {
                14: link_pb2.Credentials(
                        credentials = [credential]
                )
        }
)
org.Assign(assignment)
```

In a couple seconds (it's pretty fast), you should see your assignment propagated to the device. Scan the same credential again and you will see green.

You can now filter for new audits.

```
input("I've waited a couple of seconds for an audit")
auditRequest = link_pb2.ListAuditsRequest(
        device_ids = [14],
        server_time_after =  onlyAudit.when_server
)
auditResponse = org.ListAudits(auditRequest)
print("unlock", auditResponse)
```

Streaming audits

Say, how can we get seamless performance? Stream! So far we have polled devices for a list of devices. We can also stream audits, getting alerted each time an audit comes through from your device. We'll re use the list request.

```
for audit in org.StreamAudits(auditRequest):
        print("audit", audit, flush=True)
```